YOLO-ReT: Towards High Accuracy Real-time Object Detection on Edge GPUs

Prakhar Ganesh[1*], Yao Chen[1*], Yin Yang[2], Deming Chen[3], Marianne Winslett[3]
[1]Advanced Digital Sciences Center, Singapore
[2]College of Science and Engineering, Hamad Bin Khalifa University, Qatar
[3]University of Illinois at Urbana-Champaign, USA
{prakhar.g, yao.chen}@adsc-create.edu.sg, yyang@hbku.edu.qa, {dchen, winslett}@illinois.edu

Abstract

Performance of object detection models has been growing rapidly on two major fronts, model accuracy and efficiency. However, in order to map deep neural network (DNN) based object detection models to edge devices, one typically needs to compress such models significantly, thus compromising the model accuracy. In this paper, we propose a novel edge GPU friendly module for multi-scale feature interaction by exploiting missing combinatorial connections between various feature scales in existing state-of-the-art methods. Additionally, we propose a novel transfer learning backbone adoption inspired by the changing translational information flow across various tasks, designed to complement our feature interaction module and together improve both accuracy as well as execution speed on various edge GPU devices available in the market. For instance, YOLO-ReT with MobileNetV2$\times$0.75 backbone runs real-time on Jetson Nano, and achieves 68.75 mAP on Pascal VOC and 34.91 mAP on COCO, beating its peers by 3.05 mAP and 0.91 mAP respectively, while executing faster by 3.05 FPS. Furthermore, introducing our multi-scale feature interaction module in YOLOv4-tiny and YOLOv4-tiny (3l) improves their performance to 41.5 and 48.1 mAP respectively on COCO, outperforming the original versions by 1.3 and 0.9 mAP.

1. Introduction

Object detection is one of the most fundamental problems in computer vision, which does both localisation and classification of various objects in an image or a video. Due to its foundational nature, object detection has numerous applications across various domains like unmanned autonomous vehicle (UAV) [46], medical imaging [12], identity verification [45], robot navigation [61, 44], sports analysis [20], *etc.* and has been adapted into more complex problem statements like object tracking [58], action recognition [1], face recognition [32], *etc.* Many of these applications require highly accurate real-time feedback.

Object detection has seen major advancements in recent years both in accuracy as well as efficiency due to the adoption of deep learning algorithms [3, 29]. A majority of research in efficient object detection is dominated by single-stage object detection models that do both localisation and classification in the same network [2, 9, 49], and focuses on real-time execution using Desktop GPUs, like YOLOv4 [2], EfficientDet [49], ScaledYOLOv4 [51], *etc.* However the execution speed of such models (measured in frames per seconds or FPS) drops dramatically on edge devices due to their high computational requirement[62].

Real-time object detection is a vital functionality for modern end-node IoT devices, which due to their disconnection to a centrally located computing platform, do not pack large amounts of computing power and face stricter power supply and latency constraints. However, bringing the power of computing directly to such devices where the data is being collected or consumed, can help us save the data transfer and high computational costs, and improve on response time and other bandwidth issues that would arise in case of an off-location computing. While most of the recent work in the field of lightweight computer vision focuses on reducing the model size and computations [6, 14, 48], it is important to note that these improvements do not directly translate into a faster model. In fact, using these metrics of comparison without executing the model on target device can lead to sub-optimal design [33, 38].

Multi-scale feature interaction is at the heart of modern object detection models [27, 30, 49]. However, with the increasing complexity of these feature interaction modules, the trade-off between efficiency and accuracy is saturating (see Table 3), leaving the need for an innovative feature interaction method. Since most existing methods focus on some combination of the top-down on bottom-up approach to feature collection, these paths leave out a number of possible inter-scale interactions between non-adjacent fea-

[*]Both authors contributed equally to this research.

ture scales that can significantly improve feature refinement for further processing. Additionally, existing feature interaction methods are constrained by the number of output scales, thus missing out on important low-level features.

The direct adoption of transfer learning backbone from classification to detection has been a topic of debate for a long time [67], with a number of research papers even creating their own backbones designed and trained directly on object detection datasets [55, 57]. The increasingly task-specific nature of later layers in a backbone has also been studied extensively [60], and thus directly using pre-trained backbone during transfer learning is clearly not the best adoption of expert knowledge available at our disposal. A more effective backbone adoption is required to achieve the best trade-off between accuracy and efficiency.

In this work, we design a raw feature collection and redistribution module along with an improved truncated backbone adoption during transfer learning that is compatible with various feature extraction backbones and detection heads, and improves both model's execution speed as well as accuracy on edge GPUs. Our contributions include:

- A lightweight raw feature collection and redistribution (RFCR) module that efficiently combines multi-scale features, compatible with various backbones and detection heads. Additionally, the feature collection of our RFCR module is independent of the number of output scales in the detection head, facilitating better feature interaction.
- An extensive experimental analysis of the importance of individual transfer learning layers, together with a truncation method for improved model efficiency. Our truncation and RFCR module complement each other, allowing us to create faster and more accurate detection models.
- An in-depth ablation study with on-device execution latency experiments for edge GPUs, instead of other indirect metrics like MFLOPs or model size, thus providing an accurate comparison of various competing designs.

2. Background

Convolutional Neural Networks (CNNs) in the last decade have made several advancements in the direction of lightweight components which benefits both the feature extraction backbone as well as the head of an object detection model. We elaborate the details in the following.

2.1. Single-stage Object Detection

A modern single-stage object detection model comprises of two components, a feature extractor usually pre-trained on ImageNet [41] and an object detection head responsible for the final output. While CNNs are the go-to choice for feature extraction models, there does exist some work on the exploration of other forms of feature extractors, e.g., extreme learning machines (ELM) [59], motion probability maps [43], *etc.*. Single-stage object detection models can be further divided based on the detection head they use into anchor-based or anchor-free models. Heatmap-based detection models like CornerNet [21], CenterNet [7], *etc.* are common examples of anchor-free models. However, these models require computationally expensive backbones [35] as they rely on keeping high resolution information of the input image intact. Anchor-based detection models on the other hand are the lighter alternatives. For example, YOLOv3 detection head [10] is one of the most commonly used detection head for edge devices, and allows easy integration of lightweight backbones [9, 55, 37, 2, 51].

2.2. Building Blocks

A large section of research in real-time object detection models have been devoted to improving the basic building blocks of CNNs. The traditional CNN layers contain a large number of parameters as well as computations, forcing most such real-time detection models to be significantly shallow networks [9, 10]. Decoupling 2D convolution into depthwise separable and pointwise (1×1) convolutions is a common technique to make networks lighter [37, 16, 49, 64]. Further reducing the number of channels using 1×1 convolutions before applying the intended convolution gave birth to the idea of fire modules [19] and have been adapted in various lightweight detection models [9, 22, 24, 54, 56].

However, using multiple consecutive pointwise convolutions to reduce the computational cost of the information flow infringes on an essential rule of designing fast deep learning models, i.e., network fragmentation [33]. Network fragmentation is a phenomena in which a heavier operation is fragmented into multiple lightweight operations, and significantly hurts the model's execution speed as it interferes with its internal degree of parallelism [33]. For example, MobileDets [57] discovered that grouped pointwise convolutions are not well executed on GPU devices, while ShuffleNetV2 [33] found that pointwise convolutions are fastest when number of input and output channels are the same.

The final feature extraction backbone is formed by combining one or more of the building blocks mentioned above. A number of works have even utilized Neural Architecture Search (NAS) to build their own backbones and detection models [57, 55]. However, these models miss out on transfer learning information present in other pre-trained backbones [23, 25]. On the other hand, backbones pre-trained on existing datasets might contain classification task specific features [26, 57], which can add an unnecessary burden of feature calculation. Thus, an efficient adaptation of a pre-trained backbone from classification to object detection also plays a major role in the model's final performance.

2.3. Multi-scale Feature Fusion

Multi-scale feature interaction is a vital part of the object detection head, both in single-stage as well as two-stage object detection models. Existing methods of feature interaction take some combination of either the top-down or bottom-up approach for the flow of information across multi-scale features [17, 27, 30, 34, 36, 49, 18, 65]. Feature Pyramid Networks (FPN) [27] were the first to create a top-down path from high-level feature scales towards low-level feature scales, with the purpose of using well processed deeper features to help improve the accuracy of detection layers using shallower features. Path Augmentation Networks (PANet) [30] took it a step further and showed that an additional bottom-up path can help further improve the detection accuracy of high-level features.

Building on the success of FPN and PANet, NAS-FPN [4, 11] attempted to find the optimal paths of information flow between various multi-scale features. Since such architecture search based models are designed specifically for certain datasets and backbone networks, it is difficult to generalise them to a wider range of applications. However, these searches reveal interesting trends that can help us learn more about the inherent requirements of such models. NAS-FPN designs revealed the presence of direct connections between various feature scales not adjacent to each other, showing that the flow of information only through adjacent scales might become convoluted and warrants the need of such shortcut connections. Similarly, NAS-FPN also revealed the importance of repeatedly following the top-down and bottom-up path that was later adopted by BiFPN [49] to further improve model accuracy.

Not only the path taken to combine multi-scale features together, but a lot of work have also been done on how various features are combined. While most existing work simply concatenates feature maps from multiple scales together, weighted or attention-based fusion of features have also been proposed [23, 49] to better highlight more important feature scales. Another aspect of fusing features is bringing them to a common scale. Simpler solutions for this includes upsampling or downsampling one of the feature scales to match the other. However, this can entail a local positional mismatch between various scales, and thus multiple ways have also been explored to process the features before and after fusion to facilitate better flow of information across various scales [50, 61, 5].

3. Proposed Solution

In this section, we introduce a class of object detection models, YOLO-ReT, which uses our RFCR module and transfer learning inspired backbone truncation to improve both accuracy and efficiency on edge GPUs.

3.1. Raw Feature Collection and Redistribution

We first introduce our raw feature collection and redistribution (RFCR) module. Following our discussion in Section 2.3, we look to strengthen the raw features provided by the backbone using an improved feature interaction network to increase the detection accuracy, without causing any significant harm to the execution speed. While we focus specifically on detection in this paper, our RFCR module can be generalized to feature interaction for similar tasks.

Existing methods of multi-scale feature interaction can be broken down into some combination of the top-down and bottom-up approaches which focuses on only two adjacent feature scales at a time. This misses out on a large number of possible combinatorial pairs and makes the propagation of information between distant feature scales inefficient [4, 11]. Furthermore, when repeatedly using the top-down and bottom-up paths like in BiFPN [49], e.g., moving from BiFPNx2 to BiFPNx3, the detection accuracy of the model starts to saturate (see Table 3 for details).

Here, inspired by non-adjacent feature scale connection in NAS-FPN [4, 11], we propose a lightweight feature collection and redistribution module which fuses raw multi-scale features from the backbone together and then redistributes it back to each feature scale. Thus feature maps from each scale now contain direct connections from all the other scales. Such a layer does not involve any heavy computations or parameters, however allows a direct link between every pair of feature scales, as shown in the Figure 1. It should be noted that our RFCR module cannot replace the meticulousness that other feature aggregation methods provide, but instead we aim to provide an extremely lightweight feature processing before passing them to other multi-scale feature fusion methods, providing orthogonal improvements in accuracy.

Additionally, our module design allows us independence from the number of output scales in the detection head, as there are no constraints between the number of input and output features to our RFCR module. For example, despite YOLOv3 detection head having 3 output scales, we can use four different backbone features (3 features same as the output scales, with a fourth shallower feature 'shortcut') during feature collection stage, allowing us to utilize more fine-grained low-level features to improve model performance [53]. Similarly, even for detection heads with only 2 output scales like in YOLOv4-tiny [51], detection features are enriched by the multiple low-level features with the adoption of our RFCR module (see Section 4.4.1).

As discussed in Section 2.3, the manner of feature fusion is as important as the aggregation path. In order to keep the additional latency overhead to a minimum, we pass the raw features during collection through a single 1x1 convolution, and use a simple weighted sum to fuse features together. We pass the fused feature map through a mobilenet convo-

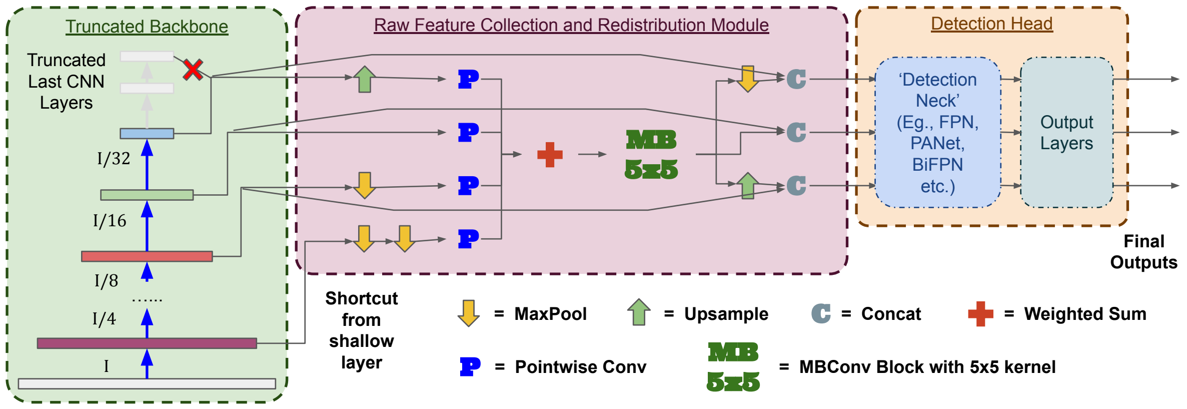

Truncated Backbone
Truncated Last CNN Layers
I/32
I/16
I/8
I/4
I
Raw Feature Collection and Redistribution Module
P
MB 5x5
C
Detection Head
'Detection Neck' (Eg., FPN, PANet, BiFPN etc.)
Output Layers
Final Outputs
Shortcut from shallow layer
= MaxPool
= Upsample
= Concat
= Weighted Sum
= Pointwise Conv
= MBConv Block with 5x5 kernel

Figure 1. Complete Architecture of YOLO-ReT.

lution block (MBConv), which is then redistributed back to various scales. Such a design allows us to keep the network fragmentation to a minimum, since our RFCR module can be represented with just four layers, a 1x1 convolution, a weighted sum and two layers in the MBConv block, along with upsampling and downsampling layers as required. The parallel collection and redistribution of features can also be easily optimized for faster execution.

When fusing features from different scales, naive upsampling and downsampling can cause inconsistent semantics and local positional mismatch [5]. Thus, we propose increasing the receptive field of the feature fusion layer by using a 5x5 kernel instead of the conventional 3x3 or 1x1, to help improve the detection accuracy of the model with negligible affect on its execution latency. We found increasing the kernel to 7x7 did not benefit the performance further.

3.2. Backbone Truncation

Most state-of-the-art lightweight image classification models [33, 42, 48, 63] attempt to keep the number of channels to a minimum by gradually increasing them after every few convolution blocks. However, towards the end, even these models start rapidly expanding the number of channels after every block in an attempt to represent features more clearly before the final fully connected layer [14, 47]. Not only are these last CNN layers the most computationally expensive and heaviest part of the backbone (see the spacing between datapoints in Figure 2), but since they are used to create a better representation for the final classification, these layers mainly contain task specific features.

The importance of transfer learning from classification models has been questioned before, with certain papers even designing specialised backbones for detection [57, 55]. This is based on the intuition that the translational information flow (i.e. across height and width of the image) through consecutive CNN layers varies across tasks. For example, classification models do not preserve spatial information and might accumulate to a spatially coarse feature. On the other hand, detection models attempt to keep the spatial information intact, required for a fine-grained detection output. We identify that the transfer learning capabilities of the initial layers of the feature extraction backbone are quite vital, and it is the last layers that do not provide critical information for the detection/recognition.

We test the importance of individual backbone convolution layers with a detailed analysis of the transfer learning capabilities of various feature extraction backbones, complete with PANet [30] feature aggregation path and YOLOv3 [10] detection head. We experiment with three commonly used backbones, MobilenetV2 ($\times$0.75 and $\times$1.4) and EfficientNet-B3 for our experiments (see Section 4.1 for more details) and divide the backbone into various blocks, which in this case are MBConv blocks for MobileNetV2 and MBConvSE blocks for EfficientNet. Next, we gradually increase, from shallower towards deeper, the number of blocks which are initialised using pre-trained weights from ImageNet dataset while the rest are initialised randomly similar to the detection head, and trained each individual model to convergence (see Section 4.1 for the training setup). The collected results can be seen in Figure 2.

It can be noted from the figure, as we increase the portion of feature extraction backbone initialised with pre-trained weights, the model performance improves, emphasizing the importance of transfer learning. However, around the 60% mark the performance starts to deteriorate and fluctuate. This shows that initializing the last layers of feature extractors with transfer learning weights from ImageNet actually damages the performance as compared to random initialization, possibly because of being stuck in a local minima due to the task-specific nature of these layers.

Since these last layers hold no transfer learning importance, they can be analysed purely from an architecture viewpoint. As can be seen in Figure 2, these last 2 or 3 layers contain over 40% of the weights due to an extreme expansion of number of channels not relevant for object detection. Thus, we propose using a truncated version of var-

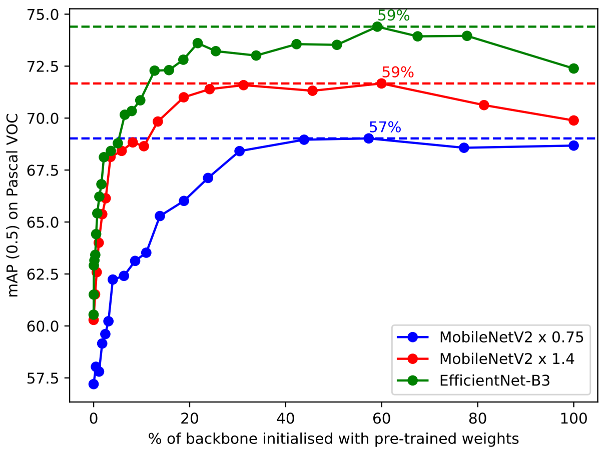


Figure 2. Transfer Learning Curve.

ious feature extraction backbones for the final object detection model, presenting it as a better alternative than reducing the scaling factor. We use the results from Figure 2 to find the point of truncation, i.e., we truncate the last two blocks from both MobileNetV2 versions, and the last three blocks from EfficientNet when we adopt them as backbones.

4. Evaluations

We evaluate our proposed solution using mean average precision [28] on multiple datasets as well as FPS achieved on various edge GPU devices. A thorough ablation study of our proposed RFCR module and backbone truncation and comparison of our model's performance against various real-time object detection models is conducted.

4.1. Evaluation Setups

Feature Extraction Backbone. The balance of execution speed and accuracy of the feature extraction backbone is crucial to the final performance of the detection model due to the transfer learning capability it offers [2, 7, 49]. As discussed in the Section 2.2, the device friendliness together with the theoretical computation and model size are all important factors that affect the model's execution speed. Instead of theoretically modeling the impact of these factors, we directly collect the FPS of various feature extraction backbones on Jetson devices in Table 1. Based on the table, we focus on three commonly used lightweight backbones, MobilenetV2 ($\times 0.75$), MobilenetV2 ($\times 1.4$) and EfficientNet-B3 for our ablation study.

Lightweight Detection Layers. We use feature aggregation path proposed by PANet [30] and YOLOv3 detection head [10]. Instead of traditional convolution blocks as in PANet, we chose single 1x1 pointwise convolution layers [15] for all forms of connections between various feature levels. All features are passed through a single MBConvSE block [48] before every feature aggregation path. We also

Backbone	Depth	Acc.	FPS Nano	FPS NX	FPS AGX	Size (MB)	Comp.
ResNet50 [13]	177	74.9%	33.44	102.14	166.17	97.8	3989
MblNetV2 x 1.4 [42]	157	74.7%	47.64	**129.96**	**188.64**	23.5	588
DarkNet19 [39]	62	72.9%	**48.37**	115.07	187.25	79.5	2764
DarkNet53 [39]	187	77.2%	20.81	71.53	142.99	158	7172
CSPDarkNet53 [52]	418	77.2%	20.60	82.49	153.08	109	5038
EfficientNet-B0 [48]	250	77.1%	36.91	112.09	177.97	**20.4**	**396**
EfficientNet-B3 [48]	407	**81.6%**	17.97	74.48	134.94	47.1	1007

Table 1. Backbone accuracy (ImageNet Top-1) and FPS

do all feature refining during feature aggregation, and again use only a single 1x1 pointwise layer for converting the final feature map into an object detection output.

Dataset and Evaluation Metrics. We choose Pascal VOC [8] and COCO [28] datasets for their popularity for similar tasks. Pascal VOC contains a total of 16,551 images, with 20 object classes and an average of 2.4 bounding boxes per image. COCO is a larger dataset that contains 117,264 images with 80 object classes and has an average of 7.4 bounding boxes per image. We conduct all ablation studies on Pascal VOC, while the final comparison with other state of the art lightweight object detection models in literature are on both datasets. We use Pascal VOC 2007 and 2012 training dataset together for training and compare results on the 2012 test split. For COCO, we use the train and validation splits of COCO 2017 dataset. For Pascal VOC, we use the default IoU threshold of 0.5, while for COCO we provide various detailed metrics as proposed by the dataset [28]. We also compare the runtime FPS of various models on Jetson Nano, Jetson Xavier NX and Jetson AGX Xavier.

Training Details. We use two forms of data augmentation during training, (i) geometric augmentations like random crop, rotation, flip, resize etc., and (ii) photometric augmentations like HSV adjustment, brightness adjustment etc. We use self-adversarial training [2] along with a cosine learning rate decay [31] for best results. For localization, we use GIoU loss [40] rather than the other alternatives (CIoU[66], MSE loss). For training, we first freeze the layers which are initialized using transfer learning weights and train for 100 epochs with a starting learning rate of 0.001. Next, we unfreeze all the layers and fine-tune the model for 150 epochs on a smaller starting learning rate of 0.0001. We do all our ablation studies on 320x320 input resolution, but we also train 224x224 and 416x416 models for final comparison.

Device Setup. For ablation study, all models together with the baselines are implemented on three Jetson devices for FPS calculation. We use TensorRT based FP16 optimization to further speed up the execution. All models are executed with batch size = 1 as we are targeting real-time applications. We process 10,000 images and take the average execution time for FPS calculations. We also provide FPS values for TensorRT based INT8 optimizations, to provide a fair comparison against baseline models designed specifi-

	x	Complete Backbone				Truncated Backbone			
		AP^{50}	FPS			AP^{50}	FPS		
			Nano	NX	AGX		Nano	NX	AGX
MblNetV2 (width = x)	1.4	69.88	19.96	62.58	90.63	69.67	24.58	67.38	95.19
	1.0	69.40	24.85	67.92	93.20	68.87	29.32	73.87	98.67
	0.75	68.67	28.16	73.25	99.75	66.58	34.02	77.67	103.77
	0.5	63.94	35.18	81.05	117.52	61.27	39.97	86.13	124.27
Efficient--Net-x	B3	72.05	9.69	45.72	73.15	72.28	11.24	49.72	78.61
	B2	71.84	12.44	50.86	80.08	71.92	13.45	55.61	82.47
	B1	71.67	13.34	52.31	82.19	71.59	14.62	57.52	86.22
	B0	71.24	18.27	60.97	95.96	70.98	19.08	64.34	99.95

Table 2. Comparing Width reduction vs Depth reduction

cally for integer computations [55]. It should be noted that Jetson Nano does not have tensor cores to support INT8 based optimization, and thus such models do not have any advantage over their floating point counter-parts on Jetson Nano. Additionally, the FPS values reported here might differ from the original reported values for various models, which can be attributed to TensorRT optimizations not performed, prediction box post-processing time not included or a bigger batch size used by certain works in literature.

4.2. Ablation Study

Following the steps to construct the final detection model, we first start with the backbone truncation and then integrate our RFCR module for final comparison.

4.2.1 Truncated feature extraction backbone

We compare two ways of compressing the MobileNetV2 and EfficientNet backbones, which are reducing the scaling factor (or width multiplier for MobileNetV2) and truncating the last parameter heavy layers, and collect the results in Table 2. It can be noted that the truncated versions of EfficientNet are able to outperform their counterparts, both in terms of accuracy as well as FPS, emphasizing on the negative impact of classification task specific backbone features.

For MobileNetV2, when comparing models with similar FPS, the one with the truncated backbone performs better than the the one with smaller scaling factor. e.g., when comparing truncated backbone MobileNetV2x1.4 with complete backbone MobileNetV2x1.0, they both provide similar FPS while the former one provides 0.27 better mAP. This attributes to the fact that reducing the width multiplier reduces the number of channels uniformly across all the layers while truncating the backbone only removes the features from last layers. This difference is exaggerated even more for lighter models on low power devices. For example, truncated backbone at width 0.75 for MobileNetV2 gives similar FPS to the complete backbone at width 0.5 (34.02 and 35.18 respectively on Jetson Nano), yet provides a 2.64 points improvement in mAP. Clearly, as we move towards real-time performing models, using a truncated backbone provides a more accurate and faster feature extraction network than the complete backbone.

Backbone	Feature Aggr. Path	Without RFCR module				With RFCR module			
		AP^{50}	FPS			AP^{50}	FPS		
			Nano	NX	AGX		Nano	NX	AGX
MblNetV2 x 0.75	None	59.92	38.24	80.17	126.42	63.97	36.93	78.38	119.41
	FPN	64.15	37.82	78.42	111.02	66.11	36.33	76.95	101.94
	PANet	66.58	34.02	77.67	103.77	68.75	33.19	71.64	95.97
	BiFPNx1	66.29	34.81	78.04	103.47	66.65	33.78	72.17	95.70
	BiFPNx2	66.69	32.98	76.10	101.78	66.83	31.43	75.61	99.40
	BiFPNx3	66.78	31.54	74.45	100.37	66.90	30.72	73.68	98.27
MblNetV2 x 1.4	None	68.15	28.21	72.97	110.24	69.26	26.50	71.81	106.43
	FPN	69.02	26.06	69.69	103.18	69.95	24.19	66.23	98.73
	PANet	69.67	24.58	67.38	95.19	70.35	23.01	65.37	93.49
	BiFPNx1	69.50	25.26	67.42	95.55	70.14	23.60	65.79	93.71
	BiFPNx2	69.84	23.25	65.77	92.96	70.53	22.18	64.43	92.03
	BiFPNx3	69.87	20.99	62.81	91.50	70.61	20.33	62.17	90.93
Efficient-Net-B3	None	71.24	12.21	62.31	87.60	72.37	11.94	59.84	84.58
	FPN	71.60	11.75	56.04	85.84	72.63	11.34	53.28	82.54
	PANet	72.28	11.24	49.72	78.61	72.96	10.96	47.07	75.61
	BiFPNx1	72.07	12.12	47.70	79.28	72.80	11.71	45.02	75.39
	BiFPNx2	72.39	11.15	43.53	76.22	73.01	10.72	42.24	72.89
	BiFPNx3	72.51	9.92	39.91	72.55	73.08	9.38	38.25	67.99

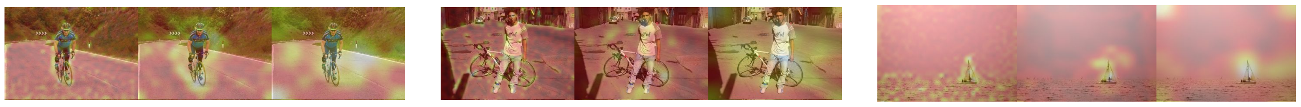
Table 3. Effectiveness of RFCR module

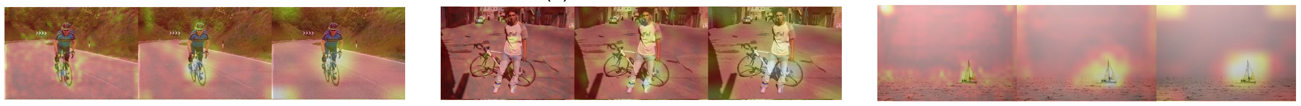
4.2.2 Raw feature collection and redistribution

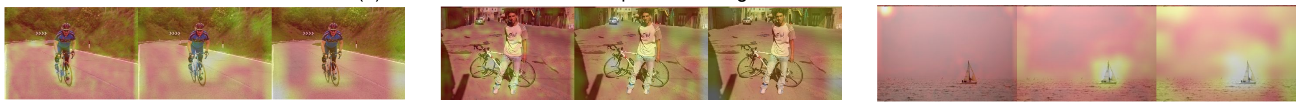
We now evaluate our raw feature collection and redistribution module, combined with various truncated backbones as well as feature aggregation paths adapted with our lightweight detection layers, in Table 3. Our method provides a consistent improvement in performance, irrespective of the backbone or the feature aggregation path that follows it. While this also comes at the cost of slight drop in FPS, overall the trade-off between the two is favorable for us, and thus the feature collection and redistribution module serves as a profitable lightweight addition to the model.

On taking a deeper dive into Table 3, we can notice that the effect of our feature redistribution is significantly more when there is no other feature aggregation method following it. This can be attributed to the fact that in the absence of any interaction between multi-scale features, except through the backbone itself, such a redistribution provides the much needed feature interaction. However, even with BiFPNx3, our method still gets a noticeable boost in performance, showing the importance of shortcut connections between non-adjacent layers.

Finally, we bring all methods discussed above together to do a combined component ablation study. The results are collected in Table 4. We start with the MobileNetV2 ($\times$0.75) backbone for Jetson Nano, MobileNetV2 ($\times$1.4) backbone for Jetson Xavier NX and EfficientNet-B3 backbone for Jetson AGX Xavier, along with a PANet feature aggregation based YOLOv3 object detection head and lightweight detection layers. Next, we test our RFCR module with and without truncating the backbone. While it is clear that RFCR module performs well in both scenarios,


(a) Multi-Scale Raw Features


(b) Multi-Scale Raw Features passed through our RFCR Module


(c) Multi-Scale Raw Features passed through our RFCR module and PANet

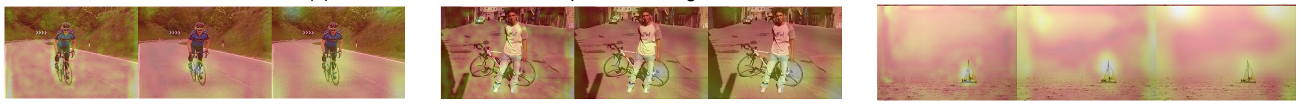

(d) Multi-Scale Raw Features passed directly through PANet without the RFCR Module

Figure 3. Qualitative study with intermediate heatmaps. For all sets of three heatmaps, the original resolution of the feature maps from low-level to high-level features, i.e., left-to-right, is 40x40, 20x20 and 10x10 respectively, for a 320x320 input image.

the drop in FPS for model with complete backbone is more as compared to the one with truncated backbone. This is due to the fact that the complete backbone has heavier layers towards the end, which makes the following feature aggregation layers heavier too.

As discussed in Section 3.1, we also introduce additional 'shortcut' connections in our RFCR module independent of the detection head's output scales. We notice that this additional 'shortcut' from shallower layers of the backbone further improves its accuracy, emphasizing the importance of low-level features for accurate detection tasks and the freedom our design provides in using more input features from the backbone than the number of output scales. Overall, we are able to both speed up the execution and improve accuracy by combining backbone truncation and RFCR module.

Baseline	+Truncate	+RFCR	+Shortcut	AP^{50}	FPS
MobileNetV2 x 0.75 on Jetson Nano	✗	✗	✗	68.67	28.16
	✓	✗	✗	66.58	34.02
	✗	✓	✗	69.50	26.97
	✓	✓	✗	68.40	33.35
	✓	✓	✓	68.75	33.19
MobileNetV2 x 1.4 on Jetson NX	✗	✗	✗	69.88	62.58
	✓	✗	✗	69.67	67.38
	✗	✓	✗	70.56	62.11
	✓	✓	✗	70.21	65.91
	✓	✓	✓	70.35	65.37
EfficientNet-B3 on Jetson AGX	✗	✗	✗	72.05	73.15
	✓	✗	✗	72.28	78.61
	✗	✓	✗	72.37	72.90
	✓	✓	✗	72.58	75.72
	✓	✓	✓	72.96	75.61

Table 4. Ablation study

4.3. Qualitative Analysis

We also conduct a qualitative assessment of our RFCR module by visualizing various intermediate feature maps using our MobileNetV2x0.75 based model. We do a channel wise max pooling of the feature maps and then plot the obtained heatmaps scaled back to the original input image, as shown in Figure 3. For all 3 images, we notice that the raw feature maps (a) directly from the backbone are very noisy. While low-level raw features create a better boundary around the object, the attention patterns are discontinuous due to their smaller receptive field. On the other hand, high-level raw features create an inaccurate boundary which sometimes extends well beyond the object. However, just by doing a simple feature collection and redistribution, we obtain feature maps (b), in which we notice that various scales of features work together in harmony to obtain a better boundary of attention around the object. We notice similar behavior between final feature maps (c) obtained right before the detection head, and final feature maps (d) obtained from a separately trained model which has the same architecture but without RFCR module, even though feature maps (d) are obtained after feature aggregation through PANet, which demonstrates that the RFCR module played an important role in filtering out the noise from raw features.

4.4. Comparison with State-of-the-art Models

We build models based on selected backbones using truncation and the RFCR module, and then compare them with 3 different input image resolutions to various state-of-the-art real-time object detection models with the same settings, as shown in Table 5. As expected, smaller input resolution results in a faster but less accurate detection model. We also provide detailed evaluation on COCO dataset in Table 6, however the scope of comparison is limited as not all state-of-the-art models provide such detailed results.

When comparing on Jetson Nano, we find that our

Model	Input Res.	Size (MB)	FPS Nano	FPS NX	FPS AGX	AP^{50} VOC	AP^{50} COCO
Tiny-YOLOv3 [10]	416	34.9	27.36	66.55	91.71	61.30	33.10
Tinier-YOLO [9]	416	8.9	30.14	68.73	92.09	65.70	**34.00**
YOLO Nano [55]	416	4.0	13.62	54.03[‡]	85.81[‡]	69.10	–
YOLO-Fastest [37]	320	1.3	**42.41**	**76.13**	**126.82**	61.02	–
YOLO-Fastest XL [37]	320	3.5	27.93	61.33	108.76	**69.43**	32.45
YOLO-ReT-M0.75	416	5.2	19.87	58.24	71.16	**72.39**	**36.44**
	320	5.2	33.19	71.64	95.97	68.75	34.91
	224	5.2	**55.16**	**84.10**	**134.87**	60.77	30.76
YOLO-ReT-M1.4	416	12.3	13.17	46.07	66.23	**73.32**	**36.52**
	320	12.3	23.01	65.37	93.49	70.35	35.77
	224	12.3	**43.16**	**84.32**	**113.94**	62.91	31.63
YOLO-ReT-EB3	416	28.3	6.35	28.83	49.07	**76.49**	**39.12**
	320	28.3	10.96	44.59	75.61	72.96	36.51
	224	28.3	**18.57**	**54.87**	**93.55**	65.52	33.11

[‡] Calculated with INT8 optimization

Table 5. Comparison with other state-of-the-art models

Model	Input Res.	COCO AP	AP^{50}	AP^{75}	AP^{s}	AP^{m}	AP^{l}
Tiny-YOLOv3 [10]	416	15.3	33.1	12.4	4.4	15.2	25.1
Tinier-YOLO [9]	416	17.0	34.0	15.7	4.8	17.3	26.8
YOLO-ReT-M0.75	320	18.4	34.9	17.3	5.4	18.7	28.9
YOLO-ReT-M1.4	320	19.1	35.8	18.4	5.8	19.6	30.2
YOLO-ReT-EB3	320	**19.7**	**36.5**	**19.3**	**6.3**	**20.3**	**31.1**

Table 6. Evaluation on COCO dataset

model YOLO-ReT-M0.75 at 320x320 resolution outperforms Tinier-YOLO by 3.05 mAP on Pascal VOC and 0.91 mAP on COCO, while executing faster by 3.05 FPS. On Jetson Xavier NX, our model YOLO-ReT-M1.4 at 320x320 resolution outperforms YOLO-Fastest-XL by 0.92 mAP on Pascal VOC and 3.34 mAP on COCO, while executing faster by 4.02 FPS. Even though our YOLO-ReT-EB3 model at 416x416 resolution is able to push for the best performance while still executing real-time on Jetson Xavier AGX, it should be noted that at similar FPS, MobileNetV2 based models outperform EfficientNet based models.

4.4.1 Comparison with YOLOv4-tiny

The state-of-the-art in object detection is being pushed constantly, with multiple parallel works being published at any moment. YOLOv4-tiny [51] has done the same for real-time object detection on edge devices, using novel training methods as well backbone scaling and customization for an improved object detection model that provides significantly better performance than any existing SOTA. While this work was done in parallel with ours, the improvements are undeniably significant to be ignored. Thus, we directly introduce our RFCR module into YOLOv4-tiny and YOLOv4-tiny (3l) without backbone truncation as these models are designed specifically for object detection and are trained from scratch without using transfer learning [51]. We still use 4 inputs to our feature collection, and redistribute them based on the number of output scales present in the model, i.e. 2 for YOLOv4-tiny and 3 for YOLOv4-tiny (3l). We execute both models on Jetson Nano device since it has the least resource compared to the other two platforms. The final results are collected in Table 7.

Model	Input Res.	FPS Nano	COCO AP	AP^{50}	AP^{75}
YOLOv4-tiny	416	29.55	21.7	40.2	22.5
YOLOv4-tiny+RFCR	416	27.81	**22.9**	**41.5**	**23.3**
YOLOv4-tiny (3l)	608	24.87	28.7	47.2	29.7
YOLOv4-tiny (3l)+ RFCR	608	21.40	**29.3**	**48.1**	**30.5**

Table 7. Comparison with YOLOv4-tiny [51]

We find that the RFCR versions of both these models are able to outperform their counterparts in terms of accuracy. Looking deeper, we find that our RFCR module provides larger improvement for YOLOv4-tiny, as compared to YOLOv4-tiny (3l). This is expected, as YOLOv4-tiny only has 2 output scales, which further necessitates multi-scale feature interaction, and helps us demonstrate the compatibility of our module with various detection models.

5. Conclusion and Future Work

This paper presents a novel raw feature collection and redistribution (RFCR) module, and a truncated backbone for improved transfer learning for object detection. These techniques complement each other, leading to both improved accuracy and efficiency for various lightweight architectures. We believe that machine learning model designs targeting edge GPU devices have opened up a new avenue of research for edge computing, and can lead to a wide range of application possibilities. Thus, further research on device specific model optimizations and neural architecture search can help push the technology forward effectively for real-time performing models. Meanwhile, an in-depth understanding of interactions between various features for object detection can further enhance the information flow and is an important aspect of improving the model accuracy while maintaining its efficiency. In future, we aim to extend our RFCR module to other vision domains and a larger variety of models. The source code of our design is opened to public at `https://github.com/prakharg24/yoloret`.

Acknowledgement

This publication was made possible by grant AICC03-0324-200005 from Qatar National Research Fund (a member of Qatar Foundation), and grant MRC-05-110 from Hamad Medical Corporation. It is also partially supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. The findings herein reflect the work, and are solely the responsibility, of the authors.

References

[1] Djamila Romaissa Beddiar, Brahim Nini, Mohammad Sabokrou, and Abdenour Hadid. Vision-based human activity recognition: a survey. *Multimedia Tools and Applications*, 79(41):30509–30555, 2020.

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[3] Karanbir Singh Chahal and Kuntal Dey. A survey of modern object detection literature using deep learning. *arXiv preprint arXiv:1808.07256*, 2018.

[4] Bo Chen, Golnaz Ghiasi, Hanxiao Liu, Tsung-Yi Lin, Dmitry Kalenichenko, Hartwig Adam, and Quoc V Le. Mnasfpn: Learning latency-aware pyramid architecture for object detection on mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13607–13616, 2020.

[5] Yimian Dai, Fabian Gieseke, Stefan Oehmcke, Yiquan Wu, and Kobus Barnard. Attentional feature fusion. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3560–3569, 2021.

[6] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11592–11601, 2020.

[7] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6569–6578, 2019.

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[9] Wei Fang, Lin Wang, and Peiming Ren. Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, 8:1935–1944, 2019.

[10] Ali Farhadi and Joseph Redmon. Yolov3: An incremental improvement. In *Computer Vision and Pattern Recognition*, pages 1804–02767. Springer Berlin/Heidelberg, Germany, 2018.

[11] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7036–7045, 2019.

[12] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. A survey of deep learning techniques for medical diagnosis. In *Information and Communication Technology for Sustainable Development*, pages 161–170. Springer, 2020.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510. IEEE, 2018.

[17] Petr Hurtik, Vojtech Molek, Jan Hula, Marek Vajgl, Pavel Vlasanek, and Tomas Nejezchleba. Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3. *arXiv preprint arXiv:2005.13243*, 2020.

[18] Yoo Jin Hyeok, Kum Dongsuk, and Choi Jun Won. Scarfnet: Multi-scale features with deeply fused and redistributed semantics for enhanced object detection. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4505–4512. IEEE, 2021.

[19] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[20] Paresh R Kamble, Avinash G Keskar, and Kishor M Bhurchandi. Ball tracking in sports: a survey. *Artificial Intelligence Review*, 52(3):1655–1705, 2019.

[21] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[22] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. Cornernet-lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900*, 2019.

[23] Ji Li and Yingdong Ma. MSPNet: Multi-level semantic pyramid network for real-time object detection. In *International Conference on Image Analysis and Recognition*, pages 76–88. Springer, 2020.

[24] Wei Li, Xianghua Ma, and Tongrui Peng. A real-time multipoint-based object detector. In *2020 5th International Conference on Computational Intelligence and Applications (ICCIA)*, pages 1–7. IEEE, 2020.

[25] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-DSOD: Lightweight object detection for resource-restricted usages. *arXiv preprint arXiv:1807.11013*, 2018.

[26] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *arXiv preprint arXiv:1804.06215*, 2018.

[27] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence

Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[29] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International journal of computer vision*, 128(2):261–318, 2020.

[30] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[31] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[32] Chen Change Loy. Face detection. *Computer Vision: A Reference Guide*, pages 1–5, 2020.

[33] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[34] Qi-Chao Mao, Hong-Mei Sun, Yan-Bo Liu, and Rui-Sheng Jia. Mini-YOLOv3: real-time object detector for embedded applications. *IEEE Access*, 7:133529–133538, 2019.

[35] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[36] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. Thundernet: Towards real-time generic object detection on mobile devices. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6718–6727, 2019.

[37] D Quiqui. Yolo-fastest. `https://github.com/dog-qiuqiu/Yolo-Fastest`, 2020.

[38] Ruslan Rakhimov, Emil Bogomolov, Alexandr Notchenko, Fung Mao, Alexey Artemov, Denis Zorin, and Evgeny Burnaev. Making densepose fast and light. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1869–1877, 2021.

[39] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[40] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[43] Mohammad Javad Shaifee, Brendan Chywl, Francis Li, and Alexander Wong. Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *Journal of Computational Vision and Imaging Systems*, 3(1), 2017.

[44] Wei A Shang. Survey of mobile robot vision self-localization. *Journal of Automation and Control Engineering Vol*, 7(2), 2019.

[45] Yichun Shi and Anil K Jain. DocFace+: ID document to selfie matching. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 1(1):56–67, 2019.

[46] Ramesh Simhambhatla, Kevin Okiah, Shravan Kuchkula, and Robert Slater. Self-driving cars: Evaluation of deep learning techniques for object detection in different driving conditions. *SMU Data Science Review*, 2(1):23, 2019.

[47] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[48] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

[49] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10781–10790, 2020.

[50] Qiankun Tang, Jie Li, Zhiping Shi, and Yu Hu. Lightdet: A lightweight and accurate object detection network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2243–2247. IEEE, 2020.

[51] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13029–13038, 2021.

[52] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 390–391, 2020.

[53] Tiancai Wang, Rao Muhammad Anwer, Hisham Cholakkal, Fahad Shahbaz Khan, Yanwei Pang, and Ling Shao. Learning rich features at high-speed for single-shot object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1971–1980, 2019.

[54] Alexander Womg, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. In *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 95–101. IEEE, 2018.

[55] Alexander Wong, Mahmoud Famuori, Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Jonathan Chung. YOLO nano: A highly compact you only look once convolutional neural network for object detection. *arXiv preprint arXiv:1910.01271*, 2019.

[56] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 129–137, 2017.

[57] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Yongzhe Wang, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. Mobiledets: Searching for object detection architectures for mobile accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3825–3834, 2021.

[58] Rui Yao, Guosheng Lin, Shixiong Xia, Jiaqi Zhao, and Yong Zhou. Video object segmentation and tracking: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(4):1–47, 2020.

[59] Yunhua Yin, Huifang Li, and Wei Fu. Faster-YOLO: An accurate and faster object detection method. *Digital Signal Processing*, page 102756, 2020.

[60] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems*, 27:3320–3328, 2014.

[61] Xiaofan Zhang, Haoming Lu, Cong Hao, Jiachen Li, Bowen Cheng, Yuhong Li, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, Wen-mei Hwu, and Deming Chen. SkyNet: a hardware-efficient method for object detection and tracking on embedded systems. In *Conference on Machine Learning and Systems (MLSys)*, 2020.

[62] Xiaofan Zhang, Anand Ramachandran, Chuanhao Zhuge, Di He, Wei Zuo, Zuofu Cheng, Kyle Rupnow, and Deming Chen. Machine learning on fpgas to face the iot revolution. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 819–826, 2017.

[63] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[64] Haipeng Zhao, Yang Zhou, Long Zhang, Yangzhao Peng, Xiaofei Hu, Haojie Peng, and Xinyue Cai. Mixed YOLOv3-LITE: A lightweight real-time object detection method. *Sensors*, 20(7):1861, 2020.

[65] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9259–9266, 2019.

[66] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU loss: Faster and better learning for bounding box regression. In *AAAI*, pages 12993–13000, 2020.

[67] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.