Spring 2023

Computer Vision in Adverse Conditions: Small Objects, Low-Resoltuion Images, and Edge Deployment

Raja Sunkara
Missouri University of Science and Technology

Department:

COMPUTER VISION IN ADVERSE CONDITIONS: SMALL OBJECTS, LOW-RESOLUTION IMAGES, AND EDGE DEPLOYMENT

by

RAJA SUNKARA

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

2023

Approved by:

Dr. Tony T. Luo, Advisor
Dr. Sid Nadendla
Dr. Ardhendu Tripathy

PUBLICATION THESIS OPTION

This thesis consists of the following two articles, formatted in the style used by the Missouri University of Science and Technology.

Paper I: Pages 3-25 have been accepted by European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD). No More Strided Convolutions or Pooling: A New CNN Building Block for Low-Resolution Images and Small Objects.

Paper II: Pages 26-53 have been accepted by Pattern Recognition Journal. YOGA: Deep Object Detection in the Wild with Lightweight Feature Learning and Multiscale Attention.

ABSTRACT

Computer vision based on deep learning is an essential field that plays a significant role in object detection, image classification, semantic segmentation, instance segmentation, and other applications. However, these models face significant challenges in adverse conditions, such as small objects, low-resolution images, and edge deployment. These challenges limit the accuracy and efficiency of computer vision algorithms, making it difficult to obtain reliable results.

The primary objective of this thesis is to assess the performance of deep learning-based computer vision models in challenging conditions and provide viable solutions to overcome the obstacles. The study will specifically address three key challenges, namely, the detection of small objects, handling low-resolution images, and deployment of models at the edge.

To address the challenges of small objects and low-resolution images, we propose SPD-Conv. This new CNN building block eliminates strided convolution and pooling layers to improve the detection of small objects and reduce the loss of fine-grained information.

To address the challenge of edge deployment, we propose YOGA, a lightweight object detection model that achieves high accuracy on low-end edge devices by using a two-phase feature learning pipeline with attention-based multi-scale feature fusion.

The proposed solutions are evaluated on COCO-val and COCO-testdev datasets and compared with state-of-the-art models, demonstrating their effectiveness in overcoming these challenging scenarios. This thesis places significant emphasis on the importance of reproducibility in research. All experiments are conducted using open-source tools and frameworks, and the code and models are made available to the research community. This ensures that the results are transparent, and others can easily reproduce and build upon the work presented in this thesis.

ACKNOWLEDGMENTS

Firstly, I would like to extend my heartfelt gratitude to my advisor, Dr. Tony Luo, for his unwavering support, invaluable advice, and infinite patience throughout my research journey. Without his guidance, I would not have achieved this milestone in my academic career.

I would also like to express my profound appreciation to the other members of my committee, Dr. Sid Nadendla and Dr. Ardhendu Tripathy, for their insightful comments, invaluable suggestions, and unwavering encouragement. Their contributions played a crucial role in shaping the direction of my research and helped me to navigate the challenges along the way.

Furthermore, I am deeply grateful to the Computer Science department, including all of the faculty and staff, for providing me with the necessary resources to conduct my research and pursue my academic goals. The support and opportunities afforded to me by this department have been truly instrumental in my success.

Finally, I would like to acknowledge the Foundry services at our university, particularly for providing access to large GPUs that enabled me to train and evaluate the proposed solutions.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

Computer vision based on deep learning has shown remarkable success in a wide range of applications. Convolutional Neural Networks (CNNs) are the backbone of modern deep learning-based computer vision systems, with models such as AlexNet, VGGNet, ResNet and others achieving state-of-the-art performance in image classification, object detection, and semantic segmentation tasks. However, despite their success, these models often face significant challenges when dealing with adverse conditions. For instance, small objects are challenging to detect, as they typically have lower resolution and limited contextual information for the model to learn from. Low-resolution images, on the other hand, can lead to a loss of fine-grained information and poorly learned features. Moreover, deploying these models on edge devices is often constrained by limited resources, such as memory and processing power. In this thesis, we investigate the effectiveness of deep learning-based computer vision models in adverse conditions and propose solutions to address these challenges, with a particular emphasis on small objects, low-resolution images, and edge deployment.

1.1. SMALL OBJECTS AND LOW-RESOLUTION IMAGES

Convolutional neural networks (CNNs) have excelled at many computer vision tasks, including image classification and object detection. However, existing CNN models need high-quality inputs (fine images, medium to large objects) in both training and inference. The use of strided convolution and pooling in CNN architecture design usually does not exhibit adverse effects because most scenarios being studied have good resolutions and objects are in fair sizes. However, in tougher tasks, such as when images are blurry or objects are small, the current design starts to suffer from the loss of fine-grained information and poorly learned features.

To address this issue, a new building block for CNNs, called SPD-Conv, is proposed in substitution of strided convolution and pooling layers altogether. SPD-Conv is a space-to-depth (SPD) layer followed by a non-strided (i.e., vanilla) convolution layer, which replaces both strided convolution and pooling in a general and unified way. SPD-Conv can be applied to most if not all CNN architectures and downsamples a feature map without any information loss. In this way, SPD-Conv can learn features well in scenarios with small objects or low-quality inputs.

1.2. EDGE DEPLOYMENT

While deep learning has revolutionized object detection in various applications, the models' increasing complexity demands more training data, tuning parameters, and longer training and inference times. This presents a significant challenge for deployment in the wild. Although researchers have attempted to address this issue through pruning and quantization methods, their effect is often limited. Thus, a clean-slate design is much desired for resource-conscious object detection at the edge. In this thesis, we propose a novel object detection model named YOGA, based on a resource-conscious design principle. YOGA achieves a reduction of up to 34% in model size, in terms of number of parameters and floating-point operations, while maintaining competitive accuracy.

PAPER

I. NO MORE STRIDED CONVOLUTIONS OR POOLING: A NEW CNN BUILDING BLOCK FOR LOW-RESOLUTION IMAGES AND SMALL OBJECTS

Raja Sunkara and Tie Luo (Corresponding author)
Computer Science Department
Missouri University of Science and Technology
{rs5cq,tluo}@mst.edu

ABSTRACT

Convolutional neural networks (CNNs) have made resounding success in many computer vision tasks such as image classification and object detection. However, their performance degrades rapidly on tougher tasks where images are of low resolution or objects are small. In this paper, we point out that this roots in a defective yet common design in existing CNN architectures, namely the use of *strided convolution* and/or *pooling layers*, which results in a loss of fine-grained information and learning of less effective feature representations. To this end, we propose a new CNN building block called *SPD-Conv* in place of each strided convolution layer and each pooling layer (thus eliminates them altogether). SPD-Conv is comprised of a *space-to-depth* (SPD) layer followed by a *non-strided* convolution (Conv) layer, and can be applied in most if not all CNN architectures. We explain this new design under two most representative computer vision tasks: object detection and image classification. We then create new CNN architectures by applying SPD-Conv to YOLOv5 and ResNet, and empirically show that our approach significantly outperforms state-of-the-art deep learning models, especially on tougher tasks with low-resolution images and small objects. We have open-sourced our code at https://github.com/LabSAINT/SPD-Conv.

1. INTRODUCTION

Since AlexNet [18], convolutional neural networks (CNNs) have excelled at many computer vision tasks. For example in image classification, well-known CNN models include AlexNet, VGGNet [30], ResNet [13], etc.; while in object detection, those models include the R-CNN series [9, 28], YOLO series [4, 26], SSD [24], EfficientDet [34], and so on. However, all such CNN models need "good quality" inputs (fine images, medium to large objects) in both training and inference. For example, AlexNet was originally trained and evaluated on 227×227 clear images, but after reducing the image resolution to 1/4 and 1/8, its classification accuracy drops by 14% and 30%, respectively [16]. The similar observation was made on VGGNet and ResNet too [16]. In the case of object detection, SSD suffers from a remarkable mAP loss of 34.1 on 1/4 resolution images or equivalently 1/4 smaller-size objects, as demonstrated in [11]. In fact, small object detection is a very challenging task because smaller objects inherently have lower resolution, and also limited context information for a model to learn from. Moreover, they often (unfortunately) co-exist with large objects in the same image, which (the large ones) tend to dominate the feature learning process, thereby making the small objects undetected.

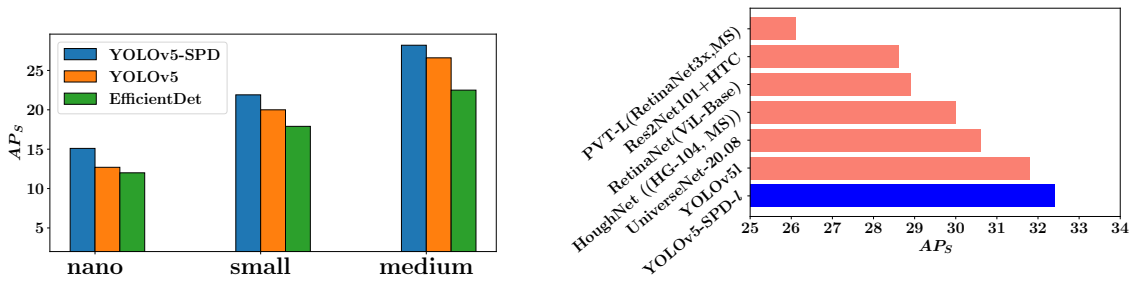
In this paper, we contend that such performance degradation roots in a defective yet common design in existing CNNs. That is, the use of strided convolution and/or pooling, especially in the earlier layers of a CNN architecture. The adverse effect of this design usually does not exhibit because most scenarios being studied are "amiable" where images have good resolutions and objects are in fair sizes; therefore, there is plenty of *redundant* pixel information that strided convolution and pooling can *conveniently skip* and the model can still learn features quite well. However, in tougher tasks when images are blurry or objects are small, the lavish assumption of redundant information no longer holds and the current design starts to suffer from loss of fine-grained information and poorly learned features.

To address this problem, we propose a new building block for CNN, called *SPD-Conv*, in substitution of (and thus eliminate) strided convolution and pooling layers altogether. SPD-Conv is a *space-to-depth* (SPD) layer followed by a *non-strided* (i.e., vanilla) convolution layer. The SPD layer downsamples a feature map X but retains all the information in the *channel* dimension, and thus there is no information loss. We were inspired by an image transformation technique [29] which rescales a raw image before feeding it into a neural net, but we substantially generalize it to downsampling *feature maps* inside and throughout the entire network; furthermore, we add a non-strided convolution operation after each SPD to reduce the (increased) number of channels using learnable parameters in the added convolution layer. Our proposed approach is both *general* and *unified*, in that SPD-Conv (i) can be applied to most if not all CNN architectures and (ii) replaces both strided convolution and pooling the same way. In summary, this paper makes the following contributions:

1) We identify a defective yet common design in existing CNN architectures and propose a new building block called SPD-Conv in lieu of the old design. SPD-Conv downsamples feature maps without losing learnable information, completely jettisoning strided convolution and pooling operations which are widely used nowadays.

2) SPD-Conv represents a general and unified approach, which can be easily applied to most if not all deep learning based computer vision tasks.

3) Using two most representative computer vision tasks, object detection and image classification, we evaluate the performance of SPD-Conv. Specifically, we construct YOLOv5-SPD, ResNet18-SPD and ResNet50-SPD, and evaluate them on COCO-2017, Tiny ImageNet, and CIFAR-10 datasets in comparison with several state-of-the-art deep learning models. The results demonstrate significant performance improvement in AP and top-1 accuracy, especially on small objects and low-resolution images. See Figure 1 for a preview.



(a) Nano, small, and medium models.

(b) Large-scale models.

Figure 1. Comparing AP for small objects (AP_S). "SPD" indicates our approach.

4) SPD-Conv can be easily integrated into popular deep learning libraries such as PyTorch and TensorFlow, potentially producing greater impact. Our source code is available at https://github.com/LabSAINT/SPD-Conv.

The rest of this paper is organized as follows. Section 2 presents background and reviews related work. Section 3 describes our proposed approach and Section 4 presents two case studies using object detection and image classification. Section 5 provides performance evaluation. This paper concludes in Section 6.

2. PRELIMINARIES AND RELATED WORK

We first provide an overview for this area, focusing more on object detection since it subsumes image classification.

Current state-of-the-art object detection models are CNN-based and can be categorized into one-stage and two-stage detectors, or anchor-based or anchor-free detectors. A two-stage detector firstly generates coarse region proposals and secondly classifies and refines each proposal using a head (a fully-connected network). In contrast, a one-stage detector skips the region proposal step and runs detection directly over a dense sampling

Table 1. A taxonomy of OD models.

Model	Anchor-based	Anchor-free
One-stage	Faster R-CNN [27], SSD [24], RetinaNet [21], EfficientDet [34], YOLO [4, 14, 26, 36]	FCOS [35], CenterNet [7], DETR [5], YOLOX [8]
Two-stage	R-CNN [10], Fast R-CNN [9]	RepPoints, CenterNet2

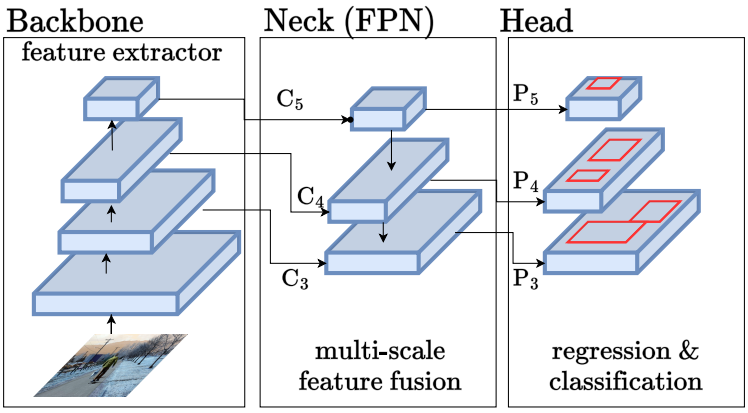


Figure 2. A one-stage object detection pipeline.

of locations. Anchor-based methods use *anchor boxes*, which are a predefined collection of boxes that match the widths and heights of objects in the training data, to improve loss convergence during training. We provide Table 1 that categorizes some well-known models.

Generally, one-stage detectors are faster than two-stage ones and anchor-based models are more accurate than anchor-free ones. Therefore, later in our case study and experiments we focus more on one-stage and anchor-based models, i.e., the first cell of Table 1.

A typical one-stage object detection model is depicted in Figure 2. It consists of a CNN-based *backbone* for visual feature extraction and a detection *head* for predicting class and bounding box of each contained object. In between, a *neck* of extra layers is added to combine features at multiple scales to produce semantically strong features for detecting objects of different sizes.

2.1. SMALL OBJECT DETECTION

Traditionally, detecting both small and large objects is viewed as a multi-scale object detection problem. A classic way is *image pyramid* [3], which resizes input images to multiple scales and trains a dedicated detector for each scale. To improve accuracy, SNIP [31] was proposed which performs *selective backpropagation* based on different object sizes in each detector. SNIPER [32] improves the efficiency of SNIP by only processing the context regions around each object instance rather than every pixel in an image pyramid, thus reducing the training time. Taking a different approach to efficiency, Feature Pyramid Network (FPN) [20] exploits the multi-scale features inherent in convolution layers using lateral connections and combine those features using a top-down structure. Following that, PANet [22] and BiFPN [34] were introduced to improve FPN in its feature information flow by using shorter pathways. Moreover, SAN [15] was introduced to map multi-scale features onto a scale-invariant subspace to make a detector more robust to scale variation. All these models unanimously use strided convolution and max pooling, which we get rid of completely.

2.2. LOW-RESOLUTION IMAGE CLASSIFICATION

One of the early attempts to address this challenge is [6], which proposes an end-to-end CNN model by adding a super-resolution step before classification. Following that, [25] proposes to transfer fine-grained knowledge acquired from high-resolution training images to low-resolution test images. However, this approach requires high-resolution training

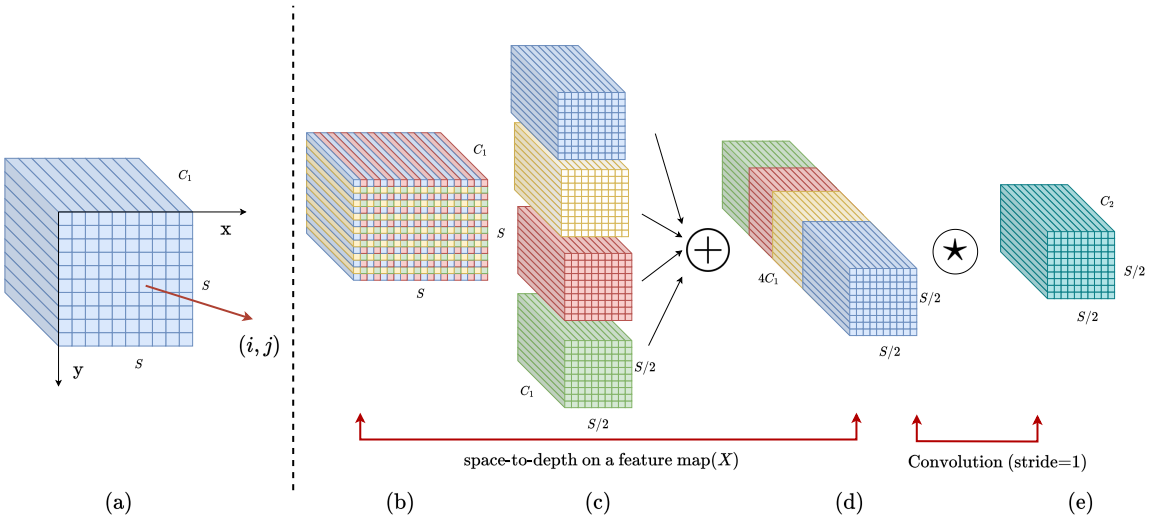


Figure 3. Illustration of SPD-Conv when $scale = 2$ (see text for details).

images corresponding to the specific application (e.g., the classes), which are not always available. This same requirement of high-resolution training images is also needed by several other studies such as [37]. Recently, [33] proposed a loss function that incorporate attribute-level separability (where attribute means fine-grained, hierarchical class labels) so that the model can learn class-specific discriminative features. However, the fine-grained (hierarchical) class labels are difficult to obtain and hence limit the adoption of the method.

3. A NEW BUILDING BLOCK: SPD-CONV

SPD-Conv is comprised of a Space-to-depth (SPD) layer followed by a non-strided convolution layer. This section describes it in detail.

3.1. SPACE-TO-DEPTH (SPD)

Our SPD component generalizes a (raw) image transformation technique [29] to downsampling feature maps inside and throughout a CNN, as follows. Consider any intermediate feature map X of size $S \times S \times C_1$, slice out a sequence of sub feature maps as

$f_{0,0} = X[0 : S : scale, 0 : S : scale], f_{1,0} = X[1 : S : scale, 0 : S : scale], \ldots,$

$f_{scale-1,0} = X[scale - 1 : S : scale, 0 : S : scale];$

$f_{0,1} = X[0 : S : scale, 1 : S : scale], \ldots, f_{scale-1,1} = X[scale - 1 : S : scale, 1 : S : scale];$

$\vdots$

$f_{0,scale-1} = X[0 : S : scale, scale - 1 : S : scale], f_{1,scale-1}, \ldots,$

$f_{scale-1,scale-1} = X[scale - 1 : S : scale, scale - 1 : S : scale].$

In general, given any (original) feature map X, a sub-map $f_{x,y}$ is formed by all the entries $X(i, j)$ that $i + x$ and $j + y$ are divisible by $scale$. Therefore, each sub-map downsamples X by a factor of $scale$. Figure 3(a)(b)(c) give an example when $scale = 2$, where we obtain four sub-maps $f_{0,0}, f_{1,0}, f_{0,1}, f_{1,1}$ each of which is of shape $(\frac{S}{2}, \frac{S}{2}, C_1)$ and downsamples X by a factor of 2.

Next, we concatenate these sub feature maps along the channel dimension and thereby obtain a feature map X', which has a reduced spatial dimension by a factor of $scale$ and an increased channel dimension by a factor of $scale^2$. In other words, SPD transforms feature map $X(S, S, C_1)$ into an intermediate feature map $X'(\frac{S}{scale}, \frac{S}{scale}, scale^2 C_1)$. Figure 3(d) gives an illustration using $scale = 2$.

3.2. NON-STRIDED CONVOLUTION

After the SPD feature transformation layer, we add a non-strided (i.e., stride=1) convolution layer with C_2 filters where $C_2 < scale^2 C_1$, and further transforms $X'(\frac{S}{scale}, \frac{S}{scale}, scale^2 C_1) \rightarrow X''(\frac{S}{scale}, \frac{S}{scale}, C_2)$. The reason we use non-strided convolution is to retain all the discriminative feature information as much as possible. Otherwise, for instance, using a 3×3 filer with stride=3, feature maps will get "shrunk" yet each pixel is sampled only once; if stride=2, *asymmetric sampling* will occur where even and odd rows/columns will be sampled different times. In general, striding with a step size greater than 1 will cause *non-discriminative loss* of information although at the surface, it appears to convert feature map $X(S, S, C_1) \rightarrow X''(\frac{S}{scale}, \frac{S}{scale}, C_2)$ too (but without X').

4. HOW TO USE SPD-CONV: CASE STUDIES

To explain how to apply our proposed method to redesigning CNN architectures, we use two most representative categories of computer vision models: object detection and image classification. This is without loss of generality as almost all CNN architectures use strided convolution and/or pooling operations to downsample feature maps.

4.1. OBJECT DETECTION

YOLO is a series of very popular object detection models, among which we choose the latest YOLOv5 [14] to demonstrate. YOLOv5 uses CSPDarknet53 [4] with a SPP [12] module as its backbone, PANet [23] as its neck, and the YOLOv3 head [26] as its detection head. In addition, it also uses various data augmentation methods and some modules from YOLOv4 [4] for performance optimization. It employs the cross-entropy loss with a sigmoid layer to compute objectness and classification loss, and the CIoU loss function [38] for localization loss. The CIoU loss takes more details than IoU loss into account, such as edge overlapping, center distance, and width-to-height ratio.

4.1.1. YOLOv5-SPD. We apply our method described in Section 3 to YOLOv5 and obtain YOLOv5-SPD (Figure 4), simply by replacing the YOLOv5 stride-2 convolutions with our SPD-Conv building block. There are 7 instances of such replacement because YOLOv5 uses five stride-2 convolution layers in the backbone to downsample the feature map by a factor of 2^5, and two stride-2 convolution layers in the neck. There is a concatenation layer after each strided convolution in YOLOv5 neck; this does not alter our approach and we simply keep it between our SPD and Conv.

4.1.2. Scalability. YOLOv5-SPD can suit different application or hardware needs by easily scaling up and down in the same manner as YOLOv5. Specifically, we can simply adjust (1) the number of filters in every non-strided convolution layer and/or (2) the repeated

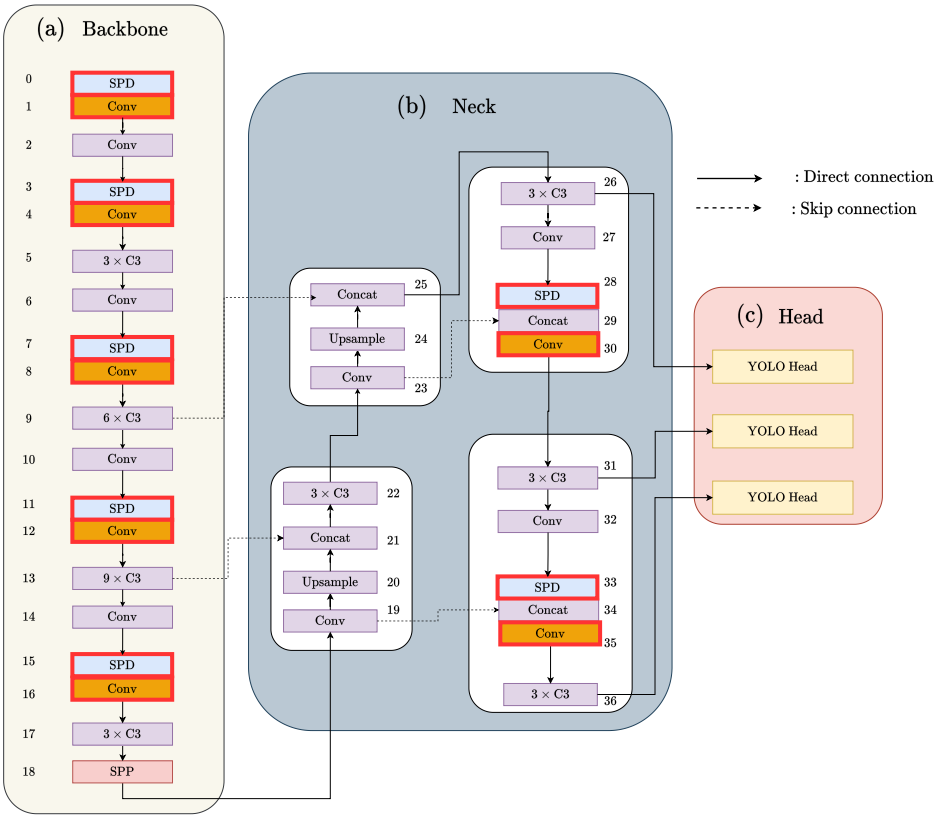


Figure 4. Overview of our YOLOv5-SPD. Red boxes are where the replacement happens.

times of C3 module (as in Figure 4), to obtain different versions of YOLOv5-SPD. The first is referred to as *width scaling* which changes the original width n_w (number of channels) to $\lceil n_w \times width\_factor \rceil_8$ (rounded off to the nearest multiple of 8). The second is referred to as *depth scaling* which changes the original depth n_d (times of repeating the C3 module; e.g., 9 as in $9 \times$ C3 in Figure 4) to $\lceil n_d \times depth\_factor \rceil$. This way, by choosing different width/depth factors, we obtain *nano, small, medium*, and *large* versions of YOLOv5-SPD as shown in Table 2, where factor values are chosen the same as YOLOv5 for the purpose of comparison in our experiments later.

Table 2. Scaling YOLOv5-SPD to obtain different versions that fit different use cases.

Models	Depth_Factor	Width_Factor
YOLOv5-SPD-*n*	0.33	0.25
YOLOv5-SPD-*s*	0.33	0.50
YOLOv5-SPD-*m*	0.67	0.75
YOLOv5-SPD-*l*	1.00	1.00

4.2. IMAGE CLASSIFICATION

A classification CNN typically begins with a stem unit that consists of a stride-2 convolution and a pooling layer to reduce the image resolution by a factor of four. A popular model is ResNet [13] which won the ILSVRC 2015 challenge. ResNet introduces residual connections to allow for training a network as deep as up to 152 layers. It also significantly reduces the total number of parameters by only using a single fully-connected layer. A softmax layer is employed at the end to normalize class predictions.

4.2.1. ResNet18-SPD and ResNet50-SPD. ResNet-18 and ResNet-50 both use a total number of four stride-2 convolutions and one max-pooling layer of stride 2 to downsample each input image by a factor of 2^5. Applying our proposed building block, we replace the four strided convolutions with SPD-Conv; but on on the other hand, we simply remove the max pooling layer because, since our main target is low-resolution images, the datasets used in our experiments have rather small images (64×64 in Tiny ImageNet and 32×32 in CIFAR-10) and hence pooling is unnecessary. For larger images, such max-pooling layers can still be replaced the same way by SPD-Conv. The two new architectures are shown in Table 3.

Table 3. Our ResNet18-SPD and ResNet50-SPD architecture.

Layer Name	ResNet18-SPD	ResNet50-SPD
spd1	**SPD-Conv**	
conv1	3×3 kernel, 64 output channels	
conv2	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
spd2	**SPD-Conv**	
conv3	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
spd3	**SPD-Conv**	
conv4	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
spd4	**SPD-Conv**	
conv5	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
fc (fully conn.)	Global avg. pooling + fc(no. of classes) + softmax	

5. EXPERIMENTS

This section evaluates our proposed approach SPD-Conv using two representative computer vision tasks, object detection and image classification.

5.1. OBJECT DETECTION

5.1.1. Dataset & Setup. We use the COCO-2017 dataset [1] which is divided into `train2017` (118,287 images) for training, `val2017` (5,000 images; also called `minival`) for validation, and `test2017` (40,670 images) for testing. We use a wide range of state-of-the-art baseline models as listed in Tables 4 and 5. We report the standard metric of average precision (AP) on `val2017` under different IoU thresholds [0.5:0.95] and object sizes (small, medium, large). We also report the AP metrics on `test-dev2017` (20,288

images) which is a subset of `test2017` with accessible labels. However, the labels are not publicly released but one needs to submit all the *predicted* labels in JSON files to the `CodaLab COCO Detection Challenge` [2] to retrieve the evaluated metrics, which we did.

5.1.2. Training. We train different versions (nano, small, medium, and large) of YOLOv5-SPD and all the baseline models on `train2017`. Unlike most other studies, we *train from scratch without using transfer learning*. This is because we want to examine the *true learning capability* of each model without being disguised by the rich feature representation it inherits via transfer learning from ideal (high quality) datasets such as ImageNet. This was carried out on our own models ($*$-SPD-n/s/m/l) and all the existing YOLO-series models (v5, X, v4, and their scaled versions like nano, small, large, etc.). The other baseline models still used transfer learning because of our lack of resource (training from scratch consumes an enormous amount of GPU time). However, note that this simply means that *those baselines are placed in a much more advantageous position* than our own models as they benefit from high quality datasets.

We choose the SGD optimizer with momentum 0.937 and a weight decay of 0.0005. The learning rate linearly increases from 0.0033 to 0.01 during three warm-up epochs, followed by a decrease using the Cosine decay strategy to a final value of 0.001. The *nano* and *small* models are trained on four V-100 32 GB GPU with a batch size of 128, while *medium* and *large* models are trained with batch size 32. CIoU loss [38] and cross-entropy loss are adopted for objectness and classification. We also employ several data augmentation techniques to mitigate overfitting and improve performance for *all* the models; these techniques include (i) photometric distortions of hue, saturation, and value, (ii) geometric distortions such as translation, scaling, shearing, fliplr and flipup, and (iii) multi-image enhancement techniques such as mosaic and cutmix. Note that augmentation is not used at inference. The hyperparameters are adopted from YOLOv5 without re-tuning.

Table 4. Comparison on MS-COCO validation dataset (val2017).

Model	Backbone	Image size	AP	AP_S (small obj.)	Params (M)	Latency (ms) (batch_size=1)
YOLOv5-SPD-*n*	-	640 × 640	**31.0**	**16.0** (**+13.15%**)	2.2	7.3
YOLOv5n	-	640 × 640	28.0	14.14	1.9	6.3
YOLOX-Nano	-	640 × 640	25.3	-	0.9	-
YOLOv5-SPD-*s*	-	640 × 640	**40.0**	**23.5** (**+11.4%**)	8.7	7.3
YOLOv5s	-	640 × 640	37.4	21.09	7.2	6.4
YOLOX-S	-	640 × 640	39.6	-	9.0	9.8
YOLOv5-SPD-*m*	-	640 × 640	**46.5**	**30.3** (**+8.6%**)	24.6	8.4
YOLOv5m	-	640 × 640	45.4	27.9	21.2	8.2
YOLOX-M	-	640 × 640	46.4	-	25.3	12.3
YOLOv5-SPD-*l*	-	640 × 640	48.5	**32.4** (**+1.8%**)	52.7	10.3
YOLOv5l	-	640 × 640	49.0	31.8	46.5	10.1
YOLOX-L	-	640 × 640	**50.0**	-	54.2	14.5
Faster R-CNN	R50-FPN	-	40.2	24.2	42.0	-
Faster R-CNN+	R50-FPN	-	42.0	26.6	42.0	-
DETR	R50	-	42.0	20.5	41.0	-
DETR-DC5	ResNet-101	800 × 1333	44.9	23.7	60.0	-
RetinaNet	ViL-Small-RPB	800 × 1333	44.2	28.8	35.7	-

5.1.3. Results. Table 4 reports the results on `val2017` and Table 5 reports the results on `test-dev`. The AP_S, AP_M, AP_L in both tables mean the AP for small/medium/large *objects*, which should not be confused with *model* scales (nano, small, medium, large). The image resolution 640 × 640 as shown in both tables is not considered high in object detection (as opposed to image classification) because the resolution on the actual objects is much lower, especially when the objects are small.

5.1.4. Results on Val2017. Table 4 is organized by model scales, as separated by horizontal lines (the last group are large-scale models). In the first category of nano models, our YOLOv5-SPD-*n* is the best performer in terms of both AP and AP_S: its AP_S is 13.15% higher than the runner-up, YOLOv5n, and its overall AP is 10.7% higher than the runner-up, also YOLOv5n.

In the second category, small models, our YOLOv5-SPD-*s* is again the best performer on both AP and AP_S, although this time YOLOX-S is the second best on AP.

Table 5. Comparison on MS-COCO test dataset (test-dev2017).

Model	ImgSize	Params (M)	AP	AP_{50}	AP_{75}	AP_S (small obj.)	AP_M	AP_L
YOLOv5-SPD-*n*	640 × 640	2.2	**30.4**	48.7	32.4	**15.1**(**+19%**)	33.9	37.4
YOLOv5n	640 × 640	1.9	28.1	45.7	29.8	12.7	31.3	35.4
EfficientDet-D0	512 × 512	3.9	33.8(Trf)	52.2	35.8	12.0	38.3	51.2
YOLOv5-SPD-*s*	640 × 640	8.7	**39.7**	59.1	43.1	**21.9**(**+9.5%**)	43.9	49.1
YOLOv5s	640 × 640	7.2	37.1	55.7	40.2	20.0	41.5	45.2
EfficientDet-D1	640 × 640	6.6	39.6	58.6	42.3	17.9	44.3	56.0
EfficientDet-D2	768 × 768	8.1	43.0(Trf)	62.3	46.2	22.5(Trf)	47.0	58.4
YOLOv5-SPD-*m*	640 × 640	24.6	**46.6**	65.2	50.8	**28.2**(**+6%**)	50.9	57.1
YOLOv5m	640 × 640	21.2	45.5	64.0	49.7	26.6	50.0	56.6
YOLOX-M	640 × 640	25.3	46.4	65.4	50.6	26.3	51.0	59.9
EfficientDet-D3	896 × 896	12.0	45.8	65.0	49.3	26.6	49.4	59.8
SSD512	512 × 512	36.1	28.8	48.5	30.3	-	-	-
YOLOv5-SPD-*l*	640 × 640	52.7	48.8	67.1	53.0	**30.0**	52.9	60.5
YOLOv5l	640 × 640	46.5	49.0	67.3	53.3	29.9	53.4	61.3
YOLOX-L	640 × 640	54.2	**50.0**	68.5	54.5	29.8	54.5	64.4
YOLOv4-CSP	640 × 640	52.9	47.5	66.2	51.7	28.2	51.2	59.8
PP-YOLO	608 × 608	52.9	45.2	65.2	49.9	26.3	47.8	57.2
YOLOX-X	640 × 640	99.1	51.2	69.6	55.7	31.2	56.1	66.1
YOLOv4-P5	896 × 896	70.8	51.8	70.3	56.6	33.4	55.7	63.4
YOLOv4-P6	1280 × 1280	127.6	54.5	72.6	59.8	36.8	58.3	65.9
RetinaNet (w/ SpineNet-143)	1280 × 1280	66.9	50.7	70.4	54.9	33.6	53.9	62.1

In the third, medium model category, the AP performance gets quite close although our YOLOv5-SPD-*m* still outperforms others. On the other hand, our AP_S has a larger winning margin (8.6% higher) than the runner-up, which is a good sign because SPD-Conv is especially advantageous for smaller objects and lower resolutions.

Lastly for large models, YOLOX-L achieves the best AP while our YOLOv5-SPD-*l* is only slightly (3%) lower (yet much better than other baselines shown in the bottom group). On the other hand, our AP_S remains the highest, which echos SPD-Conv's advantage mentioned above.

5.1.5. Results on Test-Dev2017. As presented in Table 5, our YOLOv5-SPD-*n* is again the clear winner in the nano model category on AP_S, with a good winning margin (19%) over the runner-up, YOLOv5n. For the average AP, although it appears as if

EfficientDet-D0 performed better than ours, that is because EfficientDet has almost double parameters than ours and was trained using high-resolution images (via transfer learning, as indicated by "Trf" in the cell) and AP is highly correlated with resolution. This training benefit is similarly reflected in the small model category too.

In spite of this benefit that other baselines receive, our approach reclaims its top rank in the next category, medium models, on both AP and AP_S. Finally in the large model category, our YOLOv5-SPD-*l* is also the best performer on AP_S, and closely matches YOLOX-L on AP.

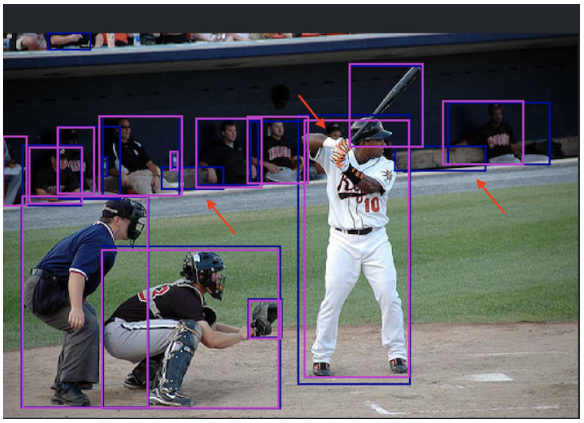
5.1.6. Summary. It is clear that, by simply replacing the strided convolution and pooling layers with our proposed SPD-Conv building block, a neural net can significantly improves its accuracy, while maintaining the same level of parameter size. The improvement is more prominent when objects are small, which meets our goal well. Although we do not constantly notch the first position in all the cases, SPD-Conv is the only approach that *consistently* performs very well; it is only occasionally a (very close) runner-up if not performing the best, and is *always* the winner on AP_S which is the chief metric we target.

Lastly, recall that we have adopted YOLOv5 hyperparameters without re-tuning, which means that our models will likely perform even better after dedicated hyperparameter tuning. Also recall that all the non-YOLO baselines (and PP-YOLO) were trained using transfer learning and thus have benefited from high quality images, while ours do not.

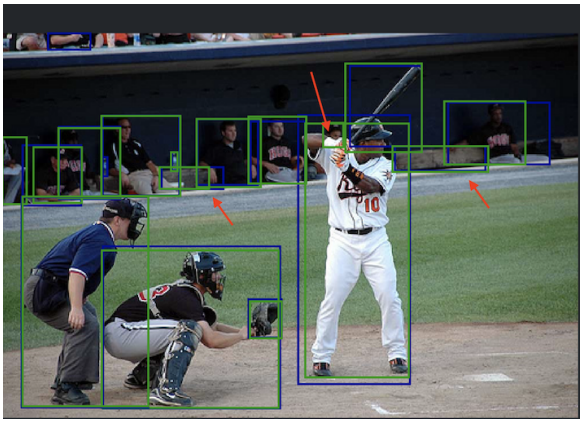
5.1.7. Visual Comparison. For a visual and intuitive understanding, we provide two real examples using two randomly chosen images, as shown in Figure 5. We compare YOLOv5-SPD-*m* and YOLOv5m since the latter is the best performer among all the baselines in the corresponding (medium) category. Figure 5(a)(b) demonstrates that YOLOv5-SPD-*m* is able to detect the occluded giraffe which YOLOv5m misses, and Figure 5(c)(d) shows that YOLOv5-SPD-*m* detects very small objects (a face and two benches) while YOLOv5m fails to.

(a) Purple boxes: YOLOv5m predictions.

(b) Green boxes: YOLOv5-SPD-*m* predictions.



(c) Purple boxes: YOLOv5m predictions.



(d) Green boxes: YOLOv5-SPD-*m* predictions.

Figure 5. Object detection examples from `val2017`. Blue boxes indicate the ground truth. Red arrows highlight the differences.

5.2. IMAGE CLASSIFICATION

5.2.1. Dataset & Setup. For the task of image classification, we use the Tiny ImageNet [19] and CIFAR-10 datasets [17]. Tiny ImageNet is a subset of the ILSVRC-2012 classification dataset and contains 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. Each image is of resolution $64 \times 64 \times 3$ pixels. CIFAR-10 consists of 60,000 images of resolution $32 \times 32 \times 3$, including 50,000 training images and 10,000 test images. There are 10 classes with 6,000 images per class. We use the top-1 accuracy as the metric to evaluate the classification performance.

5.2.2. Training. We train our ReseNet18-SPD model on Tiny ImageNet. We perform random grid search to tune hyperparameters including learning rate, batch size, momentum, optimizer, and weight decay. Figure 6 shows a sample hyperparameter sweep plot generated using the `wandb` MLOPs. The outcome is the SGD optimizer with a learning rate of 0.01793 and momentum of 0.9447, a mini batch size of 256, weight decay regularization of 0.002113, and 200 training epochs. Next, we train our ResNet50-SPD model on CIFAR-10. The hyperparameters are adopted from the ResNet50 paper, where SGD optimizer is used with an initial learning rate 0.1 and momentum 0.9, batch size 128, weight decay regularization 0.0001, and 200 training epochs. For both ReseNet18-SPD and ReseNet50-SPD, we use the same decay function as in ResNet to decrease the learning rate as the number of epochs increases.

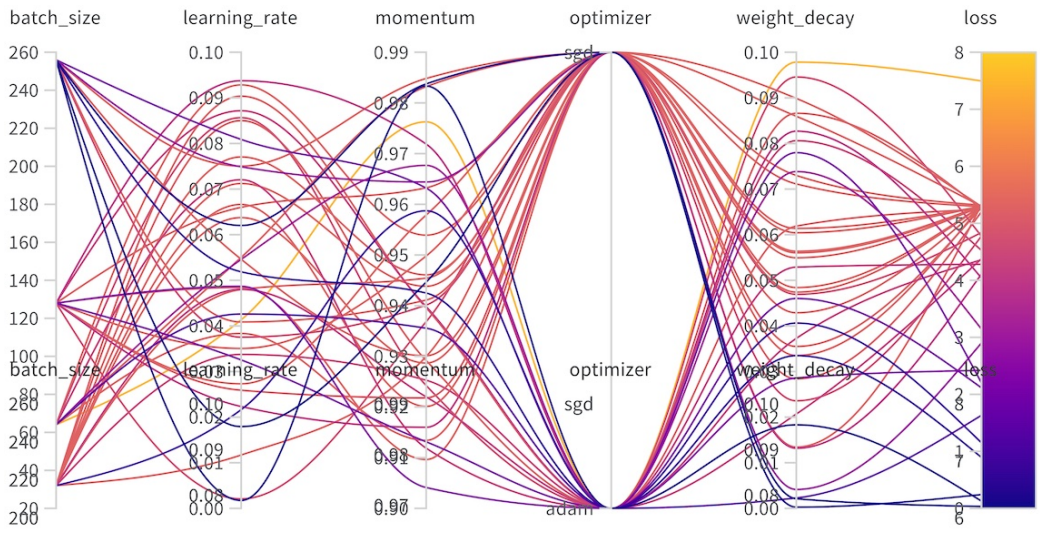


Figure 6. Hyperparameter tuning in image classification: a sweep plot using `wandb`.

5.2.3. Testing. The accuracy on Tiny ImageNet is evaluated on the validation dataset because the ground truth in the test dataset is not available. The accuracy on CIFAR-10 is calculated on the test dataset.

5.2.4. Results. Table 6 summarizes the results of top-1 accuracy. It shows that our models, ResNet18-SPD and ResNet50-SPD, clearly outperform all the other baseline models.

Table 6. Image classification performance comparison.

Model	Dataset	Top-1 accuracy (%)
ResNet18-SPD	Tiny ImageNet	**64.52**
ResNet18	Tiny ImageNet	61.68
Convolutional Nystromformer for Vision	Tiny ImageNet	49.56
WaveMix-128/7	Tiny ImageNet	52.03
ResNet50-SPD	CIFAR-10	**95.03**
ResNet50	CIFAR-10	93.94
Stochastic Depth	CIFAR-10	94.77
Prodpoly	CIFAR-10	94.90

Finally, we provide in Figure 7 a visual illustration using Tiny ImageNet. It shows 8 examples misclassified by ResNet18 and correctly classified by ResNet18-SPD. The common characteristics of these images is that the resolution is low and therefore presents a challenge to the standard ResNet which loses fine-grained information during its strided convolution and pooling operations.

6. CONCLUSION

This paper identifies a common yet defective design in existing CNN architectures, which is the use of strided convolution and/or pooling layers. It will result in the loss of fine-grained feature information especially on low-resolution images and small objects. We then propose a new CNN building block called SPD-Conv that eliminates the strided and pooling operations altogether, by replacing them with a space-to-depth convolution followed by a non-strided convolution. This new design has a big advantage of downsampling feature maps while retaining the discriminative feature information. It also represents a general

Figure 7. Green labels: ground truth. Blue labels: ResNet18-SPD predictions. Red labels: ResNet-18 predictions.

and unified approach that can be easily applied to perhaps any CNN architecture and to strided conv and pooling the same way. We provide two most representative use cases, object detection and image classification, and demonstrate via extensive evaluation that SPD-Conv brings significant performance improvement on detection and classification accuracy. We anticipate it to widely benefit the research community as it can be easily integrated into existing deep learning frameworks such as PyTorch and TensorFlow.

REFERENCES

[1] Coco dataset. https://cocodataset.org (2017)

[2] CodaLab COCO detection challenge (bounding box). https://competitions.codalab.org/competitions/20794 (2019)

[3] Adelson, E.H., Anderson, C.H., Bergen, J.R., Burt, P.J., Ogden, J.M.: Pyramid methods in image processing. RCA engineer **29**(6), 33–41 (1984)

[4] Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934 (2020)

[5] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: European Conference on Computer Vision. pp. 213–229. Springer (2020)

[6] Chevalier, M., Thome, N., Cord, M., Fournier, J., Henaff, G., Dusch, E.: Lr-cnn for fine-grained classification with varying resolution. In: 2015 IEEE International Conference on Image Processing (ICIP). pp. 3101–3105. IEEE (2015)

[7] Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., Tian, Q.: Centernet: Keypoint triplets for object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6569–6578 (2019)

[8] Ge, Z., Liu, S., Wang, F., Li, Z., Sun, J.: Yolox: Exceeding yolo series in 2021. arXiv preprint arXiv:2107.08430 (2021)

[9] Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1440–1448 (2015)

[10] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 580–587 (2014)

[11] Haris, M., Shakhnarovich, G., Ukita, N.: Task-driven super resolution: Object detection in low-resolution images. In: International Conference on Neural Information Processing. pp. 387–395. Springer (2021)

[12] He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence **37**(9), 1904–1916 (2015)

[13] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)

[14] Jocher, G., et al.: https://github.com/ultralytics/yolov5 (2021), released version available at the time of evaluation: Oct 12, 2021

[15] Kim, Y., Kang, B.N., Kim, D.: San: Learning relationship between convolutional features for multi-scale object detection. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 316–331 (2018)

[16] Koziarski, M., Cyganek, B.: Impact of low resolution on image recognition with deep neural networks: An experimental study. International Journal of Applied Mathematics and Computer Science **28**(4) (2018)

[17] Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 (canadian institute for advanced research) http://www.cs.toronto.edu/~kriz/cifar.html

[18] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. NeurIPS **25** (2012)

[19] Le, Y., Yang, X.: Tiny imagenet visual recognition challenge. CS 231N **7**(7), 3 (2015)

[20] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2117–2125 (2017)

[21] Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: IEEE ICCV. pp. 2980–2988 (2017)

[22] Liu, S., Qi, L., Qin, H., Shi, J., Jia, J.: Path aggregation network for instance segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8759–8768 (2018)

[23] Liu, S., Qi, L., Qin, H., Shi, J., Jia, J.: Path aggregation network for instance segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)

[24] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: European conference on computer vision. pp. 21–37. Springer (2016)

[25] Peng, X., Hoffman, J., Stella, X.Y., Saenko, K.: Fine-to-coarse knowledge transfer for low-res image classification. In: 2016 IEEE International Conference on Image Processing (ICIP). pp. 3683–3687. IEEE (2016)

[26] Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)

[27] Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems **28**, 91–99 (2015)

[28] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence **39**(6), 1137–1149 (2016)

[29] Sajjadi, M.S., Vemulapalli, R., Brown, M.: Frame-recurrent video super-resolution. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6626–6634 (2018)

[30] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[31] Singh, B., Davis, L.S.: An analysis of scale invariance in object detection - snip. In: IEEE CVPR. pp. 3578–3587 (2018)

[32] Singh, B., Najibi, M., Davis, L.S.: Sniper: Efficient multi-scale training. Advances in neural information processing systems **31** (2018)

[33] Singh, M., Nagpal, S., Vatsa, M., Singh, R.: Enhancing fine-grained classification for low resolution images. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2021)

[34] Tan, M., Pang, R., Le, Q.V.: Efficientdet: Scalable and efficient object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10781–10790 (2020)

[35] Tian, Z., Shen, C., Chen, H., He, T.: Fcos: Fully convolutional one-stage object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9627–9636 (2019)

[36] Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M.: Scaled-yolov4: Scaling cross stage partial network. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13029–13038 (2021)

[37] Wang, Z., Chang, S., Yang, Y., Liu, D., Huang, T.S.: Studying very low resolution recognition using deep networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4792–4800 (2016)

[38] Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., Ren, D.: Distance-iou loss: Faster and better learning for bounding box regression. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 12993–13000 (2020)

II. YOGA: DEEP OBJECT DETECTION IN THE WILD WITH LIGHTWEIGHT FEATURE LEARNING AND MULTISCALE ATTENTION

Raja Sunkara and Tie Luo (Corresponding author)
Department of Computer Science
Missouri University of Science and Technology
Rolla, MO 65409, USA
{rs5cq,tluo}@mst.edu

ABSTRACT

We introduce YOGA, a deep learning based yet lightweight object detection model that can operate on low-end edge devices while still achieving competitive accuracy. The YOGA architecture consists of a two-phase feature learning pipeline with a cheap linear transformation, which learns feature maps using only half of the convolution filters required by conventional convolutional neural networks. In addition, it performs multi-scale feature fusion in its neck using an attention mechanism instead of the naive concatenation used by conventional detectors. YOGA is a flexible model that can be easily scaled up or down by several orders of magnitude to fit a broad range of hardware constraints. We evaluate YOGA on COCO-val and COCO-testdev datasets with other over 10 state-of-the-art object detectors. The results show that YOGA strikes the best trade-off between model size and accuracy (up to 22% increase of AP and 23-34% reduction of parameters and FLOPs), making it an ideal choice for deployment in the wild on low-end edge devices. This is further affirmed by our hardware implementation and evaluation on NVIDIA Jetson Nano.

1. INTRODUCTION

Object detection empowered by deep learning has made booming success in diverse applications such as autonomous driving, medical imaging, remote sensing, and face detection. Research in this area has been thriving and the performance competition is fierce. Well-known detectors include the R-CNN series [9], YOLO series [22], SSD [20], RetinaNet [16], EfficientDet [26], YOLO-Anti [32], UDNet [6], etc. Although the fierce competition has led to better performance in general, it has also resulted in deeper neural network architectures and more complex model designs, implying a need for more training data, more tuning parameters, and longer training and inference time. This would not be suitable for resource-constrained environments such as Internet of Things (IoT) devices at the edge.

Researchers have attempted *Pruning and Quantization* methods toward this goal. However, that is an "aftermath" approach and the effect is often limited (for example, we applied PyTorch's pruning utility to three popular object detection models and observed a mere improvement of 0%, 8%, and -15%(negative), respectively). To fundamentally address this problem for edge deployment in the wild, a clean-slate design is much more desired.

In this paper, we propose YOGA, a new object detection model based on a resource-conscious design principle. YOGA cuts down model size by up to 34% (cf. Table 4), in terms of number of model parameters and FLOPs, yet notably, achieving competitive accuracy (often even better, by up to 22%; cf. Table 4). YOGA consists of (i) a new backbone called CSPGhostNet (*cross stage partial* GhostNet), (ii) a new neck called AFF-PANet (*attention feature fusion*-based *path aggregation network*), and (iii) a YOLO-based head. (The underlined letters account for the coined name, YOGA.) Our main idea is twofold. First, to slim down the neural network, we use a two-phase feature learning pipeline with a cheap linear transformation called group convolution throughout the network, which can

learn the same number of feature maps as in standard CNNs but using only half of the convolution filters. Second, to achieve high accuracy, we fuse multi-scale feature maps at the neck using a *local attention* mechanism along the channel dimension (besides global attention along the space dimension), rather than using the conventional *concatenation* which is essentially equal-weighted.

Apart from being lightweight and high-performing, YOGA also represents a flexible design in that it can be easily scaled up or down in a wide range by choosing different repetitions of one of its building blocks (CSPGhost; cf. Figure 2). This makes it easily fit for a broad range of applications with different resource constraints, from small embedded IoT systems to intermediate edge servers and to powerful clouds.

Besides the performance evaluation commonly seen in the literature, we have also implemented YOGA on real hardware, NVIDIA Jetson Nano 2GB (the *lowest-end* deep learning device from NVIDIA), and tested its performance to assess its applicability to edge deployment in the wild. The results are promising (for instance, YOGA-*n* runs at 0.57 sec per 640x640 image, which is close to real-time) and will surely be much more responsive on less-restrictive hardware (e.g., Jetson Nano 4GB, or Jetson TX2).

In summary, the contributions of this paper are:

- We propose YOGA, a new object detection model that learns richer representation (via attention based multi-scale feature fusion) with a much lighter model (reducing nearly half convolution filters via group convolution).

- We provide a theoretical explanation of how *label smoothing* facilitates backpropagation during training, by mathematically analyzing how the loss gradient vector is involved in the recursive backpropagation algorithm when label smoothing is used. We also overcome a GhostNet overfitting issue using a hyper-parameter tuning method based on Genetic Algorithm.

- We compare YOGA with a large variety of (over 10) state-of-the-art deep learning object detectors (as YOGA can be easily scaled up or down so we can make fair comparison with models at different levels of scales). The results demonstrate the superiority of YOGA on the joint performance of model size and accuracy.
- We also migrate YOGA to real hardware to assess its usability in the wild. Our experiments show that YOGA is well suited for even the lowest-end deep learning edge devices.

2. RELATED WORK AND PRELIMINARIES

Current state-of-the-art object detection models are convolutional neural network (CNN) based and can be categorized into one-stage and two-stage detectors, or anchor-based or anchor-free detectors. A two-stage detectors first use region proposals to generate coarse object proposals, and then use a dedicated per-region head to classify and refine the proposals. In contrast, one-stage detectors skip the region proposal step and run detection directly over a dense sampling of locations. Anchor-based methods use *anchor boxes*, which are a predefined collection of boxes that match the widths and heights of training data objects, to improve the loss convergence during training. We provide a classification of some well-known object detection models in Table 1. For a detailed review of such methods, the reader is referred to a comprehensive survey [18]. For a overview of deep-learning based methods for salient object detection in videos, refer to [33].

Generally, one-stage detectors are faster than two-stage ones and anchor-based models are more accurate than anchor-free ones. Thus, in the YOGA design, we focus on the one-stage and anchor-based models, i.e, the first cell of Table 1.

Table 1. A taxonomy of object detection models.

Model	Anchor-based	Anchor-free
One-stage	Faster R-CNN [23], SSD [20], RetinaNet [16], EfficientDet [26], YOLO [22]	FCOS [27], CenterNet [5], DETR [2], YOLOX [7]
Two-stage	R-CNN [9], Fast R-CNN [8]	RepPoints [35], CenterNet2 [5]

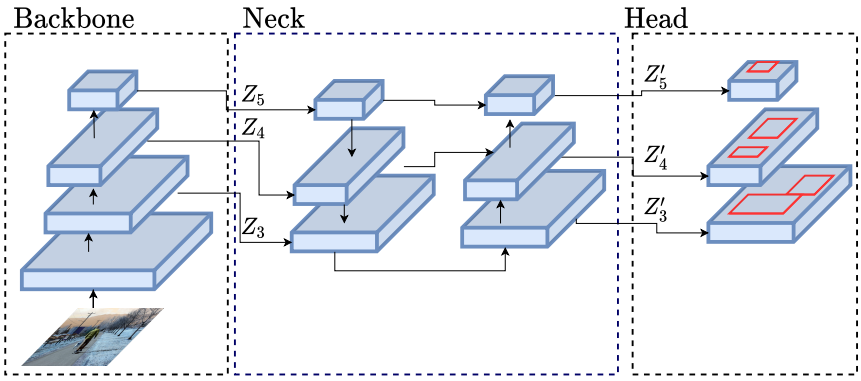


Figure 1. A one-stage object detection model generally consists of a *backbone* for feature extraction, a *neck* for feature fusion, and a *head* for regression and classification.

A typical one-stage object detection model is depicted in Figure 1. It consists of a CNN-based *backbone* for visual feature extraction and a detection *head* for predicting class and bounding box of each contained object. In between, a *neck* is added to combine features at multiple scales to produce semantically strong features for detecting objects of different sizes.

3. DESIGN OF YOGA

An overview of the YOGA architecture is given in Figure 2.

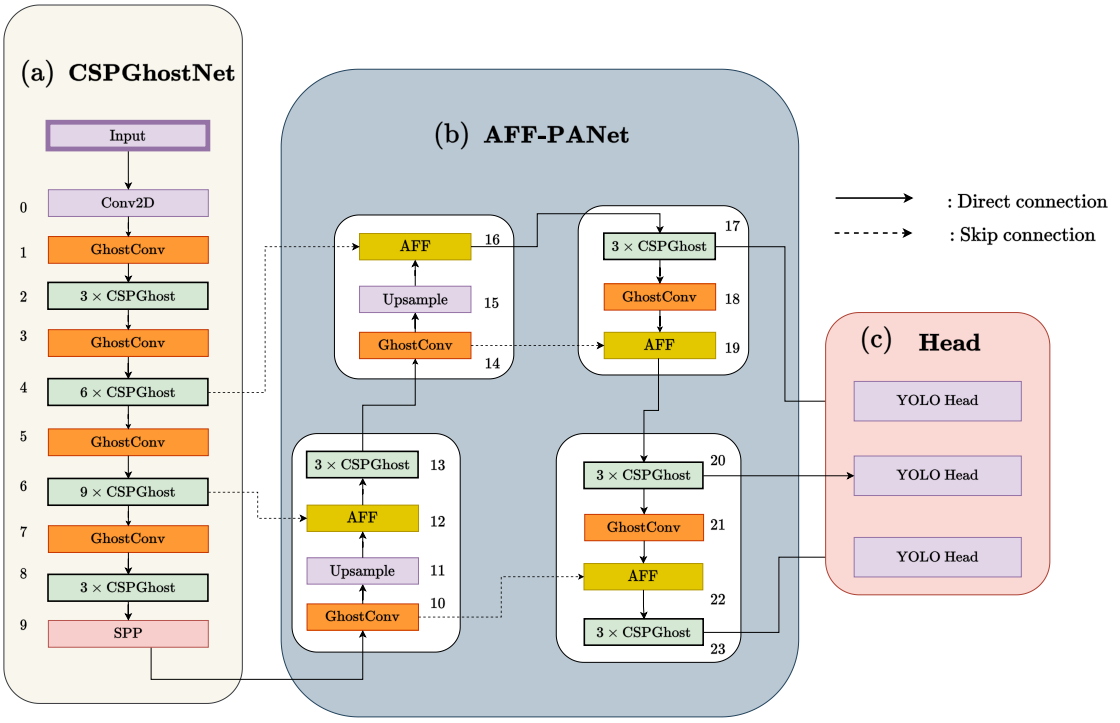


Figure 2. The YOGA architecture: (a) Backbone, (b) Neck, (c) Head. Zoom-in view of CSPGhost module (light green) is provided in Figure 3. The $n\times$ repetition allows our model to scale up and down easily.

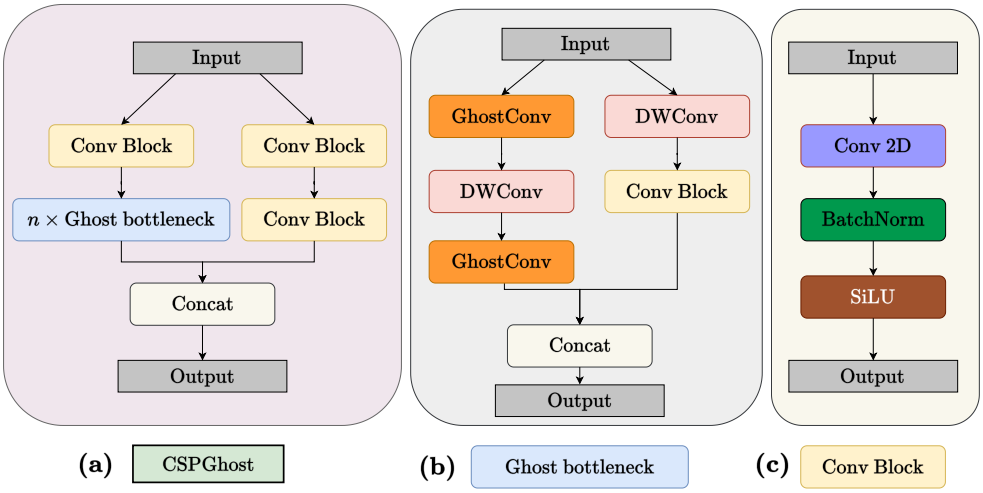


Figure 3. (a) Internal structure of our CSPGhost module. (b) The Ghost bottleneck layer (light blue in CSPGhost), where DWConv stands for depth-wise convolution. (c) The Conv Block (yellow) used in both (a) and (b).

3.1. BACKBONE: CSPGHOSTNET

Our design of backbone, called CSPGhostNet, is motivated by two observations and guided by two corresponding aims. First, we identify that standard CNNs add many redundant features in order to learn better representation of input images. For example, ResNet [12] generates numerous similar feature maps. Such designs come at a price of high computational cost and heavyweight models. Therefore, we pose the following question as our first aim: Is it possible to generate the same number of features with similar (necessary) redundancy but using much less computation and less parameters?

Second, we observe that the training and inference time of current deep learning models has a large room to improve. We are therefore motivated to also speed up training and inference processes in our context, object detection.

To achieve the first aim, we adapt GhostNet [10] to exploit a low-cost two-phase convolutional pipeline. For the second aim, we integrate half of the feature maps across backbone and neck from the beginning to the end to create a shortcut, by leveraging CSPNet [30]. Specifically, it splits each input feature map into two parts, feeds one part through a group of convolution blocks while letting the other part bypass those blocks, and merges these two branches at the end via concatenation. This shortcut also reduces the repetition of gradient information during backpropagation. In the following, we focus on explaining how we achieve the first aim using GhostNet because CSPNet can be applied without much change to its original architecture (however, we are the first that creates a new module combining GhostNet and CSPNet).

Ghost bottleneck (G-bneck) as in Figure 3(b), which we draw *based on* the GhostNet paper [10] (but does not exist in [10]), was designed specially for small CNNs. It is not trivial to use G-bneck to build medium and large CNNs. In fact, this is also the reason why GhostNet, which is built on top of G-bneck, was compared with only small neural nets

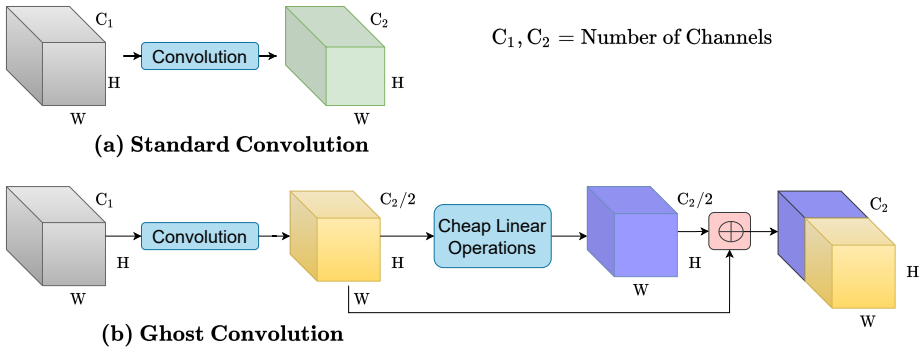


Figure 4. Ghost convolution vs. standard convolution.

like MobileNetv2 and MobileNetv3. To overcome this limit, we designed a new CSPGhost module as in Figure 3(a), where G-bneck layer is just part of it (the light blue block). This CSPGhost module allows us to build medium and large CNNs.

CSPGhost (in light green) is located at multiple positions in both our backbone and neck (see Figure 2), and its internal structure is shown in Figure 3(a). CSPGhost contains a Ghost bottleneck layer (in light blue) and multiple Conv Blocks (in light yellow). Each Conv Block consists of a 2D Convolution, a BatchNorm, and a SiLU non-linear activation function (Figure 3(c)). The Ghost bottleneck layer is similar to ResNet's basic residual block that integrates several convolutional layers and short-cut connections. It mainly consists of two GhostConv modules (in orange) with depth-wise convolution in between: the first GhostConv module acts as an expansion layer that increases the number of channels, while the second GhostConv module reduces the number of channels to match the input shortcut path, after which the input of the first GhostConv and the output of the second GhostConv is connected by the shortcut through the depth-wise convolution and Conv block.

The GhostConv block (in orange) stands for Ghost convolution and its structure is given in Figure 4. In standard convolution, C_2 filters each of depth C_1 will be used to transform an input feature map of depth C_1 to an output feature map of depth C_2, as shown in Figure 4a. In contrast, GhostConv uses only $C_2/2$ standard filters to generate

an intermediate feature map, denoted by X_a, of depth $C_2/2$, and then applies a *group convolution* with $C_2/2$ groups to X_a to generate a feature map, X_b, of identical depth $C_2/2$. Group convolution with $C_2/2$ groups is a cheap linear transformation which only performs *per-channel* instead of cross-channel convolution as in the standard convolution. Finally, the two feature maps X_a and X_b are concatenated to obtain the output feature map, which has a depth of C_2.

Mathematically, we can formulate this process as follows:

$$\begin{aligned} X_a &= \boldsymbol{w_1} \otimes \boldsymbol{x_0} \text{ (first half; std conv)} \\ X_b &= \boldsymbol{w_2} \tilde{\otimes} X_a \text{ (second half; group conv)} \\ \boldsymbol{y} &= X_a \oplus X_b \text{ (output feature maps)} \end{aligned} \tag{1}$$

where $\otimes$ denotes the standard convolution, $\tilde{\otimes}$ denotes the group convolution, and $\oplus$ denotes concatenation along the channel dimension. Thus, we can see that GhostConv adopts ordinary convolution to generate a few intrinsic feature maps and then utilizes cheap linear operations to augment the features and increase the number of channels.

Because of this, GhostConv can speed up the convolution process as well as reduce the number of parameters by 2 approximately. In general, the improvement factor is $s = C_2/D(X_a)$ where $D(X)$ is the number of channels of a feature map X. Based on the empirical analysis in [10], $s = 2$ results in the best performance. Therefore, we choose half filters to generate intermediate feature maps in GhostConv as shown in Figure 4.

Our redesigned backbone CSPGhostNet enables YOGA to substantially reduce the number of parameters and FLOPs without sacrificing its detection performance (mAP). Moreover, as a general guideline in deep learning, less parameters also tend to imply a more generalizable neural network.

Finally, we add spatial pyramid pooling (SPP) [11] to the tail of our backbone network in order to increase the receptive field.

3.2. NECK: AFF-PANET

We also design a new neck architecture called AFF-PANet that addresses a fundamental problem in object detection: *feature fusion*. An object detection task inevitably requires fusing low-level and high-level feature maps extracted from the backbone. However, current research all centers around fusing these feature maps using a naive concatenation with no learning involved. As illustrated in Figure 1, such a neck simply stacks the feature maps Z_3, Z_4 and Z_5 along the channel dimension, and then applies a standard convolution to match the output channels.

The problem with this kind of naive fusion is that *concatenation essentially treats each feature map equally*, but the features learned by the backbone have multiple scales and larger-scale ones tend to overshadow smaller-scale ones. Therefore, we propose to incorporate *learning* into the fusion process using an *attention mechanism*. However, this is non-trivial because typical attention methods such as SENet [13] cannot be directly applied to multi-scale features. The underlying reason is that those channel attention mechanisms use an extreme and coarse feature descriptor that implicitly assumes that large objects occupy a large portion of space and averages the feature maps across the spatial dimension, which would wipe out much of the image signal present in small objects. More specifically, such methods compress each feature map into a scalar and thus the average of feature maps along the spatial dimension becomes very small, resulting in poor detection of small objects. In fact, these are *global attention* mechanisms alone which cannot well handle multi-scale feature fusion.

Our design is the first that introduces AFF [4] into the area of object detection in order to add *local attention* to feature fusion (besides global attention), and the first that incorporates it in PANet [19] to shorten the pathway of passing feature information to the head. The paper [4] proposed an AFF module in Feature Pyramid Networks (FPN), but in our case, we integrated the AFF module into the Path Aggregation Network (PANet) and build a new neck architecture called AFF-PANet, which we explain the details below.

AFF uses a multi-scale channel attention module (MS-CAM) as depicted in Figure 5. Given an intermediate feature map $\mathbf{Z} \in \mathbb{R}^{C \times H \times W}$, there are two network pathways, one computing a global channel context $\mathbf{g}(\mathbf{Z})$ and the other is responsible for computing a local channel context $\mathbf{L}(\mathbf{Z})$. The two contexts $\mathbf{g}(\mathbf{Z})$ and $\mathbf{L}(\mathbf{Z})$ are then combined through a broadcasting addition operation, followed by a Sigmoid non-linearlity to map values into the range of 0-1, to obtain the attentional weights $\mathbf{M}(\mathbf{Z}) = \sigma(\mathbf{g}(\mathbf{Z}) \oplus \mathbf{L}(\mathbf{Z}))$. Therefore, by ushering AFF into object detection, we mainly exploit its local attention pathway, i.e., $\mathbf{L}(\mathbf{Z})$.

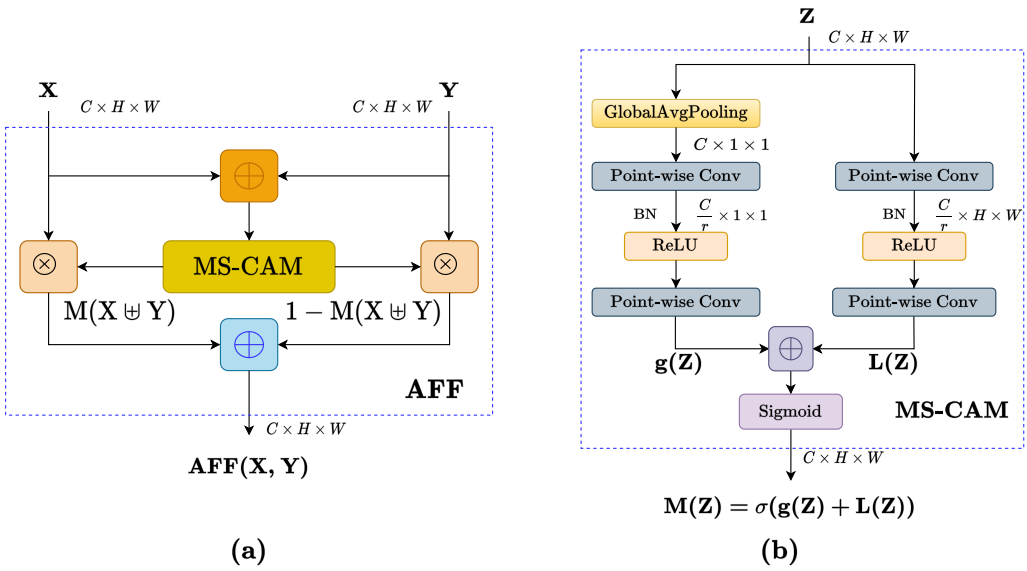


Figure 5. (a) Attention Feature Fusion (AFF). (b) Multi-scale channel attention module (MS-CAM).

Mathematically, this process can be expressed as

$$\mathbf{AFF}(\mathbf{X}, \mathbf{Y}) = \mathbf{M}(\mathbf{X} \uplus \mathbf{Y}) \odot \mathbf{X} + (1 - \mathbf{M}(\mathbf{X} \uplus \mathbf{Y})) \odot \mathbf{Y} \tag{2}$$

where $\mathbf{X} \in \mathbb{R}^{C \times H \times W}$ is a low-level feature map, and $\mathbf{Y} \in \mathbb{R}^{C \times H \times W}$ is a high-level feature map, $\uplus$ denotes the initial feature integration which we choose to be element-wise summation, $\mathbf{M}(\mathbf{X} \uplus \mathbf{Y})$ is a MS-CAM function that computes fusion weights between 0 and 1, which are

finally applied to $\mathbf{X}$ and $\mathbf{Y}$ to form a weighted average. Corresponding to Figure 2, $\mathbf{X}$ and $\mathbf{Y}$ are the outputs of blocks 4 and 15 respectively, and $\mathbf{AFF}(\mathbf{X}, \mathbf{Y}) \in \mathbb{R}^{C \times H \times W}$ is the fused feature as the output of block 16.

We also incorporate PANet in our neck. The reason is as follows. The feature pyramid network (FPN) [15] is comprised of only a top-down pathway to concatenate feature maps (cf. Figure 1 left part of neck), but PANet adds a bottom-up pathway and multiple lateral connections. This addition will help shorten the path of passing feature information from earlier layers to the head through only a few convolutional layers, thereby learning richer representations for multi-scale objects (as shorter paths have a better gradient flow from earlier CNN layers to the neck).

In summary, our AFF-based neck design introduces learning into multi-scale feature fusion by combining feature maps using learnable weights rather than naive concatenation.

3.3. HEAD: YOLO

The purpose of the head is to perform dense predictions, where each prediction consists of an object confidence score, a probability distribution of the object classes, and bounding box coordinates. A head makes these predictions based on the feature maps (Z'_3, Z'_4, and Z'_5 as shown in Figure 1), obtained from the neck.

We adopt the YOLO head architecture which consists of a 3×3 convolution layer followed by a 1×1 convolution layer. The number of filters used in this 1×1 convolution layer is $N(C + 5)$, where C is the number of classes and N is the number of anchor boxes (each prediction is made by using an anchor at one of three different scales). The output of the head is post-processed by non-maximum suppression (NMS) to eliminate redundant and low-confidence bounding boxes.

3.4. LABEL SMOOTHING

We use a regularization technique called label smoothing [25] to improve backpropagation gradients during neural network training. Unlike one-hot vector where the entire probability mass is concentrated on a single true class, label smoothing weighs $1-(K-1)\epsilon$ on the true class and ϵ on the remaining $K-1$ classes. This section provides an in-depth mathematical explanation of how this method helps backpropagation during model training, as [25] proposed it only heuristically.

Given an input sample, let $\mathbf{y}$ be its true label encoded by label smoothing, and $\mathbf{y}_n$ be its prediction made at a neural network's last (the n-th) layer. Using the cross-entropy loss $L(\mathbf{y}, \mathbf{y}_n) = -\sum_{i=1}^{K} y_i \log y_{n_i}$, the gradient of this loss with respect to the predicted output $\mathbf{y}_n$ is given by

$$\nabla_{\mathbf{y}_n} L = \begin{cases} -\frac{1-(K-1)\epsilon}{y_{n_c}}, & \text{for the true class } c \\ -\frac{\epsilon}{y_{n_i}}, & \text{any other classes } i \end{cases} \tag{3}$$

This gradient is then applied to the recursive backpropgation step given below:

$$\begin{aligned} &\nabla_{\mathbf{z}_k} L = (\nabla_{\mathbf{y}_k} L) J_{\mathbf{y}_k}(\mathbf{z}_k), \quad \nabla_{\mathbf{y}_{k-1}} L = (\nabla_{\mathbf{z}_k} L)\mathbf{W}_k \\ &\nabla_{\mathbf{W}_k} L = \mathbf{y}_{k-1} \nabla_{\mathbf{z}_k} L, \qquad \nabla_{\mathbf{b}_k} L = \nabla_{\mathbf{z}_k} L \end{aligned} \tag{4}$$

where $\mathbf{z}_k$ and $\mathbf{y}_k$ represent the pre-activation and post-activation vectors, respectively, at layer k. We compute Jacobian $J_{\mathbf{y}_n}(\mathbf{z}_n)$ using the last layer activation function, and then apply the recursion (4) from $k = n$ (last layer) to 1 (first layer) to compute all the gradients. The Jacobain matrix $J_{\mathbf{y}_k}(\mathbf{z}_k)$ and weight matrix $\mathbf{W}_k$ for the k-th layer are given, respectively, by

$$\begin{bmatrix} \frac{\partial y_{k_1}}{\partial z_{k_1}} & \frac{\partial y_{k_1}}{\partial z_{k_2}} & \cdots & \frac{\partial y_{k_1}}{\partial z_{k_D}} \\ \frac{\partial y_{k_2}}{\partial z_{k_1}} & \frac{\partial y_{k_2}}{\partial z_{k_2}} & \cdots & \frac{\partial y_{k_2}}{\partial z_{k_D}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{k_M}}{\partial z_{k_1}} & \frac{\partial y_{k_M}}{\partial z_{k_2}} & \cdots & \frac{\partial y_{k_M}}{\partial z_{k_D}} \end{bmatrix}, \begin{bmatrix} w_{11}^{(k)} & w_{21}^{(k)} & \cdots & w_{D_{k-1}1}^{(k)} \\ w_{12}^{(k)} & w_{22}^{(k)} & \cdots & w_{D_{k-1}2}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1D_k}^{(k)} & w_{2D_k}^{(k)} & \cdots & w_{D_{k-1}D_k}^{(k)} \end{bmatrix} \tag{5}$$

where D_k represents the number of neurons in the k-th layer.

For the label-smoothing based loss, all the entries of the gradient $\nabla_{\mathbf{y}_n} L$ (Eq. 3) are non-zero. As this gradient vector is multiplied with the Jacobian $J_{\mathbf{y}}(\mathbf{z})$ and the weight matrix $\mathbf{W}_k$ in the recursive backpropagation step, using such gradient to update weights during gradient descent would significantly mitigate the gradient vanishing problem and thus help the training and convergence of deep neural networks.

4. PERFORMANCE EVALUATION

For an extensive performance evaluation, we compare YOGA with a large number of state-of-the-art object detection models as our baselines, including YOLOv5, EfficientDet, YOLOX, YOLOv4, PP-YOLO, DETR, Faster-RCNN, SSD512, etc. and the complete list can be seen from Tables 3 and 4. Code is available at https://github.com/LabSAINT/YOGA.

As mentioned before, YOGA can easily scale up or down to suit different application or hardware needs. For example, we have tested that its Nano version YOGA-*n* can run near real-time on Jetson Nano and its Large version YOGA-*l* can run real-time on a V-100 GPU.

Specifically, one can scale YOGA by simply adjusting the number of filters in each convolutional layer (i.e., *width* scaling) and the number of convolution layers in the backbone (i.e., *depth* scaling) to obtain different versions of YOGA, such as Nano, Small, Medium, and Large. The width and depth scaling will result in a new width of $\lceil n_w \times \text{width factor} \rceil_8$ and a new depth of $\lceil n_d \times \text{depth factor} \rceil$, respectively, where n_w is the original width and n_d is the original number of repeated blocks (e.g., 9 as in 9× CSPGhost as in Figure 2), and

$\lceil \cdot \rceil_8$ means rounded off to the nearest multiple of 8. The width/depth factors are given in Table 2, where YOGA-*n*/*s*/*m*/*l* correspond to the Nano, Small, Medium, and Large versions of YOGA, respectively.

Table 2. Depth and Width scaling factors in YOGA.

Model	Depth Factor	Width Factor
YOGA-*n*	0.50	0.33
YOGA-*s*	0.57	0.68
YOGA-*m*	1.00	1.20
YOGA-*l*	1.50	1.41

4.1. EXPERIMENT SETUP

4.1.1. Dataset and Metrics. We use the COCO-2017 dataset [17] which is divided into `train2017` (118,287 images) for training, `val2017` (5,000 images; also called `minival`) for validation, and `test2017` (40,670 images) for testing. We use a wide range of state-of-the-art baseline models as listed in Tables 3 and 4. We report the standard metric of average precision (AP) on `val2017` under different IoU thresholds [0.5:0.95] and object sizes (small, medium, large). We also report the AP metrics on `test-dev2017` (20,288 images) which is a subset of `test2017` with accessible labels. However, the labels are not publicly released but one needs to submit all the *predicted* labels in JSON files to the `CodaLab COCO Detection Challenge` [3] to retrieve the evaluated metrics, which we did.

4.1.2. Training. We train different versions (nano, small, medium, and large) of YOGA on `train2017`. Unlike most other studies, we *train from scratch without using transfer learning*. This is because we want to examine the *true learning capability* of each model without being disguised by the rich feature representation it inherits via transfer learning from ideal (high quality) datasets such as ImageNet. This was carried out on our own models (YOGA-n/s/m/l) and all the existing YOLO-series models (v5, X, v4, and their

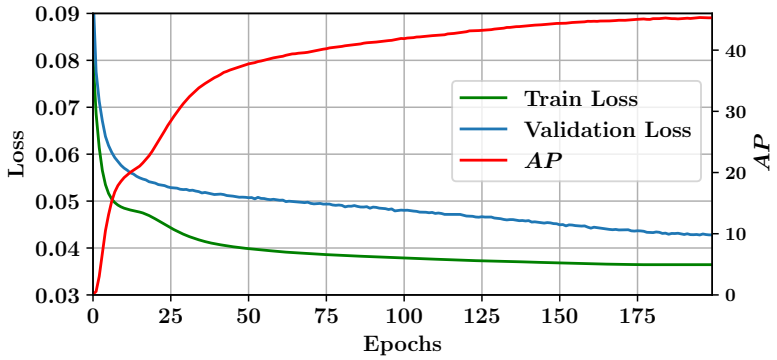


Figure 6. The training loss (green line) and validation loss (blue line) refer to the localization loss. The red line denotes AP values on validation data.

scaled versions like nano, small, large, etc.). The other baseline models still used transfer learning because of our lack of resource (training from scratch consumes an enormous amount of GPU time). However, note that this simply means that *those baselines are placed in a much more advantageous position* than our own models as they benefit from high quality datasets.

We apply this the same way to our YOLO series baselines (v5, X, v4, and their scaled versions) as well. On the other hand, other baseline models still use (and thus benefit from) transfer learning because of lack of resource (training from scratch consumes enormous time and GPU resources). However, this simply means that those baselines are in a *much more advantageous* position than YOGA.

4.1.3. Hyperparameter Tuning. We use the Genetic Algorithm (GA) to tune hyperparameters for YOGA. We ran GA on $m = 20$ hyperparameters for 200 generations on a subset of the COCO dataset. Our choice of $m = 20$ is based on the Vapnik–Chervonenkis (VC) inequality:

$$P[|E_{in} - E_{out}| > \epsilon] \leq 4m_h(2N)e^{-\frac{1}{8}\epsilon^2 N} \tag{6}$$

where E_{in} is the error on the validation set, E_{out} is the error on the test set, N is the number of validation samples, and m_h is the growth function of a hypothesis set defined by m. For a small $\epsilon = 0.05$ and with probability 95%, we choose $N \geq 10\times$ VC-dimension by rule of thumb for good generalization ($E_{in} \approx E_{out}$), where VC-dimension [1] is the number of independent parameters which is upper-bounded by m. Since COCO validation data contains $N = 5000$ images, choosing $m = 20$ satisfies the above inequality, which ensures the difference between test and validation error to be arbitrarily small according to (6). Therefore, our mAP estimates computed from validation data will be reliable to use and is a good proxy for mAP on test data.

The hyperparameters of GA-based optimization are as follows. It uses the SGD optimizer with momentum 0.937, weight decay of 0.005, a learning rate that linearly increases from 0.0033 to 0.01 for the first three epochs, and then decreases using the Cosine decay strategy to a final value of 0.001. The total number of epochs is 200, which is chosen based on our observation as shown in Figure 6, where the model enters the overfitting region beyond 200 epochs.

In neural network training, a larger batch size can lead to faster training, but it also requires more memory. On the other hand, a smaller batch size may require more training steps, but it may also be more memory efficient. When training on a GPU, the available memory is a limiting factor on the maximum batch size. Therefore, we trained our YOGA nano and small models on 4 V-100 32 GB GPU with a batch size of 128, and medium and large models with batch size 32. We employed CIoU loss for objectness and cross-entropy loss for classification. To mitigate overfitting, we applied several data augmentation techniques following YOLOv5, including photometric distortions of hue, saturation, and value, as well as geometric distortions such as translation, scaling, shearing, fliplr and flipup. Multi-image enhancement techniques such as mosaic and cutmix were also employed.

For baselines, we use their best hyperparameters stated in their respective papers, or given in their respective online repositories.

Table 3. Results on `MS-COCO Validation` Dataset. Percentages are in comparison against the closest performer.

Models	Backbone	Image-size	AP	AP_S	Params (M)	FLOPs (B)
YOGA-*n*	CSPGhostNet-*n*	640 × 640	**32.3** (**+15.35%**)	**15.2**(**+7.4%**)	1.9	4.9
YOLO-*n* [14]	-	640 × 640	28.0	14.14	1.9	4.5
YOLOX-Nano [7]	-	640 × 640	25.3	-	0.9	1.08
YOGA-*s*	CSPGhostNet-*s*	640 × 640	**40.7** (**+8.8%**)	**23.0**(**+9.0%**)	7.6 (+5%)	16.6 (+0.6%)
YOLO-*s* [14]	-	640 × 640	37.4	21.09	7.2	16.5
YOLOX-S [7]	-	640 × 640	39.6	-	9.0	26.8
YOLOv7-tiny [29]	-	640 × 640	38.7	-	6.2	13.8
YOGA-*m*	CSPGhostNet-*m*	640 × 640	45.2	28.0	**16.3** (**-23%**)	**34.6** (**-29%**)
YOLO-*m* [14]	-	640 × 640	45.4	27.9	21.2	49.0
YOLOX-M [7]	-	640 × 640	46.4	-	25.3	73.8
YOGA-*l*	CSPGhostNet-*l*	640 × 640	48.9	31.8	**33.6** (**-27.7%**)	**71.8** (**-34%**)
YOLO-*l* [14]	-	640 × 640	49	31.8	46.5	109.1
YOLOX-L [7]	-	640 × 640	50.0	-	54.2	155.6
YOLOv7 [29]	-	640 × 640	51.2	-	36.9	104.7
HTC [31]	HRNetV2p-W48	800 × 1333	47.0	28.8	79.42	399.12
HoughNet [24]	HG-104	-	46.1	30.0	-	-
DETR-DC5 [2]	ResNet-101	800 × 1333	44.9	23.7	60	254
RetinaNet [36]	ViL-Small-RPB	800 × 1333	44.2	28.8	35.68	254.8
YOLOv7-X [29]	-	640 × 640	52.9	-	71.3	189.9

4.2. RESULTS

With no test-time augmentation, we compare YOGA with baselines at the image resolution of 640 × 640. Table 3 reports the results on the validation dataset (5000 images with ground truth). Table 4 reports the results on the test-dev dataset (20000 images with no public ground truth). In order to obtain the accuracy on test-dev, we submitted all our predictions to the `CodaLab COCO Detection Challenge (Bounding Box)` [3] in JSON files. The AP_S/AP_M/AP_L in Table 3 and 4 means AP obtained on small/medium/large *objects* (not model scales). To simplify notation (e.g., in tables and figures), this section denotes by YOLO the YOLOv5 latest version v6.1 release in February 2022.

The scales of YOLO-v7 models (tiny-6.2M, base-36.9M, and X-71.3M) do not match our and and other baseline models, and thus prevent a fair comparison. For example, YOGA-*n/m/l* has only 1.9/16.3/33.6 M parameters, hence we did not compare YOGA with YOLOv7. However, we still included YOLO-v7 results in both Tables 3 and 4 for reference.

Table 4. Results on `MS-COCO Test-Dev` Dataset. Percentages are in comparison against the closest performer.

Method	ImgSize	FPS	Params (M)	FLOPs (B)	AP[0.5: 0.95]	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOGA-*n*	640	74	1.9	4.9	**32.3** (+15%)	50.3	34.6	**14.2** (+12%)	**34.7** (+11%)	**43.1** (+22%)
YOLO-*n* [14]	640	158	1.9	4.5	28.1	45.7	29.8	12.7	31.3	35.4
EfficientDet-D0 [26]	512	98.0	3.9	2.5	33.8	52.2	35.8	12.0	38.3	51.2
YOGA-*s*	640	67	7.6 (+5%)	16.6 (+0.6%)	**40.3** (+8.6%)	59.1	43.5	20.4	**43.5** (+5%)	**53.1** (+17%)
YOLO-*s* [14]	640	156	7.2	16.5	37.1	55.7	40.2	20.0	41.5	45.2
EfficientDet-D1 [26]	640	74.1	6.6	6.1	39.6	58.6	42.3	17.9	44.3	56.0
EfficientDet-D2 [26]	768	56.5	8.1	11	43.0	62.3	46.2	22.5	47.0	58.4
YOLOv7-tiny-SiLU [29]	640	286	6.2	13.8	38.7	56.7	41.7	18.8	42.4	51.9
YOGA-*m*	640	64	**16.3** (-23%)	**34.6** (-29%)	46.4	65.0	50.3	26.6	50.1	**58.9** (+4%)
YOLO-*m* [14]	640	121	21.2	49.0	45.5	64.0	49.7	26.6	50.0	56.6
EfficientDet-D3 [26]	896	34.5	12	25	45.8	65.0	49.3	26.6	49.4	59.8
YOLOX-M [7]	640	81.3	25.3	51.4	46.4	65.4	50.6	26.3	51.0	59.9
SSD512 [20]	-	-	36.1	-	28.8	48.5	30.3	-	-	-
YOGA-*l*	640	62	**33.6** (-27.7%)	**71.8** (-34%)	47.9 (-2.2%)	66.4	51.9	28.0 (-6.3%)	51.6	60.6
YOLO-*l* [14]	640	99	46.5	109.1	49.0	67.3	53.3	29.9	53.4	61.3
YOLOX-L [7]	640	69.0	54.2	115.6	50.0	68.5	54.5	29.8	54.5	64.4
YOLOv4-CSP [28]	640	73	52.9	-	47.5	66.2	51.7	28.2	51.2	59.8
PP-YOLO [21]	608	72.9	52.9	-	45.2	65.2	49.9	38.4	59.4	67.7
YOLOv7 [29]	640	161	36.9	104.7	51.4	69.7	55.9	31.8	55.5	65.0
YOLOX-X [7]	640	57.3	99.1	219.0	51.2	69.6	55.7	31.2	56.1	66.1
YOLOv4-P5 [28]	896	43	70.8	-	51.8	70.3	56.6	33.4	55.7	63.4
YOLOv4-P6 [28]	1280	32	127.59	-	54.5	72.6	59.8	36.8	58.3	65.9
YOLOv4-P7 [28]	1536	17	287.57	-	55.5	73.4	60.8	38.4	59.4	67.7
EfficientDet-D5 [26]	1280	-	34	135	51.5	70.5	56.7	33.9	54.7	64.1
ATSS	800	-	-	-	46.3	64.7	50.4	27.7	49.8	58.4
EfficientDet-D6 [26]	1280	-	52	226	52.6	71.5	57.2	34.9	56.0	65.4
RDSNet (R-101) [34]	800	-	0	-	38.1	58.5	40.8	21.2	41.5	48.2
RetinaNet (SpineNet-143)	1280	-	66.9	524.4	50.7	70.4	54.9	33.6	53.9	62.1
YOLOv7-X [29]	640	114	71.3	189.9	53.1	71.2	57.8	33.8	57.1	67.4

Figure 7 provides a comparison of YOGA with multiple SOTA models in terms of AP and number of parameters. The four YOGA points correspond to YOGA-*n*/*s*/*m*/*l*. Similarly, the points of other models correspond to their respective model sizes too. The results show that YOGA has the best AP at every model scale, or equivalently the lightest model for any target AP. For instance, PPYOLO [21] has an AP of 22.7 at 4.20M parameters, while YOGA achieves an AP of ~35 (interpolated) with the same number of parameters, amounting to a 54% improvement. More thorough and detailed comparisons are presented in Tables 3 and 4 and discussed below.

4.2.1. Nano and Small Models. With the same number (1.9M) of parameters, YOGA-*n* achieves an AP of 32.3 which is 15.35% higher than the best-performing model, YOLO-*n*, whose AP is 28.0. In the comparison on small objects, YOGA-*n* achieves an improvement of 7.4% AP_S over YOLO-*n*.

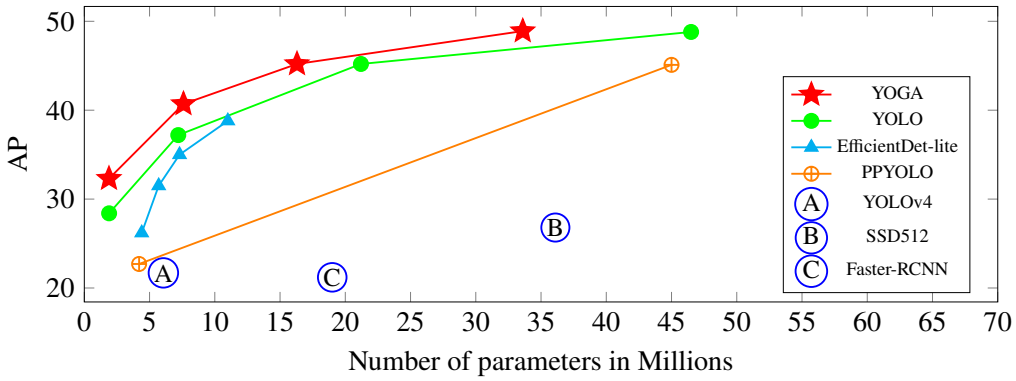


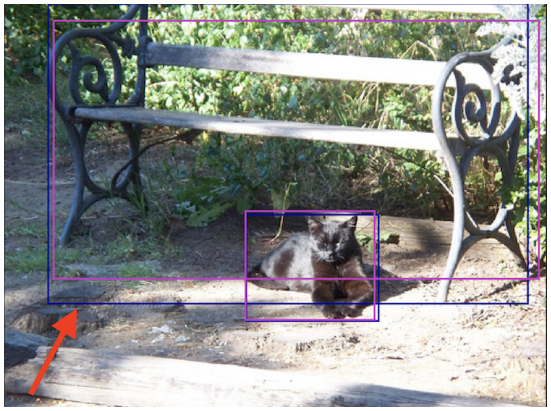
Figure 7. Comparing YOGA with state-of-the-art object detection models.

Similarly, our YOGA-*s* achieves 40.7 AP while the best-performing YOLO-*s* achieves 37.4, with almost the same number of parameters and FLOPs, which indicates a 8.8% increase in the AP value. Our YOGA-*s* achieves 23.0 AP on small objects (AP_S) while the best-performing YOLO-*s* achieves 21.09 AP_S, a +9.0% increase in the AP_S. Compared to state-of-the-art models on test-dev (Table 4), our YOGA-*n* model compared to YOLO-*n* achieves an improvement of 15% AP (+4.2 AP). When we compare on object scales, on small objects, there is an improvement of 12% AP_S (+1.5 AP_S); on medium objects, there is an improvement of 11% AP_M (+3.4 AP_M), and on large objects, there is an improvement of 22% AP_L (+7.7 AP_L).

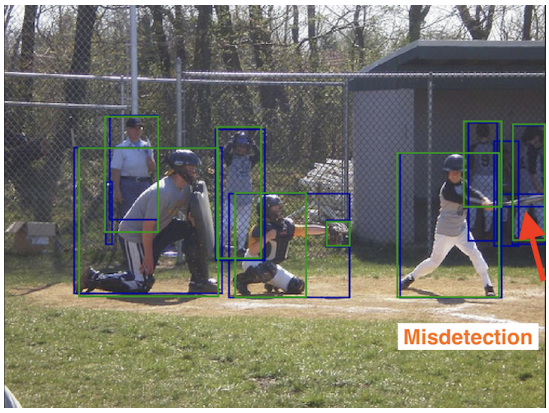
Similarly, our YOGA-*s* model compared to YOLO-*s*, achieves an improvement of 8.6% AP (+3.2 AP). When we compare on object scales, on medium objects, there is an improvement of 5% AP_M (+2.0 AP_M); on large objects, there is an improvement of 17% AP_L (+7.9 AP_L).

4.2.2. Medium and Large Models. When compared to YOLO-*m*, our YOGA-*m* achieves similar AP value of 45.2 but significantly reduces parameters and FLOPs by 23% and 29% respectively. simlarly, comparing with YOLOX-M model, our YOGA-*m* has significantly reduction in parameters and FLOPs by 35% and 53%.

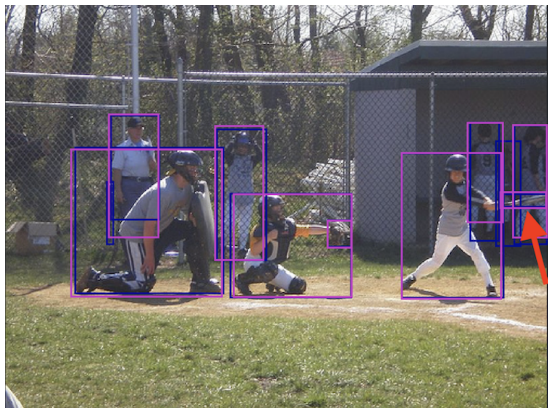
(a) Green boxes: YOLO-*n* predictions.



(b) Purple boxes: YOGA-*n* predictions.



(c) Green boxes: YOLO-*n* predictions.



(d) Purple boxes: YOGA-*n* predictions.

Figure 8. A visual comparison. Blue boxes: the COCO-17 ground truth. Red arrows highlight the differences.

Compared to YOLO-*l*, our YOGA-*l* model achieves the same AP (48.9) but with a significantly lower number of parameters and FLOPs: 27.7% and 34% respectively. similarly, comparing with YOLOX-L, our YOGA-*l* model has significant reduction in parameters and FLOPs by 38% and 53.8%.

When compared to state-of-the-art models on test-dev (Table 4), YOGA-*m* achieves AP of 46.4, a 2% improvement to YOLO-*m* and it uses only 16.3 M parameters and and 34.6 BFLOPs, which are significantly (23% and 29%) lower than YOLO-*m*.

4.2.3. Visual Comparison. We compare the bounding box predictions of the YOLO-*n* and our YOGA-*n* model on two random sample images on COCO validation dataset. In Figure 8, we see that the top row image has two ground truth objects (Blue boxes) and our YOGA-*n* detects both objects (Purple boxes), while YOLO-*n* (Green boxes) fail to detect one object (Bench). Similarly, the bottom row image has a total of eleven ground truth objects, and YOLO-*n* detects only eight objects while our YOGA-*n* model detects nine objects, an extra object called Baseball bat.

4.3. HARDWARE IMPLEMENTATION AND EVALUATION

To assess the edge suitability of YOGA and its usability in the wild, we migrate YOGA code to NVIDIA Jetson Nano 2GB, which comes with 2 GB 64-bit LPDDR4 25.6 GB/s RAM and 32 GB MicroSD storage and is the *lowest-end* deep learning hardware product from NVIDIA.

Figure 9 shows the hardware setup for our edge inference experiments, where we have set up the runtime environment (Ubuntu 20.04, PyTorch 1.12, Jetpack 4.6) for evaluation. We measure the inference time of YOGA to see how near-real-time it can be when performing object detection. The results are reported in Table 5, where we see that YOGA-n achieves an inference time of of 0.57 sec per image (each image is of large size 640x640) which is close to real-time. We highlight an important fact that, as seen from Figure 9 (in oval shapes), the 2GB memory on Jetson Nano was fully utilized at peak time and 1.828 GB swap space on the disk had to be used to compensate for the memory shortage. This means that the disk I/O had throttled the performance substantially, and it is therefore reasonable to anticipate a significantly better performance on the 4 GB Jetson Nano and even better on Jetson TX2, which would no longer or rarely need to use swap space, making the inference indeed (near)real-time.[1]

[1]Both before and at the time of writing, the global market has been undergoing a severe GPU product shortage and many products have been out of stock in the market. As a consequence, we could not procure more hardware for testing.

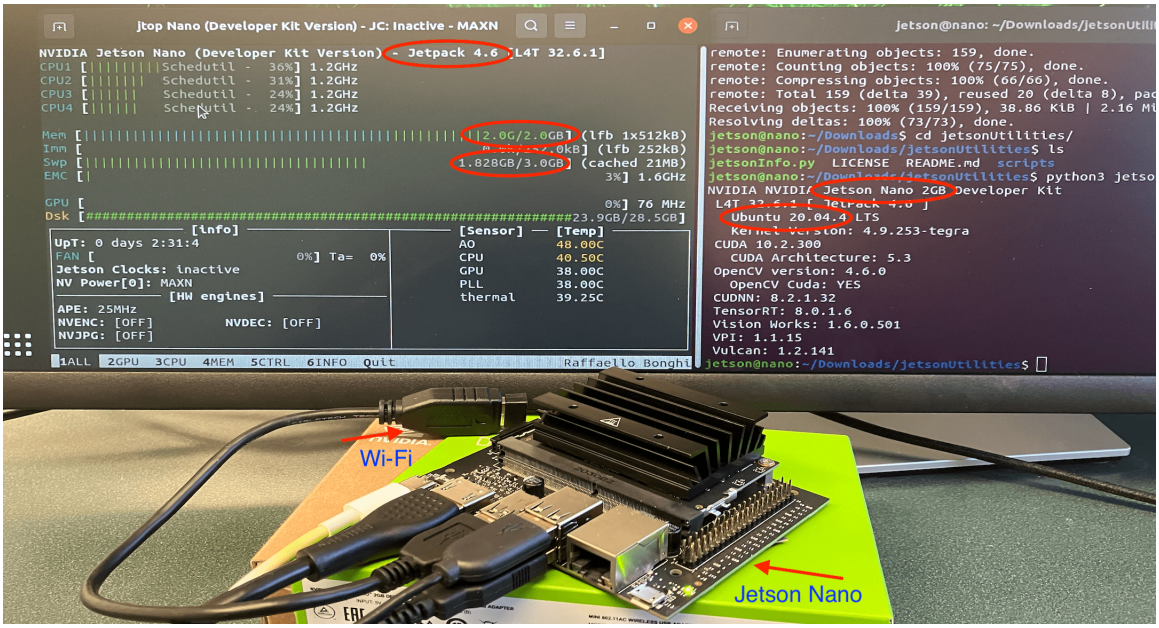


Figure 9. Our hardware testbed setup and run-time outputs.

Table 5. Performance on Jetson Nano 2GB with 640 x 640 (large) COCO images.

Model	Inference time (sec)
YOGA-*n*	0.57
YOGA-*s*	0.77
YOGA-*m*	0.98
YOGA-*l*	1.30

4.4. ABLATION STUDY

We also design experiments to investigate the individual effect of our new backbone and neck: specifically, how our CSPGhostNet backbone compares to the YOLO backbone (arguably the best backbone so far) and how our AFF-PANet neck compares to the naive concatenation as used in all SOTA architectures. Moreover, we also evaluate the effect of label smoothing on the gradient descent convergence. We conduct these ablation studies using YOGA-*n*.

Table 6. Ablation study on Backbone and Neck (YOGA-*n*).

Backbone	Neck	AP
YOLO Backbone	Naive Concat	28.4
YOLO Backbone	AFF-PANet	29.2
CSPGhostNet	Naive Concat	31.1
CSPGhostNet	AFF-PANet	**32.3**

The results for backbone and neck are given in Table 6. We observe that, using the AFF-PANet neck architecture consistently leads to improved performance compared to using the Naive Concat (PANet) neck architecture. Additionally, using our CSPGhostNet backbone leads to better performance than using the existing YOLO backbone in both cases. Overall, these results suggest that both the AFF-PANet neck and the CSPGhostNet backbone contribute positively to the performance of YOGA.

For label smoothing, we observed during our training that it helped our model training to converge to a desirable AP[0.5:0.95] and recall in ~10% less number of epochs than without label smoothing.

5. CONCLUSION

This paper presents YOGA, a novel object detection model with an efficient convolutional backbone and an enhanced attention-based neck. It is a deep yet lightweight object detector with high accuracy, which we have validated with extensive evaluation benchmarked against more than 10 state-of-the-art modern deep detectors. For instance, in its Nano version, our YOGA-*n* outperforms the current best-performing model YOLOv5n by 15.35% in AP, with similar number of parameters and FLOPs; this improvement further increases to 22% on detecting large objects (AP_L) on the test-dev dataset. In its Medium version, our YOGA-*m* achieves the same AP (45.2) as the best-performing model YOLOv5m

but with 23% fewer parameters and 29% fewer FLOPs. In its Large version, our YOGA-*l* achieves the same AP (48.9) as the best-performing model YOLOv5-l but with 27.7% fewer parameters and 34% fewer FLOPs.

We have also implemented and assessed YOGA on the *lowest-end* deep learning device from NVIDIA, Jetson Nano 2GB, and the results affirmed that YOGA is suitable for edge deployment in the wild. For instance, YOGA-*n* runs at 0.57 sec per 640x640 image, which is close to real-time.

The main limitation of YOGA is that it could be prone to overfitting when training extremely large models. Nonetheless, such extra-large models are rather unlikely to be adopted in edge deployments. Future directions for improving YOGA or object detection in general, include: (1) investigating different attention mechanisms such as incorporating self-attention or transformer architectures; (2) exploring ways to further optimize the model for specific hardware platforms such as mobile devices; (3) extending YOGA to handle additional tasks or challenges such as semantic segmentation, instance segmentation, and object tracking.

In summary, YOGA represents a new contribution to the field of object detection by ushering in high run-time efficiency, low memory footprint, and superior accuracy simultaneously. In addition, its flexible scalability makes it applicable to a wide range of applications with different hardware constraints in IoT, edge and cloud computing.

REFERENCES

[1] Abu-Mostafa, Y.S., 1993. Hints and the vc dimension. Neural Computation 5, 278–288.

[2] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S., 2020. End-to-end object detection with transformers, in: European Conference on Computer Vision, Springer. pp. 213–229.

[3] CodaLab, 2019. CodaLab COCO detection challenge (bounding box). `https://competitions.codalab.org/competitions/20794`.

[4] Dai, Y., Gieseke, F., Oehmcke, S., Wu, Y., Barnard, K., 2021. Attentional feature fusion, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 3560–3569.

[5] Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., Tian, Q., 2019. Centernet: Keypoint triplets for object detection, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 6569–6578.

[6] Fang, Y., Zhang, H., Yan, J., Jiang, W., Liu, Y., 2023. Udnet: Uncertainty-aware deep network for salient object detection. Pattern Recognition 134, 109099.

[7] Ge, Z., Liu, S., Wang, F., Li, Z., Sun, J., 2021. Yolox: Exceeding yolo series in 2021. arXiv preprint arXiv:2107.08430 .

[8] Girshick, R., 2015. Fast R-CNN, in: Proceedings of the IEEE international conference on computer vision, pp. 1440–1448.

[9] Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580–587.

[10] Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C., 2020. Ghostnet: More features from cheap operations, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1580–1589.

[11] He, K., Zhang, X., Ren, S., Sun, J., 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence 37, 1904–1916.

[12] He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

[13] Hu, J., Shen, L., Sun, G., 2018. Squeeze-and-excitation networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7132–7141.

[14] Jocher, G., et al., 2021. https://github.com/ultralytics/yolov5. Released version available at the time of evaluation: Feb 22, 2022.

[15] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017a. Feature pyramid networks for object detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117–2125.

[16] Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P., 2017b. Focal loss for dense object detection, in: ICCV, pp. 2980–2988.

[17] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft coco: Common objects in context, in: European conference on computer vision, Springer. pp. 740–755.

[18] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., Pietikäinen, M., 2020. Deep learning for generic object detection: A survey. International Journal of Computer Vision 128, 261–318.

[19] Liu, S., Qi, L., Qin, H., Shi, J., Jia, J., 2018. Path aggregation network for instance segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8759–8768.

[20] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C., 2016. SSD: Single shot multibox detector, in: European conference on computer vision, Springer. pp. 21–37.

[21] Long, X., et al., 2020. Pp-yolo: An effective and efficient implementation of object detector. arXiv preprint arXiv:2007.12099 .

[22] Redmon, J., Farhadi, A., 2018. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 .

[23] Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. NeurIPS 28, 91–99.

[24] Samet, N., Hicsonmez, S., Akbas, E., 2020. Houghnet: Integrating near and long-range evidence for bottom-up object detection, in: European Conference on Computer Vision, Springer. pp. 406–423.

[25] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826.

[26] Tan, M., Pang, R., Le, Q.V., 2020. Efficientdet: Scalable and efficient object detection, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 10781–10790.

[27] Tian, Z., Shen, C., Chen, H., He, T., 2019. Fcos: Fully convolutional one-stage object detection, in: Proceedings of the IEEE/CVF international conference on computer vision, pp. 9627–9636.

[28] Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M., 2021. Scaled-yolov4: Scaling cross stage partial network, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 13029–13038.

[29] Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M., 2022. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv preprint arXiv:2207.02696 .

[30] Wang, C.Y., Liao, H.Y.M., Wu, Y.H., Chen, P.Y., Hsieh, J.W., Yeh, I.H., 2020a. 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, pp. 390–391.

[31] Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al., 2020b. Deep high-resolution representation learning for visual recognition. IEEE transactions on pattern analysis and machine intelligence 43, 3349–3364.

[32] Wang, K., Liu, M., 2022. Yolo-anti: Yolo-based counterattack model for unseen congested object detection. Pattern Recognition , 108814.

[33] Wang, Q., Zhang, L., Li, Y., Kpalma, K., 2020c. Overview of deep-learning based methods for salient object detection in videos. Pattern Recognition 104, 107340.

[34] Wang, S., Gong, Y., Xing, J., Huang, L., Huang, C., Hu, W., 2020d. Rdsnet: A new deep architecture forreciprocal object detection and instance segmentation, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 12208–12215.

[35] Yang, Z., Liu, S., Hu, H., Wang, L., Lin, S., 2019. Reppoints: Point set representation for object detection, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 9657–9666.

[36] Zhang, P., Dai, X., Yang, J., Xiao, B., Yuan, L., Zhang, L., Gao, J., 2021. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 2998–3008.

SECTION

2. SUMMARY AND FUTURE WORK

This thesis investigates the effectiveness of deep learning-based computer vision models in challenging conditions, including small objects, low-resolution images, and edge deployment. To address these challenges, two solutions are proposed: SPD-Conv, which eliminates strided convolution and pooling layers to improve detection of small objects and classifying low-resolution images through preserving fine-grained information; and YOGA, a lightweight object detection model that achieves high accuracy on low-end edge devices by using two-phase feature learning pipeline with attention-based multi-scale feature fusion. The proposed solutions are evaluated on COCO-val and COCO-testdev datasets and compared with state-of-the-art models, demonstrating their effectiveness in overcoming these challenges.

this thesis emphasizes the importance of reproducibility by conducting experiments using open-source tools and frameworks, and making code and models available to the research community. Future directions for improving SPD-Conv include to see how it performs when the stride is grearthan than two and reducing GPU memory consumptions across the SPD-Conv block. and for YOGA include investigating different attention mechanisms, optimizing the models for specific hardware platforms, and extending them to handle additional tasks or challenges such as semantic segmentation, instance segmentation, and object tracking. Overall, this thesis contributes to advancing the field of computer vision and provides a foundation for future research in improving deep learning-based models in challenging scenarios.

In addition to proposing solutions, this thesis emphasizes the importance of reproducibility through the use of open-source tools and frameworks, and making code and models available to the research community. To further improve the proposed models, future directions for SPD-Conv include investigating its performance with strides greater than two and reducing GPU memory consumption. For YOGA, future work includes exploring different attention mechanisms, optimizing the models for specific hardware platforms, and extending them to handle additional tasks such as semantic segmentation, instance segmentation, and object tracking. Overall, this thesis contributes to advancing the field of computer vision and provides a foundation for future research in improving deep learning-based models in challenging scenarios.

VITA

Raja Sunkara is a skilled machine learning and deep learning researcher/engineer. He obtained his Master of Science in Computer Science from Missouri University of Science & Technology in May 2023, and his Bachelor's degree in Aerospace Engineering with a minor in Systems Engineering from the Indian Institute of Technology Madras in 2017.

Raja has a strong research background and has published several papers in machine learning conferences. He has also worked as a research assistant at the Securing Artificial Intelligence and IoT Lab and the Statistical Machine Learning Lab, both at Missouri S&T. Additionally, Raja has industry experience as a research engineer (AI/CV) at Matdun Labs and as a data scientist at Agrometrics. Moreover, he worked as an R&D Data Scientist Intern at Alcon, where he developed an image registration algorithm using CNN architecture and semi-supervised learning.