Depthwise Separable Convolutions and Variational Dropout within the context of YOLOv3

Joseph Chakar, Rayan Al Sobbahi, and Joe Tekli*

Lebanese American University, 36 Byblos, Mount Lebanon, Lebanon
{joseph.elchakar, rayan.alsobbahi}@lau.edu, joe.tekli@lau.edu.lb

Abstract. Deep learning algorithms have demonstrated remarkable performance in many sectors and have become one of the main foundations of modern computer-vision solutions. However, these algorithms often impose prohibitive levels of memory and computational overhead, especially in resource-constrained environments. In this study, we combine the state-of-the-art object-detection model YOLOv3 with depthwise separable convolutions and variational dropout in an attempt to bridge the gap between the superior accuracy of convolutional neural networks and the limited access to computational resources. We propose three lightweight variants of YOLOv3 by replacing the original network's standard convolutions with depthwise separable convolutions at different strategic locations within the network, and we evaluate their impacts on YOLOv3's size, speed, and accuracy. We also explore variational dropout: a technique that finds individual and unbounded dropout rates for each neural network weight. Experiments on the PASCAL VOC benchmark dataset show promising results where variational dropout combined with the most efficient YOLOv3 variant lead to an extremely sparse solution that reduces 95% of the baseline network's parameters at a relatively small drop of 3% in accuracy.

Keywords: Computer Vision, Object Detection, Convolutional Neural Network, Depthwise Separable Convolution, Network Sparsification, Variational Dropout.

1 Introduction

Convolutional Neural Networks (CNNs) have witnessed tremendous growth following the release of AlexNet [1] at the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) competition [2]. Due to their accuracy and generalizability compared with traditional techniques, CNNs have become the dominant approach for a variety of real-life applications, particularly in the field of computer vision.

Perhaps one of the most fundamental problems in this area is the task of object detection, which is characterized by two main categories of deep learning based solutions: i) two-stage and ii) single-stage detectors. In two-stage detectors like R-CNN (Regions with CNN features) [3], Fast R-CNN [4], and Faster R-CNN [5], region proposal networks generate regions of interest that are sent down a detection pipeline. In contrast, the single-stage framework treats object detection as a regression problem by learning the bounding box coordinates and class probabilities in one forward pass over a dense sampling of possible locations. State-of-the-art one-stage detectors include

* Corresponding author

SSD (Singe Shot MultiBox Detector) [6] and YOLO (You Only Look Once) [7-9]. While two-stage detectors reach higher accuracy levels, single-stage ones usually achieve higher inference speeds. In this context, the trade-off between accuracy and speed continues to be a major challenge for modern convolutional object detectors.

Among the one-stage models, YOLOv3 [9] is one of the most recent and popular when it comes to balancing these two key performance criteria for practical applications. Despite its efficient architecture, YOLOv3 still has millions of parameters that come with a heavy computational cost and a large run-time memory footprint. The computational resources required to train such a large neural network on large benchmark datasets like PASCAL VOC (Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes) [10] and MS COCO (Microsoft Common Objects in Context) [11] can often be prohibitive, and this issue of high computation overhead and power consumption often hinders its deployment on resource-constrained devices.

In this study, we reduce YOLOv3's size and induce a high state of sparsity within its network in order to produce an efficient object-detection model fit for resource-limited environments. To do so, we propose three lightweight variants of YOLOv3 by replacing its standard convolutions with depthwise separable convolutions at different strategic locations, and we evaluate their impacts on the original network's size, speed, and accuracy using the PASCAL VOC benchmark dataset. We then apply variational dropout [12] to the most efficient YOLOv3 variant, which leads to an extremely sparse solution that effectively compresses the baseline model by a factor of 20, thus reducing 95% of the latter's parameters at a relatively small decrease of 3% in its accuracy, depending on the batch size.

The remainder of the paper is organized as follows. Section 2 provides preliminary notions regarding depthwise separable convolutions and variational dropout. Section 3 briefly reviews the background and related works. Section 4 presents our proposal. Section 5 describes our experimental evaluation and results, before concluding in Section 6 with future directions.

2 Preliminaries

2.1 YOLOv3 Model

The concept of the YOLO object-detection algorithm [7-9] is built on a unique set of characteristics that stands out from traditional systems in order to reduce computational complexity and achieve real-time inference speeds while maintaining high accuracy. It reasons globally about the full image by handling the task of object detection as an integrated regression problem to predict bounding boxes and their associated class probabilities in one single evaluation. First, the input image is divided into an $S \times S$ grid, where each grid cell is responsible for detecting the object that falls into it. Furthermore, the classification network architecture relies on 1×1 reduction layers followed by 3×3 convolutional layers. Lastly, a multi-part loss function that combines: i) a confidence loss, ii) a bounding box loss whenever the prediction box contains objects, and iii) a classification loss, is used to optimize the neural network's

parameters. The original version of YOLO [7] has many shortcomings, e.g.: i) it imposes strong spatial constraints on bounding box predictions, ii) it struggles with detecting and localizing small objects, and iii) it is not able to properly generalize to objects with new or unusual aspect ratios. The second version, YOLOv2 [8], comes with several improvements and has established itself on standard detection tasks like PASCAL VOC. It introduces predefined anchor boxes that assist the prediction boxes, as well as a multi-scale training method, which offers a tradeoff between speed and accuracy by allowing the model to run at varying input image sizes. In the third and most powerful installment in this series, YOLOv3 [9], the backbone network has been upgraded to the state-of-the-art Darknet-53 feature extractor, on top of which several convolutional layers are stacked for the task of detection. YOLOv3 is capable of accurately detecting large, medium, and small objects by making predictions at three consecutive scales located at three different stages within the network. At each scale, a 3D tensor encodes: i) the four bounding box offsets, ii) the level of confidence of having an object, and iii) the corresponding class predictions.

Despite being cited as one of the fastest deep learning based object detectors, YOLOv3 has a large runtime memory footprint. In an attempt to resolve this issue, the YOLO series comes with a lightweight version called tiny YOLO, which decreases the number of floating point operations per second (FLOPS) by over 85%. However, this reduction in model size and inference time comes with a sharp drop of around 20% in the MS COCO detection accuracy. In this regard, striking a good balance between accuracy and speed remains a major computer-vision challenge to date. Various approaches to produce efficient machine learning models have been proposed in the literature e.g., [13-16], but for the sake of this study, we mainly focus on deep learning based object-detection solutions that use i) depthwise separable convolutions or ii) network sparsification as their underlying foundations.

2.2 Depthwise Separable Convolutions

A depthwise separable convolution is a form of factorized convolution. It separates the latter's spatial and channel components into two layers: i) a depthwise convolutional layer, which applies a single filter to each input channel, and ii) a pointwise or 1×1 convolutional layer, which multiplies the depthwise layer's outputs to generate the same output of the original convolution.

Compared with the standard convolution, which filters the input channels and combines them into a new set of outputs in a single step, the depthwise separable convolution substantially reduces a convolutional neural network's size and computational cost. In fact, a standard convolutional layer takes a $D_F \times D_F \times M$ feature map as input and outputs a $D_G \times D_G \times N$ feature map, where D_F and D_G are the respective spatial width and height of the square input and output feature maps, and M and N respectively designate the number of input channels and number of output channels. This standard network unit is thus parameterized by a convolutional kernel K of size $D_K \times D_K \times M \times N$. Similarly, D_K here represents the spatial dimension of the kernel, which is taken to be square. For the average convolution with a stride of one and padding where the input and output feature maps have the same spatial dimensions, the

computational cost depends multiplicatively on the input depth N, the output depth M, the kernel size $D_K \times D_K$, and the feature map size $D_F \times D_F$, and comes down to:

$$D_K^2 \times M \times N \times D_F^2 \quad \textbf{(1)}$$

Using the alternative separable representation, the standard convolution is first broken into a depthwise convolution that filters the input channels and comes at the computational cost of $D_K^2 \times M \times D_F^2$. An additional 1×1 pointwise layer is then needed to generate new features across the N output channels, at the computational cost of $M \times N \times D_F^2$. Consequently, the total computational cost of the depthwise separable convolution is equal to the sum of the two previous terms:

$$D_K^2 \times M \times D_F^2 + M \times N \times D_F^2 \quad \textbf{(2)}$$

As can be seen, this approach is significantly more efficient than the traditional one, and this translates into a drastic reduction in computation of:

$$\frac{D_K^2 \times M \times D_F^2 + M \times N \times D_F^2}{D_K^2 \times M \times N \times D_F^2} = \frac{1}{N} + \frac{1}{D_K^2} \quad \textbf{(3)}$$

Similar to MobileNets [17], YOLOv3 relies heavily on 3×3 convolutional filters. This conversion thus lowers the computation of each convolution by up to 9 times. Even though these calculations do not take the effect of having strides and valid padding into consideration, these results safely generalize to input and output feature maps of different sizes. Nonetheless, this drop in the number of parameters is associated with a minor drop in performance. In Section 4, we describe three experimental setups that produce different reductions in size and accuracy based on which YOLOv3 standard convolutions are changed to depthwise separable convolutions.

2.3 Network Sparsification using Variational Dropout

Sparsification is another leading approach to address the speed-versus-accuracy challenge of object-detection models. Sparsity is achieved when a proportion of a model is comprised of zero values. With most of the elements set to zero, sparse matrix formats can be used to store and perform efficient mathematical operations on the resulting weight matrices. Dropout [18] is a popular and empirically effective way of sparsifying a neural network and controlling over-fitting by randomly dropping out or ignoring a certain pre-defined percentage of neural network units during training. Variational dropout (VD) [19] is a more recent neural network regularization technique originally proposed as a Bayesian reinterpretation of Gaussian dropout [20], which is a more efficient approximation of the standard (binary) dropout. Simply put, variational dropout is a generalization of Gaussian dropout with learnable dropout rates. It has been later extended in [12] to include more specific dropout rates, where individual weight parameters with high dropout values can be removed post-training to get highly sparse models with a virtually identical performance.

For a training set D of N samples $(x_i, y_i)_{i=1}^N$ and a classification problem where the goal is to learn the weight parameters w of the conditional probability $p(y|x, w)$, Bayesian inference is used to update an initial belief over w in the form of a prior

distribution $p(w)$ with observed data D into a belief in the form of a posterior distribution $p(w|D)$:

$$p(w|D) = p(D|w)/\mathrm{p}(D) \quad \textbf{(4)}$$

Since computing the true posterior distribution $p(w|D)$ is computationally intractable, an approximation is used instead [12]. In variational inference, the parameters ϕ of some model $q_\phi(w)$ are optimized such that the approximated parameterized model is as close as possible to the true posterior distribution, as evaluated by the Kullback-Leibler (KL) divergence between the two distributions. In practice, this divergence is minimized by maximizing the variational lower-bound equation, which is the difference between the expected log-likelihood $L_D(\phi)$ and the KL-divergence regularization of $q_\phi(w)$ with respect to $p(w)$, as shown below:

$$L(\phi) = L_D(\phi) - D_{KL}(q_\phi(w)||p(w)) \quad \textbf{(5)}$$

where:

$$L_D(\phi) = \sum_{(x,y)\in \mathrm{D}} \boldsymbol{E}_{q_\phi}[log(p(y|x,w))] \quad \textbf{(6)}$$

Using the Stochastic Gradient Variational Bayes (SGVB) algorithm [19], the log-likelihood is reduced to the standard cross-entropy loss, which is typically used to minimize the divergence of the predicted label from the true one, while the KL divergence term serves as a regularization term. Note that in the standard formulation of VD, the weights of neural network are assumed to be drawn from a fully-factorized Gaussian approximate posterior:

$$w_{ij} \sim q_\phi(w_{ij}) = \mathcal{N}(\theta_{ij}, \sigma_{ij}^2) \quad \textbf{(7)}$$

where θ_{ij} and $\sigma_{ij}^2 = \alpha_{ij}\theta_{ij}^2$ are the mean and variance of this Gaussian distribution, with α_{ij} being a parameter that defines the dropout rate p_{ij} of the weight w_{ij} as follows:

$$p_{ij} = \frac{\alpha_{ij}}{1+\alpha_{ij}} \quad \textbf{(8)}$$

If $\alpha_{ij} = 0$, then w_{ij} is fully preserved with no dropout rate. In contrast, when $\alpha_{ij} \to +\infty$, $p_{ij} \to 1$, w_{ij} can be completely removed to sparsify the model. For each training step, the weights are sampled from the normal distribution N, and the so-called re-parameterization trick [21, 22] is used to differentiate the loss with respect to the parameters through this sampling operation:

$$w_{ij} = \theta_{ij}\left(1+\sqrt{\alpha_{ij}}\epsilon_{ij}\right) \sim \mathcal{N}(w_{ij}|\theta_{ij}, \alpha_{ij}\theta_{ij}^2) \quad \textbf{(9)}$$

where $\epsilon_{ij} \sim \mathcal{N}(0,1)$. Via this parameterization, the mean and variance of the neural network parameters can be directly optimized. For a log-uniform prior on the weights p(w), the KL divergence component of the $D_{KL}(q_\phi(\omega_{ij})||p(\omega_{ij}))$ objective function can be accurately approximated using the following equation [12]:

$$D_{KL} \approx \frac{k_1}{1 + e^{-(k_2 + k_3 log \alpha_{ij})}} - 0.5 \log\left(1 + \frac{1}{\alpha_{ij}}\right) + C \quad \textbf{(10)}$$

where k_1=0.63576, k_2 = 1.87320, k_3 = 1.48695, and C = –k_1.

The authors in [12] highlight some difficulties in training certain models with a learnable sparse architecture from a random initialization, as large portions of the model tend to adopt high dropout rates before a useful representation is learned from the data. To address this issue, they propose to start from a pre-trained network or use warm-up, i.e., re-scale the KL divergence term during the training by adding a regularizer coefficient, and then gradually increase it from 0 to 1. We adopt a similar approach in our study, but we instead apply variational dropout to an efficient and lightweight YOLOv3 variant based on depthwise separable convolutions.

3 Related Works

3.1 Depthwise Separable Convolution Approaches

Recently, several deep learning models have been built on depthwise separable convolutions. MobileNets [17] are a family of fast and small-sized deep neural networks which are based on depthwise separable convolutions and include two global hyperparameters to tune their latency and accuracy. The first parameter is a width multiplier, which can scale down the input and output channels of a given layer to thin the latter uniformly. The second is a resolution multiplier, which can be applied to the input image to reduce the internal representation of every layer. After varying these two hyper-parameters, different trade-offs for reducing the network size and accuracy are achieved, and the authors compare their results with those of popular models in various applications. Instead of building new models from scratch, many researchers have focused on redesigning YOLO's architecture in order to create lighter versions that increase inference speed while maintaining high detection accuracy. One example of a small-sized YOLOv3 variant that relies on depthwise separable convolutions is Mini-YOLOv3 [23], which consists of a new backbone network with a parameter size of only 16% that of Darknet-53. In the residual layers of Mini-YOLOv3, the authors use a 1×1 convolution to increase the input dimension and then decrease it, leaving the 3×3 layer in an inverted bottleneck with larger input/output dimensions. To compensate for the large calculations associated with these operations, the authors group the convolutions and add a channel shuffle to enable cross-group information flow. Furthermore, they introduce a Multi-Scale Feature Pyramid Network (MSFPN) based on a U-shaped structure to improve the performance of the multi-scale object detection task. In this MSFPN, a Concat model first fuses the backbone's three feature maps to generate the base feature. An Encoder-Decoder then generates a group of multi-scale features, and a Feature Fusion model finally aggregates the three feature maps and group of multi-scale features into a feature pyramid. The Mini-YOLOv3 model achieves accuracy levels comparable to those of YOLOv3 on the MS COCO dataset, but at double the inference speed. Another example of a lightweight model is YOLOv3-Lite [24]. The feature extraction backbone of this network is 13-layers deep and is built entirely on

depthwise separable convolutions. Similarly to YOLOv3, each convolution layer is followed by batch normalization and ReLU non-linearity layers. The authors also adopt the idea of a feature pyramid network that combines low- and high-resolution information at three different scales to detect large, medium, and small scale objects. Their lightweight detection network uses the YOLOv3 bounding box regression strategy, and it reaches a detection accuracy comparable to that of YOLOv3 on a custom dataset for cracks in aircraft structures.

3.2 Network Sparsification Approaches

Some of the recent solutions that have been proposed to sparsify the YOLOv3 neural network are presented in [25, 28]. In the first study [25], the authors impose channel-level sparsity on YOLOv3's convolutional layers by applying L1 regularization to the γ regularizer of the batch normalization layers. L1 regularization is a technique used to penalize a neural network's loss function in proportion to the sum of the absolute values of the weights. It helps drive the weights of irrelevant features to zero, thus sparsifying the model. The authors also integrate a spatial pyramid pooling (SPP) module, which consists of multiple parallel maxpool layers with different kernel sizes, in order to extract additional multi-scale features and further improve the detection accuracy. They then apply L1 regularization to their YOLOv3-SPP3 model and use a penalty factor to optimize the resulting L1 loss term. After the sparsity training, the authors introduce a global threshold to control the pruning ratio and carefully prune each feature channel to maintain the integrity of the network connections. Finally, they follow a fine-tuning operation and incremental pruning strategy to compensate for any performance degradation and prevent over-pruning. Their proposed SlimYOLOv3 model is evaluated on the VisDrone2018-Det benchmark dataset [26] for Unmanned Aerial Vehicles (UAV) applications [27], and experimental results using different pruning ratios show a decrease in parameter size of down to 92% with a detection accuracy comparable to that of YOLOv3. In [28], the authors use variational dropout to sparsify YOLOv3 on a self-collected dataset about road traffic in Vietnam. Both YOLOv3 and YOLOv3-VD are initialized from the pre-trained weights obtained on the COCO dataset, but during the training of YOLOv3-VD, a scaling factor is used to balance the variational dropout loss term with the network loss function. The authors successfully eliminate up to 91% of the original network weight parameters with only a 3% drop in accuracy, and their experimental results show that the sparsity level gradually increases such that the final layers can be completely pruned.

4 Proposal: Separable YOLOv3 with Variational Dropout

After careful examination of the YOLOv3 architecture, we set out to study the size-accuracy trade-off associated with substituting standard convolutions with depthwise separable ones and adding variational dropout. Our approach's overall process is depicted in Fig. 1, and consists of three main components: i) integrating depthwise separable convolutions in YOLOv3 to produce three lightweight variants, ii) training these three separable models on the PASCAL VOC dataset, and then iii) applying

variational dropout to the best performing model. We rely on a TensorFlow implementation of YOLOv3[1], which is an overall faithful reproduction of the original model [9] but with several tweaks such as: i) replacing the original loss with a GIoU (General Intersection of Union) loss, ii) using cosine scheduling for the learning rate, and iii) implementing different data augmentation techniques. Even though they drastically affect the network's performance, these modifications have no impact on this study, since all the experiments are done under the same settings.

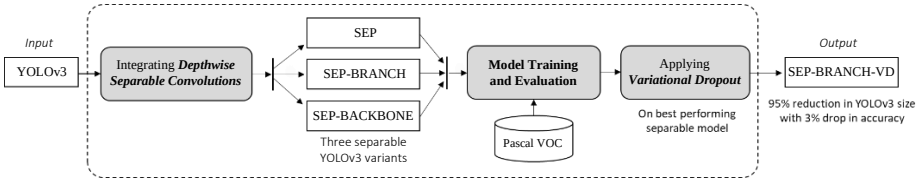


Fig. 1. Simplified activity diagram describing our approach

4.1 Separable YOLOv3 Configurations

We produce three separable models by replacing YOLOv3's standard convolutions with depthwise separable ones at different locations within the network:

- **SEP:** All standard convolutions are replaced with depthwise separable convolutions, except those with a 1×1 filter, since reducing and then restoring the dimensions of the multiple 1×1 convolutions spread throughout the network would greatly weaken the feature learning and expression ability of the model. Note that this exception is extended to the two remaining models.
- **SEP-BRANCH:** All standard convolutions are replaced with depthwise separable convolutions, except for the detection layers, which are marked by the last two convolutions of each of YOLOv3's three detection scales.
- **SEP-BACKBONE:** Only the standard convolutions of the Darknet-53 backbone are replaced with depthwise separable convolutions.

Note that each convolution in YOLOv3 is followed by a batch normalization layer and a leaky ReLU activation function. Similarly, batch normalization and the leaky ReLU nonlinearity are applied after each of the depthwise and pointwise 1×1 convolutions.

4.2 Separable YOLOv3 Model

We conduct the variational tests on the best performing model, which turns out to be SEP-BRANCH based on our empirical results reported in Section 5. We follow Gale et al.'s [29] TensorFlow implementation of variational dropout, which is publicly available online[2].

Given that the depthwise separable factorization splits the standard convolution into a depthwise convolution and a pointwise convolution, with $w_{d_{ij}}$and $w_{p_{ij}}$ as their respective weights, we apply a variational distribution with learnable parameters θ, σ^2,

[1] https://github.com/YunYang1994/tensorflow-yolov3

[2] https://github.com/google-research/googleresearch/tree/master/state_of_sparsity/layers/variational_dropout

and α on each weight. We then appropriately sum the two resulting KL-divergence terms and add them to the global network loss, along with the divergence term of the standard convolutions. The multi-part loss function becomes as follows:

$$L_{total} = L_{GIoU} + L_{Conf} + L_{Prob} + \lambda D_{KL} \quad \textbf{(11)}$$

Knowing that training the model with VD from the start is not recommended [12], we use a regularizer coefficient λ to balance YOLOv3's loss and the KL-divergence term. Similarly to the study in [27], we gradually ramp the regularizer coefficient to induce sparsity. We train the model without any VD loss ($\lambda = 0$) from the obtained VOC weights until we reach convergence after 35 epochs. We then set the divergence coefficient to $\lambda = 10^{-6}$ for 10 epochs, and raise it to $\lambda = 10^{-5}$ for an additional 10 epochs. Afterwards, we notice that additional training starts to increase the network's sparsity level at the expense of its accuracy. As a result, we lower the learning rate to 10^{-6} to fine-tune the model sparsity over 10 epochs.

5 Experimental Results

5.1 Separable YOLOv3 Models

We train YOLOv3 and its three depthwise separable variants (SEP, SEP-BRANCH, and SEP-BACKBONE) from scratch and under the same configuration settings on the PASCAL VOC dataset (VOC2007+2012 trainval for the train set, and VOC2007 test for the test set): i) a total number of 100 epochs (including 5 warm-up epochs), ii) a learning rate starting at 10^{-4} and gradually decreasing to 10^{-6} following a cosine scheduling, and iii) a batch size of 8. The training was conducted on Google Colab[3] using a Tesla P100 GPU. The performance of each separable model is compared to that of the original network in terms of i) mean average precision (mAP), ii) model size, and iii) inference speed. Our goal is to find the drop in accuracy associated with the convolutional factorization and to identify the most efficient model. Results are provided in Table 1 and visualized in Fig. 2.

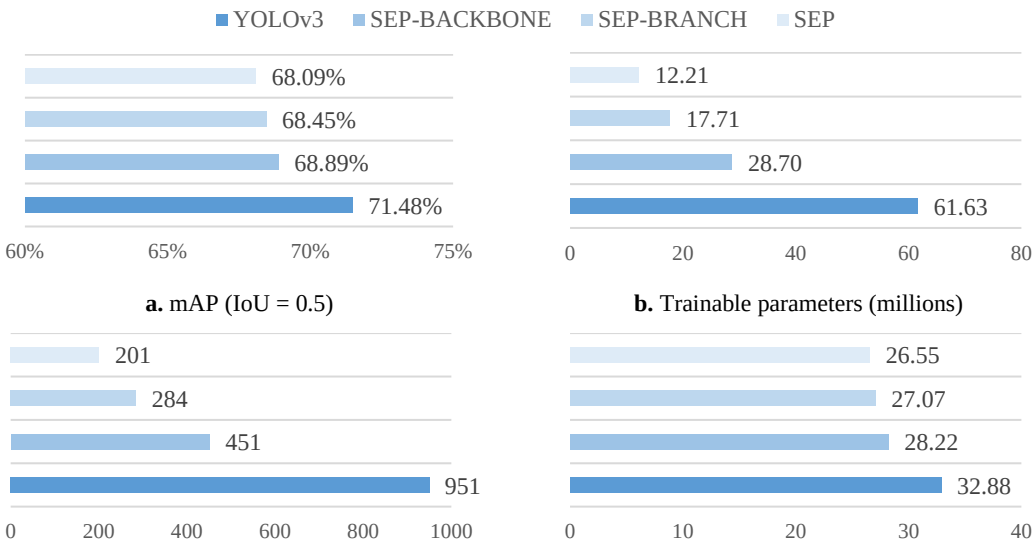
Table 1. Results for YOLOv3 and its separable variants for a 416×416 input

Model	mAP (IoU = 0.5)	# of trainable parameters	Model size (MB[4])	Inference time (milliseconds)
YOLOv3	71.48%	61,626,049	951	32.88
SEP	68.09%	12,211,426	201	26.55
SEP-BRANCH	68.45%	17,706,594	284	27.07
SEP-BACKBONE	68.89%	28,696,930	451	28.22

[3] https://colab.research.google.com/

[4] The model size represents the size of the checkpoint files obtained during the training, which are significantly larger than the compressed files used for inference.

Results show that the conversion to depthwise separable convolutions seems to be associated with a slight drop in accuracy, compared with a sharp decrease in the number of trainable parameters and model size. Even though only the Darknet-53 backbone is subject to this conversion in the SEP-BACKBONE architecture, the latter's accuracy is comparable to that of its counterparts and fails to justify its significant increase in model size. This is probably due to the fact that two types of convolutions do not mix well together, especially when they show large differences in their number of parameters and feature extraction capabilities. While all the separable models seem to be on a par with YOLOv3 in terms of accuracy, the SEP-BRANCH model reduces 70% of YOLOv3's size with only a 3% drop in accuracy, and thus offers the best trade-off between accuracy and speed.



a. mAP (IoU = 0.5)

b. Trainable parameters (millions)

c. Model size (MB)

d. Inference time (milliseconds)

Fig. 2. Results for YOLOv3 and its separable variants for a 416×416 input

5.2 YOLOv3 Model

In the second stage of this work, we apply variational dropout to the SEP-BRANCH model in order to produce a sparse and compact model suitable for real-time applications. Even though adding variational dropout doubles the total number of trainable parameters, the SEP-BRANCH model with variational dropout – noted SEP-BRANCH-VD – still amounts to less than 60% of YOLOv3's trainable parameters. We train both the SEP-BRANCH and SEP-BRANCH-VD models using the same settings adopted in the previous experiment, to the exception of batch size, which we set to 2 (instead of 8). We use a smaller batch size to cater for the large computation overhead associated with the variational dropout terms. Results obtained on the PASCAL VOC dataset are shown in Table 2. They demonstrate the effectiveness of variational dropout when applied to the depthwise separable convolutions of a deep and complex network

like YOLOv3 trained on a large benchmark dataset. They also show that this technique can drop most of the model's weights without damaging its performance. In fact, for a batch size of 2, SEP-BRANCH-VD reaches a mAP equal to that of SEP-BRANCH with only 18% of the latter's parameters, and just 5% of YOLOv3's baseline parameters. However, adding variational dropout has two major drawbacks: i) it requires more than double the training time, and ii) it requires significantly larger computational resources to run on a larger batch size of 8 and preserve the original mAP levels. Nonetheless, for a batch size of 8, we expect the SEP-BRANCH-VD model to sustain a mAP of at least 68% and thus, relatively to YOLOv3, limit the drop in accuracy to 3%.

Table 2. Results of SEP-BRANCH trained with variational dropout

Model	mAP (IoU = 0.5)	# of weight parameters	# of zero weight parameters	Sparsity level
YOLOv3 (batch size of 8)	71.48%	61,626,049	-	-
SEP-BRANCH (batch size of 8)	68.45%	17,706,594	-	-
SEP-BRANCH (batch size of 2)	65.35%	17,706,594	-	-
SEP-BRANCH-VD (batch size of 2)	65.42%	17,706,594	14,584,999	82.37%

During inference, we set to zero all weight parameters with a $log\alpha$ value greater than 3, as they correspond to weights with a dropout rate larger than 95% [12]. Accuracy can be traded for more sparsity by decreasing the $log\alpha$ threshold. For example, with a threshold of 1, the SEP-BRANCH-VD model achieves 84.3% global sparsity with 64% test set accuracy. We provide in Table 3 the test set accuracy and global sparsity levels under different thresholds, and our results show that the drop in accuracy does not justify the minor increase in sparsity.

Table 3. SEP-BRANCH-VD results under different logα thresholds

$log\alpha$ threshold	mAP	Sparsity level
3	65.42%	82.37%
2	64.85%	83.20%
1	64.04%	84.30%
0	60.41%	85.50%

Considering that variational dropout distributes sparsity non-uniformly across the neural network layers, we can make several observations regarding sparsity ratio distribution across the convolutional weights:

- First, the overall sparsity level seems to be gradually increasing throughout the network, which is consistent with the findings in [28]. The first convolutions are

almost fully condensed (sparsity levels go from 0% to 30%), whereas the last ones are almost entirely sparse (sparsity levels between 60% and up to 97%).
- Second, the average sparsity level for the depthwise convolutions is 11%, in contrast with 58% for the pointwise convolutions. This can be due to the fact that depthwise convolutions extract features from the input channels, while pointwise convolutions combine the filtered inputs into a new set of output channels.
- Third, higher sparsity ratios are achieved in standard convolutions (average sparsity level of 70%) compared with depthwise separable convolutions. In particular, the highest sparsity levels are seen at the detection layers, where they reach values greater than 90%.
- Fourth, by examining the sparsity distribution across the different convolutional layers, we notice that the zero values are spread rather randomly across the weight matrices, and do not follow any recognizable pattern.

Lastly, given that a new YOLOv4 model [30] has just been released, we hope that the redundancy seen within the convolutional weights leads to a better understanding of the workings and generalization properties of YOLOv3, and in the future helps the design of more efficient models that focus on parameter and layer quality rather than quantity.

6 Conclusion

6.1 YOLOv3 Model

This study introduces a lightweight and sparse YOLOv3-based model by combining depthwise separable convolutions with VD. We first propose three different YOLOv3 variants by integrating depthwise separable convolutions at different strategic locations within the original network. Results for all three models are satisfactory, with the most efficient model reducing YOLOv3's size by a factor of 3.5 at only a 3% drop in accuracy. We then apply VD to this compact model and further eliminate more than 82% of its weight values, thus effectively removing 95% of YOLOv3's total parameters without any additional drop in accuracy – given that the same batch size is used. The obtained results i) validate the effectiveness of depthwise separable convolutions, ii) demonstrate that a deep and complex neural network based on YOLOv3 and depthwise separable convolutions can undergo extensive sparsification on a large benchmark dataset, and iii) give insights into the relevance of the different YOLOv3 layers.

6.2 Discussion and Future Works

The scope of our present work includes evaluating certain properties related to depthwise separable convolutions and VD within the context of YOLOv3, rather than reaching a global optimum and maximizing the mAP. Therefore, the training hyperparameters need to be reviewed and carefully fine-tuned if higher accuracy levels are to be achieved for all the models evaluated in this study. Moreover, YOLOv3 is typically trained on the MS COCO dataset, and the resulting weights are usually used to initialize the training on PASCAL VOC or any custom dataset. It would be therefore interesting to replicate our experiments on the significantly larger and more varied

COCO dataset and check whether depthwise separable convolutions can leverage the transfer learning property on the VOC dataset as well as standard convolutions do. It would be also interesting to learn whether the sparse topology learned on the COCO dataset using VD can be used to initialize the training on the VOC dataset, since performing the training phase in a fully sparse manner would greatly accelerate the time-to-solution and might even allow the training to be conducted on resource-constrained embedded devices. Finally, knowing that sparsification is an intermediate but crucial step to network compression, our approach can be combined with data compression techniques like quantization and Huffman coding [31] and then integrated with light-weight deep learning frameworks such as TensorFlow Lite in order to reach real-time processing for on-device inference.

References

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90

[2] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. *Inter. Journal of Computer Vision*, 115(3), 211–252. doi:10.1007/s11263-015-0816-y

[3] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Conf. on Comp. Vision & Pattern Recogn.* (CVPR'14), 580–587

[4] Girshick, R. (2015). Fast R-CNN. *2015 IEEE Inter. Conf. on Computer Vision (ICCV'15)*, 1440–1448

[5] Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149

[6] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. *Computing Research Repository*, CoRR abs/1512.02325

[7] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. 2016 *IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR'16), 779–788

[8] Redmon, J., and Farhadi, A. (2016). Yolo9000: Better, faster, stronger. *Computing Research Repository*, CoRR abs/1612.08242

[9] Redmon, J., and Farhadi, A. (2018). Yolov3: An incremental improvement. *Computing Research Repository*, CoRR abs/1804.02767

[10] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (Voc) challenge. *Inter. J. of Computer Vision*, 88(2):303–338. doi:10.1007/s11263-009-0275-4

[11] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2015). Microsoft COCO: Common objects in context. *Computing Research Repository*, CoRR abs/1405.0312

[12] Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. *Computing Research Repository*, CoRR abs/1701.05369

[13] Salem, C., Azar, D., Tokajian, S., (2018). An Image Processing and Genetic Algorithm-Based Approach for the Detection of Melanoma in Patients. *Methods of Information in Medicine*, doi: 10.3412/ME17-01-0061

[14] F. N. Abu-Khzam, S. Li, C. Markarian, F. M. auf der Heide, P. Podlipyan, (2019). Efficient parallel algorithms for parameterized problems. *Theoretical Computer Science*, volume 786, pp. 2-12

[15] Abu-Khzam, F. N., Markarian, C., auf der Heide, F. M., and Schubert, M. (2018). Approximation and Heuristic Algorithms for Computing Backbones in Asymmetric Ad-hoc Networks. *Theory of Computing Systems*, 62(8):1673-1689

[16] Abu-Khzam, F. N., Daudjee, K., Mouawad, A. E., and Nishimura, N. (2015). On Scalable Parallel Recursive Backtracking. *Journal of Parallel and Distributed Computing*, 84:65-75

[17] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *Computing Research Repository*, CoRR abs/1704.04861

[18] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *Computing Research Repository*, CoRR abs/1207.0580

[19] Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. *Computing Research Repository*, CoRR abs/1506.02557

[20] Wang S. & Manning C., Fast dropout training (2013). *Inter. Conf. on Machine Learning* (ICML'13), 118–126

[21] Kingma, D. P., and Welling, M. (2014). Auto-encoding variational bayes. *Computing Research Repository*, CoRR abs/1312.6114

[22] Rezende, D., Mohamed, S., and Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *Inter. Conf. on Machine Learning, 32*, II-1278–II-1286

[23] Mao, Q., Sun, H., Liu, Y., and Jia, R. (2019). Mini-YOLOv3: Real-time object detector for embedded applications. *IEEE Access*, 7:133529–133538

[24] Li, Y., Han, Z., Xu, H., Liu, L., Li, X., and Zhang, K. (2019). YOLOv3-lite: A lightweight crack detection network for aircraft structure based on depthwise separable convolutions. *Applied Sciences*, 9(18):3781

[25] Zhang, P., Zhong, Y., and Li, X. (2019). SlimYolov3: Narrower, faster and better for real-time uav applications. *2019 IEEE/CVF Inter. Conf. on Computer Vision Workshop* (ICCVW'19), pp. 37–45

[26] Zhu P. et al. (2019) VisDrone-VDT2018: The vision meets drone video detection and tracking challenge Results. In: *Leal-Taixé L., Roth S. (eds) Computer Vision – ECCV 2018 Workshops* (*ECCV'18*), pp. 496-518

[27] Ebrahimi, D., Sharafeddine, S., Ho, P., Assi, C., (2020), Autonomous UAV trajectory for localizing ground Objects: A Reinforcement Learning Approach. In *IEEE Trans. on Mobile Computing*, doi: 10.1109/TMC.2020.2966989

[28] Sang, D. V., and Hung, D. V. (2019). YOLOv3-VD: A sparse network for vehicle detection using variational dropout. *Inter. Sym. on Information and Communication Technology* (SoICT'19), 280–284

[29] Gale, T., Elsen, E., and Hooker, S. (2019). The State of sparsity in deep neural networks. *Computing Research Repository*, CoRR abs/1902.09574

[30] Bochkovskiy, A., Wang, C. Y., and Liao, H. Y. (2020). YOLOv4: optimal speed and accuracy of object detection. *Computing Research Repository*, CoRR abs/2004.10934

[31] Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *Computing Research Repository*, CoRR abs/1510.00149