Lebanese American University

School of Engineering

Departments of Industrial and Mechanical Engineering & Electrical and Computer Engineering

The Case for Depthwise Separable Convolutions & Variational Dropout in YOLOv3

MEE594 UNDERGRADUATE RESERCH
Spring 2020

Student Name & ID:

Joseph Chakar – 201602957

Academic Advisor & Co-Advisor:

Dr. Joe Tekli

Mr. Rayan Al Sobbahi

Administrative Advisor:

Dr. Charbel Mansour

Date Report Presented:

06/10/2020

Table of Contents

Table of Figures

Table of Tables

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 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, 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 led to an extremely sparse solution that reduces 95% of the baseline network's parameters at little cost to its accuracy.

1. Introduction

Convolutional neural networks (CNNs) have witnessed tremendous growth following the release of AlexNet (Krizhevsky et al., 2012) at the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) competition (Russakovsky et al., 2015). 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 streams of deep learning based solutions: i) two-stage and ii) single stage detectors. In two-stage detectors like R-CNN (Regions with CNN features) (Girshick et al., 2014), Fast R-CNN (Girshick, 2015), and Faster R-CNN (Ren et al., 2015), 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 simple 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 SSD (Singe Shot MultiBox Detector) (Liu et al., 2016) and YOLO (You Only Look Once) (Redmon et al., 2016; Redmon & Farhadi, 2016; Redmon & Farhadi, 2018).

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 obstacle for modern convolutional object detectors. Among the one-stage models, YOLOv3 (Redmon & Farhadi, 2018) 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 such as PASCAL VOC (Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes) (Everingham et al., 2010) and MS COCO (Microsoft Common Objects in Context) (Lin et al., 2015) can often be prohibitive, and this issue of high computation overhead and power consumption often hinders the deployment of these models on resource-constrained embedded devices.

Over the past few years, numerous techniques for compressing YOLOv3 without notably harming its detection accuracy have been proposed (Li et al., 2019; Mao et al., 2019; Sang & Hung, 2019; Zhang et al., 2019). While most of these solutions produce small and low latency models that match the design requirements for mobile and embedded vision applications, they mostly i) require an application-specific and often complex restructuring of YOLO's architecture, ii) implement a computationally expensive iterative approach, or iii) bring about a significant increase in resource requirements and memory consumption.

In this study, we reduce YOLOv3's size and induce a high state of sparsity within this 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 (Molchanov et al, 2017) to the most efficient YOLOv3 variant, which leads to an extremely sparse solution that effectively compresses the original model by a factor of 20.

2. Related Works

Despite being cited as one of the fastest deep learning-based object detectors, YOLOv3 has a large run-time memory footprint that prevents it from being deployed on embedded devices. In an attempt to resolve this issue, the YOLO series comes with a lightweight version called tiny YOLO. This variation of the YOLO architecture successfully decreases the number of floating point operations per second (FLOPS); however, this reduction in model size and inference time comes with a sharp drop in detection accuracy. Today's object detection algorithms are still attempting to strike a balance between speed and accuracy, and this task has become under extensive investigation in both academia and industry.

Various approaches to produce lightweight and efficient object detection models for mobile and embedded vision applications have been proposed in the literature. For the sake of this study, we mainly focus on applications that use i) depthwise separable convolutions or ii) network sparsification as their underlying foundations.

2.1 Depthwise Separable Convolutions Approaches

MobileNets (Howard et al., 2017) are a class of "mobile-first" models designed to effectively maximize accuracy for a variety of use cases while accounting for the restricted resources in on-device applications. They are a family of fast and small-sized deep neural networks that are built primarily from depthwise separable convolutions, with two simple 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, and 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 were achieved, and the results were compared to a number of popular models in various applications.

Instead of building new models, many researchers have focused on redesigning YOLO's architecture in order to create lighter versions that can maintain its high detection accuracy and speed. One example of a small-sized YOLOv3 variant that also relies on depthwise separable convolutions is Mini-YOLOv3. Mao et al. (2019) constructed their new backbone network with a parameter size of only 16% that of Darknet-53 using these factorized convolutions. In the residual layers of YOLOv3, they adopted an opposite strategy to ResNets (He at al, 2015), which stack a 3×3 convolution between two 1×1 pointwise convolutions responsible for leaving the former in a bottleneck with a smaller input and output. Considering that depthwise convolutions reduce the input dimension and then restore it, they instead used a 1×1 convolution to increase the input dimension by a factor of 4 and then decrease it, leaving the 3×3 layer an inverted bottleneck with larger input/output dimensions. To compensate for the large calculations associated with these operations, they grouped convolutions and added a channel shuffle to enable cross-group information flow. Furthermore, a Multi-Scale Feature Pyramid Network (MSFPN) based on a simple U-shaped structure was introduced to improve the performance of the multi-scale object detection. 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. In summary, the compact Mini-YOLOv3 model achieved a detection accuracy on the MS COCO dataset comparable to that of YOLOv3, with half the inference time. Another example of a lightweight model is YOLOv3-Lite. Inspired by YOLOv3, Li et al. (2019) also used depthwise separable convolutions to design a feature extraction backbone and adopted the idea of a feature pyramid network that combines low- and high-resolution information at three different layers. Their lightweight crack detection network used YOLOv3 for the bounding box regression to reach state-of-the-art performance on their custom dataset for cracks in aircraft structures.

2.2 Network Sparsification Approaches

Sparsification is another leading approach to address the speed versus accuracy challenge. 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, thus storing and performing mathematical operations on weight matrices becomes more efficient processes. Deep neural networks have been shown to tolerate high sparsity levels (Han et al., 2015; Narang et al., 2017; Ullrich et al., 2017); consequently, this property has been leveraged to significantly reduce the cost associated with deploying state-of-the-art models in resource-constrained environments (Theis et al., 2018; Kalchbrenner et al., 2018; Valin & Skoglund, 2019). Different techniques exist for sparsifying neural networks, some of which can be applied on-the-fly during the training process (Lie et al., 2017). On the one hand, Zhang et al. (2019) enforced a channel-level sparsity on YOLOv3 by imposing L1 regularization on the gamma regularizer of the network's batch normalization layers. By pruning YOLOv3's convolutional layers iteratively, they were able to find a sparse model for real-time Unmanned Aerial Vehicles (UAV) applications, referred to as SlimYOLOv3. On the other hand, Sang & Hung (2019) gradually introduced variational dropout to YOLOv3. After training it on a self-collected dataset about traffic in Vietnam, they managed to eliminate up to 91% of the original network weight parameters with a negligible drop in accuracy.

3. Background

3.1 YOLOv3 Architecture

The concept of the YOLO object-detection algorithm (Redmon et al., 2016) is built on a unique set of characteristics that stands out from traditional systems in order to reduce the computational complexity and achieve real-time inference speeds while maintaining high average precision (Redmon et al., 2016). 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. 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. Each cell predicts B bounding boxes and their confidence scores, as well as one set of C conditional class probabilities. The confidence scores reflect how confident the model is of the presence of an object inside that box as well as the accuracy of its prediction. Furthermore, the classification network architecture was inspired by GoogLeNet (Szegedy et al., 2014) and Lin et al.'s (2014) work, relying 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 (Redmon et al., 2016) 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. In (Redmon & Farhadi, 2016), the authors introduce some improvements to the YOLO model: i) adding batch normalization, ii) increasing the classifier resolution, and iii) proposing a new classification model called Darknet-19. Some of the most interesting contributions that truly redefined YOLO are perhaps the use of anchor boxes and multi-scale training. In YOLOv2 (Redmon & Farhadi, 2016), the prediction boxes are assisted with predefined anchors or priors chosen via k-means clustering on the training dataset. As for the multi-scale training method, the model is able to run at varying input image sizes, which offers an easy tradeoff between speed and accuracy. This improved version of YOLO established itself as a state-of-the-art model on standard detection tasks like PASCAL VOC.

The third and most powerful installment in this series is YOLOv3 (Redmon & Farhadi, 2018), which adds several incremental improvements to its predecessors. First, the backbone network is upgraded to Darknet-53. As the name implies, this feature extractor is 53 layers deep, where each layer is followed by a batch normalization layer and Leaky ReLU activation. Darknet-53's tabulated summary is shown in Figure 1.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 1: Darknet-53 Architecture (Redmon & Farhadi, 2018)

For the task of detection, several more convolutional layers are stacked on top of the backbone, and the new YOLOv3 architecture incorporates several elements that are now staple in most state-of-the-art algorithm such as residual blocks, skip connections, and upsampling. YOLOv3 is also capable of accurately detecting objects of various sizes by making predictions at three different scales located at three separate places in the network. The system extracts features from those scales using a concept similar to feature pyramid networks in (Lin et al., 2017) and the detection is marked by a 1×1 kernel applied on feature maps of three different sizes. At each scale, a 3D tensor encodes i) the four bounding box offsets, ii) the level of confidence of having an object (referred to as objectness), and iii) the corresponding class predictions, resulting in a final tensor shape of $N \times N \times [3 * (4 + 1 + C)]$, where C is the number of classes. Predictions for each consecutive scale build upon prior computations and the fine-grained information from previous feature maps in order to extract more meaningful semantic information and accurately detect small-scale objects. The overall network architecture is visualized in Figure 2.

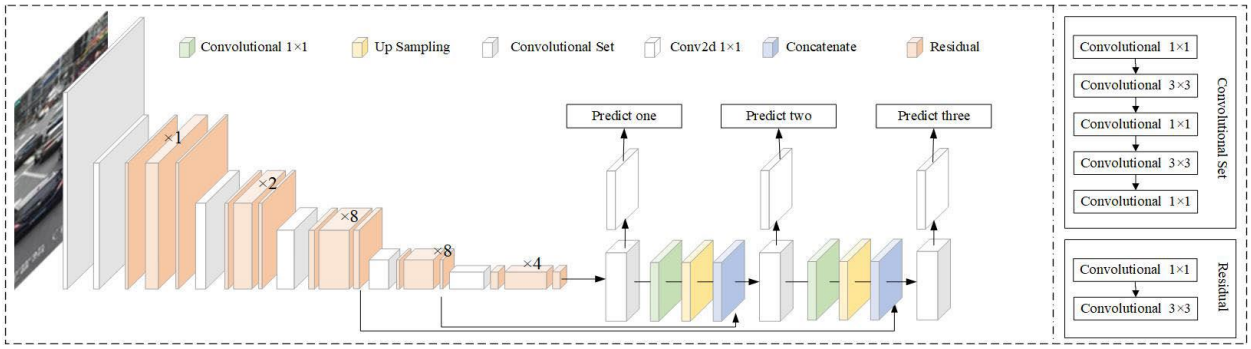


Figure 2: YOLOv3 Architecture (Mao et al., 2019)

3.2 Depthwise Separable Convolution

YOLOv3's size is significantly reduced simply by swapping its standard 3×3 convolution with a depthwise separable convolution, which is a form of a factorized convolution split into two compact layers, as adopted in MobileNets (Howard et al., 2017). A depthwise convolution first applies a single 3×3 filter to each input channel, then a pointwise convolution applies a 1×1 convolution on the former's output channels. Compared with the standard convolution, which filters its inputs and combines them into a new set of outputs in a single step, the depthwise separable convolution substantially reduces the model size and computational cost.

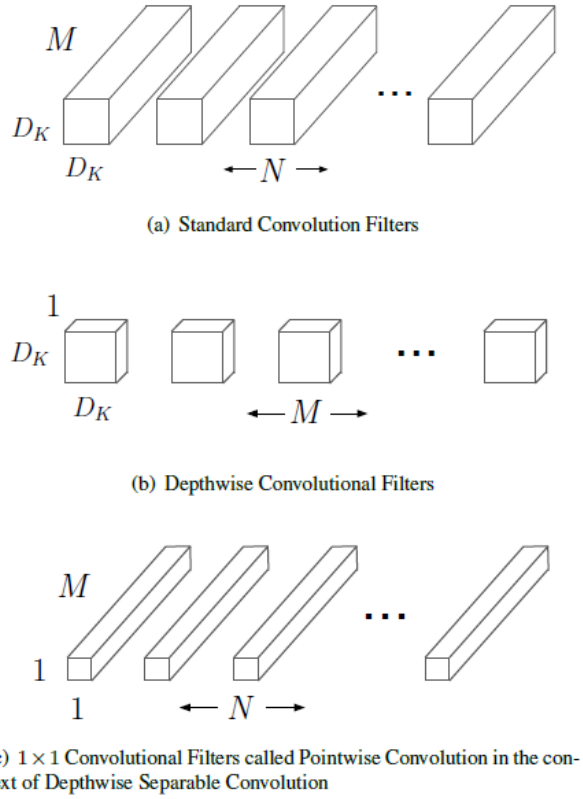


Figure 3: Comparison of a standard convolution to depthwise and pointwise convolutions (Howard et al., 2019)

Figure 3 illustrates how a standard convolution can be factorized into depthwise and pointwise convolutions. 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 are the number of input channels and number of output channels, respectively. 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:

$$D_k^2 . M . N . D_F^2 \tag{1}$$

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

$$D_K^2 . M . D_F^2 + M . N . D_F^2 \tag{2}$$

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

$$\frac{D_K^2 . M . D_F^2 + M . N . D_F^2}{D_K^2 . M . N . D_F^2} = \frac{1}{N} + \frac{1}{D_K^2} \tag{3}$$

Similar to MobileNets (Howard et al., 2017), 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 trainable parameters is also associated with a minor drop in performance. In Section 5, we describe various experimental results that produce different reductions in size and accuracy based on which standard convolutions are changed to separable convolutions within YOLOv3's architecture.

3.3 Network Sparsification using Variational Dropout

Due to their high flexibility, neural networks tend to learn complex patterns that could be fit for the training dataset but not necessarily for new data. When left unchecked, this flexibility often strips the model from its ability to generalize to new unseen data. Dropout (Hinton et al., 2012) is a popular and empirically effective way of controlling this over-fitting by randomly dropping out or ignoring a certain pre-defined percentage of neural network units during training. First introduced in (Kingma et al. 2015), variational dropout is a more recent neural network regularization technique originally proposed as a reinterpretation of Gaussian dropout (Wang & Manning, 2013) – a more efficient approximation of the standard (binary) dropout – through variational inference under the Bayesian view. Simply put, it is a generalization of Gaussian dropout but with learnable dropout rates. Variational dropout has been later extended in (Molchanov et al., 2017) to include individual parameter dropout rates, where weights with high

dropout rates can be removed post-training to get highly sparse models with a virtually identical performance.

For a training set $\mathcal{D}$ of N samples $(x_i, y_i)_{i=1}^{N}$and a classic 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 $\mathcal{D}$ into a belief in the form of a posterior distribution $p(w|\mathrm{D})$:

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

Since computing the true posterior distribution $p(w|\mathrm{D})$ is computationally intractable, an approximation is used instead (Molchanov et al., 2017). In variational inference, the parameters ϕ of some model $q_\phi\ (w)$ are optimized such that the approximated parameterized model is as close 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:

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

where:

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

Using the Stochastic Gradient Variational Bayes (SGVB) algorithm (Kingma et al., 2015), 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.

In the standard formulation of variational dropout, the weights of neural network are assumed to be drawn from a fully-factorized Gaussian approximate posterior:

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

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:

$$p_{ij} = \frac{\alpha_{ij}}{1 + \alpha_{ij}} \tag{8}$$

Clearly, if $\alpha_{ij} = 0$, then w_{ij} is fully preserved with no dropout rate. In contrast, when $\alpha_{ij} \to +\infty$, $p_{ij} \to 1$, thus w_{ij} can be completely removed to sparsify the model.

For each training step, the weights are sampled from the normal distribution $\mathcal{N}$, and the so-called re-parameterization trick (Kingma & Welling, 2013; Rezende et al., 2014) is used to differentiate the loss with respect to the parameters through this sampling operation.

$$w_{ij} = \theta_{ij}\big(1 + \sqrt{\alpha_{ij}}\epsilon_{ij}\big) \sim \mathcal{N}(w_{ij}|\theta_{ij}, \alpha_{ij}\theta_{ij}^2). \tag{9}$$

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 (Molchanov et al., 2017):

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

where $k_1 = 0.63576$, $k_2 = 1.87320$, $k_3 = 1.48695$, and $C = -k_1$

The authors in (Molchanov et al., 2017) found some difficulty 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 proposed 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 gradually increasing it from 0 to 1.

4. Proposal: Separable YOLOv3 with Variational Dropout

After careful examination of the YOLOv3 architecture and its limitations, we set out to study the size-accuracy trade-off associated with substituting standard convolutions with depthwise separable ones and adding variational dropout. We rely on Yung Yang's TensorFlow implementation of YOLOv3, which is publicly available online[1].While this implementation is overall a faithful reproduction of Redmon & Farhadi's work (2018), Yang adds several tweaks to his model, 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 in the same environment and under the same settings.

4.1 Separable YOLOv3 Configurations

We consider three configurations by replacing YOLOv3's standard convolutions with depthwise separable ones at different locations within the network, as described in Table 1:

Configuration name	Description
SEP	Taking into consideration that there are multiple convolutions with (1×1) kernels across the network, particularly in the residual layers, and that reducing and then restoring the convolution dimension would greatly weaken the expression ability of the model, all standard convolutions are replaced with depthwise separable convolutions, except those with a (1×1) filter. Note that this exception was extended to the two remaining configurations.
SEP-BRANCH	All standard convolutions are replaced with depthwise separable convolutions, except 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.

Table 1: Description of the different separable YOLOv3 configurations considered

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

Note that each convolution in YOLOv3 is followed by a batch normalization layer as well as a leaky ReLU activation function. As such, batch normalization and the leaky ReLU nonlinearity were also applied after each of the depthwise and pointwise (1×1) convolutions. This convolutional layer transformation is illustrated in Figure 4.

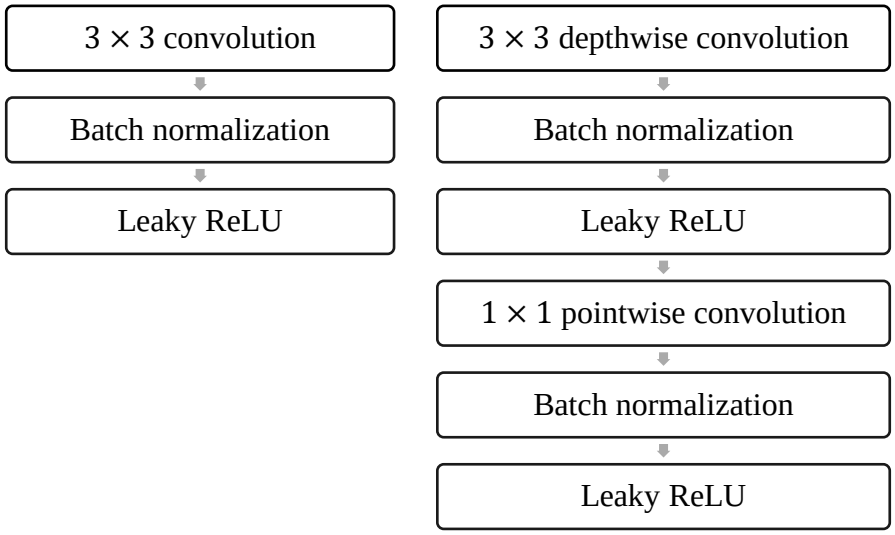


Figure 4: Standard convolutional layer (left) vs. depthwise separable convolution layer (right)

4.2 Variational Dropout (VD)

Since our empirical results show SEP-BRANCH to be the most efficient model (cf. Section 5), the variational dropout tests are directly carried out on this configuration. We rely on Gale et al.'s (2019) 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, 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:

$$L_{total} = L_{GIoU} + L_{Conf} + L_{prob} + \lambda D_{KL} \tag{11}$$

Knowing that training the model with variational dropout from the start is not recommended (Molchanov et al., 2017), we use a regularizer coefficient λ to balance the YOLOv3 loss and the KL-divergence term. Similarly to the study in (Sang & Hung, 2019), we gradually ramp the regularizer coefficient to induce sparsity. We train the model without any variational dropout 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 an additional training starts to increase the sparsity level at

[2] https://github.com/ars-ashuha/variational-dropout-sparsifiesdnn.

the expense of the model performance. We therefore set the learning rate to 10^{-6} and fine-tune the model sparsity over 10 epochs.

Even though adding variational dropout doubles the total number of trainable parameters, the SEP-BRANCH-VD model still amounts to less than 60% of YOLOv3's trainable parameters. However, training with variational dropout results in a drastic memory overhead. Consequently, for the same computational resources, we can only use a batch size of 2. Knowing that the batch size impacts the model performance, we train both the SEP-BRANCH and SEP-BRANCH-VD models and compare their performance for a batch size of 2.

5. Experimental Results

We trained YOLOv3 and its separable configurations from scratch under the same configuration settings on PASCAL VOC, mainly:

- Total number of epochs: 100 epochs (with 5 warm-up epochs)
- Learning rate: starts at 10^{-4} and gradually decreases to 10^{-6} following a cosine scheduling
- Batch size: 8

5.1 Separable YOLOv3 Configurations Results

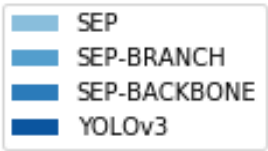
The performance of each separable YOLOv3 configuration is compared to that of the original network in terms of i) mean average precision (mAP), ii) model size, and iii) inference speed. Our goal is to find the drop in accuracy associated with the convolutional factorization and to identify the most efficient configuration. We conducted the training on Google Colab using a Tesla P100 GPU, and the experimental results are outlined and visualized in Table 2 and Figure 5, respectively.

Model	mAP (IoU = 0.5)	Trainable parameters (millions)	Model size (MB)[3]	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

Table 2: Comparison between YOLOv3 and its separable variants for a 416x416 input

Model	ΔmAP	ΔModel size	ΔInference speed
SEP	-3.39%	-80%	+20%
SEP-BRANCH	-3.03%	-70%	+18%
SEP-BACKBONE	-2.59%	-53%	+14%

Table 3: Drops in accuracy, model size, and inference time of each YOLOv3 variant



[3] 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.

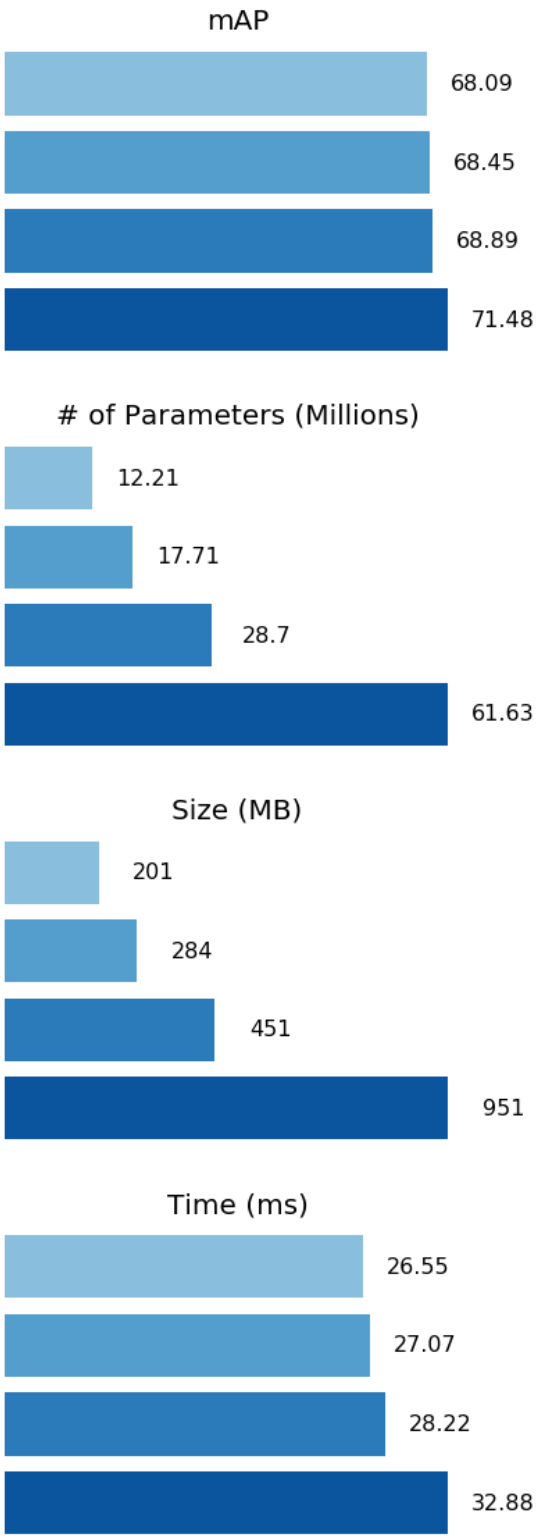


Figure 5: Comparison between YOLOv3 and its separable variants for a 416x416 input

Based on the above results, the conversion to depthwise separable convolutions seems to be associated with a sharp decrease in the model size but only a slight drop in accuracy. Indeed, all the separable configurations are on a par with YOLOv3. Even though only the Darknet-53 backbone is subject to this conversion in the SEP-BACKBONE architecture, the latter's performance is comparable to that of its counterparts and fails to justify the increase in model size it brings about. This can be due to two types of convolution' inability to mix well together, given the large difference in number of parameters and thus feature extraction capability between them. Lastly, the SEP-BRANCH configuration reduces 70% of YOLOv3's size with only a 3% drop in accuracy, and thus seems to offer the best trade-off between accuracy and speed.

5.2 Separable YOLOv3 VD Results

In the second stage of this work, we apply variational dropout to the SEP-BRANCH model in order to get a sparse and compact model suitable for real-time applications. As previously mentioned, the SEP-BRANCH-VD model can only be trained using a batch size of 2 due to the memory overhead brought on by the variational dropout parameters. Results obtained from training on the VOC dataset are shown in Table 7:

Model	mAP (IOU = 0.5)	Trainable parameters (millions)	Non-zero weight parameters (millions)	Sparsity level
YOLOv3	71.48%	61,626,049	61,626,049	0%
SEP-BRANCH (batch size of 8)	68.45%	17,706,594	17,706,594	0%
SEP-BRANCH (batch size of 2)	65.35%	17,706,594	17,637,627	0%
SEP-BRANCH-VD (batch size of 2)	65.42%	17,706,594	3,121,595	82%

Table 4: Results of SEP-BRANCH trained with variational dropout

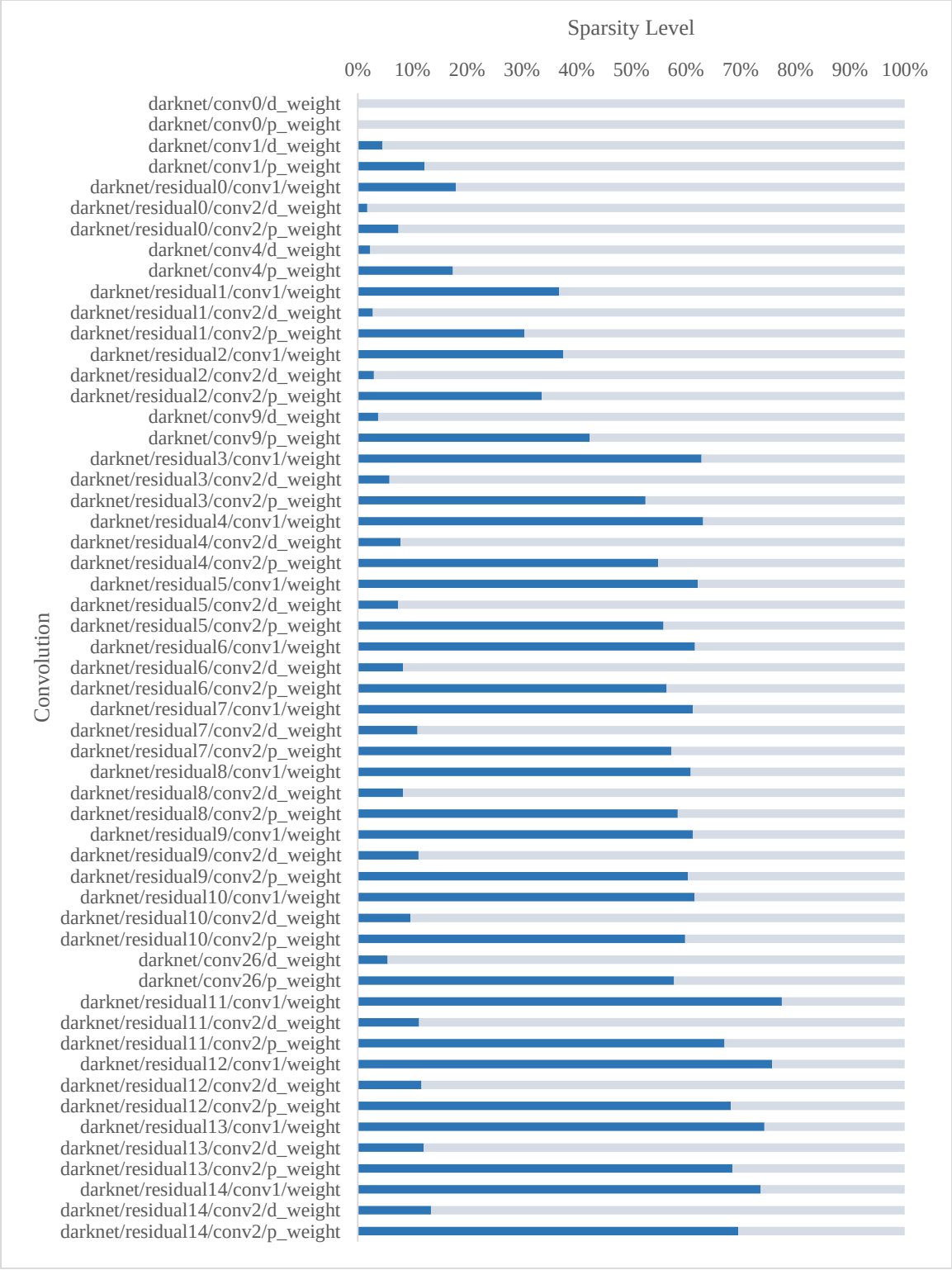
During inference, we set to zero all weight parameters with a $log\alpha$ value greater than 3, as they correspond to weights with a dropout rate larger than 95% (Molchanov et al., 2017). Accuracy can be traded for more sparsity by decreasing the logα threshold. For example, with a threshold of 1, the SEP-BRANCH-VD model achieves 84.3% global sparsity with 64.04% test set accuracy. The test set accuracy and global sparsity level under different thresholds are reported in Table 4 below, and they show that the drop in accuracy does not justify the minor increase in sparsity.

logα threshold	mAP	Sparsity level
3	65.42%	82%
2	64.85%	83.2%
1	64.04%	84.3%
0	60.41%	85.5%

Table 5: SEP-BRANCH-VD results under different $log\alpha$ thresholds

Results in Tables 3 and 4 demonstrate the effectiveness of variational dropout when applied to the depthwise separable convolutions of a deep and complex network like YOLOv3 and trained on a large benchmark dataset. They also show that this technique can cut out 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 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 batch size of 8 and preserve the original mAP obtained.

Given that variational dropout distributes sparsity non-uniformly across the neural network layers, further investigation of the weight-specific sparsity ratios shown in Figure 6 below highlights the relevance of each YOLOv3 layer.


Sparsity Level
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Convolution
darknet/conv0/d_weight
darknet/conv0/p_weight
darknet/conv1/d_weight
darknet/conv1/p_weight
darknet/residual0/conv1/weight
darknet/residual0/conv2/d_weight
darknet/residual0/conv2/p_weight
darknet/conv4/d_weight
darknet/conv4/p_weight
darknet/residual1/conv1/weight
darknet/residual1/conv2/d_weight
darknet/residual1/conv2/p_weight
darknet/residual2/conv1/weight
darknet/residual2/conv2/d_weight
darknet/residual2/conv2/p_weight
darknet/conv9/d_weight
darknet/conv9/p_weight
darknet/residual3/conv1/weight
darknet/residual3/conv2/d_weight
darknet/residual3/conv2/p_weight
darknet/residual4/conv1/weight
darknet/residual4/conv2/d_weight
darknet/residual4/conv2/p_weight
darknet/residual5/conv1/weight
darknet/residual5/conv2/d_weight
darknet/residual5/conv2/p_weight
darknet/residual6/conv1/weight
darknet/residual6/conv2/d_weight
darknet/residual6/conv2/p_weight
darknet/residual7/conv1/weight
darknet/residual7/conv2/d_weight
darknet/residual7/conv2/p_weight
darknet/residual8/conv1/weight
darknet/residual8/conv2/d_weight
darknet/residual8/conv2/p_weight
darknet/residual9/conv1/weight
darknet/residual9/conv2/d_weight
darknet/residual9/conv2/p_weight
darknet/residual10/conv1/weight
darknet/residual10/conv2/d_weight
darknet/residual10/conv2/p_weight
darknet/conv26/d_weight
darknet/conv26/p_weight
darknet/residual11/conv1/weight
darknet/residual11/conv2/d_weight
darknet/residual11/conv2/p_weight
darknet/residual12/conv1/weight
darknet/residual12/conv2/d_weight
darknet/residual12/conv2/p_weight
darknet/residual13/conv1/weight
darknet/residual13/conv2/d_weight
darknet/residual13/conv2/p_weight
darknet/residual14/conv1/weight
darknet/residual14/conv2/d_weight
darknet/residual14/conv2/p_weight

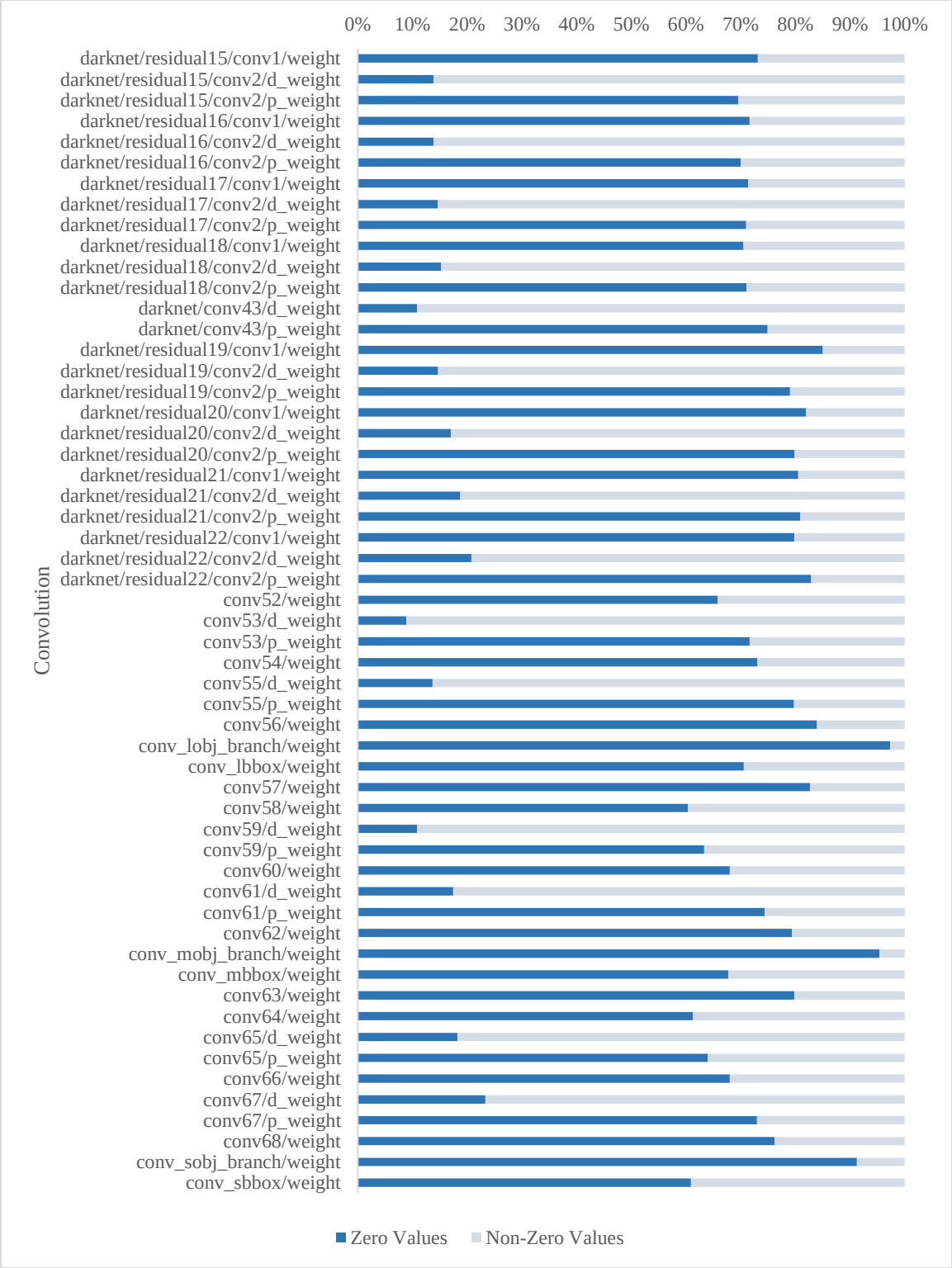


Figure 6: Weight-specific sparsity ratios

Several observations can be made from the results shown above. First, the overall sparsity level seems to be gradually increasing throughout the network, which is consistent with the findings of Sang & Hung (2019). The first convolutions are almost fully condensed (sparsity levels between 0% and 30%), whereas the last ones are almost entirely sparse (sparsity levels between 60% and 97%). Second, the average sparsity level for the depthwise convolutions is 11%, in contrast with 58% for the pointwise convolutions. This can be explained by 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. Moreover, higher sparsity ratios are achieved in standard convolutions (average sparsity level of 70%) than in depthwise separable convolutions. In particular, the highest sparsity levels are seen at the detection layers, where they reach values greater than 90%. After examining the sparsity distribution across the standard convolutions, 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 (Bochkovskiy et al, 2020) has just been released, we hope that the redundancy seen within the individual weights leads to a better understanding of the workings and generalization properties of YOLOv3, and in the future helps researchers design more efficient models that focus on parameter and layer quality rather than quantity.

In summary, our results show that replacing standard convolutions with depthwise separable ones can simply yet effectively reduce YOLOv3's size without needing to redesign its backbone or add new features, as was done in (Mao et al., 2019) and (Sang & Hung, 2019) (cf. Section 2). They also show that variational dropout can successfully eliminate the majority of the model weights without any additional drop in accuracy – but only if the training is done using the same batch size.

6. Conclusion

6.1 Recap

This study introduces and evaluates various factorized models based on YOLOv3, by combining depthwise separable convolutions with variational dropoup in an effort to reduce model size and inference time. We propose three different models integrating depthwise separable convolutions at different strategic locations within the original network. Results for all three models are satisfactory, with the best variant reducing YOLOv3's network size by a factor of 3.5, without any noticeable drop in accuracy. In addition, after applying variational dropout to this compact model, more than 80% of its weight values are further eliminated, thus effectively eliminating 95% of YOLOv3's total parameters at little cost to its accuracy. The obtained results i) give insights into the relevance of every individual YOLOv3 layer, ii) are promising for real-time and embedded systems applications with limited processing capabilities, and iii) demonstrate depthwise separable convolutions' ability to fit a deep and complex neural network like YOLOv3 and undergo extensive sparsification on a large benchmark dataset.

6.2 Future Directions

Several improvements to this study lay ahead in the near future. First, we aim to generalize our findings by replicating our experimental results on the MS COCO benchmark dataset, which contains a significantly larger number images with more varied object sizes and classes. To the best of our knowledge, the mAP of YOLOv3 trained from scratch on VOC is not officially documented by Redmon & Farhadi nor validated in the adopted TensorFlow implementation. Indeed, YOLOv3 is typically trained on the COCO dataset, and the resulting weights are usually used to initialize the training on VOC or any custom dataset. Since Redmon & Farhadi (2016) reported a mAP of 76.8% for YOLOv2 trained on VOC from scratch, and given the numerous upgrades present in YOLOv3, it is clear that the hyperparameters and their ranges of values should be reviewed and carefully hand-tuned to reach a comparable performance. It is well known that the number of possible training settings cannot all be practically explored, and thus the possibility of getting improved results under unexplored settings should not be eliminated. Nevertheless, the reported results fit the scope of our study, which is not to reach a global optimum and maximize the mAP, but rather to evaluate certain model properties. Second, we plan to investigate whether depthwise separable convolutions can leverage the transfer learning property as well as standard convolutions do, by initializing the training of the separable configurations on PASCAL VOC from the obtained COCO weights. Third, we would like to determine whether the sparse topology learned using variational dropout could be used to initialize a custom training, 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. Lastly, knowing that sparsification is an intermediate but crucial step to network compression, we can combine this process with modern techniques like quantization and Huffman coding (Han et al., 2016; Ullrich et al., 2017) as well as deep learning frameworks such as TensorFlow Lite for on-device inference.

References

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv:2004.10934*.

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

Gale, T., Elsen, E., & Hooker, S. (2019). The state of sparsity in deep neural networks. *ArXiv:1902.09574*.

Girshick, R. (2015). Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1440–1448. doi:10.1109/ICCV.2015.169

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580–587. doi:10.1109/CVPR.2014.81

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ArXiv:1510.00149*.

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning Both Weights and Connections for Efficient Neural Networks. *Proceedings of the 28th International Conference on Neural Information Processing Systems, 1*, 1135–1143.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *ArXiv:1512.03385*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv:1207.0580*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv:1704.04861*.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning, 37*, 448–456.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., Oord, A., Dieleman, S., & Kavukcuoglu, K. (2018). Efficient Neural Audio Synthesis. *Proceedings of the 35th International Conference on Machine Learning*, 2410–2419.

Kingma, D. P., Salimans, T., & Welling, M. (2015). Variational dropout and the local reparameterization trick. *ArXiv:1506.02557*.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. *ArXiv:1312.6114*.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM, 60*(6), 84–90. doi:10.1145/3065386

Li, Y., Han, Z., Xu, H., Liu, L., Li, X., & Zhang, K. (2019). YOLOv3-lite: A lightweight crack detection network for aircraft structure based on depthwise separable convolutions. *Applied Sciences, 9*(18), 3781. doi:10.3390/app9183781

Lin, M., Chen, Q., & Yan, S. (2014). Network in network. *ArXiv:1312.4400.*

Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie S. (2017). Feature Pyramid Networks for Object Detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 936–944.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). Microsoft coco: Common objects in context. *ArXiv:1405.0312.*

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. *ArXiv:1512.02325.*

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning efficient convolutional networks through network slimming. *ArXiv:1708.06519.*

Mao, Q., Sun, H., Liu, Y., & Jia, R. (2019). Mini-YOLOv3: Real-time object detector for embedded applications. *IEEE Access, 7*, 133529–133538. doi:10.1109/ACCESS.2019.2941547

Molchanov, D., Ashukha, A., & Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. *ArXiv:1701.05369*.

Narang, S., Elsen, E., Diamos, G., & Sengupta, S. (2017). Exploring sparsity in recurrent neural networks. *ArXiv:1704.05119.*

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. 2016 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779–788. doi:10.1109/CVPR.2016.91

Redmon, J., & Farhadi, A. (2016). Yolo9000: Better, faster, stronger. *ArXiv:1612.08242*.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *ArXiv:1804.02767*.

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*(6), 1137–1149. doi:10.1109/TPAMI.2016.2577031

Rezende, D., Mohamed, S., & Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning, 32*, II-1278–II-1286.

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. *International Journal of Computer Vision, 115*(3), 211–252. doi:10.1007/s11263-015-0816-y

Sang, D. V., & Hung, D. V. (2019). YOLOv3-VD: A sparse network for vehicle detection using variational dropout. *Proceedings of the Tenth International Symposium on Information and Communication Technology – SoICT 2019*, 280–284. doi:10.1145/3368926.3369691

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions. *ArXiv:1409.4842.*

Theis, L., Korshunova, I., Tejani, A., & Huszár, F. (2018). Faster gaze prediction with dense networks and Fisher pruning. *ArXiv:1801.05787.*

Ullrich, K., Meeds, E., & Welling, M. (2017). Soft weight-sharing for neural network compression. *ArXiv:1702.04008*.

Valin, J.-M., & Skoglund, J. (2019). Lpcnet: Improving neural speech synthesis through linear prediction. *ArXiv:1810.11846.*

Wang S., & Manning, C. Fast dropout training. (2013). *Proceedings of the 30th International Conference on Machine Learning*, 118–126.

Zhang, P., Zhong, Y., & Li, X. (2019). Slimyolov3: Narrower, faster and better for real-time uav applications. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 37–45. doi:10.1109/ICCVW.2019.00011