CONVOLUTIONAL NEURAL NETWORKS FOR VISUAL INFORMATION ANALYSIS WITH LIMITED COMPUTING RESOURCES

Paraskevi Nousi *Emmanouil Patsiouras* *Anastasios Tefas* *Ioannis Pitas*

Department of Informatics, Aristotle University of Thessaloniki

ABSTRACT

Over the past decade, Deep Convolutional Neural Networks with heavy architectures and large numbers of parameters have achieved state-of-the-art results and eclipsed other methods in multiple visual analysis tasks, including object detection. However, the real-time requirements of such tasks directly conflict with the restricted computational capabilities of embedded systems, prohibiting the immediate deployment of bulky models, and necessitating their optimization for inference. Among popular inference optimization methods, parameter pruning techniques reduce the number of parameters leading by extension to fewer computational operations. Furthermore, inference-targeted optimization schemes provided by Deep Learning frameworks can yield significant speed ups, for example by allowing half-precision floating point operations. We investigate the behavior of various model configurations in object detection tasks and perform a comparative study on inference optimization methods which aim to reduce the computational cost of Convolutional Neural Networks, while examining the effect of such methods on their performance.

Index Terms— Convolutional Neural Networks, Embedded Systems, Inference Optimization

1. INTRODUCTION

Deep Convolutional Neural Networks (CNNs) have been excelling continuously on various challenging visual analysis tasks and competitions, including the ILSVRC object recognition and detection challenges [1], and the PASCAL VOC challenges [2]. Deep models with parameter-heavy architectures have been successfully trained and deployed on such tasks, partly due to the availability of large collections of annotated, such as the ImageNet or COCO datasets [3], and partly due to the continuous development of increasingly more powerful GPUs. However, the power consumption and sheer size of such models inhibit their use on mobile and embedded systems, e.g., on Unmanned Aerial Vehicles (UAVs) and autonomous robots in general. The GPUs available for deployment on such systems are inadequate in terms of computational power, thus severely slowing down the performance of large models and rendering real-time deployment almost impossible. Moreover, memory constraints prohibit the direct deployment of large models even when real-time requirements can be relaxed. On the other hand, applications related to visual analysis tasks have become progressively more popular, increasing the demand of deployment of large CNNs on mobile devices.

In this paper, we investigate the behavior of Deep Learning (DL) models in the context of deployment on UAVs, by examining the effect of various hyperparameters on both the speed and accuracy of such models. Furthermore, we explore post-training, inference-focused optimization schemes, to facilitate the use of deep models for visual analysis tasks on devices with limited computational capabilities. We focus our experiments on visual object detection tasks using the NVIDIA Jetson TX2 module. Finally we provide pragmatic solutions, and model configurations which perform at real-time or near real-time, while closely monitoring the accuracy trade-off.

The rest of this paper is organized as follows. Section 2 introduces the motivation behind our work and describes previous work related to deploying large Deep Learning models on mobile devices. In Section 3, we describe the models we experiment with and their parameters which may be tuned to allow their deployment on mobile devices. In Section 4, we present and discuss the experiments conducted, while providing insights into the effect of the various hyperparameters on the speed-accuracy trade-off. Finally, Section 5 concludes our findings and discusses solutions to the problems arising when deploying DL models to mobile devices.

2. RELATED WORK

In the context of UAVs, visual analysis tasks may be of assistance in UAV-based intelligent cinematography, e.g., by detecting and tracking a desired target, as well as in flight safety related tasks, such as obstacle detection and avoidance. However, the deployment of large models on mobile devices is subject to computational and memory constraints.

Methods developed for the purpose of allowing large models to be utilized in mobile devices can in general be separated into two categories based on whether the optimization occurs during or after the training phase. The first case entails training smaller or otherwise more efficient models.

Reducing the number of layers or parameters per layer, and thus calculations, is one such approach. Algorithmic approaches, such as depthwise separable convolutions [4], also belong to this category. MobileNets [5] exploit both of the aforementioned approaches and were proposed specifically for feature extraction for the purpose of deployment on mobile devices, as their name suggests. In [6], MobileNets are pitted against other popular feature extractors, including the Inception V2 model [7], in the context of feature extraction for object detection, and the effect of altering the input size on the detection precision is examined, among other factors. Larger input sizes lead to larger heatmaps and denser object detection, but impose heavy memory and computational constraints. In contrast, smaller input sizes are processed faster but lead to coarser, less accurate predictions.

In the second case, a large model is optimized post-training, thus separating the training and deployment phases. For example, weight quantization methods may be applied to reduce the memory requirements of a network while attempting to maintain the information contained in the original parameters [8]. NVIDIA's TensorRT library offers weight quantization techniques among other optimizations, including layer/tensor fusion schemes, leading to significant speed gains. Knowledge distillation methods [9] also belong to this category.

Like [6], we investigate speed-accuracy trade-offs for various models and examine the effect of altering their hyperparameters on this trade-off. However, we explore more extreme hyperparameter choices to facilitate the deployment of DL models on mobile devices. We include the model implementation as a practical consideration, as it can heavily influence the performance of a model. Finally, we also experiment with both during-training and post-training methods as well as combinations of the two.

3. LOW-COST CONVOLUTIONAL NEURAL NETWORKS

For the purpose of object detection, we focus our study on single-stage detectors without region proposals, namely the Single Shot Detector (SSD) [10], and the You Only Look Once (YOLO) detector [11]. Although region-based detectors, such as Faster R-CNN [12], are more accurate, they tend to be slower than single-stage detectors as demonstrated in [6], motivating our choice of evaluated detectors.

SSD [10] is a single stage multiobject detector, meaning that a single feed forward pass of an image suffices for the extraction of multiple bounding boxes with coordinate and class information and no Region of Interest (ROI) pooling occurs internally. In its original form, the detector relied on the VGG16 [13] architecture for feature extraction, and added a number of layers upon it to extract better defined boxes. Two versions were proposed, one running at 300×300 input size and one at 500×500, with the latter producing the best results in terms of detection precision while being significantly slower than the first.

In [6], SSD was used as a meta-architecture for single stage object detection and compared against region-based detectors. Among the findings of that work, was that SSD with MobileNets and Inception V2 for the feature extraction step provided the best time performance at the cost of lower detection precision, as evaluated on the challenging COCO dataset.

Another factor which affects the speed of detection is the number of classes to be recognized, since recognizing more classes require more parameters increasing the size of a model and decreasing its speed. Thus, in applications were only one or a few classes of objects are to be detected, the detector should be trained to only detect those, to save time. Furthermore, since the last step of detection typically involves using Non Maximum Suppression (NMS) to collapse overlapping bounding boxes, which depends on the number of detected boxes, this approach can lead to significant speed gains when applicable.

Although similar in nature to SSD, YOLO [11] is a widely used object detector, whose popularity may be attributed to its simplicity stemming from its ability to detect multiple objects with a single forward pass of an image, in combination with its speed which surpasses that of SSD. YOLO relies on a custom architecture for its feature extraction step, which is pretrained on ImageNet and publicly available. Its fully-convolutional architecture allows the network to be trained and deployed at any resolution, although odd multiples of 32 — the network's total subsampling factor — are preferred.

A smaller version, named Tiny YOLO, is also available and performs object detection based on the same principles. Using half the convolutional layers, Tiny YOLO sacrifices precision for the sake of speed. The tiny version is also fully convolutional and subsamples the input image by a factor of 32. Thus, for an input of 416×416, a 13×13 final heatmap is produced, whose depth depends on the number of classes to be detected and the number of precomputed anchors. Note that only this final output depends on the number of classes. However, NMS is once again used as a post-processing step, and is affected by the total number of detected boxes which naturally grows as the number of classes to detect increases.

4. EXPERIMENTAL STUDY

In the following subsections, we discuss our experimental protocol and report results, offering a speed-accuracy trade-off analysis for the aforementioned detectors. We provide results for optimizations made during the training phase and post-training optimizations as well as combinations of the two. All time-dependent measurements were made on an NVIDIA Jetson TX2 platform.

4.1. During-training optimizations

For SSD, we use its Tensorflow implementation for its versatility with regards to the choice of feature extraction network and values of hyperparameters. We study the case of face detection, a well established visual analysis task, with numerous applications on mobile devices. Tensorflow's Object Detection API allows users to easily perform transfer learning from pretrained models to other domains. We focus on the input size and more specifically on inputs smaller than 300×300, the default for this detector. For this purpose, we utilize the two feature extraction architectures found in [6] to be the fastest, namely MobileNets and Inception V2. We exploit the publicly available detectors pretrained on the COCO dataset, which contains the general class 'human', and finetune the detector for the purpose of face detection, using the FDDB dataset [14]. For evaluation purposes, we follow the default protocol for this dataset and perform 10-fold cross-validation on the predefined folds.

For both feature extractors, we perform experiments at five input sizes: 300×300, the baseline, 224×224, 192×192, 160×160 and finally 128×128. To measure the detection speed in a fair yet realistic manner, we run all models on the same video frames and report the overall time divided by the number of frames (FPS).

The FPS and corresponding recall averaged over the 10 folds for the Inception V2 version are reported in Table 1, while Figure 1 depicts the true positive rate versus the number of false positive detections. At the baseline input size of 300×300, the detector achieves a maximum recall of 85.2 with fewer than 100 false positive detections on the entirety of the FDDB dataset. At about half the resolution (160×160), a 9% drop in recall is observed at double the FPS. However, even the smallest input size evaluated, while maintaining an acceptable recall, leads to a maximum of about 17 FPS, somewhat far from real-time performance.

Input Size	FPS	Recall
300	8.5	85.2
224	12.38	84.0
192	13.79	81.1
160	15.65	76.6
128	17.15	73.6

Table 1. Frames per second and mean Average Precision for various input sizes for the SSD with Inception V2 feature extractor model.

As for the MobileNets models, the respective FPS and recall are reported in Table 2, whereas the true positive rate to false positives curves are depicted in Figure 2. At the baseline input size, this version achieves a 84.1 recall at about 11 FPS, very close to the performance of the Inception V2 version at 224×224 input size. At 224×224, a 2 FPS gain is somewhat insignificant given the decrease in recall. However,

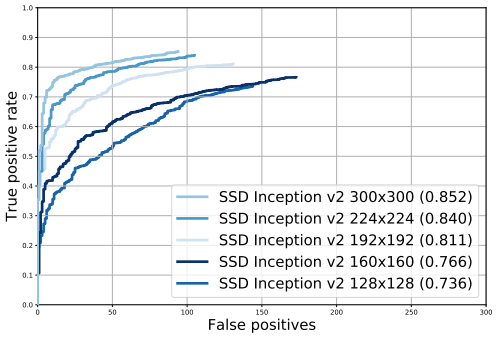

Fig. 1. True positive rate versus false positives for the SSD with Inception V2 detector on the FDDB dataset.

at 192×192 the gain in speed to recall decrease is very balanced, allowing the detector to run at 18 FPS, more than any of the evaluated Inception V2 models, while attaining a recall of 80.0. Finally, at an extremely small 128×128 input size, the detector achieves about 24 FPS, that is near real-time performance, at the cost of a significant loss in recall. Depending on the application, however, a recall of 71.4 may still be acceptable.

Input Size	FPS	Recall
300	11.64	84.1
224	13.47	81.1
192	18.20	80.0
160	21.04	77.4
128	23.85	71.4

Table 2. Frames per second and mean Average Precision for various input sizes for the SSD with MobileNets feature extractor model.

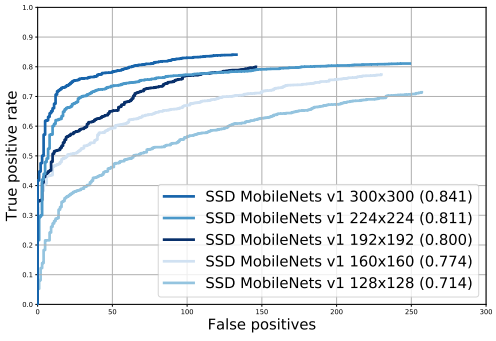

Fig. 2. True positive rate versus false positives for the SSD with MobileNets detector on the FDDB dataset.

For the YOLO detector, we use its original and tiny version and make some modifications. First, the Leaky ReLU activations are replaced with ReLUs in both versions. In the YOLO model, the reorganizing layer which reshapes the input of a higher-level layer to match that of a lower-level layer

Input Size	FPS	mAP
608	2.9	71.26
544	3.2	73.64
480	5.4	74.50
416	6.4	73.38
352	7.8	71.33
320	8.5	70.02

Table 3. Frames per second and mean Average Precision for various input sizes for the modified YOLO model.

Input Size	FPS	mAP
608	6.5	51.28
544	8.2	52.93
480	13.4	55.00
416	16.5	56.28
352	20	55.05
320	23	53.81

Table 4. Frames per second and mean Average Precision for various input sizes for the modified Tiny YOLO model.

is replaced with a max pooling operation. In the Tiny YOLO model, a max pooling layer of size 2 and stride 1 right before the final detection layers is removed, leaving the size of the final layer unchanged. These modifications serve the purpose of allowing straightforward post-training optimizations on the models using TensorRT. We found that for both models, these changes lead to about 1% absolute loss in mean Average Precision (mAP), measured on the VOC dataset. We train both models using multi-scale training, which chooses a random input size from 320 to 608 (multiples of 32) every ten batches, starting from the respective networks pretrained on ImageNet. The detector may then be evaluated at each of these scales. We report the mAP and FPS for various input sizes for YOLO in Table 3 and for Tiny YOLO in Table 4.

4.2. Post-training optimizations

For post-training inference optimization we use NVIDIA's TensorRT library, which offers capabilities such as layer/tensor fusion and half-point (FP16) or INT8 precision computations among others. We begin by converting the YOLO and Tiny YOLO models to Caffe [15] model definitions, and the modifications we made allow for their straightforward benchmarking using TensorRT's tools.

The speed up of TensorRT, and especially that of FP16 arithmetic, is significant and could allow the deployment of the original YOLO model even at large input sizes, provided the memory constraints of the device allow it.

Input Size	No TensorRT	TensorRT	FP16
608	241.5	128.8	69.3
544	214.4	121.2	64.3
480	155.4	62.3	35.7
416	155.3	56.5	32.5
352	111.0	45.0	24.3
320	103.0	40.4	22.8

Table 5. Forward times (in ms) for the modified YOLO model ported in Caffe with and without TensorRT and with FP16 precision arithmetic.

Input Size	No TensorRT	TensorRT	FP16
608	76.5	37.5	22.1
544	68.4	34.8	20.5
480	50.1	17.2	11.7
416	49.9	15.7	10.3
352	37.1	13.0	7.9
320	34.0	11.7	7.2

Table 6. Forward times (in ms) for the modified Tiny YOLO model ported in Caffe with and without TensorRT and with FP16 precision arithmetic.

5. CONCLUSIONS

In this paper, we have discussed practical difficulties convening the deployment of large Deep Learning models on mobile devices in the presence of low computational capabilities and memory constraints. We have investigated the factors affecting the speed-accuracy trade-offs in object detection using convolutional neural networks and provided pragmatic solutions for real-time or near real-time object detection in mobile devices, using UAVs as a case study. We have shown, through our experiments, that despite the heavy memory and computational requirements of such models, post-training and during-training optimization methods can be successfully deployed to significantly boost detection speed while maintaining performance above accepted thresholds.

6. ACKNOWLEDGMENTS

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE). This publication reflects the authors views only. The European Commission is not responsible for any use that may be made of the information it contains.

7. REFERENCES

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej

Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[2] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," .

[3] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[4] Francois Chollet, "Xception: Deep learning with depthwise separable convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

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

[6] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al., "Speed/accuracy trade-offs for modern convolutional object detectors," in *IEEE CVPR*, 2017.

[7] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015, pp. 448–456.

[8] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[11] Joseph Redmon and Ali Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, vol. 1612, 2016.

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[13] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[14] Vidit Jain and Erik Learned-Miller, "Fddb: A benchmark for face detection in unconstrained settings," *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009*, vol. 2, no. 7, pp. 8, 2010.

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.