Face detection with YOLO on edge.

ALI-GOMBE, A., ELYAN, E., MORENO-GARCIA, C.F. and ZWIEGELAAR, J.

2021

Face Detection with YOLO on Edge *

Adamu Ali-Gombe[1], Eyad Elyan[1], Carlos Francisco Moreno-García[1], and Johan Zwiegelaar[2]

[1] Robert Gordon University Aberdeen, United Kingdom
[2] Mintra Group Oslo, Norway
https://www.rgu.ac.uk, https://www.mintragroup.com
{a.ali-gombe,e.elyan,c.moreno-garcia}@rgu.ac.uk,
johan.zwiegelaar@mintragroup.com

Abstract. Significant progress has been achieved in objects detection applications such as Face Detection. This mainly due to the latest development in deep learning-based approaches and especially in the computer vision domain. However, deploying deep-learning methods require huge computational power such as graphical processing units. These computational requirements make using such methods unsuitable for deployment on platforms with limited resources, such as edge devices. In this paper, we present an experimental framework to reduce the model's size systematically, aiming at obtaining a small-size model suitable for deployment in a resource-limited environment. This was achieved by systematic layer removal and filter resizing. Extensive experiments were carried out using the "You Only Look Once" model (YOLO v3-tiny). For evaluation purposes, we used two public datasets to assess the impact of the model's size reduction on a common computer vision task such as face detection. Results show clearly that, a significant reduction in the model's size, has a very marginal impact on the overall model's performance. These results open new directions towards further investigation and research to accelerate the use of deep learning models on edge-devices.

Keywords: Deep Learning · YOLO · Face Detection.

1 Introduction

Face detection is a common task across many computer vision applications such as face recognition systems [2], border control systems [16] and others. Early face detector systems employed features engineering techniques such as SIFT [13] and HOG [5], which are fast and lightweight with low computation requirements. However, these models were limited and do not generalise well enough when compared to more recent deep learning-based approaches. Although deep learning models are state-of-the-art as seen in many domains [14] [6], deploying them in the real world is resource-intensive. Most of these models require Graphical Processing Units (GPU) and a large memory to achieve performance goals.

* Supported by InnovateUK, Mintra Group and Robert Gordon University.

In return, this hinders their use in a resource constraint environment, such as a CPU only system. Therefore, there is an evident need to account for other computational metrics, such as memory footprint, parameters, operation count, inference time and power consumption [3].

Over-parametrisation is a major concern in deploying deep models [3]. This leads to a significant increase in model size and inference time, thereby making the algorithms less viable in constrained environments. Many approaches to reducing the storage size and parameters of models have been proposed in the literature. These include quantisation, pruning parameters/layers, neural architectural search, architecture parallelisation and others [1], [4]. While these approaches are effective, they are a trade-off between size and performance. Moreover, it is also difficult to ascertain the best approach due to varying benchmarks and evaluation criteria. Again, these methods and some architectural choices can produce the most compact model, but may not necessarily produce the best accuracy.

In this paper, we present a reconfiguration of the You Only Look Once (YOLO) v3-tiny architecture to achieve reduced model storage sizes and parameters. We target the model size because the amount of computations directly affects the inference time, which is crucial in real-world application [3]. Different light models were also evaluated, with each model trading a slight performance for size. This was achieved through systematic filters resizing and trimming large layers that contribute to large parameters. Filter resizing was inspired by SqueezeNet [11], and trimming was based on experimentation. While similar researches such as [19], [10] and [7] were conducted on general object detection, our experiments are focused on face detection task. Furthermore, we compare different lighter model versions to a YOLO v3-tiny baseline on two different datasets and using several metrics. Our experiments showed that we are able to make the model 16 times smaller and still maintain comparable performances.

The rest of the paper is organised as follows. In Section 2, related literature is reviewed and discussed. Section 3 presents the model used in this work. Section 4 discusses in details experimental set-up and the datasets used. Findings are discussed in section 5. Finally, we conclude and suggest future directions in Section 6.

2 Literature

A common approach to size reduction is pruning, which focuses on removing redundant connections after training. This technique finds a sub-network within the model, with performances similar to the large model. Weight removing is based on magnitude [8] or even a learning criterion [17]. Salvi et al. [1] provided an in-depth analysis among these approaches, and highlighted that pruning produces sparse matrices which may not necessarily be optimised. Efficient convolution and activations also reduce parameter sizes. For instance, SqueezeeNet [11] achieved AlexNet level performance with 50 times less parameters and a model size of about 0.5 MB, by replacing 3×3 with 1×1 filters in convolution layers.

Other approaches use network distillation in an ensemble with a large teacher to guide a smaller student using a hint layer [9].

YOLO v3-tiny model was proposed as a lighter version of the original YOLO v3 model [15]. This was built to target embedded systems and scenarios with computation and memory constraints. However, at 33 MB, the model may not scale well for restrictive systems like mobile devices and client-web applications. This has lead to many research extension to the original and tiny YOLO frameworks. These extensions target either speed in terms of Frames Per Second (FPS), storage size, or the number of operations.

YOLO Nano [19] extended the original YOLO v3 with a more compact architecture, using a human-machine collaborative design strategy. This model employs customisable module-level microarchitecture to a specific task. The network contains stacked convolution layers with short-cut connections between some layers and outputs bounding box, class and confidence in the final layer. An interesting extension was the use of generative synthesis to determine the optimal macro and microarchitecture designs from the human-specific requirements and constraints. This set up was found to reduce the model size($\approx$ 4.0 MB) and computational cost (4.57 BFLOP) without compromising performance.

YOLO LITE [10] focused on speed only in terms of how much FPS could the model achieve in a constrained environment (ignoring both size and weights). By speeding up, this alternative prunes some layers and, along with batch normalisation, significantly improves speeds up to 21 FPS. From YOLO v2-tiny with 9 convolution layers, 3181 filter and 6.97 BFLOPS, the LITE version had only 7 layers with 749 filters and 482 MFLOP in the best model version.

Tinier-YOLO [7] was presented as an optimised YOLO v3-tiny with the aim of achieving a smaller, faster and more accurate model in a constrained environment. This was achieved by adding a fire module to the SqueezeNet backbone for reduced parameters without losing significant accuracy. That is, tinier-YOLO keeps the first five layers of v3-tiny but replaces the middle and final part. After that, the authors presented a model size of 8.9 MB compared with 25.1 FPS on Jetson TX1.

Fast-Yolo [18] targeted real-time object detection in embedded devices. The new framework introduced motion-adaptive inference to improve the inference frequency. In this setup, not all of the frame sequences are processed. Thus, a motion probability map is produced by stacking frames with a reference frame using 1×1 convolution. This is used to determine if the frame is unique enough before a new inference is generated. It also leads to reduced power consumption at the expense of a slower running time per frame. The authors also applied an evolutionary deep intelligence framework on YOLO v2 by synthesising the Fast-YOLO architecture. Fast-YOLO is 3.3 times faster than YOLO v2 and 2.8 times more compact.

3 Method

YOLO is an anchor-based detector that uses a set of pre-computed bounding boxes to refine detection. YOLO is one of the most widely used detectors and a fast unified detector that outputs bounding box coordinates without the need for region proposals and proposal networks. Detection starts by dividing the image into grids. These grids could be on the scale of 52×52, 26×26 and 13×13 in the original YOLO architecture. Each grid cell outputs four predictions; the confidence (class probability or *objectness*), the centre of the object (x, y), the height and the width of the bounding box (w, h). Figure 1 summarises the YOLO prediction pipeline. YOLO v3-tiny is a reduced version that targets improved speed and reduced size at the expense of model accuracy. The YOLO v3-tiny has nine convolution layers and uses 26×26 and 13×13 grids to perform detection. The YOLO loss is calculated using Equation 1. YOLO uses a convolution network backbone, that is, a set of convolution layers with a fully connected layer for prediction.

$$
\begin{aligned}
L = \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} l_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2] \\
+\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} l_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w_i}})^2 + (\sqrt{h_i} - \sqrt{\hat{h_i}})^2] \\
+ \sum_{i=0}^{s^2} \sum_{j=0}^{B} l_{ij}^{obj} (C_i - \hat{C_i})^2 \\
+\lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} l_{ij}^{noobj} (C_i - \hat{C_i})^2 \\
+ \sum_{i=0}^{s^2} l_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p_i}(c))^2
\end{aligned}
\tag{1}
$$

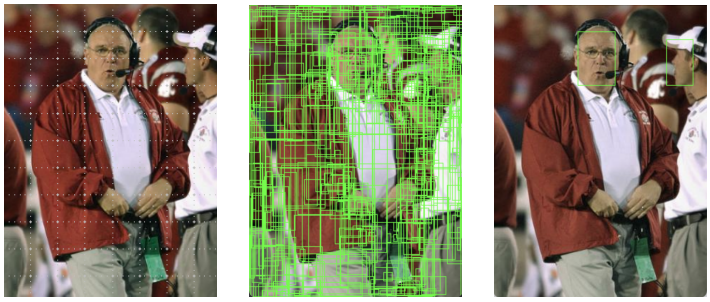
where l_i^{obj} denotes the presence of object in cell i, l_{ij}^{obj} the jth bounding box in cell i, C is a set of classes with $p(c)$ probability, B is the set of bounding boxes, S^2 is the grids and x, y, w, h are coordinates.

4 Experiment

All experiments were carried out using the original YOLO repository from DarkNet, keeping the same hyper-parameter settings.

4.1 Dataset

The first dataset considered is WiderFaces dataset [20], which is a popular public face detection benchmark. The dataset contains face samples across 61 scenarios



(a) (b) (c)

Fig. 1: YOLO detection pipeline. The image on the left shows the input image divided into grids, the second figure is the image after prediction and the rightmost figure shows the image and the location face location after post-processing.

under varying conditions such as high occlusion, pose angle and illumination. WiderFaces contains 32,203 images, with 393,703 labelled faces. The dataset is split into a train, validation and a test set (40-10-50 split). For each experiment, the training set was used to train detectors, and the validation set was kept as a hold out set for evaluation.

The second dataset is the Face Detection Data Set and Benchmark (FDDB) [12]. FDDB contains 2845 images with a total of 5171 annotated faces. The dataset is created from faces in the wild, and images are less complex compared to WiderFaces. FDDB images are a mixture of a front-facing facial pose, different view angles, multiple faces per images with occasional half and tiny faces. All images in this dataset were used to evaluate models trained on WiderFaces.

4.2 Models

The base model we used is the YOLO v3-tiny, which we refer to as 33m-YOLO in these experiments. The model can be divided into two parts. The first part contains 10 convolution layers. The first six layers of this part are succeeded with 2×2 max-pooling layer. All convolution layers have 3×3 filters except for the eighth and the tenth layer. This part ends with the first output layer of 13×13 grid predictions. The second part contains a 1×1 convolution, followed by a $2\times$ up-sample layer, a convolution layer with 3×3 filters and a convolution layer with a 1×1 filter. This section ends with a final 26×26 output layer. The 33m-YOLO is 34.8MB in size and has $8,676,244$ parameters with 5.448 BFLOPS.

The first model was generated by removing the seventh, eighth and ninth convolution layer along with the last max-pooling from the first part. These are the layers with the largest number of filters and parameters. We refer to this model 10m-YOLO in our experiments, with a size of 10.1 MB, $2,508,692$ parameters and 3.365 BFLOPS. The second model was build from the first model by introducing a 1×1 convolution in the sixth layer. The number of filters in the

sixth layer was also changed from 512 to 256. We refer to this model 5m-YOLO with a size of 5.7 MB, $1,388,948$ parameters 2.987 BLOPS. The final model in this experiment is the 2m-YOLO. This was generated by using 1×1 filters in all convolution in the second section. 2m-YOLO has a size of 2.7 MB, $602,516$ parameters with 1.924 BFLOPS.

5 Results and Discussion

Figures 2 shows some sample detections from the trained models, and Table 1 shows the performance of the models on different detection metrics. Results were obtained by using the official YOLO evaluation script with 0.5 detection threshold.

Table 1: Models performances on WiderFaces dataset.

dataset	model	TP	FP	FN	avg. IoU	mAP	precion	recall	F1-score	size(MB)
WiderFaces	33m-yolo	14196	13043	25509	0.38	0.34	0.52	0.36	0.42	34.8
	10m-yolo	13660	14628	26045	0.35	0.31	0.48	0.34	0.40	10.1
	5m-yolo	12650	10499	27055	0.40	0.31	0.55	0.32	0.40	5.7
	2m-yolo	13057	12822	26648	0.36	0.30	0.50	0.33	0.40	2.4
FDDB	33m-yolo	2785	2265	2386	0.38	0.44	0.55	0.54	0.54	34.8
	10m-yolo	2566	2173	2605	0.38	0.36	0.54	0.50	0.52	10.1
	5m-yolo	2219	2025	2952	0.35	0.27	0.52	0.43	0.47	5.7
	2m-yolo	2147	2143	3034	0.33	0.30	0.50	0.42	0.45	2.4

All three model versions, 10m-YOLO, 5m-YOLO and 2m-YOLO have sizes significantly smaller than the original 33m-YOLO model. It is worth mentioning that this was at the expense of a small loss in performance. However, the trade-off in performance is not always symmetric to reduction in model size. For instance, on WiderFaces, the 2m-YOLO is about 16 times smaller than 33m-YOLO, but the mean Average Precision (mAP) is only 4 points off. This is arguably manageable for none critical applications. Again, this was achieved with lower false positives across all models than the original.

The results on FDDB are better, with a noticeably higher F1-score across all models. This is not surprising, given that this dataset is not as challenging as WiderFaces. However, there is a distinctly high false positively in this experiment. This issue was investigated, and it was found that it had more to do with the nature of the FDDB dataset. More precisely, the models were detecting *half faces* correctly. These faces were not part of the ground-truth. For instance, the sample in Figure 2 has four ground-truth faces, and it can be seen that the models detected five or more faces. Therefore, these additional faces were counted as false positives by the evaluation script, since no annotations are provided for these faces. Nonetheless, we consider that these detections are desired, as these

(a) 33m-YOLO (b) 10m-YOLO

(c) 5m-YOLO (d) 2m-YOLO

Fig. 2: Samples from FDDB detection.

characteristics are required in an uncontrolled environment where faces may be occluded due to a number of different reasons.

In terms of speed, we ran the models against a pre-recorded one minute video on core i5 MacBook pro-2019. The video was recorded at $720p \times 1080p$ resolution with a 30 frame rate camera. Unsurprisingly, there was also an improvement in the speed of detection. 2m-YOLO was fastest at 25.9 FPS, followed by 5m-YOLO at 23.5 FPS. Meanwhile, 10m-YOLO was running at 23.3 FPS and the lowest was 33m-YOLO at 19.3 FPS.

Removing layers typically reduces model size while affecting performance significantly. Using 1×1 convolution filters works well, but there is an obvious hard limit without a rigorous architectural search. In our experiments, we found

that these two approaches can complement each other to give a reduced model with minimal performance loss possible.

6 Conclusion

In this paper, we propose a reduced YOLO v3-tiny model for face detection in resource-constrained environments, such as CPU only systems or edge devices. Extensive experiments were carried out using public benchmark datasets, namely WiderFaces and FDDB. Performance was evaluated on different face detection metrics. The results showed models produced comparable performances with a significant reduction in model sizes. These also indicated that the proposed approach is effective in reducing the model size with improve speed, but at the expense of some performance loss. In the future, we intend to combine these approaches with other techniques, such as quantisation and parallelisation, to further reduce the model size. There is also the need for an extensive architectural search to regain some of the performances lost.

References

1. de Aguiar Salvi, A., Barros, R.C.: An experimental analysis of model compression techniques for object detection. In: Anais do VIII Symposium on Knowledge Discovery, Mining and Learning. pp. 49–56. SBC (2020)
2. Ali-Gombe, A., Elyan, E., Zwiegelaar, J.: Towards a reliable face recognition system. In: International Conference on Engineering Applications of Neural Networks. pp. 304–316. Springer (2020)
3. Canziani, A., Paszke, A., Culurciello, E.: An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678 (2016)
4. Cortés Gallardo Medina, E., Velazquez Espitia, V., Chípuli Silva, D., Fernández Ruiz de las Cuevas, S., Palacios Hirata, M., Zhu Chen, A., González González, J., Bustamante-Bello, R., Moreno-García, C.F.: Object Detection, Distributed Cloud Computing and Parallelization Techniques for Autonomous Driving Systems. Applied Sciences **11**(7), 2925 (2021). https://doi.org/https://doi.org/10.3390/app11072925, https://www.mdpi.com/2076-3417/11/7/2925
5. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). vol. 1, pp. 886–893. Ieee (2005)
6. Elyan, E., Jamieson, L., Ali-Gombe, A.: Deep learning for symbols detection and classification in engineering drawings. Neural Networks **129**, 91–102 (2020). https://doi.org/10.1016/j.neunet.2020.05.025, https://doi.org/10.1016/j.neunet.2020.05.025
7. Fang, W., Wang, L., Ren, P.: Tinier-yolo: A real-time object detection method for constrained environments. IEEE Access **8**, 1935–1944 (2019)
8. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)
9. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)

10. Huang, R., Pedoeem, J., Chen, C.: Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 2503–2510. IEEE (2018)
11. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)
12. Jain, V., Learned-Miller, E.: Fddb: A benchmark for face detection in unconstrained settings. Tech. Rep. UM-CS-2010-009, University of Massachusetts, Amherst (2010)
13. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International journal of computer vision **60**(2), 91–110 (2004)
14. Moreno-García, C.F., Elyan, E., Jayne, C.: New trends on digitisation of complex engineering drawings. Neural Computing and Applications **31**(6), 1695–1712 (2019). https://doi.org/10.1007/s00521-018-3583-1
15. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement (2018)
16. del Rio, J.S., Moctezuma, D., Conde, C., de Diego, I.M., Cabello, E.: Automated border control e-gates and facial recognition systems. computers & security (2016)
17. Savarese, P., Silva, H., Maire, M.: Winning the lottery with continuous sparsification. arXiv preprint arXiv:1912.04427 (2019)
18. Shafiee, M.J., Chywl, B., Li, F., Wong, A.: Fast yolo: A fast you only look once system for real-time embedded object detection in video. arXiv preprint arXiv:1709.05943 (2017)
19. Wong, A., Famuori, M., Shafiee, M.J., Li, F., Chwyl, B., Chung, J.: Yolo nano: A highly compact you only look once convolutional neural network for object detection. arXiv preprint arXiv:1910.01271 (2019)
20. Yang, S., Luo, P., Loy, C.C., Tang, X.: Wider face: A face detection benchmark. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5525–5533 (2016)