Baltic J. Modern Computing, Vol. 9 (2021), No. 3, pp. 333–344
https://doi.org/10.22364/bjmc.2021.9.3.07

Investigation of YOLOv5 Efficiency in iPhone Supported Systems

Daniel DLUŽNEVSKIJ[1], Pavel STEFANOVIČ[2], Simona RAMANAUSKAITĖ[3]

[1] Department of Electronic Systems, Vilnius Gediminas Technical University, Naugarduko g. 41, LT-03227 Vilnius, Lithuania
[2] Department of Information Systems, Vilnius Gediminas Technical University, Saulėtekio al. 11, LT-10223 Vilnius, Lithuania
[3] Department of Information Technology, Vilnius Gediminas Technical University, Saulėtekio al. 11, LT-10223 Vilnius, Lithuania

daniel.dluznevskij@stud.vilniustech.lt, pavel.stefanovic@vilniustech.lt
simona.ramanauskaite@vilniustech.lt

Abstract. Object detection gaining popularity and is more used on mobile devices for real-time video automated analysis. In this paper, the efficiency of the newly released YOLOv5 object detection model has been investigated. Experimental research has been performed to find out the efficiency of YOLOv5 using a mobile device with real-time object detection tasks. For this reason, four YOLOv5 model sizes have been used: small, medium, large, and extra-large. The experiments have been performed with a well-known COCO dataset. The original dataset consists of a huge number of images, so the dataset has been reduced to fit the mobile device requirements. The experimental investigation results have shown, that reducing the COCO dataset has no significant influence on the model accuracy, but the model performance is highly influenced by the hardware architecture and system where the model is used. Apple Network Engine usage might significantly increase the YOLOv5 model performance in comparison to CPU usage.

Keywords: object detection, COCO dataset, real-time detection, iPhone systems, mobile device, YOLOv5.

1 Introduction

These days, there are more mobile devices in the world than a decade ago. Mobile devices are used in various day-to-day activities. The machine learning and deep learning model solutions are realized in the newest devices, which helps us to secure devices, and also, in many different personalized options where the device adapts to our behavior or habits. Over the last decade, scientists achieved exceptional results in the computer vision field using convolutional neural networks (Rghioui et al., 2017). In the last couple of years, many interesting ideas came to our life, such as residual neural networks (Chang et al., 2018), End-to-End Object Detection using Transformers, MuZero (Schrittwieser et al., 2020), AlphaFold, GPT-3 (Floridi et al., 2020), and many more. Mobile devices rapidly progress and naturally will replace conventional computers, resulting it will be necessary of building small, and at the same time efficient models. Nowadays, the main computer vision task is image classification, object segmentation or detection, and object localization. In this paper, object detection has been analyzed. Object detection is a model that locates objects, draws bounding boxes around them, and assigning the classes to the detected objects in the image. For example, the neural network takes an image and returns one or many drawn bounding boxes around detected objects. There is a lot of various type of object detection algorithms (Zhao et al., 2019) which is used in different tasks. For example, traffic light detection, which is used by self-driving cars (Kulkarni et al., 2018); a person recognition used by police (Wilkowski et al., 2019), etc. In our experimental investigation, the newest YOLOv5 model has been used.

While artificial intelligence solutions gain their popularity, their usage is closely related to hardware capabilities to apply the model within the given time. Mobile devices have limited resources, therefore real-time usage of complex, cloud computing dedicated solutions might cause undesired delay. This issue might be very important in mobile apps, where object detection is executed from a live video stream and requires minimization of response time. Therefore it is important to prepare a mobile device-oriented object detection model and investigate its performance in a certain type of mobile device. Therefore the paper aims to investigate YOLOv5 based object detection model performance in iPhone mobile devices. The structure of the paper is as follows: In Section 2, the related works are reviewed. In Section 3 the dataset and its preparation are presented, as well as the experimental investigation and obtained results. Section 4 concludes the paper.

2 Related works

2.1 Evolution of object detection algorithms

A region-based convolutional neural network family consists of R-CNN, Fast R-CNN, Faster R-CNN, and Mask R-CNN. The first time the R-CNN (Girshick et al., 2014) was presented was in 2014. The main issue of the R-CNN algorithm is the slow performance, so to improve the algorithm and make it faster, in the same year the Fast R-CNN has been introduced by Girshick (Girshick, 2015). This proposal improved the model's training procedure by consolidating three independent models into a single framework, which speeds up the algorithm. However, speed is not the only factor of a modern solution, so the Fast R-CNN architecture was further improved to reach a higher computational speed of model training process and object detection accuracy. Ren et al., improved it by integrating the region proposal algorithm into the CNN model and named it Faster R-CNN. The Mask R-CNN extends Faster R-CNN to pixel-level image segmentation. The goal was to separate the classification and pixel-level mask prediction tasks (He et al., 2017).

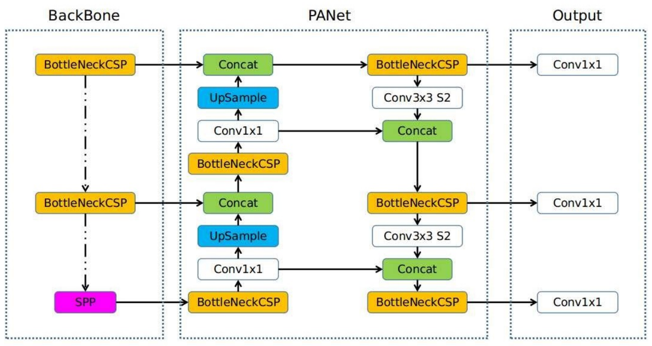

Fig. 1. Overview of YOLOv5 architecture (WEB (a), 2021

Another family of object detection models is called You Only Look Once (YOLO). The YOLO models are faster than the R-CNN family models, so usually are used in various real-time tasks, for example, object detection in video records. The YOLO model was first time presented by Redmon et al. in 2015. R-CNN's key difference is that YOLO was the first to build a fast real-time object detector and involve a single neural network trained end-to-end. It takes an image as an input and predicts bounding boxes and class labels for each bounding box. This model offers lower prediction accuracy but can operate at 45 frames per second. In 2017 Redmon and Farhadi introduced an improved version and named it YOLOv2 (sometimes called YOLO9000) capable of predicting 9000 object classes (Redmon et al., 2017). Later the same authors Redmon and Farhadi proposed further improvements in 2018 (Redmon et al., 2018). The authors proposed minor improvements, including a deeper feature detector network and minor changes to the representational layer, but it slightly improved the performance of the YOLOv2 algorithm. The latest method YOLOv4 introduced by Bochkovskiy et al. last year (Bochkovskiy et al., 2020). The experimental investigation has been performed to analyze the various combination of usage backbones (CSPResNext50, CSPDarknet53, EfficientNet-B3, etc.), and object detection models (YOLOv4, YOLOv3, SSD, RFBnet, etc.). The research showed, that YOLOv4 gives better speed performance and accuracy compared to the other combinations.

Nowadays the newest version of YOLO is the fifth version developed by the company Ultralytics. The difference of the new YOLOv5 compared to other same type models is that YOLOv5 uses a CrossStage Partial Network (CSPNet) (Liu et al., 2018) as the model backbone and Path Aggregation Network (PANet) (Wang et al., 2020) as the neck for feature aggregation. The model architecture is presented in Fig. 1. These new improvements give better feature extraction and a significant boost in the mAP score. Because of the reason that YOLOv5 is a new model, there is no much research made, where this model efficiency has been evaluated.

In 2017, Howard et al., developed a class of efficient models called MobileNet, which mainly focuses on mobile and embedded vision applications. Their main focus was to increase the model's efficiency of the network by decreasing the number of parameters by not compromising on performance. Recently, a researchers team from Facebook AI proposed a new object detection method – object detection using the Transformers. Transformers are a deep learning architecture that has gained popularity in recent years. Transformers based on a mechanism called attention, enabling artificial intelligence models to focus on certain parts of their input selectively and thus reason more effectively. Transformers are widely applied to sequential data problems: natural language processing tasks such as language modeling and machine translation. The Transformers were introduced by Vaswani et al. to perform the machine translation. The Transformer's core building block is an attention mechanism that enables it to have a long-term memory. The model's architecture is encoding and decoding blocks. An encoding block takes an input and yields output probabilities correspondingly. In DETR, the encoder takes image features, and the decoder outputs class and linked bounding box to that class. In the paper, Carion et al., authors use backbone CNN, the ResNet CNN, and smaller feature versions of the image. Before combining image features and positional encoding, the image features must be represented in a specific format (converted from a matrix representation into a single sequence). The transformers are naturally a sequence processing unit that takes a sequence of vectors. The transformer encoder then transforms this sequence into an equally long sequence of features. The transformed sequence has attention layers and can attend from each position to each position in a one-shot manner. As the Transformer transforms the representation layer up the transformer layers at each step, it can aggregate information from everywhere in the sequence to anywhere else. Therefore, this approach is compelling if a sequence requires a global connection across the sequence. The bounding boxes can be quite large, hence requiring long-range dependencies. If one part of the bounding box needs to communicate with other parts of the image, utilizing long-range dependencies of the transformer architecture becomes a convenient way of dealing with this problem. The transformer encoders output goes as a side input into the transformer decoder (conditioning information). The decoder does the same actions: it takes a sequence and outputs a sequence in one shot. The input sequence consists of object queries, for example, four random pre-trained vectors. The output sequence yields a sequence previously used in the transformer decoder, and additionally the conditioning information. The decoder's output goes through a classifier that outputs the class label and bounding box. Each of the input queries ends up being one of the bounding boxes either defining an object or stating that there is no object.

2.2 Object detection solutions evaluation metrics

Artificial intelligence and specifically object detection solutions might be evaluated based on different metrics. While model training and interference time is the main performance metric, the accuracy in object detection solutions and general classification solutions are different. Object detection must take into account not only whether the object is suitably classified, but does it found the correct bounds of the object. Therefore the main accuracy metric in object detection is the mean average precision (mAP) of the area under the precision-recall curve above.

In the object detection task, each detected area has a score associated (object likelihood in the area). Based on the predictions a precision-recall curve (PR curve) is computed for each class by varying the score threshold. The average precision (AP) is the area under the PR curve. First, the AP is computed for each class. As there are multiple classes average for all the classes is calculated to obtain the mAP.

To get the mAP value, the intersection over union (IoU) is used too. IoU measures the overlap between two areas, how much the predicted boundary overlaps with the dataset labeled area. For object detection accuracy measurement IoU threshold (for example 0.5) might be defined to analyze mAP, whether the match between predicted and labeled areas is greater than the defined threshold. This is used to evaluate true positives. Therefore while using mAP some IoU values are assigned as well (for example $mAP_{0.5}$ refers to the mAP where IoU is equal to 0.5 or above). In the experimental investigation described in Section 3 the time has been evaluated too. It indicates how fast object detection has been performed in various experimental cases.

2.3 The review of object detection methods efficiency

As was mention before, the object detection methods are often analyzed in different applications, such as measuring the efficiency, usability, methods suitability for specific tasks, etc. In the object detection tasks, the CNN methods perform a lot of computation and highly is using computer storage, so GPU started to be used for real-time object detection. By solving one problem, then other challenges arise such as the high power consumption of GPU, it is difficult to adopt GPU in mobile applications for example automatic driving. Solutions to solve such kind of problems was made by Yu et al. The authors' proposed techniques to lower the power consumption of object detection on mobile GPU or FPGA achieved the best result with mAP/Energy on mobile GPU platforms. The deep learning methods became one of the most widely used in various fields such as natural language processing, image classification, machine learning, and object detection over the past decade. The analysis of object detection algorithms Region-based Convolutional Neural Networks (RCNN), Faster-RCNN, Single Shot Detector, and You Only Look Once has been performed by Chandan et al. The analysis showed that Faster-RCNN and SSD have better accuracy results, while the YOLO algorithm performs better when speed is given preference over accuracy. Also, the various combination of methods was analyzed by measuring their performance.

In the paper Wang et al., the object detection methods were analyzed that can be used in real-time object detection by mobile devices. The authors propose an efficient architecture named PeleeNet, which base is conventional convolution. The experimental investigation was performed using the ILSVRC 2012, VOC 2007, and COCO datasets, where the PeleeNet achieves higher accuracy and over 1.8 times faster speed than MobileNet and MobileNetV2. Moreover, the proposed architecture is only 66% of the model size of the original MobileNet. The comparative analysis has been performed using various models such as SSD, YOLOv2, YOLOv3, MobileNetv2, but in all cases, by authors proposed PeleeNet architecture obtained the higher accuracy compared to the other models. The other researches performed by Tan et al., focuses on the study of neural network architecture choices and proposes several key optimizations to improve the efficiency of object detection. The authors proposed to combinate a weighted bi-directional feature pyramid network, which allows easy and fast multiscale feature fusion, and a compound scaling method that uniformly scales the resolution, depth, and width for all backbone, feature network, and box/class prediction networks at the same time. This architecture is named the EfficientDet, which consistently achieves better accuracy and efficiency than the prior art.

One more network architecture of object detection named PVANET has been proposed by Hong et al. The network architecture allows lightweight feature extraction, which achieves real-time object detection performance without losing accuracy compared to the other state-of-the-art systems. The base of proposed architecture construction is the usage of fewer channels and more layers. The obtained results showed that the object detection model trained on the VOC2007 dataset achieves 83.8% mean average precision, and using VOC2012 – 82.5% mAP. The network requires only 12.3% of the computational resource compared to ResNet-101. Also, the authors proved that a simple technique like truncated SVD could achieve a notable improvement in the runtime performance based on the proposed network. YOLO algorithm and its modifications are constantly developed to improve performance and efficiency. In the paper, Lu et al., was proposed the real-time object detection algorithm for videos based on the YOLO architecture, the so-called Fast YOLO. In the image preprocessing stage authors eliminate the image background which highly influences the final object detection results. As the base, the authors use Google Inception Net architecture to improve the YOLO network by using a small convolution operation to replace the original convolution operation. It helps to reduce the number of parameters and greatly shorten the time for object detection. Most of the related research uses already tested models, so in our work, we pay more attention to the new YOLOv5 architecture to find out its effectiveness in solving real-time object recognition problems.

3 Experimental investigation

3.1 Dataset

While the size of the training dataset does not necessarily influence the model size and complexity, usage of a smaller training dataset allows reduction of model training time. At the same time, the reduction of the training dataset might reduce the object detection accuracy, therefore a balance between model performance and accuracy should be defined for each situation. In this research, we are concentrating on the analysis of YOLOv5 model application performance on different iPhone architectures, therefore accuracy is not the main goal. A smaller training dataset was selected to be used in comparison to other accuracy-oriented solutions.

In this paper, the Microsoft Common Object in Context (COCO) dataset has been used to perform an experimental investigation. This dataset is collected by Lin et al. and it contains 330 000 images, where more than 200 000 images are labeled by human annotators. The dataset is available in four different types: training, validation, testing, unlabelled images, and annotated images. To prevent some bias or dataset unbalance, the reduction of the original COCO dataset was executed by selecting the COCO mini-train dataset and by using the cocosplit tool (Karazniewicz, 2021) spitting it into the 60/40 ratio, where 60% of the original dataset are train annotations (15 000 annotations), and 40% of the original train dataset are validation annotations (10 000 annotations). This

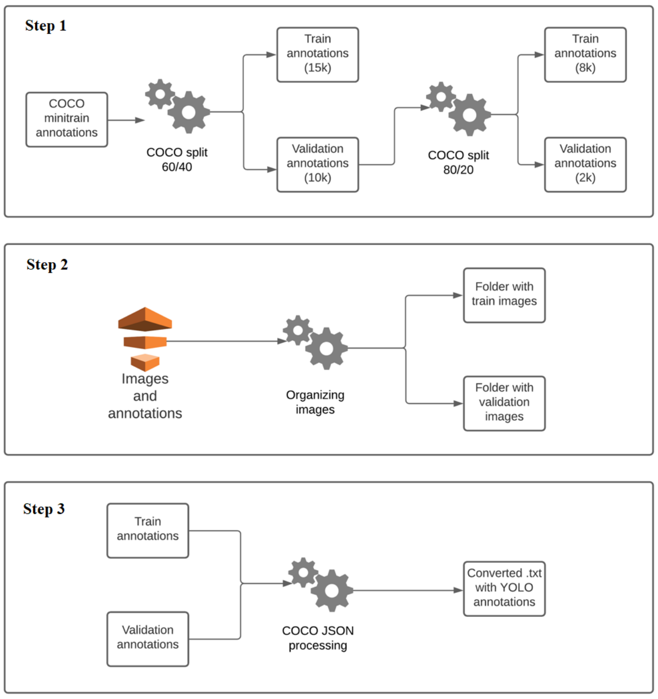

Fig. 2. The main steps of dataset preparation.

step allows gathering of the COCO dataset subset, where 10 000 annotations are presented as only validation annotations will be used for our experiment. After the selection of 10 000 annotations for further experiments, the subset of the COCO dataset was split into training and validation data by using the same cocosplit tool. 80% of the data was selected as training and 20% – as validation. The idea of COCO dataset reduction for our experiment is presented in Fig. 2 (the first two steps).

Each image in the COCO dataset has a unique identifier (ID) that corresponds to the file itself. Annotations store multiple values that are relevant to the specific image. The information applicable to object detection is category ID with 80 annotated object categories (classes), reference image ID, and bounding box coordinates (x, y, width, and height). The COCO dataset uses annotations stored in a single JSON file as key-value attributes. Meanwhile, the YOLO models use a different format – each image has a corresponding .txt file with the following fields one row per object, object class, and bounding box coordinates (x center, y center, width, height). Therefore a transformation from one file format to another is done by storing the gathered COCO dataset subject training and validation data into different catalogs and using the JSON2YOLO tool (Jocher, 2021) to get YOLO needed file format. The summary of dataset preparation is presented in Fig. 2, where the Step 1 includes COCO dataset reduction, the Step 2

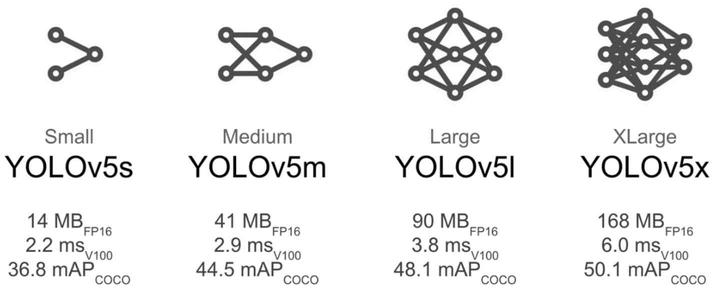

Fig. 3. YOLOv5 different model sizes, where FP16 stands for the half floating-point precision, V100 is an inference time in milliseconds on the Nvidia V100 GPU, and mAP based on the original COCO dataset (WEB (b), 2021).

prepares the dataset data to transformation suitable form, and the Step 3 converts the reduced COCO dataset to YOLO suitable format.

3.2 Model performance evaluation

The YOLOv5 provides different models with different configurations and parameter sizes (see Fig. 3). Respectively the YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x mean small (s), medium (m), large (l), and extra-large (x) models. Each model has its pros and cons, but eventually, the differences are in their complexities, performances, and overall accuracy. The (WEB (b), 2021) results show the size of the models varies from 14MB to 168MB, mAP values for the full COCO dataset is from 36.8 to 50.1 and the inference time on the Nvidia V100 GPU varies from 2.2 to 6.0 milliseconds. These results are suitable for real-time usage, however, achieved especially for artificial intelligence solutions dedicated architecture.

As we use a reduced COCO dataset, some additional experiments have to be performed to get the baseline model accuracy and performance values. As well, to reflect the mobile device specifics, the data input size for YOLOv5 models was set to 416×416 pixels. During the training of four models (s, m, l, x), each model trained on 200 epochs, default parameters for each model where left, 80/20 split on the previously defined dataset was used, and an output image size of 416×416 pixels was set. For each tested model performance was evaluated based on different metrics: mAP, and mAP with specified intersection over union (IoU), losses for both training and validation runs, speed, parameters amount, and GFLOPS. As most of the existing experiments on object detection concentrate on mAP values with 0.5:0.95 and 0.5 were used as the main accuracy metrics.

The models were trained and tested on the paid version of Google Colab, which provides the Tesla V100's GPU, a professional GPU with 16GB of fast HBM2 of video memory. Because of the cloud-based Google Colab specific, the inference time values varied during different periods, therefore an average value of several experiments is presented for each model (see Table 1).

Table 1. Results of YOLOv5 models, used on paid Google Colab environment.

Model	**Dataset**	**Size (pixels)**	**mAP 0.5:0.95**	**mAP 0.5**	**Inference time (ms)**	**Parameters**	**GFLOPS**
YOLOv5s	Reduced COCO datset	416×416	23.6	38.3	27	7.3	17
YOLOv5m			28.7	43.7	32	21.4	51.3
YOLOv5l			31.5	46.8	41	47	115.4
YOLOv5x			32.8	48.5	49	87.7	218.8

Our measured YOLOv5 accuracy and performance results demonstrate the same tendencies, as experiments by other authors (WEB (b), 2021): the accuracy, model inference time, and complexity of the model increase from the lowest values in the YOLOv5s model, to YOLOv5m, YOLOv5l, and the highest values in YOLOv5x model. Other factors, such as the size and number of dataset images, environment capabilities, influenced the accuracy. Therefore the performance of the models is not as good as achieved by other authors (Zhao et al., 2019). In comparison to YOLOv5 authors announced accuracy, the model accuracy decrease is not statistically significant ($p = 0.898$), while the performance, interference time changed statistically significant ($p = 0.005$) and increased several times. Even tho, the performance in the Google Colab platform is on the edge for real-time video processing – more than 20 images (in worse case, when YOLOv5x model is used) can be processed within one second, therefore if some frames of the video stream will be skipped, the object detection could be implemented in real-time.

3.3 Results of model application in different iPhone architectures

YOLOv5 models provide enough performance for usage in Google Colab, however, it is unknown how the created models will perform on mobile devices. The model accuracy is not influenced by the execution platform, however the model performance, inference time might be different because of different resources and its architecture. For experiments on whether YOLOv5 models are suitable for real-time usage in mobile devices, we used iPhone 12 model while utilizing the different system on a chip (SoC) components: an Apple Neural Engine (ANE), graphical processing unit (GPU), and central processing unit (CPU). The iPhone for this experiment was selected because of the variety of supported SoC – it has a specific neutral engine, which is hardware, dedicated for neural network applications, with the performance of up to 600 billion operations per second.

The created object detection models were applied for usage in iPhone 12 environment – all models were converted to the Core ML and quantized to int 8. The experimental investigation source code for the mobile object detector, an iOS application for real-time object detection available in (Dlužnevskij, 2021). These four models were tested with the same testing samples as in the Google Colab environment (see Table 2).

Comparing all models revealed the same tendencies as in the previous experiment – the YOLOv5s model is the smallest, therefore has the smallest inference time, while increasing the complexity of the model its inference speed is reduced. ANE architecture is the most suitable to run YOLOv5 models in iPhone 12, as shows the fastest inference

Table 2. Results of YOLOv5 models, used on iPhone 12 environment.

Model	**Average time, ANE (ms)**	**Average time, GPU (ms)**	**Average time, CPU (ms)**
YOLOv5s Int8	77	82	80
YOLOv5m Int8	106	114	148
YOLOv5l Int8	145	181	263
YOLOv5x Int8	341	321	441

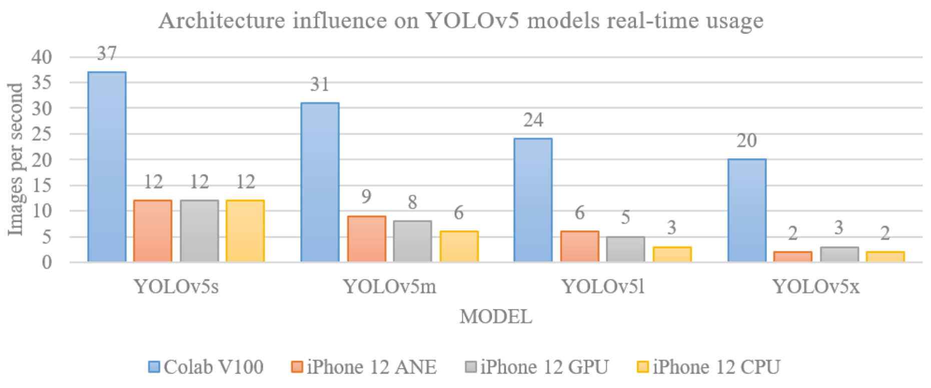

Fig. 4. Image processing frequency in different architectures of YOLOv5 models.

speed for most of the models. The difference between ANE and GPU architecture is not statistically significant ($p = 0.923$). Another interesting metric to analyze is how many images could be processed per second by using each of the technologies (see Fig. 4).

The figure above illustrates none of the YOLOv5 models would be able to process all 24 image frames in real-time by using iPhone 12. Even in the Google Colab platform, it is risky to use large or extra-large YOLOv5 models. Meanwhile, if some frames might be skipped, the iPhone 12 architecture is capable to apply the YOLOv5 model and process up to 12 images per second with the YOLOv5s model and up to 2 images per second with the YOLOv5x model. It is worth to mention the results are achieved by applying the models, defined in the previous subsection and using default parameters and image size of 416×416 pixels. It does not take into account the ANE specifics. As Leon De Andrade states (Andrade, 2021) the "kernel sizes above 7 are not supported by the Neural Engine and will force the device to switch back to the CPU/GPU". Therefore by adjusting the model configuration, a higher performance can be achieved. For example, by using the optimized for ANE YOLOv5s model, which uses 192×320 pixel image size, the inference time in ANE architecture is 10 ms, in GPU – 47 ms, in CPU – 27 ms. Such a performance allows the processing of up to 100 images per second. These results are more than enough for real-time usage in iPhone 12 devices.

4 Conclusions

Object detection in images is well developed for scientific research as provides needed datasets and tools for transformation from one dataset to another. The existing dataset and tools not only simplifies the object detection task but at the same time provide a clear benchmark for different research conditions and result comparison. The reduction of image size to 416×416 pixels does not reduce YOLOv5 model accuracy statistically significantly, even by reducing the dataset size to 8 000 records for training and 2000 records for validation. This proves YOLOv5 is suitable for mobile device-oriented images to detect objects. YOLOv5 model performance is highly influenced by the used hardware architecture and system on a chip. The performance of models in Google Colab with V100 architecture and iPhone 12 devices is not comparable because of different purpose architectures. Meanwhile, Apple Network Engine usage might increase the developed YOLOv5 models' performance in comparison to CPU usage.

Standard configuration YOLOv5 models with 416×416 pixels image size are not able to perform real-time object detection in iPhone 12 device as can process just up to 12 images per second. Meanwhile, by taking into account the Apple Network Engine specifics and optimizing the model, the object detection in iPhone 12 devices might reach up to 100 images per second, which is more than enough for real-time image detection in a video stream.

References

Andrade, L. D. `https://github.com/ultralytics/yolov5/issues/2526`.

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

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S. (2020). End-to-end object detection with transformers. *InProc. Eur. Conf. Comp. Vis*.

Chandan, G, Jain, A, Jain, H, et al. (2018). Real time object detection and tracking using deep learning and OpenCV. *In 2018 international conference on inventive research in computing applications (ICIRCA)*, Coimbatore, India, **11–12**, IEEE, pp. 1305–1308.

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., Holtham, E. (2018). Reversible architectures for arbitrarily deep residual neural networks. *In:Thirty-Second AAAI Conference on Artificial Intelligence*, Vol. **32**, AAAI Press, Palo Alto, pp. 2811–2818.

Dlužnevskij, D. *Source code of experimental investigation*. `https://github.com/danikkm/MobileObjectDetector`.

Floridi, L., Chiriatti, M. (2020). GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds Mach* **30**, pp. 681–694.

Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *In: Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Girshick, R. (2015). Fast R-CNN. *In: Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

He, K., Gkioxari, G., Dollár, P., Girshick, R. (2017). Mask R-CNN. *In: 2017 IEEE international conference on computer vision (ICCV)*, pp. 2980-8.

Hong, S., Roh, B., Kim, K. H. (2016). PVANet: Lightweight deep neuralnetworks for real-time object detection. *In Proc. IEEE Conf. Comput.Vis. Pattern Recognit (CVPR)*, Las Vegas, NV, USA, Jun, pp. 412–425.

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.

Jocher, G. *ultralytics/JSON2YOLO*. `https://github.com/ultralytics/JSON2YOLO`.

Karazniewicz, A. *akarazniewicz/cocosplit*. `https://github.com/akarazniewicz/cocosplit`.

Kulkarni, R., Dhavalikar, S., Bangar, S. (2018). Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning. *in Proc. of 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA).*

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Doll, P., Zitnick, C. L. (2014). *Microsoft COCO:common objects in context*, CoRR, Vol. **abs/1405.0312**.

Liu, S., Qi, L., Qin, H., Shi, J., Jia, J. (2018) Path aggregation network for instance segmentation. *In Proc. IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, Utah, USA, pp. 8759-8768.

Lu, S., Wang, B., Wang, H., Chen, L., Linjian, M., Zhang, X. (2019). A real-time object detection algorithm for video. *Computers & Electrical Engineering*, Vol. **77**, pp. 398-408.

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

Redmon, J., Farhadi, S. (2017). YOLO9000: better, faster, stronger. *inProceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, Honolulu, Hawaii, July.

Redmon, J., Farhadi, A. (2018). Yolov3: An Incremental Improvement. *arXiv: Computer Vision and Pattern Recognition*.

Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *In: Advancesin Neural Information Processing Systems*, pp. 91–99.

Rghioui, A., Oumnad, A. (2017). Internet of Things: Visions, Technologies, and Areas of Application, *Automation, Control and Intelligent Systems*, Vol. **5**, No. 6, pp. 83-91.

Schrittwieser, J., Antonoglou, I., Hubert, T. et al. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. Nature **588**, pp. 604–609.

Tan, M., Pang, R., Le, V. Q. (2019). EfficientDet: Scalable and Efficient Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, abs/1911.09070.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. (2017). Attention is all you need. *in: Advances in Neural Information Processing Systems*, pp. 5998–6008.

Wang, R. J., Li, X., Ling, C. X. (2018). Pelee: A real-time object detection system on mobile devices. *NeurIPS31*, pp. 1963–1972.

Wang, C. Y., Mark Liao, H. Y., Wu, Y. H., Chen, P. Y., Hsieh, J. W., Yeh, I. H. (2020). Cspnet: A new backbone that can enhance learning apability of cnn. *In: Proceedings of the IEEE/CVF conference oncomputer vision and pattern recognition workshops*, pp. 390–391.

WEB (a). *Overview of model structure about YOLOv5. 8.* `https://github.com/ultralytics/yolov5/issues/280`.

WEB (b). *YOLOv5 models*. `https://github.com/ultralytics/yolov5/issues/2990`.

Wilkowski, A., Kasprzak, W., Stefańczyk, M. (2019). Object detection in the police surveillance scenario. /textit2019 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 363-372.

Yu, J. et al. (2018). Real-time object detection towards high power efficiency. *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 704-708.

Zhao, Z. Q., Zheng, P., Xu, S. T., Wu, X. (2019). Object detection with deep learning: a review. *IEEE Trans. Neural Netw. Learn. Syst.*, **30(11)**, pp. 3212-3232.

Received September 7, 2021 , accepted September 9, 2021