Comparative Analysis of YOLOv5 and YOLOv6 Models Performance for Object Classification on Open Infrastructure: Insights and Recommendations

Marko Horvat, Ljudevit Jelečević, Gordan Gledec
University of Zagreb
Faculty of Electrical Engineering and Computing
Department of Applied Computing
Unska 3, HR-10000 Zagreb, Croatia
{marko.horvat3, ljudevit.jelecevic, gordan.gledec}@fer.hr

Abstract. *In this paper, the performance of YOLOv5 and YOLOv6 models is evaluated on standardized object classification tasks using Kaggle open-access cloud processing infrastructure. The performance of these architectures is evaluated on a balanced subset of COCO dataset considering precision, recall, $mAP_{0.5}$, and $mAP_{0.5:0.95}$ metrics, learning and processing time. Our results indicate that the choice of the optimal model in the rapidly evolving field of computer vision and machine learning is heavily influenced by the dataset size, complexity, and restrictions imposed by the data processing platform. It is not always advisable to select the most recent algorithm for all applications. The obtained results, program code and the implemented architectures are available for non-commercial use at: https://github.com/mhorvat/yolov5_yolov6_comparison*

Keywords. computer vision, image classification, deep learning, algorithm evaluation, performance comparison, YOLO

1 Introduction

Identifying and categorizing objects in an image or video stream is a fundamental task in computer vision (Guo et al., 2022; Mohan et al., 2023). Recently, there has been significant research interest in advancing and refinement of deep neural network models for object classification tasks in different domains because of their superior performance compared to previous methods, as per example in medicine (Boaro et al., 2022). The You Only Look Once (YOLO) model is unique in that it performs object recognition in a single pass (Redmon et al., 2016). This makes it significantly faster than other recognition models such as R-CNN and its variants (Srivastava et al., 2021). In this regard, the YOLO model family of deep learning networks has emerged as the most popular and effective open-code solution for various real-time computer vision tasks including object detection and categorization (Jiang et al., 2022). It's well-suited for applications such as video surveillance, self-driving cars, and augmented reality (Sukkar, Kumar & Sindha, 2021; Zaghari et al., 2021; Zhang et al., 2023). The YOLOv5 model, released in 2020, achieved outstanding results on several benchmarks and has been widely accepted by the computer vision community (Jocher, 2020; Solawetz, 2020; Jiang et al., 2022). More recently, the YOLOv6 model was released in 2022 and has been proposed to further improve performance compared to its predecessor (Li et al., 2022). Given the advancements in both models, it is beneficial to conduct a thorough comparison between YOLOv5 and YOLOv6 to determine their respective strengths and weaknesses on a common dataset and in real-world applications.

In our previous study, the authors investigated the performance of different YOLOv5 models (Horvat, Jelečević & Gledec, 2022). Through a comparative analysis using an everyday image dataset, the authors provided recommendations for selecting the most appropriate YOLOv5 model based on the specific problem type, helping researchers make informed decisions. The authors continued this line of research with a comparative analysis of the already accepted YOLOv5 and the newer YOLOv6 architecture.

To evaluate the performance of the YOLOv5 and YOLOv6 models, experiments were performed on a subset of the COCO dataset (Lin et al., 2014). The performance of the models was evaluated based on several key metrics, including precision, recall, $mAP_{0.5}$, and $mAP_{0.5:0.95}$ metrics, as well as learning and processing time (Padilla et al., 2020). The analysis was facilitated by the open data storage and Graphical Processing Unit (GPU) hardware infrastructure provided by the online data science and machine learning platform Kaggle (Bojer & Meldgaard, 2021). In addition to the primary goal of comparing YOLOv5 and v6 models, this research also aimed to demonstrate

the practicality of utilizing a modern, freely available cloud computing infrastructure for conducting realistic deep learning computer vision tasks. It demonstrates that high-quality research can be carried out without the need for expensive, dedicated hardware resources.

The authors believe that evaluating YOLO models through openly accessible processing resources is crucial for researchers with access to limited resources. Advancements in computer vision methods may not be as beneficial for these practitioners, due to their inability to access high-end infrastructure and extensive image datasets for deep model training. Consequently, such users often resort to working with datasets containing only a few thousand images, which is not ideal for achieving optimal results. By resorting to open, cloud-based GPU infrastructure, these users can effectively overcome their computational and financial constraints for developing deep learning applications. This democratizes access to advanced machine learning techniques and promotes innovation and research.

The remainder of the paper is organized as follows: Section 2 describes our experimental setup, starting with an overview of the COCO and COCO minitrain in Section 2.1, followed by Section 2.2, which provides a comprehensive explanation of the dataset sampled specifically for training of the YOLOv5 and YOLOv6 models. The specific process of training these models is described in Section 2.3. Then, in Section 3, the results of our object classification experiments are presented. Section 3.1 discusses the processing time while Section 3.2 discusses the object recognition results for all classes. In addition, Section 3.3 focuses on the recognition results for just one class. Section 4 provides a detailed discussion of the obtained results and provides recommendations for choosing the best model for a given application. Finally, Section 5 concludes the paper, summarizes main findings, and provides outlook for further research.

2 Experimental setup

In this section the undertaken experimental approach is outlined which involves four primary steps: (i) the construction of the experimental dataset, (ii) the selection of YOLOv5 and v6 models for comparative analysis, (iii) the deployment of the selected models on a cloud-based platform, and finally, and (iv) the training of these models.

When deciding on an optimal dataset for testing, it is crucial to consider the standards set by the published literature. In Deep Learning (DL) and Artificial Neural Networks (ANNs) research, it is common to use purpose-built and established datasets to benchmark the performance of the models. One commonly employed dataset for DL models in the field of computer vision, including YOLO, is the COCO (Common Objects in Context) (Lin et al., 2014).

Experimental procedures in similar studies reported in the published literature usually require 256 or 300 epochs, which necessitates the use of one or more dedicated GPUs. A prominent example of such hardware is the Nvidia P100 and V100 GPUs, which have been specifically designed for machine learning (ML) applications and DL ANN training (Markidis et al., 2018).

It should be noted, however, that despite the availability of specialized hardware, the training process is anything but fast. Despite the simplicity of some models, the time required can still be considerable as investigated in this study.

2.1 COCO and COCO minitrain

The COCO dataset is a large-scale object detection, segmentation, and captioning dataset often used in computer vision research and applications (Lin et al., 2014). It is specifically designed for object recognition, segmentation, and captioning in the context of scene understanding. The dataset contains over 200,000 labeled images with over 1.5 million object instances and 80 object categories. The dataset is notable for its quality, diversity and complexity, with images featuring objects in a wide range of orientations, sizes, and contexts.

COCO minitrain is a carefully curated training set derived from the much larger COCO train2017 dataset, containing 25,000 labeled images (~20% of train2017), ~184,000 annotations and 80 object categories (Samet, Hicsonmez & Akbas, 2020). This smaller dataset, and the accompanying data processing software support, was developed to enable efficient hyperparameter tuning and reduce the computational cost associated with experiments in computer vision. Designed as a more compact alternative to the complete COCO dataset, the minitrain dataset allows researchers and engineering to optimize their models and perform initial tests in a time-saving manner without compromising the quality of their work. A sample of the COCO and COCO minitrain images are shown in Figure 1.

Typically, smaller datasets may be useful for rapid prototyping as training models on smaller datasets can be much faster, which allows for more iterations and fine-tuning of the model and its hyperparameters. Also, large datasets require significant computational resources (CPU/GPU power, memory, storage space) that may not always be available, especially in low-resource environments. More compact datasets are also used for model architecture debugging and performance checking before migrating to larger datasets or production environments.

The COCO minitrain is a valuable resource for the computer vision community because it enables faster experimentation and model development while providing a representative sample of the larger COCO dataset. This important feature supports reproducibility

and simplifies validation of the obtained research by other teams.

Figure 1. Examples of labeled and semantically segmented images in the COCO and COCO minitrain.

The statistical distribution of object classes in the minitrain dataset closely mirrors those of the COCO2017 dataset, ensuring that the performance and behavior of models trained on the minitrain dataset generalize well to the entire dataset. To ensure the quality of sampling, labels within the COCO2017 dataset are further divided into three size groups: S, M, and L, representing small, medium, and large object regions, respectively, as they appear in the images. The dimensions defining these size categories are defined by the specifications of the COCO metric (Lin et al., 2014).

To maintain a similar distribution, the minitrain images were randomly sampled from the full COCO2017 set while preserving the following three quantities as much as possible: 1) the proportion of object instances from each class, 2) the overall ratios of S, M, and L objects, and 3) the per-class ratios of S, M, and L objects.

In the original COCO minitrain version, sampling was done by iterative random sampling while minimizing the fitness function – the discrepancy between the largest and smallest ratio of the number of tags in both the COCO2017 set and the sample. To increase the likelihood of detecting a sample whose distribution is very similar to the label distribution in the COCO2017 dataset, the penalty function was modified (Jelečević, 2023). The Kullback-Leibler divergence was used along with Laplace smoothing as a more effective measure of similarity between the distributions (Zhuang et al., 2015). This approach helped to ensure that the selected training sample accurately represents the general class distribution of the COCO2017 dataset.

2.2 Dataset for YOLOv5 and YOLOv6 models training

The experimental dataset was generated using balanced sampling from the COCO train2017. The original dataset was sampled 200,000 times before generating a subset of 1,000 images with a total of 7,587 labels that was used for model training. The sampling pseudocode for generating the training subset is given here:

```
best_sample = none;
for i in range(0, 200,000):
 sample = rnd_select_sample(1000 images,
  from COCO_train2017_dataset);
 eval_score =
  eval_sample_representation(sample, based
  on label distribution);
 if (best_sample = none or eval_score >
   best_eval_score):
  best_sample = sample;
  best_eval_score = eval_score;
```

And the actual Python code run to obtain the experimental dataset uses COCO minitrain libraries:

```
!python sample_coco.py --coco_path
"/kaggle/datasets/coco" --save_file_name
"instances_train2017_minicoco_1k" --
save_format "json" --sample_image_count
1000 --run_count 200000 --
allow_empty_sample_classes
```

The statistical chi-square test was performed to verify the COCO train2017 dataset and the sampled subset had the same class distribution. The test resulted in a p-value $p=4.2\cdot10^{-17}$, well below the significance of $\alpha=0.05$, This led to the acceptance of the null hypothesis, indicating a correlation between the distribution of the COCO train2017 dataset and the sampled subset. Therefore, the chi-square test confirmed that the COCO train2017 dataset and the sampled subset indeed have the same distribution.

Also, the Pearson correlation between the population and the subset is $r=0.9938$, indicating a strong linear dependence between number of instances of all classes in the full COCO train2017 dataset and the sampled subset used for training.

Figure 2 shows the distribution of the 30 most frequent classes in the sampled dataset and in the original COCO train2017 dataset. The 30 most common object categories and their IDs are: Person (1), Car (3), Chair (62), Book (84), Dining table (67), Bottle (44), Cup (47), Umbrella (28), Bowl (51), Carrot (57), Horse (19), Kite (38), Handbag (31), Banana (52), and Potted plant (64). Some classes are repeated in different image sizes (S, M, or L).

2.3 YOLOv5 and YOLOv6 training process

Each YOLO model was trained on the entire training subset stored on the online platform Kaggle using the

virtualized Nvidia P100 GPU available in the programming environment[1].

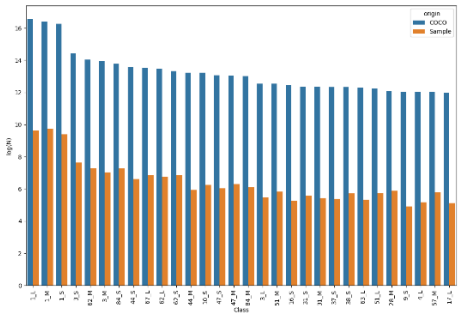

Figure 2. The distribution of the 30 most frequent classes in the COCO2017 and the experimental subsample.

Three pairs of mutually equivalent models were chosen for the training process: YOLOv5s and YOLOv6-N, YOLOv5m6 and YOLOv6-M, as well as YOLOv5x and YOLOv6-M6. Each pair consists of one model from the YOLOv5 and one from the YOLOv6 series.

Each training session lasted for 300 epochs, meaning that the Kaggle early stopping feature was not activated, which would halt the training if no progress was made in the last 100 epochs. Early stopping is a form of regularization method used to prevent overfitting in an iterative training. Thanks to the early stopping feature, at least some learning took place across all the models during the initial 200 epochs.

The scripts for training the YOLOv5 and YOLOv6 models were run with the following parameters:

```
!python train.py --data coco.yaml --epochs
300 --weights '' --cache --cfg
yolov5m6.yaml  --batch-size 16

!python tools/train.py --img 640 --batch 16
--epochs 300 --conf configs/yolov6n.py --
data data/coco.yaml
```

Tables 1 and 2 show the training durations, indicated in hours (h) and minutes (m), as well as the training speeds, denoted in the number of parameters per second (N/s), for the selected S, M, and L pairs of YOLOv5 and YOLOv6 models. The better results are highlighted in green.

Table 1. The duration of YOLO models training in hours and minutes

	YOLOv5		YOLOv6	
	Model	Duration	Model	Duration
S	YOLOv5s	05h 50m	YOLOv6-N	07h 26m
M	YOLOv5m6	09h 12m	YOLOv6-M	10h 21m
L	YOLOv5x	17h 20m	YOLOv6-M6	11h 18m

Table 2. The speed of YOLO models training in parameters per second

	YOLOv5		YOLOv6	
	Model	N/s	Model	N/s
S	YOLOv5s	19.05	YOLOv6-N	9.76
M	YOLOv5m6	59.88	YOLOv6-M	52.04
L	YOLOv5x	76.57	YOLOv6-M6	108.71

As expected, the more complex models require a longer training period. However, as can be seen in Table 2, complex models prove to be more efficient during training, effectively training more parameters in the same amount of time. Of all the models, YOLOv6-M6 stands out as the most efficient in terms of training.

3 Object classification results

As already explained, for the comparative analysis and benchmarking of YOLOv5 and YOLOv6 models the COCO val2017 dataset was used which includes a total of 5,000 images. It should be emphasized that the sampled training dataset with 1,000 images is not a subset of COCO val2017 thus ensuring a comprehensive testing process.

It should be noted that the experimental results were obtained without the use of Nvidia TensorRT, an SDK for high-performance inference runtime optimizer that delivers low latency and higher throughput for DL applications (Markidis et al., 2018; Shafi et al, 2021). However, the TensorRT SDK is commonly used in reference literature for YOLOv6 benchmarking (Li et al., 2022). This SDK is available as an optional feature on the Kaggle platform and, as reported in the published literature, can improve processing speed by three times (Markidis et al., 2018; Shafi et al, 2021).

Given the large number of image classes in the COCO2017 dataset, the relatively limited size of the evaluation dataset, and the significant proportion of the class “Person”, which accounts for one-third of all labels in the test subset, object classes detection results are presented separately in two parts: performance metrics calculated for all classes in the subset, and only to the “Person” class, in sections 3.2 and 3.3, respectively. This approach was chosen because of the unique characteristics and challenges presented by the size and class distribution of the evaluation subset.

3.1 Processing time

In the YOLO architecture, an input image goes through three different processing stages, each of which contributes significantly to the speed and accuracy of the output (Diwan, Anirudh & Tembhurne, 2023; Jiang et al., 2022).

1 Kaggle, https://www.kaggle.com/ .

Preprocessing is the first stage in which the input image is prepared for further processing. This includes resizing the image to the required dimensions, depending on the YOLO version, and normalizing the pixel values. During normalization, the pixel intensities are usually scaled to a range between 0 and 1 or -1 and 1 so that the model converges faster during training.

Inference is the core stage where the model makes predictions on the preprocessed image and the actual object recognition takes place. The preprocessed image is passed through the YOLO neural network, which outputs a three-dimensional tensor. This tensor contains the bounding box coordinates, class probabilities, and object scores for each grid cell in the image.

Non-Maximum Suppression (NMS) is the final stage that refines the results of the inference phase and helps to reduce the number of overlapping detections. The YOLO model can predict multiple bounding boxes for the same object, and NMS helps eliminate redundancies. To do this, first all bounding boxes whose objectness score is below a certain threshold are discarded. Then, for the remaining boxes, the box with the highest score is selected and all other boxes with a high overlap – measured by the Intersection over Union (IoU) metric – with the selected box are eliminated. This process is repeated until all boxes have either been selected or discarded. In Table 3 the duration of preprocessing of all input images in the evaluation dataset on the Kaggle's Nvidia P100 cloud infrastructure is shown in milliseconds [ms], separately for all three stages and in total. The best scores are highlighted in green.

Table 3. Duration required for processing all input images utilizing the Nvidia P100 cloud infrastructure

	Preprocessing [ms]	Inference [ms]	NMS [ms]	Total [ms]
YOLOv5s	0.20	2.80	1.90	4.90
YOLOv5m6	0.20	6.90	1.70	8.80
YOLOv5x	0.20	18.10	1.90	20.20
YOLOv6-N	0.12	1.57	18.40	20.09
YOLOv6-M	0.12	7.96	28.16	36.24
YOLOv6-M6	0.15	9.47	31.65	41.27

From Table 3, it can be seen that the preprocessing times are relatively consistent across all models, ranging from 0.12 ms to 0.2 ms. However, there are significant differences in the inference times and the Non-Maximum Suppression (NMS) times. The YOLOv5s model has the shortest total processing time of 4.9 ms, which is primarily due to the low inference time of 2.8 ms. On the other hand, the YOLOv6-M6 model has the longest total processing time of 41.27 ms, with the NMS stage taking up a significant portion of this time (31.65 ms).

Figure 3 shows the expected approximate number of frames per second (FPS) for each model. The YOLOv5s model outperforms all others with 204 FPS score, which is consistent with its shortest total processing time in Table 3. This is a very significant difference that results in the YOLOv5x processing video signals at twice as many frames per second as the YOLOv6-M6 model with the same complexity. However, the YOLOv6-M6 model still achieves 24 FPS despite its longer processing time which is marginally enough for real-time applications.

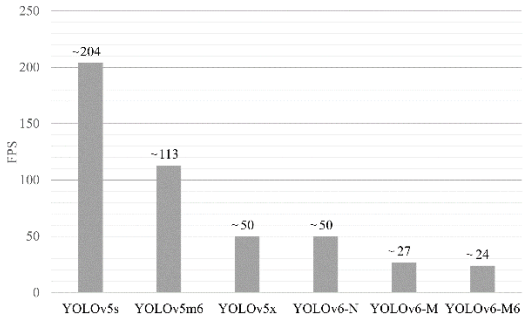

Figure 3. Approximate number of frames per second (FPS) for each model achieved on the Nvidia P100 cloud infrastructure

3.2 All class detection results

Table 4 shows precision, recall, $mAP_{0.5}$, and $mAP_{0.5:0.95}$ results for object detection of all COCO2017 classes in the evaluation dataset. The best results are again highlighted green.

Table 4. Object detection results for all COCO2017 classes in the evaluation dataset

	Precision	Recall	$mAP_{0.5}$	$mAP_{0.5:0.95}$
YOLOv5s	21.9	10.8	7.79	3.48
YOLOv5m6	24.8	12.7	9.28	4.52
YOLOv5x	22.1	12.5	9.10	4.59
YOLOv6-N	14.6	17.0	6.74	3.47
YOLOv6-M	15.9	20.0	8.51	4.81
YOLOv6-M6	15.2	20.0	8.14	4.59

From Table 4 it can be observed that the YOLOv5m6 model has the highest precision, indicating that it has the highest accuracy in detecting objects among all models. However, the YOLOv6-M and YOLOv6-M6 models have the highest recall, suggesting they are the most capable of identifying all relevant instances. In terms of $mAP_{0.5}$, the YOLOv5m6 and YOLOv5x models perform comparably well, with scores of 9.28 and 9.10 respectively. However, the YOLOv6-M model outperforms all others in the $mAP_{0.5:0.95}$ metric with a score of 4.81, indicating its superior performance across a range of IoU thresholds.

Figure 4 presents a comprehensive analysis of the relationships concerning the detection of all classes within the evaluation dataset. The top diagram in Figure 4 shows the relationship between $mAP_{50:95}$ and

the number of model parameters. This relationship is significant because it provides insight into how the complexity of the model, indicated by the number of parameters, influences its precision performance. The bottom diagram illustrates the correlation concerning $mAP_{50:95}$ and the duration of image processing. This correlation is crucial because it shows the balance between the precision of the model and the processing speed. Essentially, it provides an understanding of how fast the model can process images without compromising its precision.

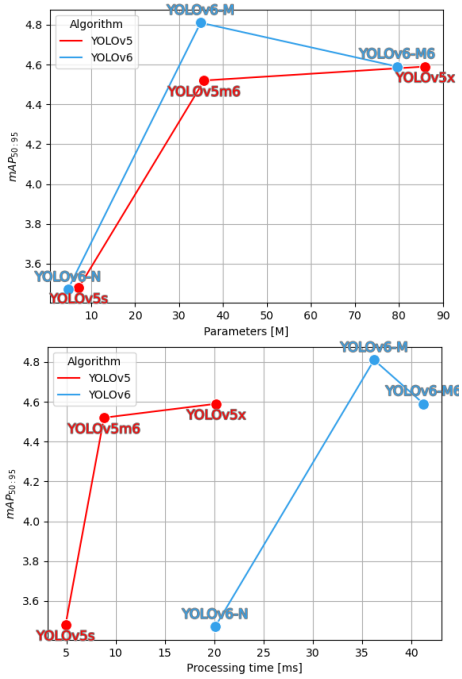

Figure 4. Relationships between $mAP_{50:95}$ and the number of model parameters (top), and $mAP_{50:95}$ and the duration of image processing (bottom) for detection of all classes

3.3 One class detection results

Table 5 presents object detection results only of "Person" class in the evaluation dataset.

Table 5. Object detection results only for class "Person" in the evaluation dataset

	Precision	Recall	$AP_{0.5}$	$AP_{0.5:0.95}$
YOLOv5s	39.6	45.4	39.8	16.9
YOLOv5m6	42.6	48.3	44.2	20.4
YOLOv5x	45.1	47.7	43.1	20.7
YOLOv6-N	54.6	41.0	41.9	19.7
YOLOv6-M	60.3	46.0	49.1	25.2
YOLOv6-M6	62.7	44.0	48.1	24.6

Figure 5 shows relationships between $mAP_{50:95}$ and the number of model parameters, and $mAP_{50:95}$ and the duration of image processing in the top and bottom diagrams, respectively, for object detection of only one class "Person" in the evaluation dataset.

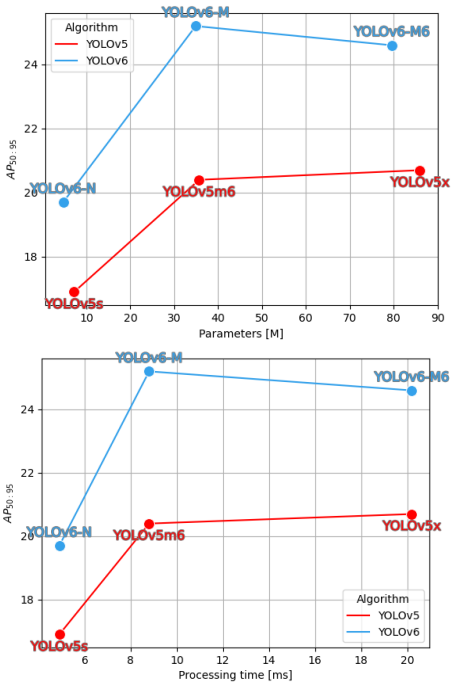

Figure 5. Relationships between $mAP_{50:95}$ and the number of model parameters (top), and $mAP_{50:95}$ and the duration of image processing (bottom) for detection of only one class "Person"

Based on the detection results for the "Person" class alone, the YOLOv6-M model outperforms others, irrespective of whether the evaluation is based on the constant Jaccard index metric ($AP_{0.5}$) or the metric that considers a range of index values ($AP_{0.5:0.95}$). This indicates that the YOLOv5 model is better at handling unbalanced labels, while the YOLOv6 performs better in object localization.

An important difference compared to the results for detection of all classes (in Section 3.2) is the significantly stronger correlation between the time required for image processing and the number of parameters in both the YOLOv5 and YOLOv6 models. This suggests that the computational efficiency of the YOLOv5 and YOLOv6 models is significantly positively affected by the number of parameters used, due to the complexity introduced by a larger parameter space. The increased time requirement could potentially impact the real-time capability of the models. Consequently, optimizing the number of parameters can significantly improve the processing time and overall performance of these models without sacrificing object detection accuracy.

4 Discussion

It is tempting to believe that a more recent deep learning architecture is always the preferable choice for

any research or software platform, but selecting the optimal data processing model is more delicate than that. To make an informed and correct decision, it is necessary to consider three crucial factors: 1) the YOLO model intended function, 2) available hardware, and 3) training time limits.

When using a pretrained model for image processing and processing time is not an issue, opting for the most complex model with the highest possible accuracy is a viable option. However, for real-time video signal processing involving localization and classification with multiple labels, the YOLOv5m6 model is recommended because it demonstrates a favorable speed-accuracy ratio. If the task is less demanding with fewer labels and the focus is on localization, the YOLOv6-M6 model is a more suitable alternative.

On the other hand, in scenarios where training time is limited, the YOLOv6 network is a better choice compared to the marginally slower YOLOv5, especially when working with multi-class models. YOLOv6, with its more advanced architecture, is designed to efficiently process complex object detection tasks, offering a balance between speed and accuracy.

Although this paper provides initial guidelines for choosing a suitable YOLOv5 or YOLOv6 model, it is advisable for researchers to adapt their selection to their specific requirements and use-cases. This can be achieved by consecutively testing different YOLO models on a subset of their data and evaluating the performance on the intended hardware after deployment. By conducting such iterative experimental assessments on a limited dataset, researchers can gather the necessary insights to make an informed decision about the most suitable YOLO model for their specific needs.

5 Conclusion

The YOLO framework deployed in cloud environment provides an out-of-the-box solution that offers great benefits to engineers and researchers who may lack extensive experience in computer vision algorithms and access to high-end processing infrastructure. However, to improve the efficiency of object recognition without delving into optimization of deep learning and machine learning algorithms, it would be recommended to improve the understanding of the scene. One possible method to improve classification performance is to neglect redundant image sections and focus exclusively on authentic regions of interest before integrating the image into the YOLO workflow. The use of knowledge graphs and ontologies, together with automated reasoning services such as expert systems is strongly recommended to obtain a truly semantically rich scene description. These methods facilitate a comprehensive understanding of the semantic regions detected in the image and their conceptual relationships (Horvat, Grbin & Gledec, 2013a; Horvat, Grbin & Gledec, 2013b).

In conclusion, the choice of a DL architecture such as YOLOv5 or YOLOv6 depends on several factors, including the intended function, available hardware, and time constraints for training. For image processing tasks where time is not an issue, more complex models with high accuracy can be used. For real-time video processing requiring classification and localization with multiple labels, YOLOv5m6 is recommended due to its optimal speed-to-accuracy ratio. For tasks with fewer labels where localization is the primary concern, YOLOv6-M6 is recommended. When training time is limited, YOLOv6 outperforms the slightly slower YOLOv5, especially for multiclass models, by providing a balance between speed and accuracy. Despite these guidelines, it is recommended that researchers tailor their selection to specific needs by testing different YOLO models on a subset of data and evaluating performance on the intended hardware after deployment. This iterative testing allows for informed decisions about the most appropriate YOLO model for their unique requirements.

References

Boaro, A., Kaczmarzyk, J. R., Kavouridis, V. K., Harary, M., Mammi, M., Dawood, H., Shea, A., Cho, E. Y., Juvekar, P., Noh, T., Rana, A., Ghosh, S., & Arnaout, O. (2022). Deep neural networks allow expert-level brain meningioma segmentation and present potential for improvement of clinical practice. *Scientific Reports*, *12(1)*, 1–3. https://doi.org/10.1038/s41598-022-19356-5

Bojer, C. S., & Meldgaard, J. P. (2021). Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, *37*(2), 587-603. https://doi.org/10.1016/j.ijforecast.2020.07.007

Diwan, T., Anirudh, G., & Tembhurne, J. V. (2023). Object detection using YOLO: Challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, *82*(6), 9243-9275. https://doi.org/10.1007/s11042-022-13644-y

Guo, M. H., Xu, T. X., Liu, J. J., Liu, Z. N., Jiang, P. T., Mu, T. J., Zhang, S. H., Martin, R. R., Cheng, M. M., & Hu, S. M. (2022). Attention mechanisms in computer vision: A survey. *Computational Visual Media*, *8(3)*, 331–368. https://doi.org/10.1007/s41095-022-0271-y

Horvat, M., Grbin, A., & Gledec, G. (2013). *WNtags: A web-based tool for image labeling and retrieval with lexical ontologies*. *Frontiers in artificial intelligence and applications*, *243*, 585–594. https://doi.org/10.3233/978-1-61499-105-2-585

Horvat, M., Grbin, A., & Gledec, G. (2013). Labeling and retrieval of emotionally-annotated images using WordNet. *International Journal of Knowledge-based and Intelligent Engineering Systems, 17*(2), 157–166. https://doi.org/10.3233/KES-130269

Horvat, M., Jelečević, Lj., & Gledec, G. (2022). A Comparative study of YOLOv5 models performance for image classification. In *Proceedings of the 33rd Central European Conference on Information and Intelligent System (CECIIS 2022).* (pp. 349–356).

Jelečević, Lj. (2023). Modified COCO minitrain. Retrieved from https://github.com/thelcrysis/coco-minitrain

Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2022). A Review of Yolo algorithm developments. *Procedia Computer Science*, *199*, 1066–1073. https://doi.org/10.1016/j.procs.2022.01.135

Jocher, G. (2020). YOLOv5. Retrieved from https://github.com/ultralytics/yolov5

Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., ... & Wei, X. (2022). YOLOv6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755). Springer, Cham.

Markidis, S., Der Chien, S. W., Laure, E., Peng, I. B., & Vetter, J. S. (2018, May). Nvidia tensor core programmability, performance & precision. In *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (pp. 522-531). IEEE. https://doi.org/10.1109/IPDPSW.2018.00091

Mohan, R., Elsken, T., Zela, A., Metzen, J. H., Staffler, B., Brox, T., Valada, A., & Hutter, F. (2023). Neural Architecture Search for Dense Prediction Tasks in Computer Vision. *International Journal of Computer Vision*, 1–24. https://doi.org/10.1007/s11263-023-01785-y

Padilla, R., Netto, S. L., & Da Silva, E. A. (2020, July). A survey on performance metrics for object-detection algorithms. In *2020 international conference on systems, signals and image processing (IWSSIP)* (pp. 237–242). IEEE.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779–788).

Samet, N., Hicsonmez, S., & Akbas, E. (2020). Houghnet: Integrating near and long-range evidence for bottom-up object detection. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16* (pp. 406-423). Springer International Publishing.

Shafi, O., Rai, C., Sen, R., & Ananthanarayanan, G. (2021, November). Demystifying tensorrt: Characterizing neural network inference engine on nvidia edge devices. In *2021 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 226-237). IEEE.

Solawetz, J. (2020). Yolov5 new version-improvements and evaluation. *Roboflow. Seach date*. Retrieved from https://blog.roboflow.com/yolov5-improvements-and-evaluation/

Srivastava, S., Divekar, A. V., Anilkumar, C., Naik, I., Kulkarni, V., & Pattabiraman, V. (2021). Comparative analysis of deep learning image detection algorithms. *Journal of Big Data*, *8*(1), 1-27.

Sukkar, M., Kumar, D., & Sindha, J. (2021, July). Real-Time Pedestrians Detection by YOLOv5. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 01–06). IEEE.

Zaghari, N., Fathy, M., Jameii, S. M., & Shahverdy, M. (2021). The improvement in obstacle detection in autonomous vehicles using YOLO non-maximum suppression fuzzy algorithm. *The Journal of Supercomputing*, *77*(11), 13421-13446.

Zhang, L., Xu, F., Liu, Y., Zhang, D., Gui, L., & Zuo, D. (2023). A posture detection method for augmented reality–aided assembly based on YOLO-6D. *The International Journal of Advanced Manufacturing Technology*, *125*(7-8), 3385-3399.

Zhuang, F., Cheng, X., Luo, P., Pan, S. J., & He, Q. (2015, June). Supervised representation learning: Transfer learning with deep autoencoders. In *Twenty-fourth international joint conference on artificial intelligence*