Computer Vision: YOLO Version Performance on Varying Processors

Michael Tolmajian, Youssef Ali, Ali Bou Haidar, Javier Ramos
matolmajian@cpp.edu, yaali@cpp.edu, abouhaidar@cpp.edu, javierramos2@cpp.edu
The Department of Electrical and Computer Engineering
College of Engineering
California State Polytechnic University, Pomona

***Abstract*—This report delves into the field of computer vision and features the use of the You Only Look Once (YOLO) object detection algorithm family. The paper also analyzes and applies its variations on different processing units. We tested the versions, YOLOv5, YOLOv7, and YOLOv8, using Intel's i7 and i9 central processing units (CPUs) to compare preprocess, inference, and postprocess speeds. The Python programming language and additional packages, including Ultralytics and OpenCV, were used to complete this analysis. Other methods, including creating a virtual environment from the Python standard library and cloning, were also utilized to conduct the experiments.**

I. Introduction

YOLO is a family or set of algorithms designed to keep object detection precise and quick at the same time. Compared to the other object detection models, the You-Only-Look-Once algorithms stand out, as they are well-rounded. The family gets its name from its single-stage characteristic, where the algorithm makes quick predictions, using an input image, of different classes' locations and sizes along with their probabilities [1].

Single-stage objection detection models tend to have shorter inference times than their double-stage counterparts, which precedes detection by generating proposals using a convolutional neural network, or CNN, for classification of specific regions [1]. Despite YOLO being a faster algorithm than double-stage models, they tend to be less accurate with their quick decision-making.

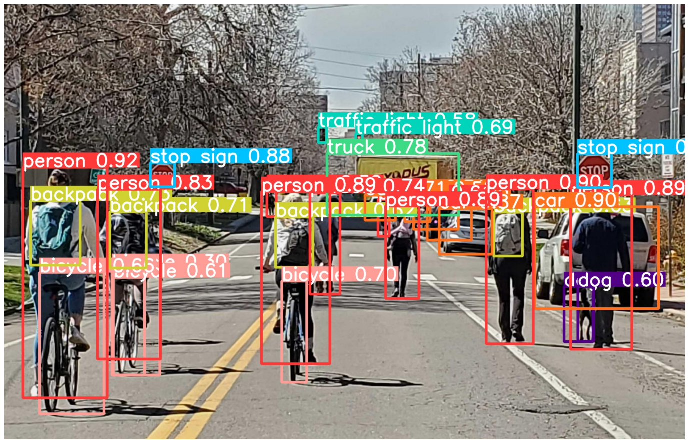

Fig. 1. Sample of bikers on a Denver street object detected using YOLOv8.

Inference time is the execution time of an object detection occurrence with the ability to measure speed, its reciprocal. A shorter inference time leads to a higher speed, not performance. A high performance, in the cases of image processing and object detection, considers both speed and accuracy.

Figure 1 shows the different objects identified using YOLO in a single stage. A few people, along with bikes and stop signs, had been identified with a number. This number is the probability that is generated by the CNN, and it represents the algorithm's confidence a certain object resembles other objects it had trained to see.

II. Methodology

The goal of the project was to compare and contrast between the performance among different versions of the You-Look-Only-Once algorithm for both the Intel Core i7 and Intel Core i9 processors. Every execution of the YOLO algorithm provides the user with an inference time, or the time needed to produce data.

Execution of the YOLO algorithms happened using the Windows Command Prompt. For this examination, the group needed components of both software and hardware, considering the YOLO versions themselves as software.

A. Software

For this examination, the project used the Python high-level, programming language version 3.8.0 with several installations using `pip` package manager on the Windows 11's Command Prompt. These installations were Ultralytics, OpenCV, and Virtualenv. Each of the used YOLO versions had similar installation steps, in terms of packages and methods.

With installing Python also came installing the necessary YOLO versions: YOLOv5, YOLOv7, and YOLOv8. Downloading these variations was possible using the `git clone` command provided by Git for Windows. The cloning command uses a mentioned website address to a Git or GitHub repository for installation. However, the link varies based on the version of YOLO needed.

Additionally, pre-trained models, using the PyTorch framework, were chosen to be the models of their corresponding YOLO versions.

1) YOLOv5: For Version 5, two models from the GitHub repository were used: `yolov5n6.pt` and `yolov5x6.pt`. The group showed the difference in average precision by using two different models of YOLOv5.

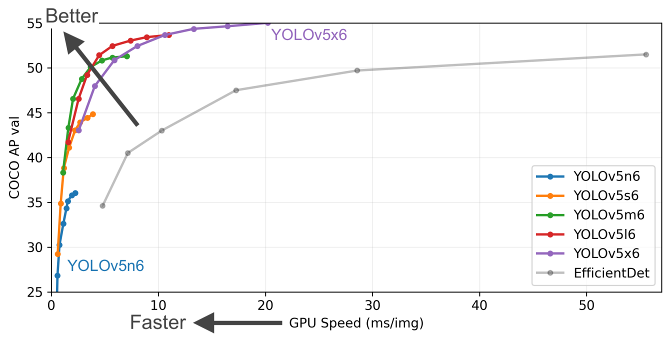

Fig. 2. Model variations of YOLOv5 compared using average precision values.

Figure 1 demonstrates the average precision, or detection accuracy, of each of the models provided by YOLOv5. Using Figure 2, `yolov5n6.pt` and `yolov5x6.pt` compares a fast, lesser accurate model to a slower yet precise one.

2) YOLOv7: With Version 7, this project only experimented on the `yolov7.pt` model.

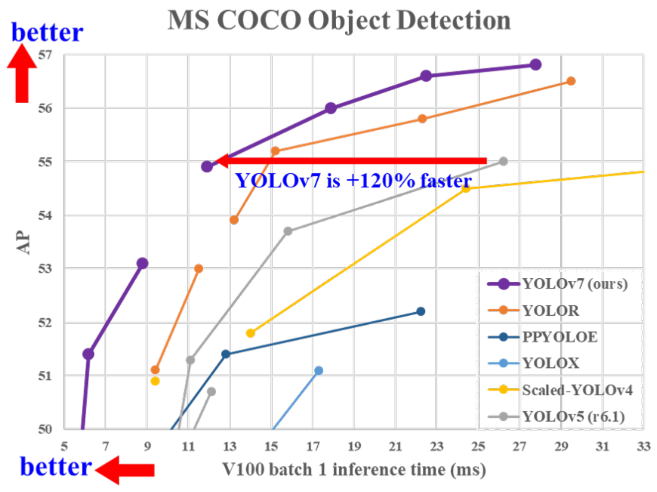

Fig. 3. Detection accuracy among different versions of YOLO.

In Figure 3, mainly YOLOv5 and YOLOv7 are compared with red indicators, determining the change in performance between them. For instance, both YOLO variations can measure an average precision of 0.55, but version 5 requires 14 milliseconds more inference time to catch up [3]. As we trace their plots toward the origin, the behavior repeats [3]. By testing YOLOv7, the project would demonstrate this behavior.

3) YOLOv8: From the Ultralytics GitHub repository, the models used for object detection were `yolov8n.pt` and `yolov8x.pt`.

Similar to YOLOv5's examination, YOLOv8's fastest model and most precise model will be compared amongst each other.

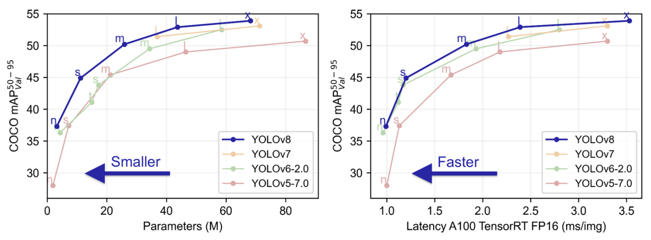

Fig. 4. (left) Accuracy of YOLOv8 models using number of parameters. (right) Accuracy of YOLOv8 models using inference time per frame.

Figure 4 shows that the model with the shortest inference time per frame will achieve the lowest precision, while the model with the longest inference time per frame will achieve the highest precision [4].

B. Hardware

The project not only compares performance between each of the YOLO algorithms but also between each of the CPUs used. It used the Intel Core i7 and Core i9 processors for experimentation. The Intel Core i7 processor used was the Core i7-10750H, while the Core i9 processor used was the Core i9-11900H.

Before beginning the project, the group hypothesized that the Intel Core i9 processor performs at a higher level due to its reputation as Intel's ”highest end processor” [5].

Because the YOLO algorithms, by default, use the graphics processing unit (GPU) during inference, the owners of the Intel Core computers set the `device` parameter equal to `'cpu'`.

III. Analysis

With the number of different YOLO models entirely along with the different processors used, there were many inference calculations. With image processing, an input image must be used for object detection to occur. Therefore, the sample media set consisted of two videos and three images. The content of the videos used were traffic in Sheffield, England for one and a football match segment for the other. The images consisted of a living room interior design with different furniture, a set of domestic cats and dogs on a couch, and bikers on an open street in Denver.

These images had many different objects. The object detection was only measured with YOLO versions 5 and 8, since we used multiple models for those.

A. YOLOv5

The project uses two of YOLOv5's models, `yolov5n6.pt` and `yolov5x6.pt`, to examine its performance.

Looking at Table I, there is hardly any evaluation on which Intel CPU would perform more efficiently. Due to the simplicity and haste of the `yolov5n6.pt` model, the core processors in question were not able to be put in proper use. It is too rapid for any notable difference.

When using a larger-scale model for YOLOv5, as shown in Table II, deviation begins between the two CPUs. Despite

TABLE I
INFERENCE TIME (MS/IMAGE) USING MODEL **YOLOV5N6.PT.**

	Intel Core i7	**Intel Core i9**
Traffic in Sheffield	128.1	53.6
Football Segment	61.6	62.5
Cats and Dogs	96.0	102.2
Denver Open Streets	72.0	71.8
Living Room	72.0	73.1

TABLE II
INFERENCE TIME (MS/IMAGE) USING MODEL **YOLOV5X6.PT.**

	Intel Core i7	**Intel Core i9**
Traffic in Sheffield	715.1	695.5
Football Segment	693.3	703.5
Cats and Dogs	1093.9	840.1
Denver Open Streets	721.7	584.1
Living Room	789.1	601.3

the Core i9 taking longer than the Core i7 for the Football Segment entry, it takes a notably shorter time in the other samples.

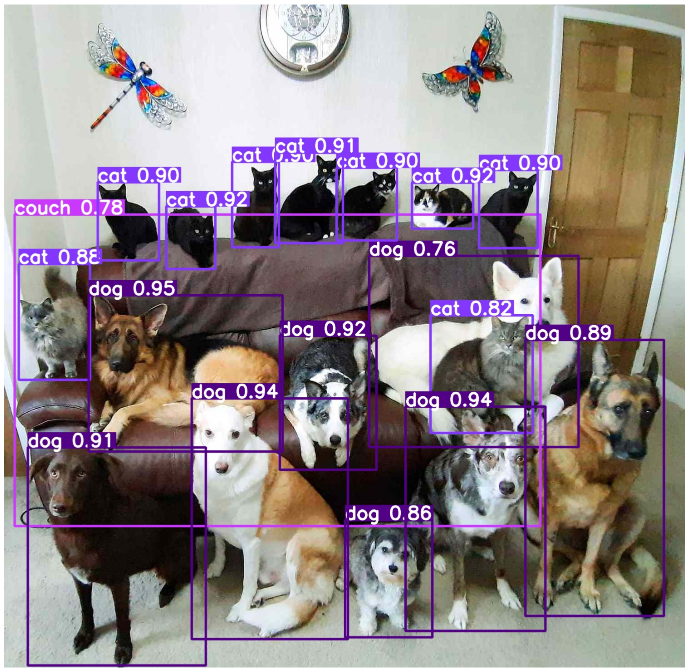

Fig. 5. Sample of cats and dogs object detected using YOLOv5's largest model.

Figure 5 shows the accuracy that YOLOv5 had achieved using its largest model. As the image consisted of 9 cats and 8 dogs, the extra large model, `yolov5x6.pt`, identified them each with high confidence.

However, if YOLOv5 uses its smallest model, `yolov5n6.pt`, as in Figure 6, the algorithm will be drastically less precise. Figure 6 demonstrates the low average precision measured, as it identified many dogs as sheep.

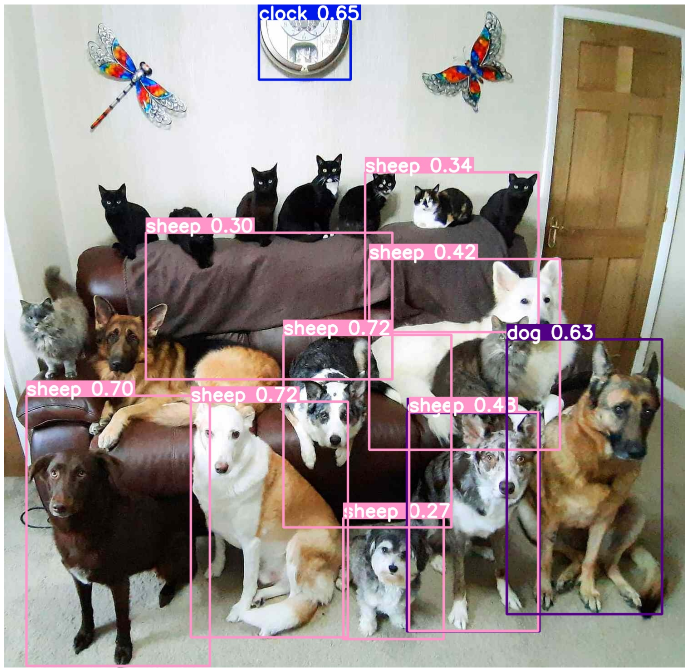

Fig. 6. Sample of cats and dogs object detected using YOLOv5's smallest model.

B. *YOLOv7*

The project tested YOLOv7 for only one model: `yolov7.pt`.

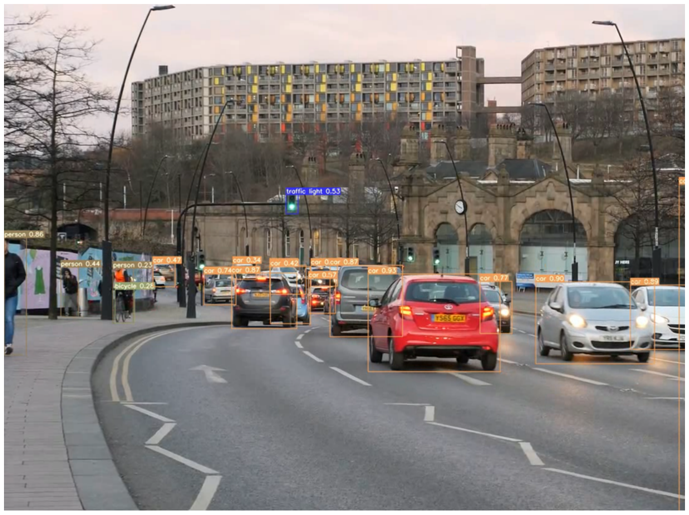

Fig. 7. A screen capture of sampled traffic video with YOLOv7 executed.

Looking at Table III, it was quick to notice the big difference between the CPU's inference times. While the Intel Core i7 took at average, half of a second, to detect objects, the Intel Core i9 took a tenth of a second, at most. Along with comparing performances of the CPUs in question, the demonstration also includes comparisons in appearance.

Figure 7 shows a frame capture of the objected-detected video using YOLOv7. From the method Version 7 uses to box detected classes, the project evaluated that YOLOv7 is user-compatible, meaning that the user can analyze the detected

TABLE III
INFERENCE TIME (MS/IMAGE) USING MODEL **YOLOV7.PT.**

	Intel Core i7	Intel Core i9
Traffic in Sheffield	473.4	57.9
Football Segment	443.0	31.5
Cats and Dogs	697.6	96.4
Denver Open Streets	687.3	22.7
Living Room	440.5	21.8

objects themselves without any box labels intercepting. This is done by minimizing the size of the class confidence and name displays, in order for the users themselves to analyze.

C. YOLOv8

As mentioned in the Methodology section, this project experimented YOLOv8 using its most simplistic model, `yolov8n.pt` and most advanced model, `yolov8x.pt`.

TABLE IV
INFERENCE TIME (MS/IMAGE) USING MODEL **YOLOV8N.PT.**

	Intel Core i7	Intel Core i9
Traffic in Sheffield	69.1	56.6
Football Segment	65.9	58.8
Cats and Dogs	103.4	102.6
Denver Open Streets	80.8	71.6
Living Room	77.8	73.2

In Table IV, the inference time, in milliseconds. is displayed for each processed medium for both cases of the Core i7 and Core i9 processors.

TABLE V
INFERENCE TIME (MS/IMAGE) USING MODEL **YOLOV8X.PT.**

	Intel Core i7	Intel Core i9
Traffic in Sheffield	703.7	737.4
Football Segment	694.3	655.8
Cats and Dogs	1015.6	1009.8
Denver Open Streets	697.8	686.8
Living Room	696.5	636.4

After performing object detection with larger weight files, `yolov5x6.pt` in Table II and `yolov8x.pt` in Table V, notice that each frame processed takes half of a second at fastest. This is due to their properties of well-trained and file size. The more weight a model trains, the bigger file size it has.

IV. CONCLUSION

The You-Only-Look-Once object detection algorithm family is seen as a beneficial algorithm, as it is well-rounded in comparison to other algorithms with speed and precision. By comparing the YOLO versions, YOLOv5, YOLOv7, and YOLOv8, using five different models, the group examined the performances in inference time along with how accurate the detections are.

This project revealed the performance difference between the Intel Core i7 and i9 processors with the Intel Core i9 finishing superior, as predicted before the project. With the different model sizes, the group realized how rapid the smallest editions processed images and how accurate the largest ones did. The little differences, as small models switch to large ones, began deviating in execution time.

V. ROLES AND ACKNOWLEDGEMENTS

Youssef and Ali had worked on downloading the YOLO algorithm versions needed on Youssef's Intel Core i9 central processing unit. Michael and Javier had also processed samples using the same YOLO versions but on Michael's Intel Core i7 CPU instead. After execution, Michael was given Youssef's samples from the i9 processor to enter the data into tables. He then compiled the data and analyzed it throughout the paper.

REFERENCES

[1] P. Soviany and R. T. Ionescu, "Optimizing the trade-off between single-stage and two-stage object detectors using image difficulty prediction," arXiv [cs.CV], 2018.
[2] G. Jocher, "YOLOv5 by Ultralytics," GitHub, https://github.com/ultralytics/yolov5 (accessed Nov. 27, 2023).
[3] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv.org, https://arxiv.org/abs/2207.02696 (accessed Nov. 27, 2023).
[4] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," GitHub, https://github.com/ultralytics/ultralytics (accessed Nov. 27, 2023).
[5] "Processor guide," UPenn ISC, https://www.isc.upenn.edu/how-to/processor-guide (accessed Nov. 27, 2023).