**ADVANCES IN CYBER-PHYSICAL SYSTEMS**

**Vol. 9, No. 2, 2024**

# PERFORMANCE ANALYSIS OF DIFFERENT TYPES OF NN MODELS FOR TARGET RECOGNITION

***Bohdan Tsiunyk, Oleksandr Muliarevych***

*Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine*
Authors' e-mails*: bohdan.s.tsiunyk@lpnu.ua, oleksandr.v.muliarevych@lpnu.ua*

https://doi.org/10.23939/acps2024.02.101

*Submitted on 26.08.2024.*



***Abstract*: The objective of this research is to conduct a comprehensive performance analysis of various types of neural network (NN) models for target recognition. Specifically, this study focuses on evaluating the effectiveness and efficiency of yolov8n, yolov8s, yolov8m models in target recognition tasks. Leveraging cutting-edge technologies such as OpenCV, the research is aimed at developing a robust methodology for assessing the performance of these NN models. Through meticulous analysis, this study aims to provide insights into the strengths and weaknesses of each model, facilitating informed decision-making for practical applications[1]. This paper presents the process of designing and conducting the performance analysis. The study discusses the implications of the findings for future developments in target recognition systems.**

***Index Terms*: yolov8, YOLO, OpenCV, NN model.**

## I. INTRODUCTION

In today's world, ensuring the safety and security of our communities has become more imperative than ever. Residents' lives and general well-being depend on how well their environment is protected. This aspect stands as one of the fundamental pillars of everyone's existence, as it underpins the essence of peace and stability.

The significance of addressing home security resonates deeply in the deployment of various security measures, encompassing electronics, equipment, and other resources. In situations where there is a risk to a person's health or life, it is imperative to warn and protect those who might be vulnerable to emergencies, home invasions, or theft. Furthermore, enhancing this process to actively engage ordinary citizens in safeguarding their security and improving their community's welfare is paramount. To fulfill this mission, the development of a surveillance camera movement monitoring system has been envisioned as the core objective and primary task of the software system under consideration.

This system aims to leverage advanced technologies, particularly neural network (NN) models, to analyze and recognize targets efficiently. By conducting a comprehensive performance analysis of various NN models, including yolov8n, yolov8s, yolov8m, and YOLO, this research endeavors to provide insights into their effectiveness and efficiency in target recognition tasks.

According to ADCIS, a target recognition system [1] comprises an amalgamation of software-based utilities, components, and hardware, tailored to cater to the requirements of distinct user groups within a designated software ecosystem, all housed within a shared user repository.

The Automated Imaging Association (AIA) defines machine vision as the integration of hardware and software to guide devices in executing tasks based on image capture and processing across industrial and non-industrial applications [2]. While industrial computer vision shares algorithms with academic and governmental sectors, it faces unique constraints. Machine vision systems rely on digital sensors within industrial cameras, paired with specialized optics, to capture images for subsequent processing, analysis, and decision-making [3]. Consequently, industrial machine vision entails achieving low cost, maintaining acceptable accuracy, and ensuring high robustness, reliability, and mechanical and thermal stability. Over the years, the fruits of Artificial Intelligence (AI) research have yielded numerous small-scale benefits [4]. From ATMs reading checks autonomously to cameras auto-focusing on faces and social media auto-tagging friends based on facial recognition, AI's reach continues to expand. Consider coffee: brewing a pot only to find it bitter can be disappointing. By allowing AI to streamline crucial aspects of coffee or data consumption, we ensure every cup is perfect. With machines parsing through hours of video data, translating languages instantaneously, and executing commands seamlessly, AI proves its capability to tackle specific, time-consuming tasks. While we are yet to achieve generalized AI, modern Machine Learning (ML) displays its adeptness in addressing specific challenges.

[1] This article uses the materials and results obtained by the authors during the research work "Intelligent design methods and tools for the modular autonomous cyber-physical systems" (0124U002340), which is carried out at the Department of Electronic Computing Machines of Lviv Polytechnic National University in 2024-2028.

In recent years, there has been significant progress in the automatic analysis of visual data using computer algorithms, which is largely due to the progress of convolutional neural networks (CNN). The success of these models can be attributed in part to their biological design. The classical artificial neuron is a simplified version of its biological counterpart [5]. Furthermore, the neural computational models used are only partially inspired by biological structures and connections. In contrast, human visual analysis involves the transfer of information between cortical areas of the brain via feedforward and feedback pathways.

Computer vision systems are widely used in various areas of everyday life. They provide visual perception and modeling with the help of computing tools. These systems use single, stereo, or multiple camera setups to perform visual tasks [6]. Stereo vision systems using dual or multi-camera configurations allow complex computer vision operations to be performed. Vision systems serve as vital imaging tools used in a wide range of applications, including portable autonomous robotics, 3D measurement, object tracking, the entertainment industry, augmented reality, and object recognition [7]. These systems, dual in nature and dependent on multiple visions, play a key role in enhancing spatial perception and facilitating a variety of visual tasks in different sectors.

## II. LITERATURE REVIEW AND PROBLEM STATEMENT

Python stands out as a programming language that embodies simplicity and power, making it a favorite choice among developers. Its intuitive nature allows users to focus on solving problems rather than struggling with complex syntax and structure. The official introduction to Python emphasizes its simplicity and efficiency, describing it as a powerful programming language with high-level data structures and a simple approach to object-oriented programming. The elegance of Python lies in its minimalist design, reminiscent of the English language with strict rules. This pseudocode-like quality simplifies problem-solving by emphasizing functionality over complexity. Its user-friendly syntax makes it extremely accessible even for beginners. In addition, Python's status as free and open-source software (FLOSS) emphasizes its collaborative nature, fostering a community focused on continuous improvement. With Python, developers are free from worrying about low-level details like memory management owing to its open-source framework. Python's versatility extends to many platforms, including GNU/Linux, Windows, macOS, and others, making it a ubiquitous tool for software development. Its adaptability even covers gaming platforms such as PlayStation and mobile devices such as iPhone, iPad, and Android, further cementing its status as a universal and widely accessible programming language [8].

A program written in a compiled language such as C or C++ goes through a process where it is translated from source code into binary code (0s and 1s) that a computer can understand, using a compiler along with various flags and parameters. Subsequently, the linker / loader software moves the program from the hard disk into memory and begins its execution after startup. In contrast, Python does not require binary compilation. Instead, you can directly execute the program from its source code. Internally, Python translates source code into an intermediate form known as bytecodes, which is then converted to the computer's native language before execution. This simplified process removes the worry of compiling your application and managing library dependencies, increasing Object-oriented programming, on the other hand, revolves around objects that encapsulate both data and functionality. Despite its simplicity, Python provides a robust framework for object-oriented programming, offering a powerful yet simple approach compared to larger languages such as C++ or Java.

If there is an important piece of code that needs to be executed quickly, or if you want to keep certain algorithms proprietary, you can develop that particular part of your program in C or C++ and easily integrate it into your Python code base. Python's appeal lies in its exciting combination of performance and functionality, which makes the process of writing Python programs both enjoyable and easy. The benefits of 3D vision-based technology extend to numerous fields, including 3D shape measurement, visual inspection, medical imaging, robot control, and more. It serves as a core component of industrial camera calibration to obtain internal and external parameters for further measurement or detection tasks. The accuracy of these measurements depends on the accuracy of the camera calibration, especially in precision industrial applications. Common camera calibration methods such as direct linear transform (DLT), Tsai's method, and Zhang's planar calibration method are used to effectively reconstruct local or full 3D profile information. Although DLT creates a basic camera perspective model using 3D points in space, it often does not account for lens distortion. Tsai's method, on the other hand, involves radial alignment constraints to calculate external camera parameters, followed by nonlinear optimization to account for radial distortions. The Zhang method, known for its flexibility, imposes minimal constraints on the spatial position of the calibration object and is favored for its adaptability and efficiency in camera calibration tasks [9].

OpenCV (Open-Source Computer Vision Library) is a freely available open-source software library designed for computer vision and machine learning tasks. It was started with the goal of creating a unified platform for computer vision applications and accelerating the integration of machine perception into commercial products. With the Apache 2 license, OpenCV offers companies a straightforward way to use and modify their code base. With over 2,500 optimized algorithms, the library contains a wide range of both traditional and state-of-the-art computer vision and

machine learning algorithms. These algorithms enable users to perform a variety of tasks such as face detection and recognition, object identification, human action classification in video, camera motion tracking, object motion tracking, 3D object modeling, 3D point cloud creation based on a stereo camera, and combining images for panoramic scenes, searching for image similarities from databases, eliminating red-eye when shooting with flash, tracking eye movements, scene recognition, and setting markers to influence the algorithm's behavior. With a thriving user community of over 47,000 and estimated over 18 million downloads [10], OpenCV has become a cornerstone tool used by companies, research institutions, and government agencies around the world. Most computer vision programs involve image input and output. Interactive applications may require a camera as an input source and a display window as a destination. Image files, video files, and raw byte data can serve as input and output sources, enabling a wide variety of application scenarios, including network transmission or procedural graphics generation.

The last and crucial step in application development is testing. It enables us to detect and fix common bugs early in the development process. Typically, testers perform this type of work. Testers are those who oversee application testing. They should simulate all possible user workflow scenarios. It allows you to go through all the functions that the application offers, testing them all.

There are two main types of testing: manual and automated. However, the disadvantage is that the development of automated tests usually requires a significant investment of development time and resources. On the other hand, manual testing involves testers running the tests. Although manual testing takes more time, it is highly regarded for its flexibility, reliability, and security. It is widely used throughout the development process, allowing developers to immediately test newly implemented features. Performance testing of the developed program requires special tools. One such tool is Desktop Studio Profiler, which facilitates the monitoring of CPU resource usage, RAM consumption, network activity, and its impact on battery life [11]. The main characteristic of such NN model for target recognition is the amount of memory, which should be from up to 20 GB of total memory allocation.

## III. SCOPE OF WORK AND PROBLEM STATEMENT

In the context of target recognition systems, there is a critical need to thoroughly analyze the performance of different types of neural network (NN) models. The main topic of this research is to conduct an in-depth study of the effectiveness and efficiency of different mesh network architectures for target recognition tasks. This requires a systematic evaluation of the performance characteristics of NN models, including factors such as accuracy, resource utilization, and computational efficiency.

The problem under consideration is the selection and optimization of NN models to achieve superior performance in target recognition applications. Given the variety of mesh network architectures available, a complete understanding of their comparative performance in real-world scenarios is lacking. Therefore, the main goal of this study is to address this gap by conducting a detailed analysis of mesh network models, focusing on their performance and suitability for target recognition tasks.

The main challenges include optimizing resource allocation, such as memory usage, and CPU load, to ensure efficient performance of NN models in target recognition systems. In addition, benchmarks should be established to evaluate performance, considering factors such as detection accuracy, processing speed, and scalability in different environments. By systematically evaluating and comparing the performance of different NM models, this study aims to provide valuable information that will inform decision-making processes in the selection and deployment of NM-based target recognition systems. The goal is to identify the most effective and efficient NN model for target recognition, which will contribute to the advancement of surveillance, security, and related fields.

The purpose of this work is to systematically evaluate and compare the effectiveness and efficiency of various neural network (NN) models in the domain of target recognition. The aim is to identify the most resource efficient and effective NN model for target recognition tasks. This research endeavors to provide valuable insights into the performance characteristics of different NN architectures, such as yolov8n, yolov8s, and yolov8m, with a focus on their ability to recognize targets accurately and efficiently in diverse environments. Through meticulous analysis and experimentation, the objective is to inform decision-making processes regarding the selection and deployment of NN models in real-world target recognition applications, ultimately contributing to the advancement of surveillance and security systems.

A critical consideration for the system entails program optimization, particularly due to the intensive computations and video stream rendering involved RAM usage, memory allocation and CPU load often pose challenges in such scenarios. Typically, the standard NN model for target recognition allocates memory up to 20 GB. Additionally, swift calculation times are essential, with the detection of moving objects typically accomplished within 0.1ms.

The result of the research work is to find the best solution NN model that meets all the requirements above and returns the best result of target recognition.

## IV. ALGORITHM FOR CONVERTING VIDEO BASED ON NN MODEL THREAD INTO CV OBJECT

The main goal and guiding principle of the development of this software system was the use of modern technologies that are popular today for such

implementations. Python was chosen as the programming language for the development of the entire system, and PyCharm IDE serves as the development environment.

After carefully studying the advantages and disadvantages of different technologies, it was decided to adopt the following technology stack for implementing neural network (NN) models: Python, OpenCV, YOLO, and PyCharm.

This algorithm describes the steps to convert a neural network model stream-based video into OpenCV objects for target recognition. It involves continuously processing frames from an input video stream, using an NN model for target recognition, and converting the results into CV objects for visualization and analysis (see Fig. 1).

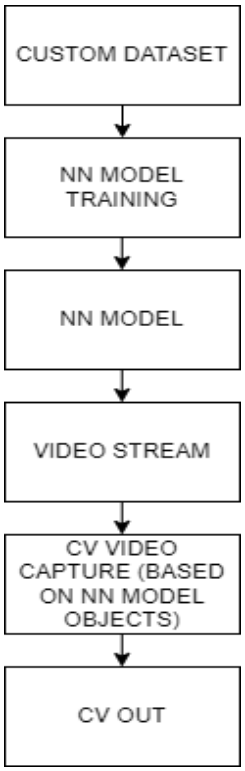


*Fig. 1. Video thread converting scheme*

Finally, an integral aspect of program development is testing. It is noteworthy that the testing phase is of primary importance in the development process, allowing the identification and correction of common errors during the development stage. Performance testing is used to ensure optimal functionality and performance.

## V. CREATION CUSTOM DATASET BASED ON REAL PHOTOS AND IMPORT IT TO THE NN MODEL TRAINING

Creating a custom dataset based on real photos and importing it into a neural network (NN) model training process is a crucial step in analyzing the performance of several types of NN models for target recognition.

Start by collecting a diverse set of real-world photos that match target recognition tasks. Ensure that the images cover a wide range of scenarios and conditions that the NN model is expected to encounter in real-world applications. Use the annotation tools to mark objects or targets of interest in each image.

Assign appropriate labels or classes to annotated objects to facilitate supervised learning during training. Resize all images to a standardized size of 640x640 pixels to ensure uniformity and compatibility with the NN model architecture.

Additionally, perform image pre-processing as needed, such as normalization or augmentation, to improve model reliability and generalization capabilities. Divide the annotated images into training, validation, and testing sets. Dedicate a sizable portion of the training dataset to ensure sufficient access to a variety of examples. The validation set is used to tune the hyperparameters and estimate the model, while the test set evaluates the final performance of the model on unseen data.

Organize annotated images and corresponding labels in a structured format compatible with popular deep learning frameworks such as TensorFlow or PyTorch. This usually involves creating separate directories for the training, validation, and test sets, with each directory containing subdirectories for different classes or labels. Implement data loaders or generators to efficiently load image and label packets during the training process. Ensure that the data loading pipeline is optimized for performance to prevent bottlenecks during model training. Train the NN model with a custom dataset created from real photos. Use state-of-the-art architectures such as YOLO (You Look Only Once) for efficient and accurate target recognition (see Fig. 2).

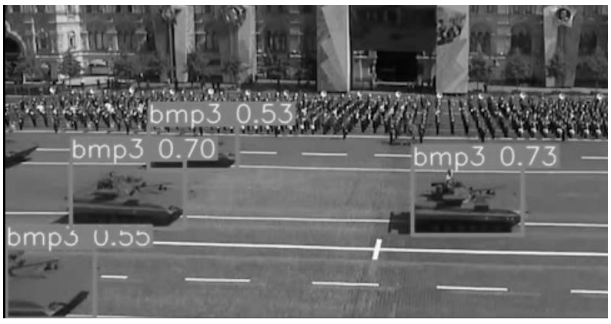


*Fig. 2. Visualization of YOLOv8S model movement target recognition status detection*

Fine-tune model parameters and architecture based on performance metrics obtained during training. Evaluate the performance of the trained model on validation and test sets to evaluate its accuracy, precision, recall, and other relevant metrics. Repeat the training process as it is needed to improve the model's performance and generalization capabilities.

The metrics based on different yolov8 models for target recognition are presented in Fig. 3.

## VI. FINE-TUNING THE YOLOV8 NN MODEL BASED ON METRICS FOR BEST PERFORMANCE RESULTS OF TARGET RECOGNITION

Start by selecting appropriate performance metrics to evaluate the performance of the YOLOv8 model for target recognition. Common measures include precision, accuracy, recall, F1 score, and mean average precision (mAP).

These metrics provide insight into the model's ability to accurately detect and classify targets in a variety of environments. Train the baseline YOLOv8 model using a standardized dataset and default hyper-parameters. This serves as an initial benchmark for evaluating model performance before fine-tuning.

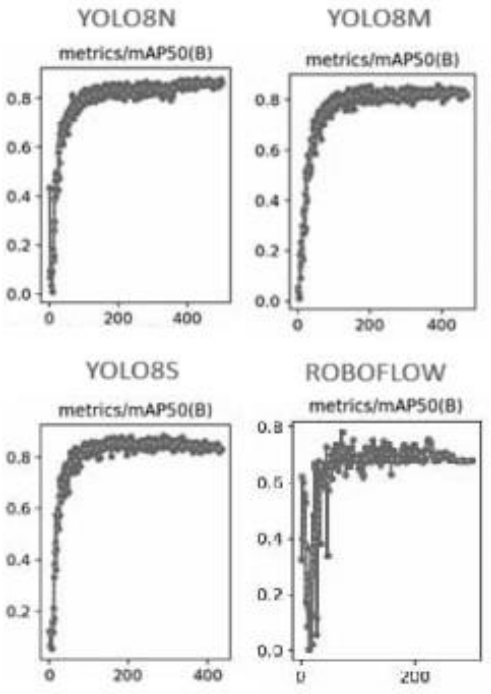


*Fig. 3. NN models YOLOv8 metrics*

Evaluate the performance of the base model using selected metrics. Identify areas for improvement and specific target recognition scenarios where the model may fall short. Fine-tune the hyperparameters of the YOLOv8 model to optimize its performance for target recognition.

This may include adjusting parameters such as learning rate, batch size, optimizer selection, binding block sizes, and expansion methods. Implement data augmentation strategies to improve the model's ability to generalize unseen data and improve performance in a variety of target recognition scenarios.

Common zoom methods include random cropping, rotation, scaling, and color variation. Use transfer learning techniques to initialize the YOLOv8 model with pre-trained weights from a larger dataset or related task. Fine-tune the model on the target recognition dataset to adapt its features to the specific characteristics of the target objects.

Use cross-validation techniques to assess the robustness of the fine-tuned YOLOv8 model on different subsets of the dataset. This helps to ensure consistency and robustness of the model across different data distributions.

Evaluate the performance of the tuned YOLOv8 model using selected validation metrics and test datasets. Compare the results to the baseline model to quantify improvements in target recognition accuracy and other relevant metrics. Iterate through the fine-tuning process, adjusting hyperparameters, expansion strategies, and other factors based on observed performance. Continuously monitor the performance of the model and improve its configuration to achieve optimal results.

Document the fine-tuning procedure, detailing chosen hyperparameters, expansion methods, and performance metrics. Prepare a detailed report with the conclusion and improvements achieved by fine-tuning the YOLOv8 model for target recognition.

## VII. ALGORITHM FOR TRAINING NN MODELS

Compile and preprocess a diverse image dataset containing relevant target objects for a recognition task. Add bounding boxes to the dataset to specify the location and class labels of the target objects. Divide the dataset into training, validation, and testing sets for model evaluation. Select the prebuilt YOLOv8 (YOLO version 8) model as the base architecture.

Include pre-trained weights in the model to benefit from learned features from a large dataset. Establish a fine-tuning approach to adapt the pre-trained YOLO model to the target recognition task. Determine if you need to immobilize certain layers or perform end-to-end training. Select the appropriate loss function configured for the target recognition task. Adjust the loss function to emphasize certain aspects of target recognition. View mini-packages of images and corresponding annotations from the training set.

Do a forward pass: feed the image to the YOLO model and calculate the predicted bounding boxes and class probabilities. Calculate the loss between predicted and actual annotations. Perform a backward pass: Review the model weights using gradient descent optimization. Repeat the training process for several epochs. Refine hyperparameters such as learning rate, batch size, and optimizer parameters. Conduct an experiment to determine optimal hyperparameter configurations. Evaluate the trained YOLO model on the validation set. Evaluate performance measures such as accuracy, precision, recall, and mean average precision (mAP). Analyze performance in a variety of target recognition scenarios. Conduct a comprehensive performance analysis to compare the performance of different YOLO variants. Measure metrics like inference speed, memory usage, and accuracy. Use a learning algorithm based on the information obtained from the performance analysis (see Fig. 4).

In the first line, import the YOLO class from the Ultralytics library, which allows you to use the YOLO object for object detection tasks. Then initialize a new YOLO object called model using the specified "yolov8.yaml" configuration file. It loads the configuration settings for the YOLOv8m model. The training process for the YOLO model uses the specified training data configuration file "config.yaml" and trains it for 500 epochs. It updates the weights and parameters of the model based on the training data to improve its performance in object detection.

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8m.yaml")  # build a new model from

# Use the model
results = model.train(data="config.yaml", epochs=500)
```

*Fig. 4. Using the YOLOv8 model with Python script*

## VIII. NN MODEL TESTING

For evaluating the current system, manual testing was chosen due to its flexibility and reliability, allowing for comprehensive bug identification and resolution. Performance testing utilized the Studio Profiled instrument integrated within the IDE, eliminating the need for external tools.

To assess the application's performance thoroughly, stress testing was conducted, aiming to gauge system stability under maximum load. This testing type involves executing operations that push application resource usage to its limit.

During stress testing, resource-intensive operations included calculating the location of moving objects within the entire image and live frame conversion from the video stream. The maximum CPU load recorded was 75 %, with RAM usage peaking at 20 GB (see Fig. 5). Additionally, the frame transformation rate reached up to 0.0125 milliseconds per frame. The speed and efficiency of these models were almost identical, so the overall efficiency comes down to resource efficiency.

CPU efficiency can be found by subtracting the percentage of CPU load from 100 %:

$$E_{Ci} = 100\,\% - L_{CPUi}, \quad (1)$$

where $L_{CPUi}$ is CPU load of model *i*. By using CPU load percentages in Fig. 5: $E_{C8n} = 100–62 = 38$ %, accordingly: $E_{C8m} = 100–67 = 33$ %, in much the same way: $E_{C8s} = 100–76 = 24$ %.

Memory efficiency can be found in an equivalent way:

$$E_{Mi} = 100\,\% - L_{MEMi}, \quad (2)$$

where $L_{MEMi}$ is percentage of used memory of model i. So $E_{M8n} = 100–52 = 48$ %, and $E_{M8m} = 100–58 = 42$ %, finally: $E_{M8s} = 100–49 = 51$ %.

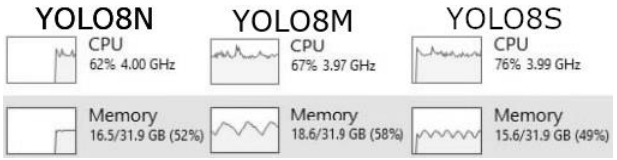


*Fig. 5. NN models resource usage test in stress mode*

Comparative efficiency calculated similarly, by subtracting usage efficiency from one another:

$$E_{ij} = E_i - E_j, \quad (3)$$

where $E_i$ is efficiency of model *i* and $E_j$ is efficiency of model *j*. By using test data in Fig. 5: $E_{C8nm} = 38–33 = 5$ %, so yolo8n is 5 % more CPU efficient than yolo8m. In the same way $E_{C8ns} = 38–24 = 14$ %, and $E_{C8ms} = 33–24 = 9$ %. As for memory usage: $E_{M8nm} = E_{M8n} – E_{M8m} = 48–42 = 6$ %, $E_{M8ns} = 48–51 = –3$ %, and $E_{M8ms} = 42–51 = –9$ %, so yolo8m is 9 % less memory efficient than yolo8s. Comparative efficiency is bidirectional, so yolo8s is also 9 % more memory efficient than yolo8m. This way, those values are all we need to compile a complete dataset.

Comparative resource usage efficiency can be found by averaging CPU and memory efficiencies:

$$E_{ij} = \frac{E_{Cij} + E_{Mij}}{2}, \quad (4)$$

where $E_{Cij}$ is comparative CPU efficiency of model *i* over model *j* and $E_{Mij}$ is comparative memory efficiency of model *i* over model *j*. With that $E_{8nm} = (5+6)/2 = 5.5$ %, $E_{8ns} = (14+3)/2 = 5.5$ %, $E_{8ms} = (9+–9)/2 = 0$ %, so yolo8m and yolo8s have same overall effectiveness, while yolo8n is 5.5 % more overall effective than the other two.

## IX. CONCLUSION

As a result of this work, these efforts culminated in a comprehensive analysis of various neural network (NN) models for target recognition. In particular, yolov8n, yolov8s, and yolov8m were thoroughly tested to determine their performance in this area.

The speed and efficiency of these models were almost identical, so the overall efficiency comes down to resource efficiency. Yolov8n overall was 5.5 % more resource-effective than other options. It also has 5 % better CPU efficiency over yolov8m $E_{C8nm} = 5$ %, and 14 % over yolov8s, $E_{C8ns} = 14$ %. Yolov8m uses way too much memory for the minor improvement in CPU efficiency: $E_{C8mn} = –5$ %, $E_{C8ms} = 9$ %, $E_{M8mn} = –6$ %, $E_{M8ms} = –9$ %, and has the same overall efficiency as yolo8s. While yolo8s is the most memory-efficient option, that loses in CPU efficiency $E_{M8sm} = 9$ %, $E_{M8sn} = 3$ %, $E_{C8sm} = –9$ %, $E_{C8sn} = –14$ %.

Among these models, yolov8s proved to be the optimal choice for this research, primarily due to its memory-efficient nature. Because the tests were done on a general-purpose CPU, the difference in CPU usage was greater than it would be on a specialized CPU with better

multithreading. Conversely, an upgrade in memory type would not impact the total memory consumption. Hence, YOLOv8 was deemed the optimal choice, over yolo8n that had 5.5 % better overall efficiency. In parallel with the development of motion detection and environmental tracking systems, the use of advanced technologies ensured relevance and ease of maintenance for future iterations. Adopting a cross-platform approach and using Python frameworks contributed to flawless integration and user interaction. The integration of OpenCV services has extended the functionality, enabling real-time authorization and monitoring.

A key aspect of the system's design was its adaptability, allowing users to customize their monitoring and tracking configurations. This flexibility emphasizes the system's usefulness and user-oriented approach. In addition, strict optimization measures have been implemented to eliminate resource constraints, particularly in terms of RAM usage when rendering the video stream.

The current NN models meet the requirements for standard target recognition models. In general, all yolov8 models meet the basic requirements based on memory usage but the most efficient for this task is the yolov8s type.

## References


[1] Das, S., Saha, S., Coello Coello, C. A., Bansal, J. C. (2023). "Deep Neural Network Based Performance Evaluation and Comparative Analysis of Human Detection in Crowded Images Using YOLO Models". In *International Conference on Advances in Data-Driven Computing and Intelligent Systems. ADCIS. Lecture Notes in Networks and Systems, vol. 893. Springer, Singapore*, pp. 508–509. DOI: 10.1007/978-981-99-9518-9_37.

[2] Delleji, T., Slimeni, F., Fekih, H. (2022). "An Upgraded-YOLO with Object Augmentation: Mini-UAV Detection Under Low-Visibility Conditions by Improving Deep Neural Networks". *Oper. Res. Forum 3, 60*, pp. 3–5. DOI: 10.1007/s43069-022-00163-7.

[3] Tattari, J., Donthi, V. R., Mukirala, D., Komar Kour, S. (2021). "Deep Neural Networks Based Object Detection for Road Safety Using YOLO-V3". In *Smart Computing Techniques and Applications. Smart Innovation, System and Technologies, vol. 225. Springer, Singapore*, pp. 731–733. DOI: 10.1007/978-981-16-0878-0_71.

[4] Poskart, B., Iskierka, G., Krot, K. (2024). "Logistics 4.0 – Monitoring of Transport Trolley in the Factory Through Vision Systems Using the YOLO Model Based on Convolution Neural Networks", In *International Conference on Intelligent Systems in Production Engineering and Maintenance III. ISPEM 2023. Lecture Notes in Mechanical Engineering, Springer, Cham*., pp. 348–350. DOI: 10.1007/978-3-031-44282-7_27.

[5] Priyankan, K. and Fernando, T. G. I. (2021). "Mobile Application to Identify Fish Species Using YOLO and Convolutional Neural Networks." In *Proceedings of International Conference on Sustainable Expert Systems: ICSES 2020, volume 176. Springer, Singapore*, pp. 304–308. DOI: 10.1007/978-981-33-4355-9_24.

[6] Ayob, A. F., Khairuddin, K., Mustafah, Y. M., Salisa, A. R., Kadir, K. (2021). "Analysis of Pruned Neural Networks (MobileNetV2-YOLOv2) for Underwater Object Detection". In: *Proceedings of the 11th National Technical Seminar on Unmanned System Technology 2019. NUSYS 2019. Lecture Notes in Electrical Engineering, vol. 666. Springer, Singapore*, pp. 87–88. DOI: 10.1007/978-981-15-5281-6_7.

[7] A Byte of Python (2022). [Electronic resource]. Available at: https://homepages.uc.edu/~becktl/byte_of_python.pdf. (Accessed: March 29, 2024).

[8] Bansal, J. C. and Uddin, M. S. (2023). "Computer Vision and Machine Learning in Agriculture, Vol. 3". In: *Algorithms for Intelligent Systems*, pp. 120–125. DOI: 10.1007/978-981-99-3754-7.

[9] Learning OpenCV (2022). [Electronic resource]. Available at: https://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf. (Accessed: March 29, 2024).

[10] Huang, D. S., Premaratne, P., Jin, B., Qu, B., Jo, K. H. and Hussain, A. (2023). "Advanced Intelligent Computing Technology and Application". *Springer, Singapore*, pp. 83–88. DOI: 10.1007/978-981-99-4742-3

[11] Lys, R., Opotyak, Y. (2023). "Development of a Video Surveillance System for Motion Detection and Object Recognition". *Advances in Cyber-Physical Systems, 8(1)*, pp. 50–53. DOI: 10.23939/acps2023.01.050.


**Bohdan Tsiunyk** is a graduate student pursuing a Master of Science degree in Computer Engineering at Lviv Polytechnic National University. Concurrently, he serves as a senior AQA engineer at n-ix. His areas of expertise and research interests encompass computer vision, machine learning, computer engineering, and the development of Python frameworks.

**Oleksandr Muliarevych** holds a Ph. D. in Computer Systems and Components and serves as an Associate Professor at the Computer Engineering Department of Lviv Polytechnic National University. His research interests encompass a wide array of fields, including distributed highly scalable microservice systems, swarm intelligence, Internet of Things (IoT), cloud computing, parallel computing technologies, computer vision, machine learning, and applications of multi-agent systems.