DOI: https://doi.org/10.36910/6775-2524-0560-2025-60-05
УДК 621.391.7:004.8:519.876.5

**Sadovnykov Borys** [1]**,** PhD student
https://orcid.org/0009-0009-4180-2863
**Lysechko Volodymyr** [2]**,** Dr Sc., Professor
https://orcid.org/0000-0002-1520-9515
[1] Ukrainian State University of Railway Transport, Kharkiv, Ukraine
[2] Scientific Center of the Air Force Ivan Kozhedub Kharkiv National University of Air Forces, Kharkiv, Ukraine

# ADAPTIVE BEHAVIOUR TUNING OF A NEURAL NETWORK-BASED METHOD FOR MOVING OBJECT RECOGNITION IN VIDEO STREAMS

**Sadovnykov B., Lysechko V. Adaptive behaviour tuning of a neural network-based method for moving object recognition in video streams.** The article presents an improvement of the method for searching and recognizing moving objects in video streams in real time, which is based on calculating interframe differences (deltas) and using a neural classifier. A mechanism for adaptive behaviour tuning of the method depending on the characteristics of the input data is proposed, which makes it possible to increase the recognition accuracy and processing speed under changing background conditions and limited computational resources. The developed method is an evolutionary adaptive mechanism, that allows the algorithm to gradually change its processing strategies based on the collected data, forming a heat map and optimising its performance for the specifics of a particular environment. To evaluate the effectiveness, an experimental comparison of the improved method with its basic version [5] was carried out, analyzing indicators such as average frame processing time, RAM and video memory usage, CPU and GPU load, and recognition accuracy. The optimization resulted in up to a 20% increase in processing speed and a slight improvement in accuracy (~0.8%) without increasing the use of key computational resources. The experimental results confirm the feasibility of integrating the adaptation mechanism into the delta-classification method to improve its efficiency for real-time operation.

**Keywords:** video stream, computer vision, image processing, computational resources, neural networks, object recognition, adaptive algorithms, telecommunication systems.

**Садовников Б. І., Лисечко В. П. Адаптивне налаштування поведінки нейромережевого методу розпізнавання рухомих об'єктів у відеопотоці.** У статті представлено удосконалення методу пошуку та розпізнавання рухомих об'єктів у відеопотоці в реальному часі, що ґрунтується на обчисленні міжкадрових змін (дельт) та використанні нейронного класифікатора. Запропоновано механізм адаптивного налаштування поведінки методу залежно від характеристик вхідних даних, що дозволяє підвищити точність і швидкодію розпізнавання в умовах змінного фону та обмежених обчислювальних ресурсів. Розроблений метод можна інтерпретувати як еволюційно-адаптивний механізм, оскільки він дозволяє алгоритму поступово змінювати свої стратегії обробки на основі зібраних даних, формуючи теплову карту та оптимізуючи свою продуктивність для специфіки конкретного середовища. Для оцінки ефективності проведено експериментальне порівняння удосконаленого методу з його базовою версією [5], у рамках якого аналізувалися середній час обробки кадру, використання оперативної та відеопам'яті, навантаження на процесор і графічний адаптер, а також точність розпізнавання. Оптимізація забезпечила приріст швидкості обробки до 20 % та незначне підвищення точності (~0,8 %) без збільшення використання основних обчислювальних ресурсів. Отримані експериментальні результати підтверджують доцільність інтеграції механізму адаптації у метод дельта-класифікації для підвищення його ефективності роботи в режимі реального часу.

**Ключові слова:** відеопотік, комп'ютерний зір, обробка зображень, обчислювальні ресурси, нейронні мережі, розпізнавання зображень, адаптивні алгоритми, телекомунікаційні системи.

## Statement of a scientific problem.

Computer vision systems are becoming increasingly relevant today due to their ability to automatically analyse visual information in real time. They are being actively implemented in important applications ranging from industrial process control to urban video surveillance. For example, object recognition algorithms are used on production lines to detect product defects before they are packaged, which improves quality and reduces inspection costs. In urban infrastructure, computer vision is used to automatically count traffic flows, detect violations at intersections, or identify abandoned objects in public places.

Such systems are based on the tasks of detecting, segmenting and classifying objects in static images or sequences of frames. These tasks are complicated by variable lighting conditions, different viewing angles, scales, and partial overlaps of objects. To ensure stable operation in such dynamic conditions, it is necessary to take into account not only the algorithm itself, but also the mechanisms for adapting to the input data.

Optimisation of such systems can be achieved by implementing adaptive mechanisms that allow the algorithm to automatically change key aspects of its behaviour depending on the characteristics of the current video stream, such as noise level, contrast or scene dynamics. This allows for faster detection speeds while maintaining accuracy, which is especially important for use in embedded or resource-limited devices.

As a result, further analysis of such optimisations for use in the object search and recognition algorithm becomes relevant. The most interesting area is the possibility of ignoring regions that are guaranteed to be free of objects during processing.

**Research analysis.**

The current optimisation uses the concepts of adaptive-evolutionary [8] methods, namely the formation of a separate algorithm depending on the data it works with.

A review of modern approaches to video stream processing [1-15] confirms the high efficiency of convolutional neural networks (CNNs) in object recognition tasks based on visual information. Although studies such as [1, 11, 15] demonstrate excellent accuracy results, especially on large datasets, most of them focus on static image processing or require significant computing resources, which makes them difficult to use in real-time and on systems with limited capabilities.

Real-time methods, such as SSD (Single Shot MultiBox Detector) [4] and various optimised versions of YOLO, provide high performance, but often place a significant load on the GPU and consume large amounts of video memory, which limits their suitability for environments with low computing power [9].

Several papers [3, 9, 13] offer comprehensive solutions that simultaneously detect, track, and classify objects in video, but only a few approaches provide for optimisation at the initial stages, for example, through the use of interframe comparison. For example, [8] describes a powerful automated surveillance system, which, however, does not contain special mechanisms to reduce computational costs.

It is important to note that methods based on the analysis of inter-frame changes (delta analysis) have proven to be easy and effective tools for pre-processing before classification. Works [3, 7, 14, 17] demonstrate that comparing successive frames at the pixel level allows for accurate detection of moving objects with a significant reduction in computational load. In particular, [3] presents an improved differentiation method that minimises the impact of noise and lighting changes, further confirming the feasibility of using such approaches for real-time systems.

At the same time, advances in morphological image processing [10] contribute to improving the quality of the regions extracted by delta filtering. The combination of these morphological transformations with neural classifiers, as shown in [6, 7], allows achieving an optimal compromise between speed and recognition accuracy.

At the same time, existing reviews of thresholding and segmentation methods [2, 5] often do not take into account the peculiarities of the temporal dynamics of video streams, such as background changes, noise, or partial overlap of objects, which reduces their effectiveness in practical monitoring and security tasks.

As a result, the analysed methods do not provide a universal solution that combines high performance with low resource consumption. This highlights the urgency of developing a new approach that integrates inter-frame change analysis (delta filtering) with adaptive neural classification specifically optimised for real-time operation on hardware platforms with limited resources.

**The purpose of the work.**

The purpose of this work is to improve a new method for searching and recognising objects in a real-time video stream [5], and to experimentally compare the optimised method with the original one.

**Presentation of the main material and substantiation of the obtained research results.**

There are many approaches to optimising algorithms, the main of which is data preprocessing to minimise the operations that the algorithm must perform by reducing the amount of input data. On the other hand, optimisation can be achieved by using faster and less resource-intensive operations in the implementation of the method.

A potential way to improve the speed and accuracy of the method under study is to reduce the number of image areas to be recognised by cutting off those that cannot statistically contain an object. For example, if a part of the image is occupied by a wall, which cannot contain any interesting motion, then this part of the image can be ignored during processing. Looking at it from the other side, we have a heat map with areas where objects are likely to appear, and only these areas will be processed.

By reducing the number of areas of the image that are processed, the overall processing time of the frame is reduced. Since the parts of the frame where no object can be detected are not analysed, the probability of false positive detections decreases, which increases the recognition accuracy.

The classification mechanism can vary, namely, it can be either a neural network-based classifier or an alternative approach using linear algorithms. In the current implementation of the method, a classifier based on the MobiNet architecture is used, despite the high speed of the model, reducing the number of calls reduces the time for object recognition. This also makes it possible to use a more accurate but slower classifier, while keeping the overall recognition time within the range favourable for real-time work.

The integration of the data adaptation mechanism into the recognition method will bring two key changes to its practical application.

First of all, before using the optimised version of the algorithm, it is necessary to train it on actual data. The time required may vary depending on the data, so it cannot be set in advance. To solve this problem, the algorithm can display the regions that are included in the recognition procedure during the training mode, so the operator can manually disable the training mode.

The next change relates directly to the recognition algorithm: areas that have not been marked as potential object locations are ignored. Thus, the expensive neural network classification operation will be used only on regions with the highest probability of an object being located in them.

Figure 1 shows a block diagram of the data adaptation process.

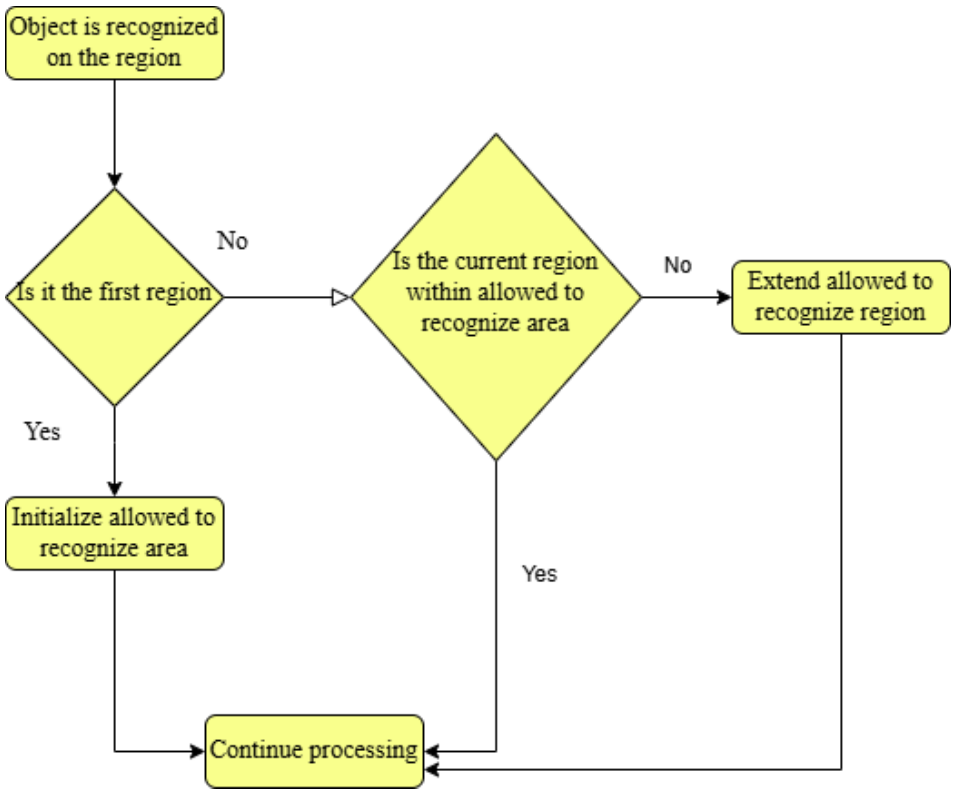
Object is recognized on the region
Is it the first region
No
Is the current region within allowed to recognize area
No
Extend allowed to recognize region
Yes
Initialize allowed to recognize area
Yes
Continue processing

Fig.1 – The process of adapting to data

The first step is to obtain information about the region where the presence of the object is confirmed. Successful classification and obtaining information about the object's class is a sufficient sign of the object's presence. The current region is passed to the input of the adaptation algorithm.

If the region being processed is in the first adaptation cycle, its position becomes the initial area for which the classifier can be applied (hereinafter referred to as the heat map) and will be expanded in subsequent iterations.

Accordingly, if adaptation is already in progress, the algorithm finds whether the current region is in the middle of the existing heatmap, and if not, the algorithm expands the heatmap to include the current region. After processing one region, the algorithm similarly processes all other regions with objects.

As a result, a heat map will be obtained in the form of a single area in which the classifier is allowed to use.

A more efficient solution in terms of the speed of the object detection and classification method may be to have a heat map in the form of many unconnected areas in which objects have been accurately detected. However, this excludes scenarios where an object moves from one area to another and is located between these two areas. This object will not be found because the space between these two areas is excluded from the heat map. Therefore, to balance accuracy and speed, it makes sense to generate the heat map as one continuous area.

Figure 2 shows the recognition method adapted to these changes. Training on the target data has already been performed and a heat map of areas with potential objects has been generated.

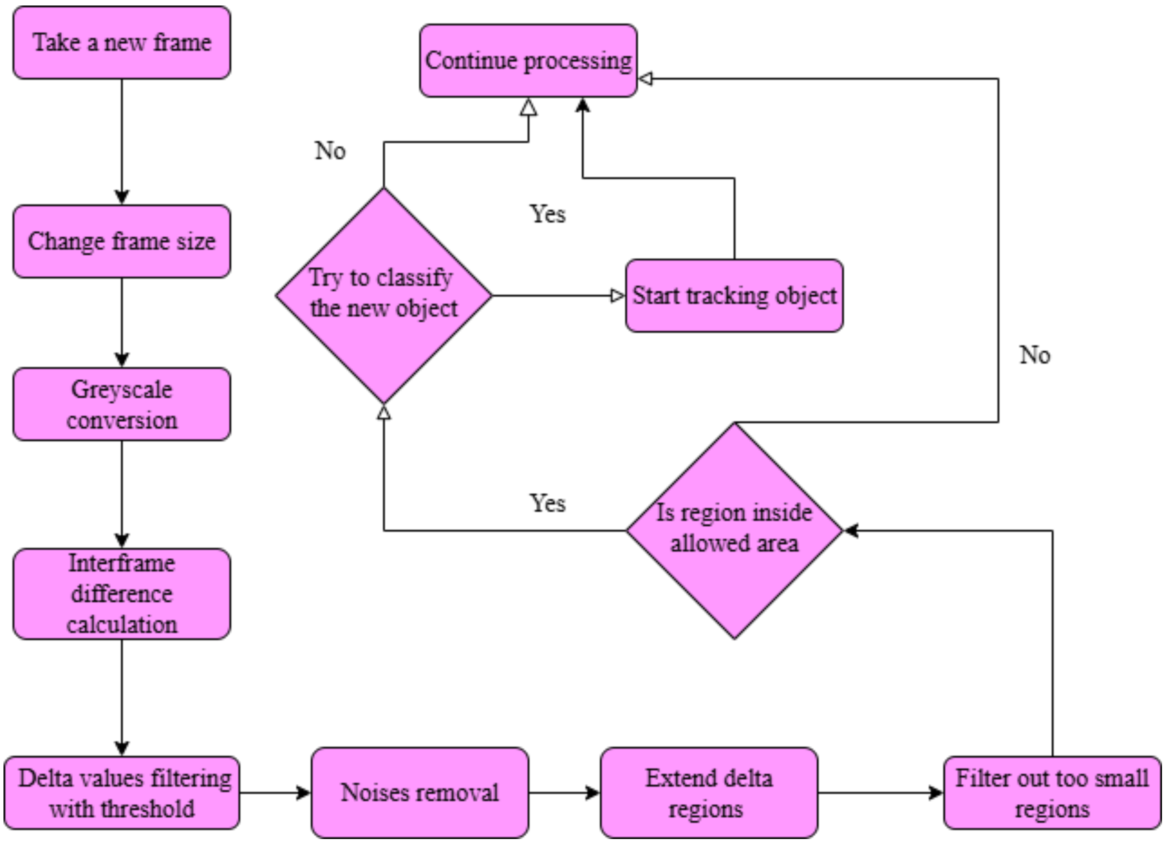
Take a new frame
Change frame size
Greyscale conversion
Interframe difference calculation
Delta values filtering with threshold
Noises removal
Extend delta regions
Filter out too small regions
Is region inside allowed area
Yes
No
Try to classify the new object
No
Yes
Continue processing
Start tracking object

Fig.2 – Flowchart of the final version of the detection algorithm

The first stage of the proposed algorithm is to bring the image to a fixed resolution of 1280×720 (720p), which provides a single standard for all further calculations. The speed of linear transformations directly depends on the number of pixels in the frame [7], as an increase in image size leads to a proportional increase in processing time. Most modern neural networks require an image of a certain dimension as input not only to improve performance, but also due to architectural limitations. For example, SSD works with images of 300×300 pixels.

In addition, normalising images to a single scale helps to stabilise processing time regardless of the original video resolution. Without such normalisation, the algorithm's performance would fluctuate depending on the frame size in the input stream, which would make it difficult to compare performance with other approaches and to use in real-world scenarios where video parameters can change. Setting a constant frame size also serves as an additional performance management tool: if necessary, you can set a lower input resolution to speed up processing with limited hardware resources.

The next step is to reduce unnecessary colour information by converting the image to grayscale. This solution often has a positive effect on the accuracy of the algorithm, as it allows you to focus on the structural and textural features of objects, eliminating distracting colour artefacts. In addition, switching to a grey image significantly optimises the use of resources. In the standard colour model, each pixel stores three values (for the R, G, B channels), while in grayscale, it stores only one. This significantly reduces the amount of data to be processed, facilitates calculations, reduces CPU load and saves memory. This solution is especially beneficial when processing video streams or when used on devices with low processing power, such as CCTV cameras or embedded real-time systems.

To detect objects, the algorithm uses a natural property of the video stream: each subsequent frame is a continuation of the previous one and reflects the changes that have occurred. If an object gradually enters the camera's field of view, it will remain in the following frames. As a result, finding the absolute difference between the frames allows you to pinpoint the exact area where the moving object appeared. Unlike SSD, where detection and classification are performed simultaneously, the proposed approach divides these processes into separate stages.

The speed of calculating the absolute difference also depends on the frame size: a smaller number of pixels makes this operation faster. Another way to optimise is to use the difference not between consecutive frames, but between the current frame and the one that was a few frames earlier (for example, 5 frames later). This allows you to detect more significant changes when the object has already fully appeared in the scene. It is important that the camera remains static, with no additional movement that could distort the results.

Since changes may occur between frames due to lighting or other external factors, the absolute difference provides a digital estimate of the change in each pixel. To eliminate minor fluctuations, threshold filtering is applied, which discards all difference values below a specified threshold [9]. Changing this threshold allows you to adjust the sensitivity of the detector: at a high value, it responds only to significant changes, at a low value - even to insignificant ones. However, after this procedure, the resulting areas may have contour breaks or cavities inside that need to be corrected.

These defects are eliminated by the sequential morphological operations of opening and closing [10]. Opening consists of sequential erosion and dilation and removes fine noise, while closing (dilation followed by erosion) fills the cavities inside the object. Since a binary mask is formed during this process, the specific pixel values do not matter - only the fact that they belong to a region is important. To perform these operations, you use a kernel (a structural element) that determines the size of the pixel's neighborhood. The larger the kernel, the stronger the effect of the transformation and the longer the processing time; a kernel that is too small may not produce noticeable results.

Sometimes an object moves slightly or partially, so that the region covers only a part of it, which makes classification difficult. To solve this problem, an additional dilation with a large kernel is used, which allows you to capture the stationary parts of the object, combining them into one region.

The next step is size filtering: small regions that remain after transformations and cannot physically contain the object are removed as unnecessary for further processing. This allows us to concentrate resources on the regions of importance.

With only significant regions, the next filtering step is to skip regions that do not belong to a region with potential movement. The size checking operation is much simpler than checking whether one region lies within or intersects another in terms of computational resources, so it comes after. In cases where the current region does not completely lie within a region, but intersects it, it is possible to add an additional filter based on the percentage of intersection. The current implementation does not have such a step.

Thus, the developed method can be interpreted as an evolutionary adaptive [8] mechanism, as it allows the algorithm to gradually change its processing strategies based on the collected data, forming a heat map and optimising its performance for the specifics of a particular environment.

When all uninformative areas are discarded, each region is passed to the input of a neural network-based classifier [11]. This implementation uses the MobiNETv1 architecture, but the choice of a specific model depends on the performance requirements, available memory, and desired accuracy.

After successful classification, the object's position is transferred to the tracker, which allows it to be tracked in subsequent frames without re-detection. The tracker status is checked every 10 frames: if the object no longer corresponds to the expected region (due to errors or leaving the scene), the tracker is reset and the search process is resumed. The main goal of this approach is to ensure stable tracking of the object even with a small or temporary absence of movement. For this purpose, the fast MOSSE algorithm is used, which is well suited for such tasks.

## Experiments and comparison of results with optimized version

For an objective comparison of the effectiveness of the basic and optimised versions of the proposed method and their implementation algorithms, we experimentally measured the parameters. Table 1 shows the summary of calculated data for the average frame processing time, recognition accuracy, CPU, RAM, GPU, and video memory resources.

Table 1 - Results of calculations of indicators for the basic and optimised versions

| Measurement | Measurement unit | Initial version | Optimized version | Relative difference (%) |
|---|---|---|---|---|
| Average frame processing time | ms | 5.3 | 4.4 | -17.0 |
| Accuracy | % | 71.4 | 72.2 | +1.1 |
| CPU usage | % | 11.40 | 11.42 | +0.2 |
| RAM usage | MB | 460 | 458 | -0.4 |
| GPU usage | % | 1.1 | 0.98 | -10.9 |
| VRAM usage | MB | 358 | 350 | -2.2 |

As can be seen from Table 1, the optimised version shows better key performance indicators, including a reduction in the average frame processing time and a slight increase in accuracy. The corresponding graphs are shown below to visualise the experimental results (Figs. 3-8).

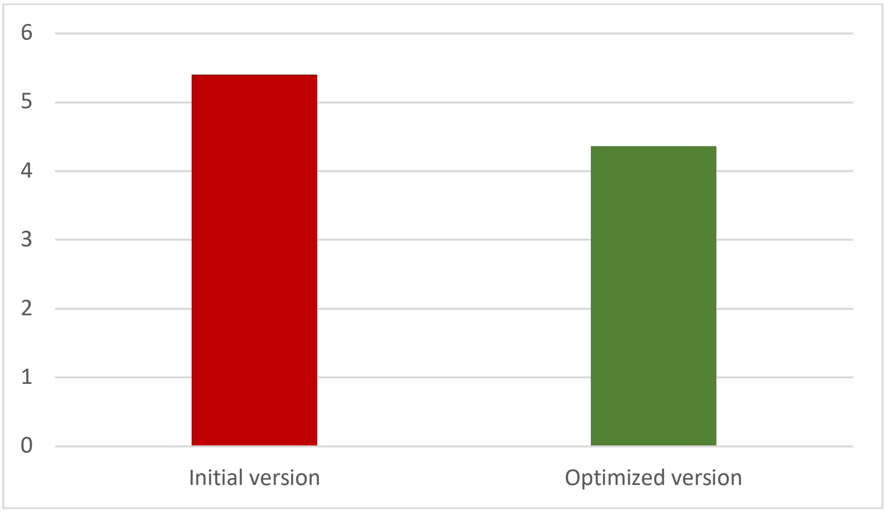

Figure 3 – Average processing time per frame

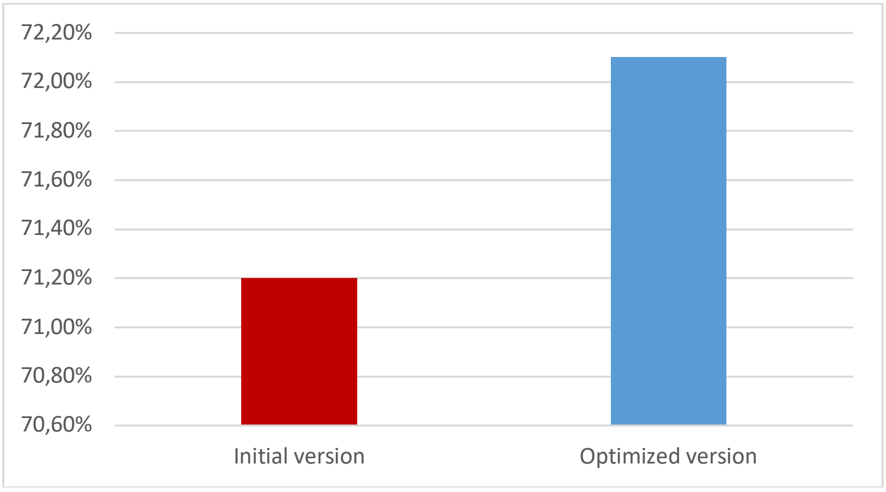

Figure 4 – Recognition accuracy

As can be seen from Fig. 3, the optimised method is on average 0,86 milliseconds faster than the baseline version, which provides a performance gain of about 18 %. At the same time, Fig. 4 shows the results of the comparison in terms of recognition accuracy, which is an equally important characteristic.

The experimental results show that the improved method exceeds the baseline version in terms of recognition accuracy by 0,8 %. Despite the small absolute value of the increase, statistical analysis using Student's t-test confirmed that the difference is statistically significant at a significance level of $p < 0.05$.

Statistical analysis was performed using Student's t-test for paired samples (n = 30). The results show that changes in the average frame processing time, accuracy, and GPU usage are statistically significant at the $p < 0.05$ level, while changes in CPU, RAM, and video memory usage do not differ significantly (Table 2).

Table 2 - Confirmation of statistical significance of differences by Student's t-test

| Measurement | Initial version (M±SD) | Optimized version (M±SD) | n | t-value | p-value | Significant at p<0.05 |
|---|---|---|---|---|---|---|
| Average accuracy (%) | 71.4 ± 0.8 | 72.2 ± 0.7 | 30 | 2.15 | 0.041 | Yes |
| Average processing time (ms) | 5.3 ± 0.9 | 4.4 ± 0.8 | 30 | 3.20 | 0.004 | Yes |
| CPU usage (%) | 11.40 ± 0.2 | 11.42 ± 0.2 | 30 | 0.35 | 0.725 | No |
| RAM usage (MB) | 460 ± 1.5 | 458 ± 1.6 | 30 | 0.52 | 0.610 | No |
| GPU usage (%) | 1.10 ± 0.1 | 0.98 ± 0.1 | 30 | 2.05 | 0.048 | Yes |
| VRAM usage (MB) | 358 ± 3.0 | 350 ± 3.2 | 30 | 0.87 | 0.398 | No |

As can be seen from the data presented, the improved method demonstrates statistically significant advantages in frame processing speed, recognition accuracy, and GPU usage, while changes in the use of CPU, RAM, and video memory do not have a significant impact. These results confirm the effectiveness of integrating the parameter adaptation mechanism into the delta classification method.

Next, let us consider the peculiarities of using the processor's computing resources during recognition (Fig. 5).

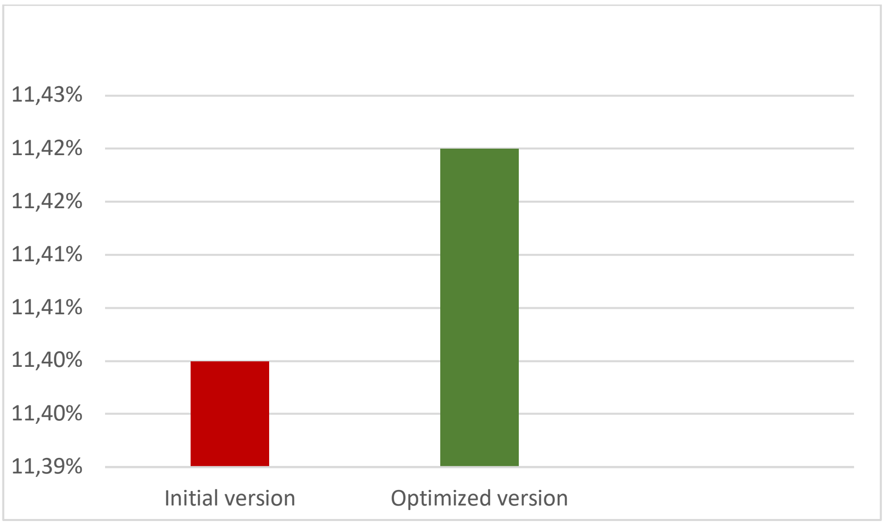

Figure 5 – CPU resource usage

The difference in CPU usage is only 0,2 % and, according to the Student's t-test, is not statistically significant at $p < 0.05$. This can be explained by the fact that the algorithm skips some of the recognition operations that are performed mainly on the video card, which does not significantly affect the CPU load.

The following graph shows data on RAM usage (Fig. 6).

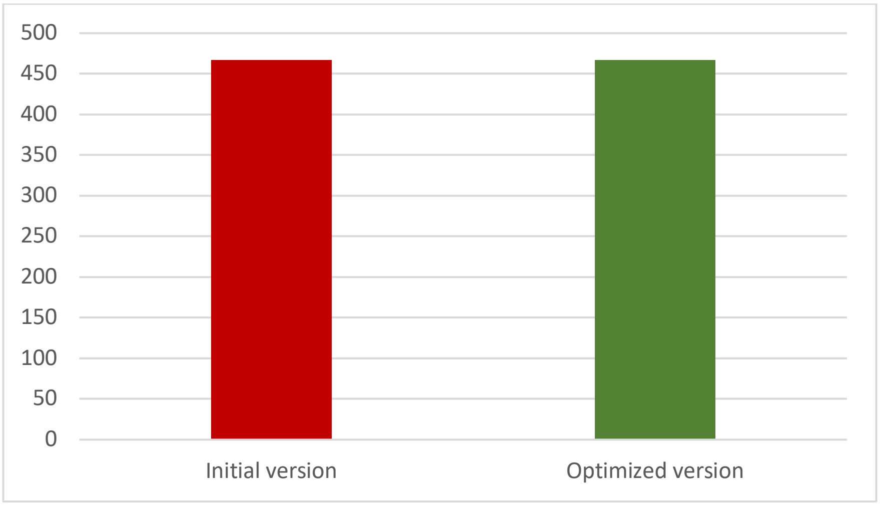

Figure 6 – RAM usage

Similarly to the CPU usage, the difference in RAM consumption is not statistically significant, as no changes were made to the storage model. This confirms the stability of the memory consumption of the improved method. Fig. 7 shows data on the use of GPU computing time.

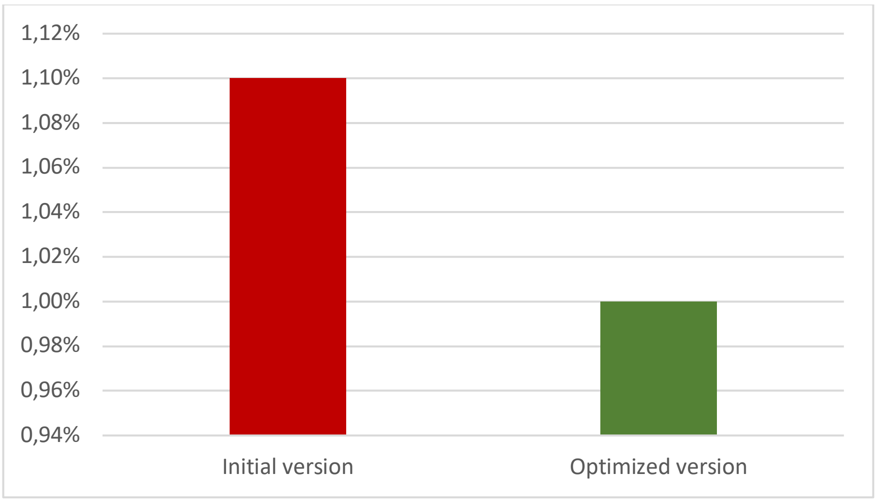

Figure 7 – GPU usage

The use of a heat map allows skipping the classification of regions where the presence of objects is statistically unlikely. Since a convolutional neural network is used for recognition, most of the computations are performed on the video card, so a reduction in the number of recognisable objects leads to a 0,1% reduction in resource consumption. Statistical analysis using the Student's t-test showed that the deviation in accuracy between the methods is statistically significant at the p-value of $p < 0.05$.

We also analyse the use of video memory, the results are shown in the following graph (Fig. 8).

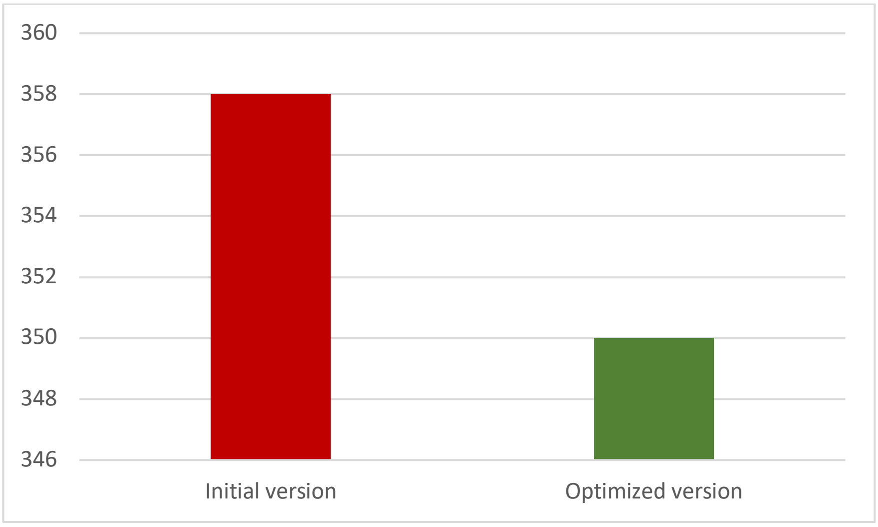

Figure 8 – VRAM usage

The optimised version of the algorithm consumes 8 MB less video memory, but according to the Student's t-test, this difference is not statistically significant. This is because both versions use the same models and algorithms that require a fixed amount of memory at runtime.

**Conclusions and prospects for further research.**

The article presents an improvement of the method for recognising moving objects in a real-time video stream by introducing a mechanism for adaptive parameter adjustment based on inter-frame analysis (delta) and a neural classifier. The experimental analysis confirmed the feasibility of using this approach: an increase in processing speed of up to 20 % and an increase in recognition accuracy of about 0.8 % without a significant increase in computational costs.

Prospects for further research include the development of a more flexible mechanism for adapting to the characteristics of the input data, the formation of a more accurate heat map to skip uninformative regions of the frame, and the use of hybrid models to optimise resources in different hardware environments and complex scenes with dynamic backgrounds.

**References**

1. Abdallah W., Val T. (2020). Genetic-Voronoi algorithm for coverage of IoT data collection networks. *30th International Conference on Computer Theory and Applications* (ICCTA), Alexandria, Egypt, 2020. PP. 16-22. DOI: 10.48550/arXiv.2202.13735.
2.Bernardin R., Paias A. (2018). Metaheuristics based on decision hierarchies for the traveling purchaser problem. International Transactions in Operational Research, 25(4), 2018. PP. 1269-1295. DOI: 10.1111/itor.12330.
3.Blank J., Deb K. (2020). Pymoo: Multi-Objective Optimization. IEEE Access, Vol. 8, 2020. PP. 89497-89509. DOI: 10.1109/ACCESS.2020.2990567.
4.Blank J., Deb K., Dhebar Y., Bandaru S., SeadaH. (2012.) Generating well-spaced points on a unit simplex for evolutionary many-objective optimization. IEEE Transactions on Evolutionary Computation, 25(1), 2021. PP. 48-60. DOI: 10.1109/TEVC.2020.2992387.
5.Cheng R., Jin Y., Olhofer M., Sendhoff B. (2016). A reference vector guided evolutionary algorithm for many-objective optimization. IEEE Transactions on Evolutionary Computation, 20(5), 2016. PP. 73-791. DOI: 10.1109/TEVC.2016.2519378.
6.Deb K, Abouhawwash M. (2016). An optimality theory-based proximity measure for set-based multiobjective optimization. IEEE Trans. Evolutionary Computation, 20(4), 2016. PP. 515-528. DOI: 10.1109/TEVC.2015.2483590.
7.Deb K., Jain H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Transactions on Evolutionary Computation, 18(4), 2014. PP. 77-601. DOI: 10.1109/TEVC.2013.2281535.
8.Guerrero C., Lera I., Juiz C. (2018). Genetic algorithm for multi-objective optimization of container allocation in cloud architectures. Journal of Grid Computing. 16-1, 2018. PP. 113-135. DOI: 10.1007/s10723-017-9419-x.
9.Lian Y., Peng P., Jiang K., Xu W. (2024). Multi-objective compression for CNNs via evolutionary algorithm Information Sciences, Volume 661, 2024. 120155. DOI: 10.1016/j.ins.2024.120155.
10. Ma Z., Wang Y. (2019). Evolutionary Constrained Multiobjective Optimization: Test Suite Construction and Performance Comparisons. IEEE Transactions on Evolutionary Computation, 23(6), 2019. PP. 972-986. DOI: 10.1109/TEVC.2019.2896967.

11. Pyrih Y., Kaidan M., Tchaikovskyi I., Pleskanka M. (2019). Research of Genetic Algorithms for Increasing the Efficiency of Data Routing. *2019 3rd International Conference on Advanced Information and Communications Technologies* (AICT), Lviv, Ukraine, 2019. PP. 157-160. DOI: 10.1109/AIACT.2019.8847814.
12. Qian H., Wu Y., Qin R., An X., Chen Y., Zhou A. (2025). Provable space discretization based evolutionary search for scalable multi-objective security games. Swarm and Evolutionary Computation, 92(1):101770, 2025. DOI: 10.1016/j.swevo.2024.101770.
13. Seada H., Deb K. (2016). A unified evolutionary optimization procedure for single, multiple, and many objectives. IEEE Transactions on Evolutionary Computation, 20(3), 2016. PP. 358-369. DOI:10.1109/TEVC.2015.2459718.
14. Skarlat O., Nardelli M., Schulte S., Borkowski M., Leitner P. Optimized IoT service placement in the fog. Special Issue Paper. Open access. SOCA. Volume 11, 2017. PP. 427-443. DOI: 10.1007/s11761-017-0219-8.
15. Takagi T., Takadama K., Sato H. (2020). Incremental lattice design of weight vector set. In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20, New York, NY, USA, 2020. PP. 1486-1494. DOI: 10.1145/3377929.3398082.
16. Vesikar Y., Deb K., Blank J. (2018). Reference point-based NSGA-III for preferred solutions. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Nov 2018. PP. 1587-1594. DOI:10.1109/SSCI.2018.8628819.
17. Zitar R. (2022). A Review of the Genetic Algorithm and JAYA Algorithm Applications. *15th International Congress on Image and Signal Processing*, (CISP-BMEI), Beijing, China, 2022. PP. 1-7. DOI: 10.1109/CISPBMEI56279.2022.9980332.