UDC 519.6

PARALLEL AND DISTRIBUTED COMPUTING TECHNOLOGIES FOR AUTONOMOUS VEHICLE NAVIGATION

Mochurad L. I. – PhD, Associate Professor, Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine.

Mamchur M. V. – Student, Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine.

ABSTRACT

Context. Autonomous vehicles are becoming increasingly popular, and one of the important modern challenges in their development is ensuring their effective navigation in space and movement within designated lanes. This paper examines a method of spatial orientation for vehicles using computer vision and artificial neural networks. The research focused on the navigation system of an autonomous vehicle, which incorporates the use of modern distributed and parallel computing technologies.

Objective. The aim of this work is to enhance modern autonomous vehicle navigation algorithms through parallel training of artificial neural networks and to determine the optimal combination of technologies and nodes of devices to increase speed and enable real-time decision-making capabilities in spatial navigation for autonomous vehicles.

Method. The research establishes that the utilization of computer vision and neural networks for road lane segmentation proves to be an effective method for spatial orientation of autonomous vehicles. For multi-core computing systems, the application of parallel programming technology, OpenMP, for neural network training on processors with varying numbers of parallel threads increases the algorithm's execution speed. However, the use of CUDA technology for neural network training on a graphics processing unit significantly enhances prediction speeds compared to OpenMP. Additionally, the feasibility of employing PyTorch Distributed Data Parallel (DDP) technology for training the neural network across multiple graphics processing units (nodes) simultaneously was explored. This approach further improved prediction execution times compared to using a single graphics processing unit.

Results. An algorithm for training and prediction of an artificial neural network was developed using two independent nodes, each equipped with separate graphics processing units, and their synchronization for exchanging training results after each epoch, employing PyTorch Distributed Data Parallel (DDP) technology. This approach allows for scalable computations across a higher number of resources, significantly expediting the model training process.

Conclusions. The conducted experiments have affirmed the effectiveness of the proposed algorithm, warranting the recommendation of this research for further advancement in autonomous vehicles and enhancement of their navigational capabilities. Notably, the research outcomes can find applications in various domains, encompassing automotive manufacturing, logistics, and urban transportation infrastructure. The obtained results are expected to assist future researchers in understanding the most efficient hardware and software resources to employ for implementing AI-based navigation systems in autonomous vehicles. Prospects for future investigations may encompass refining the accuracy of the proposed parallel algorithm without compromising its efficiency metrics. Furthermore, there is potential for experimental exploration of the proposed algorithm in more intricate practical scenarios of diverse nature and dimensions.

KEYWORDS: computer vision, neural networks, navigation methods, CUDA technology, PyTorch DDP technology.

ABBREVIATIONS

NN is a Neural Network;
OpenMP is an Open Multi-Processing;
CUDA is a Compute Unified Device Architecture;
DDP is a Distributed Data Parallel;
LiDAR is a Light Detection and Ranging;
ACO is an Ant Colony Optimization;
CNN is a Convolutional Neural Network;
IoT is an Internet of Things;
FPN is a Feature Pyramid Network;
CPU is a central processing unit;
GPU is a graphics processing unit.

NOMENCLATURE

N is the number of records in the dataset;
p is the cores count;
$T_1()$ is an execution time of a sequential algorithm;
$T_p()$ is an execution time of a parallel algorithm.

INTRODUCTION

With the advancement of autonomous vehicles, the demand for high-precision navigation systems and efficient algorithms is becoming increasingly crucial. Optimization and enhancement of existing artificial intelligence methods to improve navigation accuracy, along with the application of parallel computing to boost algorithm speed, have the potential to unlock new opportunities and contribute significantly to the development of the autonomous transportation sector [1, 2]. Algorithm and navigation method optimization can contribute to ecological and economic development as autonomous vehicles have the potential to reduce fuel costs and facilitate efficient infrastructure utilization.

Improvements in navigation algorithms can have a positive impact on various sectors, including logistics, automated warehouses, and robotics, where precise localization and navigation are critically important for effective operations [3, 4].

Since autonomous vehicles must react to real-time road situations, parallel computing helps ensure the swift execution of algorithms, which is vital for road safety and efficiency [5].

DOI 10.15588/1607-3274-2023-4-11

This work investigates the effectiveness of parallelizing the training and prediction algorithm of an artificial neural network for road lane segmentation across various devices: processor, graphics processing unit (GPU), and two GPUs simultaneously. Different distributed and parallel computing technologies are considered, including OpenMP [6], CUDA [7], and PyTorch DDP [8].

The object of this research is the navigation system of an autonomous vehicle, which encompasses various contemporary distributed and parallel computing technologies.

The subject of investigation comprises existing algorithms utilized for enhancing spatial navigation in autonomous vehicles, as well as parallel computing technologies aimed at improving the performance of these algorithms.

The purpose of this work is to enhance current navigation algorithms for autonomous vehicles in space by means of parallel training of artificial neural networks, and to determine the optimal combination of technologies and devices to increase speed and enable real-time decision-making capabilities.

1 PROBLEM STATEMENT

Let $D_{train} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training dataset, where x_i denotes the input data and y_i denotes the corresponding ground truth lane segmentation labels for S samples, i.e., $x_i = \{x_{i,s}\}_{s=1}^{S}$ and $y_i = \{y_{i,s}\}_{s=1}^{S}$, by analogy let D_{test} be the test dataset.

Suppose θ represents the parameters of the artificial neural network model, C represents the configuration space, encompassing parameters such as thread counts and parallel programming methodologies.

The functions $J(D_{train}, \theta, C)$ and $J(D_{test}, \theta, C)$ quantify the optimization criterion for the training and test dataset, encapsulating metrics that measure the efficiency of the parallelized algorithm. These metrics include aspects such as training duration, and prediction error.

In summary, the problem is to determine optimal parameters of the artificial neural network model θ and configuration space C that lead to the most efficient parallelization of the artificial neural network-based lane segmentation on test dataset, so that $J(D_{test}, \theta, C) \to \min$.

2 REVIEW OF THE LITERATURE

During the analysis of scientific papers and sources, a list of facts that served as the foundation for this research was identified. The first fact is that artificial intelligence methods are finding increasing applications in autonomous vehicles. This is due to their potential to significantly enhance road safety and make vehicle control more efficient and comfortable for drivers. According to a study, the use of autonomous vehicles with a 50% share can reduce the number of road accidents by 29% [9]. The second fact is that spatial navigation in autonomous vehicles relies on a combination of various technologies, including computer vision and artificial neural networks [10]. The third fact is that different parallel computing technologies are employed for training artificial neural networks [11, 12]. Other navigation methods for autonomous vehicles that utilize artificial intelligence include deep learning and image recognition-based techniques, such as the use of LiDAR sensors and cameras to gather environment data and create a 3D road map. This enables the vehicle to determine its location and identify surrounding objects [10].

Convolutional neural networks are effective when working with input data like images, including those from autonomous vehicle cameras [13]. Convolutional encoder-decoder architectures, on the other hand, are used to reduce the dimensionality of images and are often employed for image segmentation tasks [14].

For training neural networks used in autonomous vehicles, large datasets are required, particularly images of roads and road markings. One way to increase data volume is through data augmentation, which involves applying random transformations to images, such as scaling and rotation [15]. When using neural networks for vehicle navigation, factors like changes in lighting and atmospheric conditions must be considered. To address this, neural network models with additional layers responsible for data normalization (Batch Normalization) and the introduction of random noise to input data can be used [16]. To combine data from different sensors, Kalman filters and enhanced versions of these filters can be employed [17].

In [18], a method for lane boundary detection is presented, which operates by extracting candidate lane segments from an image and subsequently selecting the most prominent lane using dynamic programming. The authors utilize real road videos for experiments, demonstrating the effectiveness of their proposed approach. However, this method is considered outdated and does not account for factors such as changes in lighting and atmospheric conditions.

In [19], authors propose a hybrid approach based on Ant Colony Optimization (ACO) for line detection in images, using the Canny edge detection method and Hough transform for line extraction. The proposed system operates quickly but, as noted by the authors, is confidently applicable only on straight roads.

In [20], authors address road scene segmentation for RGB images using recent advancements in semantic segmentation through convolutional neural networks (CNN) and convolutional encoder-decoders. They introduce several architecture improvements that balance segmentation quality and computational speed. Experimental results indicate that their model provides accurate lane predictions in the original image size..

In [21], the authors addressed the problem of lane detection using an Internet of Things (IoT) system for

DOI 10.15588/1607-3274-2023-4-11

interaction between different modules, including the car module, cloud module, and remote car controller. The method for lane detection and tracking is executed initially on the car module and then transmitted to the cloud module for additional processing. The authors achieved a processing speed of approximately 31 ms per frame. An explicit drawback of this approach is the requirement for the car to be within the cellular network coverage area and have access to the internet, which is not always guaranteed, particularly on remote highways.

In [22], the authors utilize deep learning to tackle the lane segmentation problem, employing deep convolutional neural networks. The system achieves a respectable accuracy of 96%, but it requires 132 ms for processing a single frame.

One of the limitations of the previously presented algorithm and many others analyzed by us is that authors consider training and operating neural networks on a single powerful node (video processor) for lane segmentation, which can significantly limit the speed of learning and predictions of such networks in real-world scenarios. Our approach involves multiple independent nodes with separate video processors for parallel training and prediction, ensuring greater scalability and speed. Additionally, our approach utilizes PyTorch DDP technology for efficient communication and synchronization between nodes.

In summary, while the literature review reveals several promising approaches to lane segmentation in autonomous driving, most of them do not explore multi-node (video processor) training. Our proposed algorithm, based on parallel training and prediction on multiple independent nodes with separate video processors and PyTorch DDP communication, represents significant progress in terms of efficiency and scalability.

3 MATERIALS AND METHODS

To solve image processing tasks, CNNs (Convolutional Neural Networks) are widely used, including for object segmentation in images [13]. CNN consists of a sequence of convolutional and pooling layers, allowing the model to automatically identify important features of images at different levels.

Convolutional layers perform the convolution operation, which involves moving filters (of varying sizes and shapes) over the image to extract different image features such as edges, corners, textures, and more. The result of convolution is a feature map that highlights important regions of the image. Pooling layers reduce the size of the feature map and retain the most important features from each region, reducing the number of network parameters and preventing overfitting. CNNs leverage internal pixel relationships within the image for effective object segmentation. Operations of this type form the basis of convolutional encoders-decoders and Feature Pyramid Networks (FPN), which are used to address the lane segmentation task in the present study.

This network was proposed in 2017 with the aim of improving the object segmentation process in images. FPN (Feature Pyramid Network) consists of several convolutional layers that interact with the object in the image at different scales [23].

For the segmentation task, a slightly modified version of the FPN network is used, where each FPN level is gradually increased using convolutional functions and bilinear upscaling until it reaches a scale of 1/4. Then, these outputs are added and finally transformed into pixel-level output [24]. In general, the use of the FPN network for segmentation tasks allows for improved accuracy of results by optimally utilizing features at different scales of image resolution.

The time complexity of the convolutional encoder-decoder (FPN network) depends on the size of the input and output data, as well as the number of layers and filters in the network. In general, the time complexity can be expressed as a function of the number of operations required to process the input data.

Assuming that the convolutional encoder-decoder has L layers, filters with a size of $F \times F$ at each layer, and input data with a size of $D \times D$, then the general time complexity of the algorithm is:

$$O(N * L * D^2 * F^2).$$

Due to the fact that for each layer, the input data is processed by filters of size $F \times F$, each of size and this process is repeated L times (number of layers).

During parallelization using the CPU, the time complexity decreases proportionally to the number of physical threads engaged in computing mathematical operations during training (for example, calculating activation functions), where $N = N / THREADS$.

During parallelization using a GPU, the time complexity decreases proportionally to the number of computational units (CUDA cores) on the graphics processor and their speed, where $N = N / GRID\_SIZE$.

During parallelization using GPU and additionally PyTorch DDP, the time complexity decreases proportionally to the CUDA cores and is further divided by the number of nodes with video processors used, since the data (*N*) is shared among them for processing, where $N = N / (GRID\_SIZE * N\_DEVIES)$.

For training the neural network, the PyTorch library was utilized, which achieves parallelization of the training process through OpenMP. During the course of the research, the specific directives that were employed and the manner in which basic operations are parallelized during training were analyzed, specifically:

– To determine the number of used threads, the function omp_set_num_threads() is employed;

– The #pragma omp parallel for directive is used for parallelizing the ReLU activation function;

© Mochurad L. I., Mamchur M. V., 2023
DOI 10.15588/1607-3274-2023-4-11

– The $\#\text{pragme omp parallel for schedule}(\text{static})$ directive is utilized for parallelizing the loop of the Adam optimizer;

– The $\#\text{pragma omp parallel}$ directive is applied for parallelizing the operations of the convolutional layer;

– Matrix multiplication operations (batch_matmul) are parallelized using the $\#\text{pragma omp parallel for collapse}(2)$ directive.

Regarding the parallelization of the algorithm using CUDA technology, in our case, we utilized the CUDA kernel implementation in PyTorch, which, in turn, leverages existing NVIDIA solutions such as CUBLAS and CUDNN.

Since we have a single kernel, PyTorch by default employs the following values for constructing the grid:

```
THREADS_PER_BLOCK = 512;
BLOCKS_PER_SM = 4;
```

As a result of parallelizing the algorithm using CUDA technology:

– For parallelizing the operations of the convolutional layer, cudnnConvolutionBackwardFilter was used, which is a part of CUDNN;

– For parallelizing pooling operations, we utilize cudnnPoolingForward, which is provided with an array of tensors;

– For batch normalization, we employ cudnnBatch-NormalizationForward and cudnnBatchNormalization-Backward.

The PyTorch DistributedDataParallel (DDP) technology allows distributing computations across multiple devices, such as servers or GPUs, to accelerate the training speed of models. PyTorch DDP employs an asynchronous approach to data and computation distribution among devices. It utilizes collective communication mechanisms, enabling each device to exchange data with others in the group. Additionally, PyTorch DDP ensures automatic parameter synchronization among devices during training. With PyTorch DDP, computations can be distributed across multiple servers or nodes, thereby enhancing computational power and reducing model training time. Moreover, PyTorch DDP supports automatic scaling of computational resources based on demand, facilitating efficient utilization of limited resources. DDP follows the CUDA algorithm, with the only difference being that the dataset is evenly split between two nodes, and the weights are synchronized using gradient aggregation at the end of each epoch, resulting in a 2x acceleration.

In the process of training machine learning models, input data plays a significant role. The quality and quantity of data can impact the accuracy and performance of the model. Therefore, it is crucial to properly select and prepare input data for model training.

One of the most popular applications for creating datasets for autonomous driving models is the CARLA Simulator [25]. This open-source software allows simulating urban traffic and autonomous vehicles. CARLA enables the creation of diverse scenarios for model training and testing, including simulations of various weather conditions, road traffic, pedestrian behavior, and other objects on the road.

The dataset created using the CARLA Simulator consists of images of road lanes with markings and other road elements. Each image is sized 1024x512 pixels and is presented in color. The total size of the dataset is 2.05 GB.

The images for training are captured by a camera mounted on a simulated vehicle. The annotated images provide segmentation masks. Each pixel in the annotated image is classified as:

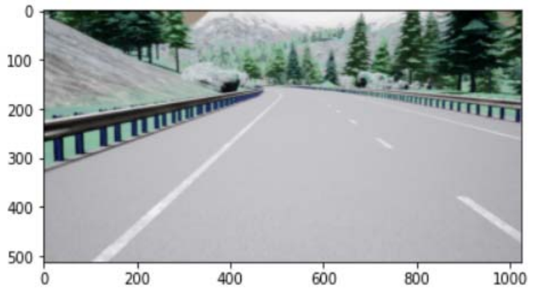
– part of the left lane boundary;

– part of the right lane boundary;

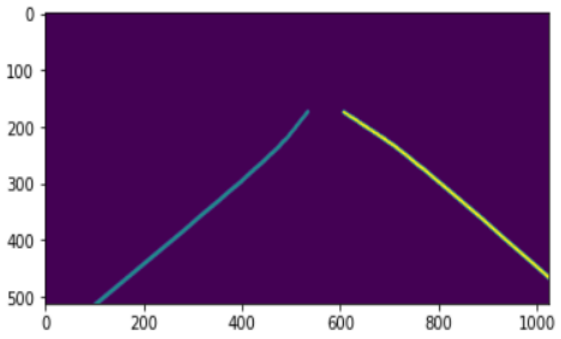
– none of the above (background).

The dataset was intentionally divided into a training and validation set: 3075 images for training and 129 for validation, along with 3075 and 129 corresponding mask images, respectively.

The challenge associated with this dataset is to train the model to accurately predict the segmentation masks for the validation dataset (see Fig. 1).



a



b

Figure 1 – The example of training samples:
a – original image; b – corresponding image-mask

DOI 10.15588/1607-3274-2023-4-11

Data augmentation can be used to increase the amount of training data and improve the quality of the model [26]. For example, images can be altered using techniques such as cropping, different positioning, color adjustments, resizing, and other transformations. Data augmentation can be particularly useful when working with limited data. In cases where the dataset lacks sufficient data for a specific task, augmentation can create new data by manipulating existing examples. This can help prevent overfitting, provide a more diverse dataset, and enhance the model's generalization ability.

One tool for data augmentation is the Python library "imgaug". It offers a variety of functions for image transformations, including rotation, scaling, color changes, noise addition, and more. Additionally, there are other libraries and tools available for data augmentation that can be used to enhance the quality of both data and machine learning models.

The developed algorithm performs data augmentation before training the network to enhance the accuracy of network training (an example of this is illustrated in Figure 2). The operations used in this sequence are as follows:

– **ShiftScaleRotate:** shifts, scales, and rotates the image with random parameters.

– **IAAAdditiveGaussianNoise:** adds Gaussian noise to the image with a probability of 0.2.

– **CLAHE:** applies the Contrast Limited Adaptive Histogram Equalization algorithm to enhance image contrast.

– **RandomBrightness:** changes the brightness of the image by a random amount.

– **RandomGamma:** adjusts the gamma of the image to a random value.

– **IAASharpen:** sharpens the image.

– **Blur:** applies blurring to the image to reduce sharpness.

– **MotionBlur:** adds motion blur to the image.

– **RandomContrast:** changes the contrast of the image by a random amount.

– **HueSaturationValue:** changes the hue and saturation of the image to random values.

After applying this sequence of operations, the images will slightly differ from each other, which allowed us to improve the model's performance on the validation dataset.

Step-by-Step Description of the Proposed Algorithm:

1. Import necessary libraries.

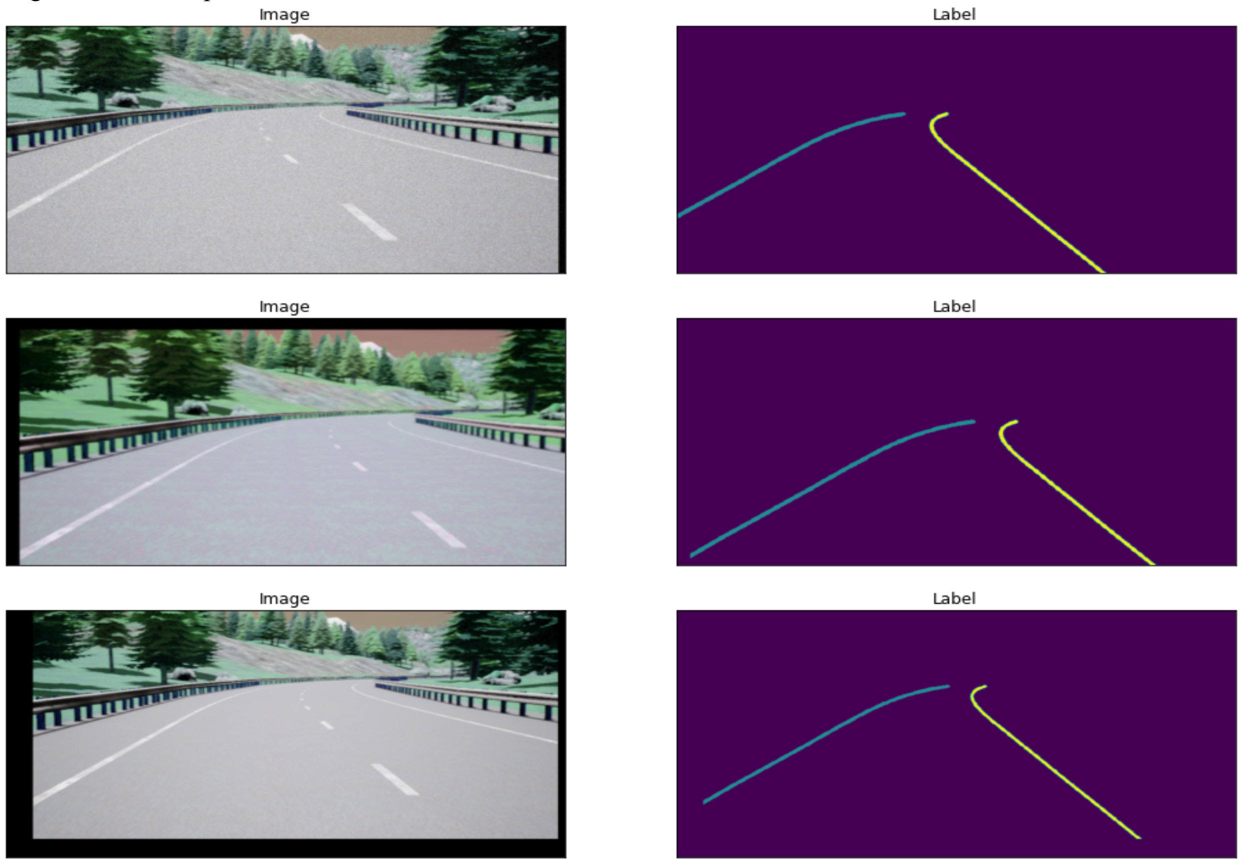


Figure 2 – Resulting augmented input image (left) and corresponding masks (right)

DOI 10.15588/1607-3274-2023-4-11

2. Declare the class CarlaLanesDataset with methods `__init__`, `__getitem__`, and `__len__`, which will be used to retrieve and preprocess the data.

3. Load the dataset into memory (training and validation sets).

4. Declare methods get_training_augmentation, get_validation_augmentation, and get_preprocessing to define transformations for data augmentation and preprocessing.

5. Apply augmentation to the loaded training dataset.

6. Initialize the artificial neural network model object.

7. Initialize the loss function and optimizer objects.

8. Parallel execution of the training process:

– On CPU using OpenMP technology.

– On GPU using CUDA technology.

– On multiple GPUs using CUDA technology and PyTorch DDP.

9. Validate on the testing dataset.

Next step taken is to calculate the theoretical acceleration and efficiency metrics for parallel algorithms using different numbers of threads when parallel computations are performed on the CPU. Additionally, we calculated the acceleration metrics for parallel algorithms on the GPU. To compute these metrics, we used the following formulas:

$$S_p(N) = \frac{T_1(N)}{T_p(N)}, \tag{1}$$

$$E_p(N) = \frac{S_p(N)}{p}. \tag{2}$$

Here, Equation (1) is used to calculate the speedup, and Equation (2) – the efficiency.

First, let's perform a theoretical speedup estimation for various numbers of processors used for training the neural network. It should be noted that calculating the theoretical speedup for training a model on the GPU is analytically impossible since the number of graphic cores used during training is unknown.

$$S_2(N) = \frac{T_1(3075)}{T_2(3075)} = \frac{O(3075 * L * D^2 * F^2)}{O\left(\frac{3075}{2} * L * D^2 * F^2\right)} = 2,$$

$$S_4(N) = \frac{T_1(3075)}{T_4(3075)} = \frac{O(3075 * L * D^2 * F^2)}{O\left(\frac{3075}{4} * L * D^2 * F^2\right)} = 4,$$

$$S_8(N) = \frac{T_1(3075)}{T_8(3075)} = \frac{O(3075 * L * D^2 * F^2)}{O\left(\frac{3075}{8} * L * D^2 * F^2\right)} = 8.$$

Let's derive the theoretical estimates of efficiency:

$$E_2(N) = \frac{S_2(N)}{2} = 1,$$

$$E_4(N) = \frac{S_4(N)}{4} = 1,$$

$$E_8(N) = \frac{S_8(N)}{8} = 1.$$

It should be noted that the provided estimates apply to a system with p processor (core) computational units.

4 EXPERIMENTS

During the research, the training of the neural network was conducted on a CPU using the OpenMP technology. For this purpose, a computer with an Intel(R) Xeon(R) CPU @ 2.20GHz processor with 2 cores and 4 threads was utilized. The network training was carried out over 5 epochs.

Parallelization was achieved through the inter-op functionality of OpenMP – a specific thread pool is allocated for performing individual tasks, such as processing one of the input parameters. Inter-op allows us to handle micro-operations like pooling, batch operations, or matrix multiplication by dividing sub-tasks among threads. As a result of inter-op, the tasks of an iteration are synchronized, marking its completion.

The results of training the neural network on the CPU using OpenMP technology revealed a test dataset accuracy of approximately 96%. The network's prediction results are depicted in Fig. 3.

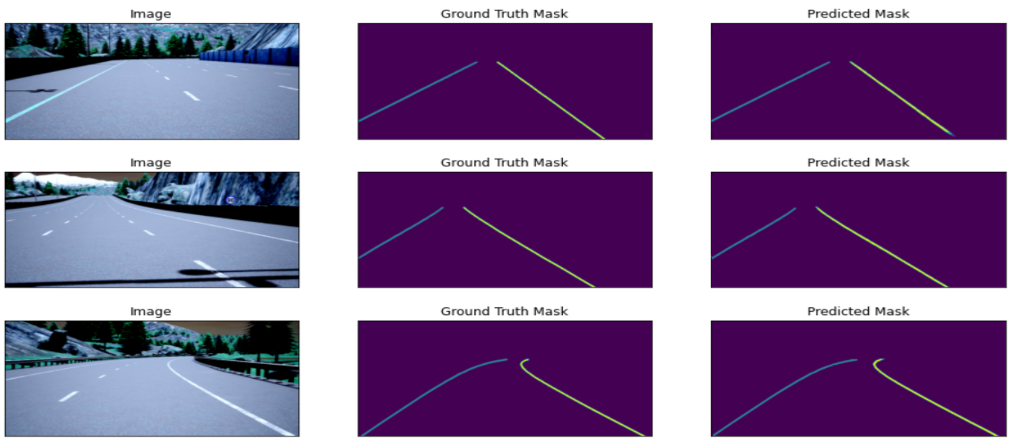


Figure 3 – The prediction results of the trained neural network. The first column displays images from the test dataset, the middle column shows the corresponding ground truth lane masks, and the left column presents the predicted lane masks

© Mochurad L. I., Mamchur M. V., 2023
DOI 10.15588/1607-3274-2023-4-11

The training algorithm was also parallelized on GPU using CUDA technology. Specifically, the network training was conducted on an NVIDIA T4 graphics card, a professional GPU released in 2018 designed for high-performance computing and artificial intelligence applications. It is built on the Nvidia Turing architecture and boasts 2560 CUDA cores, enabling it to handle large datasets and perform tasks such as deep learning and machine learning with high precision. The GPU comes equipped with 16 gigabytes of fast video memory.

PyTorch leverages pre-existing NVIDIA cores along with CUBLAS and CUDNN frameworks. These cores receive requests for executing intra-op tasks from CUBLAS and inter-op tasks from the CUDNN framework, and then perform these operations using available CUDA cores.

By utilizing PyTorch DDP technology, we were able to utilize a second device with a similar GPU module, effectively harnessing a total of 5120 CUDA cores for training. This parallel execution approach not only allows us to combine different GPUs but also avoids being restricted to a single physical device [27], which might limit the number of GPUs that can be attached. This approach enables us to scale the network to any desired size. The primary device serves as a synchronization point, while other nodes are launched with specified IP addresses, and they receive work ranges and necessary computation data from the controlling device to perform calculations.

5 RESULTS

In Table 1, we will present the time costs of sequential and parallel implementations on CPU using OpenMP technology, on GPU with CUDA, and on GPU with PyTorch DDP technology. The results of Table 1 are also visualized in Fig. 4.

Table 1 – Training Time of Neural Network on CPU, Single GPU, and Dual GPUs (in minutes)

Sequential excution	**CPU + OpenMP**			**GPU + CUDA**	**2xGPU + DDP**
	2	**4**	**8**		
333.5	144	123.6	144.6	9.1	4.8

From Table 1 and Figure 4, it is evident that the results of multi-threaded training demonstrate that increasing the number of threads up to 4 leads to improved performance, followed by a decline at 8 threads. Transitioning from 1 to 2 threads doubles the speed due to the presence of only 2 physical cores, indicating the validity of the obtained results. Moving from 2 to 4 threads results in a slight speed increase since the number of physical cores remains the same, but the logical cores provide additional cache memory for the threads. However, utilizing 8 threads depletes the available cache memory, prompting the processor to reload data to allow both threads to share a cache memory, resulting in cache miss penalties.

6 DISCUSSION

Considering the achieved speedup through the use of OpenMP technology, it can be concluded that employing OpenMP for neural network training on CPU is an effective method to reduce training time, especially in cases where GPU usage is not feasible.

Training the neural network using GPU is 36.6 times more efficient than sequential CPU training and 14 times more efficient than CPU training with 4 threads. With two GPUs, the training time per epoch is reduced to 4.8

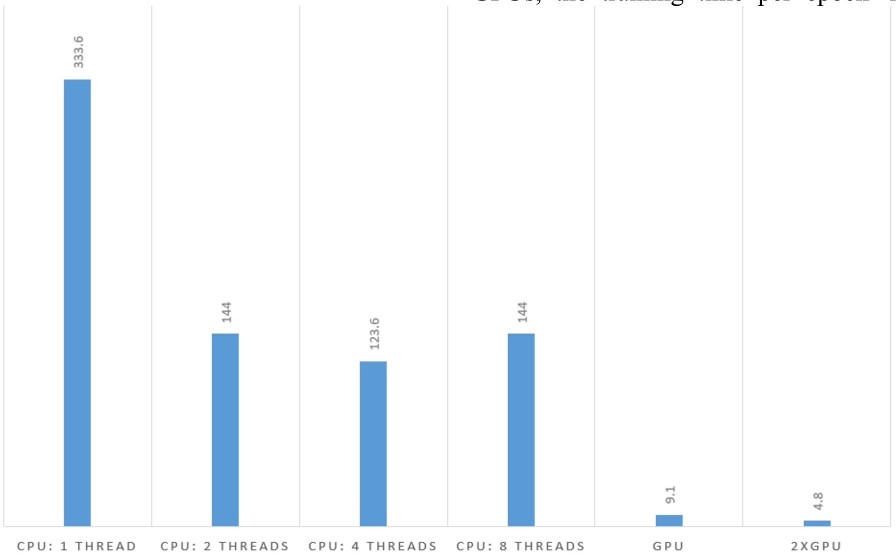


Figure 4 – Comparative diagram illustrating the training time dependency of a neural network on CPU with the involvement of threads, on a single GPU, and on two GPUs

DOI 10.15588/1607-3274-2023-4-11

minutes, which is 1.86 times more efficient compared to using a single GPU. However, the acceleration is not precisely twofold, as each device has independent video memory, and we need to synchronize training results between the devices in our local network of compute nodes after each epoch, which consumes some time.

Table 2 – Actual acceleration metrics of the parallel algorithm depending on the number of utilized threads on CPU, as well as the parallel algorithm on GPU and on two GPUs simultaneously

CPU, number of threads			**GPU**	**2xGPU**
2	4	8		
2.33	2.69	2.31	36.6	69.5

From Table 2, we observe that we achieved higher acceleration metrics for two threads compared to the theoretical values, which is due to the specifics of the Linux kernel scheduler prioritizing tasks with multiple active threads, thereby resulting in a single-threaded program having significantly lower computational speed than expected.

Since the process with one thread resulted in reduced performance, parallel execution with two threads yielded higher acceleration metrics.

When transitioning from 2 to 4 threads, the efficiency growth is marginal, as the number of physical cores remains unchanged, but the logical cores provide additional cache memory for threads.

With the utilization of 8 threads, the available cache memory is exhausted, prompting the processor to reload data to allow both threads to use the same cache memory, resulting in cache miss penalties.

Consequently, training the neural network using DDP is 69.5 times more efficient than sequential CPU training and 25.8 times more efficient than CPU training with 4 threads. Furthermore, it is 1.86 times more efficient than training with a single GPU.

Table 3 – Actual efficiency metrics of the parallel algorithm depending on the number of threads used on CPU

Number of threads		
2	**4**	**8**
1.16	0.67	0.29

Analyzing the results of Table 3, it can be observed that the actual efficiency metrics do not align with the theoretical ones. This discrepancy arises from the fact that the considered CPU has only 2 physical cores, but 4 logical cores provide additional cache memory for threads. However, as the number of threads increases to 8, the overhead of supporting these threads becomes predominant. Hence, the obtained results are reliable.

CONCLUSIONS

The paper analyzes contemporary approaches and methods for solving the problem of road lane segmentation to localize vehicles. During the analysis of scientific articles and sources, a list of facts was identified upon which this research is based.

It has been found that the utilization of modern parallel and distributed computing technologies on both CPU and GPU can significantly reduce the training time of neural networks for addressing the problem outlined in this study.

The scientific novelty of the obtained results lies in the introduction of a parallel algorithm for solving the road lane segmentation task using multiple GPUs with CUDA technology and PyTorch DDP. It has been established that the use of DDP expands computational capabilities by adding new independent nodes that can utilize both GPUs and CPUs. Therefore, this technology allows bypassing the limitations of calculations on a single device and achieving acceleration by orders of magnitude, sacrificing time only for exchanging intermediate training results between nodes.

In this work, based on the proposed algorithm, it was possible to achieve approximately a 90% increase in acceleration when using training on two nodes with NVIDIA T4 GPUs compared to one node. This is around 25 times faster compared to using the OpenMP technology for multi-core computer systems.

Furthermore, it was found that the time required for lane prediction for a single road frame by the model reached 19 ms, which is 1.63 times faster than in [20] and 6 times faster than in [21].

The algorithm employed in this study enabled achieving an accuracy of 96%, which is similar to [22]. However, it can be confidently stated that without compromising accuracy, significant acceleration of solving the road lane segmentation task for vehicle localization was achieved, specifically by a factor of 7.

The practical significance of the obtained results lies in the development of software that implements the proposed algorithm, as well as conducting a series of numerical experiments aimed at comparing the use of modern distributed and parallel computing technologies for autonomous vehicle navigation. The findings of this research can have a positive impact on road safety, cost-effectiveness, environmental friendliness, and transportation accessibility. Furthermore, they can contribute to the advancement of smart cities, integration of transportation systems, and enhance the competitiveness of automotive manufacturers. This research can also provide insights into the most efficient hardware and software tools to employ for implementing AI-based navigation systems in autonomous vehicles, depending on the situation [28].

The prospects for further research involve exploring the proposed parallel algorithm for a wide range of practical tasks.

DOI 10.15588/1607-3274-2023-4-11

REFERENCES

1. Varsi A., Taylory J., Kekempanosz L., Pyzer-Knapp E., Maskell S. A Fast Parallel Particle Filter for Shared Memory Systems, *IEEE Signal Process. Lett,* 2020, Vol. 27, pp. 1570–1574.
2. Huang K., Cao J. Parallel Differential Evolutionary Particle Filtering Algorithm Based on the CUDA Unfolding Cycle, *Wireless Communications and Mobile Computing,* 2021, Vol. 2021, pp. 1–12.
3. Kretzschmar H., Kuderer M., Burgard W. Learning to predict trajectories of cooperatively navigating agents, *2014 IEEE International Conference on Robotics and Automation (ICRA).* Hong Kong, China, 2014, pp. 4015–4020.
4. Wang P., Yang J., Zhang Y., Wang Q., Sun B., Guo D. Obstacle-Avoidance Path-Planning Algorithm for Autonomous Vehicles Based on B-Spline Algorithm, *World Electr. Veh. J,* 2022, Vol. 13, №12:233, pp. 1–15.
5. Xu L., Ouyang Y., Ford J., Banerjee S., Chen M. Real-time 3D traffic scene understanding from moving vehicles, *IEEE Transactions on Intelligent Transportation Systems,* 2018, Vol. 19, № 9, pp. 2827–2838.
6. Mochurad L. Optimization of Regression Analysis by Conducting Parallel Calculations, *COLINS-2021: 5th International Conference on Computational Linguistics and Intelligent Systems.* Kharkiv, Ukraine, 22–23 April, 2021, pp. 982–996.
7. Mochurad L. Canny Edge Detection Analysis Based on Parallel Algorithm, Constructed Complexity Scale and CUDA, *Computing and Informatics,* 2022, Vol. 41, № 4, pp. 957–980.
8. Li Sh., Zhao Yanli, Varma Rohan, Salpekar Omkar, Noordhuis Pieter, Li Teng, Paszke Adam, Smith Jeff, Vaughan Brian, Damania Pritam, Chintala Soumith PyTorch Distributed: Experiences on Accelerating Data Parallel Training, *Proceedings of the VLDB Endowment,* 2020, Vol. 13, № 12, pp. 3005–3018.
9. Petrović Đ., Mijailovic R., Pešić D. Traffic Accidents with Autonomous Vehicles: Type of Collisions, Manoeuvres and Errors of Conventional Vehicles' Drivers, *Transportation Research Procedia*, 2020, Vol. 45, pp. 161–168.
10. Haris M., Glowacz A. Navigating an Automated Driving Vehicle via the Early Fusion of Multi-Modality, *Sensors,* 2022, Vol. 22, № 4, pp. 1–18.
11. Hoffmann R. B., Löff J. , Griebler D. et al. OpenMP as runtime for providing high-level stream parallelism on multicores, *J. Supercomput*, 2022, Vol. 78, pp. 7655–7676.
12. Sierra-Canto X. Madera-Ramirez F., V. Uc-Cetina//Parallel Training of a Back-Propagation Neural Network Using CUDA, *Ninth International Conference on Machine Learning and Applications.* Washington, DC, USA, IEEE, 2010, pp. 307–312.
13. Sewak M., Karim Md. R., Pujari P. Implement advanced deep learning models using Python. *Practical convolutional neural networks*. Birmingam-Mumbai:Packt Publishing, 2018, 218 p.
14. Badrinarayanan V., Kendall A., Cipolla R. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, *in IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017, Vol. 39, № 12, pp. 2481–2495.
15. Shijie J., Ping W., Peiyi J. and Siping H. Research on data augmentation for image classification based on convolution neural networks, 2017 *Chinese Automation Congress (CAC).* Jinan, China, IEEE, 2017, pp. 4165–4170.
16. Bjorck N. Gomes Carla P, Bart Selman, and Kilian Q Weinberger Understanding batch normalization, *In Advances in Neural Information Processing Systems: 32nd Conference on Neural Information Processing Systems (NeurIPS 2018).* Montréal, Canada, 2018, pp. 7694–7705.
17. Chen J., Chen K., Ding C., Wang G., Liu Q. and Liu X. An Adaptive Kalman Filtering Approach to Sensing and Predicting Air Quality Index Values, *in IEEE Access,* 2020, Vol. 8, pp. 4265–4272.
18. Kang D.-J. Jung M.-H. Road lane segmentation using dynamic programming for active safety vehicles, *Pattern Recognition Letters,* 2003, Vol. 24, Issue 16, pp. 3177–3185.
19. Daigavane P. M., Bajaj P. R. Road Lane Detection with Improved Canny Edges Using Ant Colony Optimization, *3rd International Conference on Emerging Trends in Engineering and Technology*. Goa, India, 2010, pp. 76–80.
20. Oliveira G. L., Burgard W., Brox T. Efficient deep models for monocular road segmentation, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Daejeon, Korea (South), 2016, pp. 4885–4891.
21. Ghanem S., Kanungo P., Panda G. et al. Lane detection under artificial colored light in tunnels and on highways: an IoT-based framework for smart city infrastructure, *Complex Intell. Syst.,* 2023, Vol. 9, pp. 3601–3612.
22. Kortli Ya., Gabsi Souhir, Lew F. C., Lew Ya. V., Jridi M., Merzougui M., Atri M. Deep embedded hybrid CNN-LSTM network for lane detection on NVIDIA Jetson Xavier NX, *Knowledge-Based Systems*, 2022, Vol. 240, pp. 1–33.
23. Hu M., Li Y., Fang L., Wang S. A2-FPN: Attention Aggregation Based Feature Pyramid Network for Instance Segmentation, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* 2021, pp. 15343–15352.
24. Zapata M.A.D., Erkent Ö., Laugier Ch. YOLO-based Panoptic Segmentation Network, *COMPSAC 2021 – Intelligent and Resilient Computing for a Collaborative World 45th Anniversary Conference.* Madrid, Spain, Jul 2021, pp. 1–5.
25. Gómez-Huélamo C., Del Egido J., Bergasa L. M. et al. Train here, drive there: ROS based end-to-end autonomous-driving pipeline validation in CARLA simulator using the NHTSA typology, *Multimed Tools Appl.*, 2022, Vol. 81, pp. 4213–4240.
26. Kostrikov I., Yarats D., Fergus R.. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels [Electronic resource]. Access mode: https://arxiv.org/abs/2004.13649.
27. Li Sh. et al. Pytorch distributed: Experiences on accelerating data parallel training [Electronic resource]. Access mode: https://arxiv.org/abs/2006.15704.
28. Severino A., Curto S., Barberi S., Arena F., Pau G. Autonomous Vehicles: An Analysis Both on Their Distinctiveness and the Potential Impact on Urban Transport Systems, *Appl. Sci.*, 2021, Vol. 11, № 8:3604, pp. 1–17.

Received 18.08.2023.
Accepted 21.11.2023.

DOI 10.15588/1607-3274-2023-4-11

УДК 519.6

ТЕХНОЛОГІЇ ПАРАЛЕЛЬНИХ І РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ ДЛЯ АВТОНОМНОЇ НАВІГАЦІЇ ТРАНСПОРТНИХ ЗАСОБІВ

Мочурад Л. І. – канд. техн. наук, доцент, доцент кафедри систем штучного інтелекту національного університету «Львівська політехніка», Львів, Україна.

Мамчур М. В. – студент кафедри систем штучного інтелекту національного університету «Львівська політехніка», Львів, Україна.

АНОТАЦІЯ

Актуальність. Автономні автомобілі стають все більш популярними і одним з важливих сучасних завдань розробки таких автомобілів є забезпечення ефективної навігації останніх у просторі та їх руху у своїй виділеній проїзній смузі. У даній роботі розглянуто метод орієнтування у просторі автомобіля за допомогою комп'ютерного зору та штучних нейронних мереж. Об'єктом дослідження була система навігації автономного автомобіля, що включає в себе використання сучасних технологій розподілених та паралельних обчислень.

Мета роботи – вдосконалення сучасних алгоритмів навігації автономного автомобіля у просторі на основі паралельного навчання штучних нейронних мереж та визначення найоптимальнішої комбінації технологій та пристроїв для збільшення швидкості та можливості отримання рішення в режимі реального часу.

Метод. У роботі встановлено, що використання комп'ютерного зору та нейронних мереж для сегментації смуги дорожнього руху є ефективним методом орієнтації автономного автомобіля у просторі. При цьому для багатоядерних обчислювальних систем застосування технології паралельного програмування OpenMP для тренування нейронної мережі на процесорі з різним числом паралельних потоків збільшує швидкість виконання алгоритму. Проте використання технології CUDA для навчання нейронної мережі на відеопроцесорі дозволило значно збільшити швидкість передбачень в порівнянні з OpenMP. Також досліджено можливість використання технології PyTorch DDP для навчання нейронної мережі на декількох відеопроцесорах (вузлах) одночасно, що , в свою чергу, ще більш покращило час виконання передбачень в порівнянні з використанням одного відеопроцесора.

Результати. Розроблено алгоритм навчання та передбачення штучної нейронної мережі на двох незалежних вузлах з окремими відеопроцесорами та їх синхронізацією задля обміну результатами навчання після кожної епохи із використанням технології PyTorch DDP, що дозволяє масштабувати розрахунки при наявності більшої кількості потужностей і значно пришвидшити навчання моделі.

Висновки. Проведені експерименти підтвердили ефективність запропонованого алгоритму і дозволяють рекомендувати дане дослідження для подальшого розвитку автономних автомобілів та покращення їх навігаційних можливостей. Зокрема результати дослідження можуть знайти застосування у різних сферах, включаючи автомобільну транспортну промисловість, логістику та транспортну інфраструктуру міст. Отримані результати повинні допомогти наступним дослідникам зрозуміти, які апаратні та програмні засоби найефективніше використовувати для реалізації навігаційних систем на основі штучного інтелекту в автономних автомобілях. Перспективами подальших досліджень може бути покращення точності запропонованого паралельного алгоритму не погіршуючи показників ефективності, а також експериментальне дослідження запропонованого алгоритму на більш складних практичних задачах різної природи та розмірності.

КЛЮЧОВІ СЛОВА: комп'ютерний зір, нейронні мережі, методи навігації, технологія CUDA, технології PyTorch DDP.

ЛІТЕРАТУРА

1. A Fast Parallel Particle Filter for Shared Memory Systems / A. Varsi, J. Taylory, L. Kekempanosz et al.] // IEEE Signal Process. Lett. – 2020. –Vol. 27. – P. 1570–1574.
2. Huang K. Parallel Differential Evolutionary Particle Filtering Algorithm Based on the CUDA Unfolding Cycle / K. Huang, J. Cao // Wireless Communications and Mobile Computing. – 2021. – Vol. 2021. – P. 1–12.
3. Kretzschmar H. Learning to predict trajectories of cooperatively navigating agents / H. Kretzschmar, M. Kuderer, W. Burgard // 2014 IEEE International Conference on Robotics and Automation (ICRA). – Hong Kong, China, 2014. – P. 4015–4020.
4. Obstacle-Avoidance Path-Planning Algorithm for Autonomous Vehicles Based on B-Spline Algorithm / [P. Wang, J. Yang, Y. Zhang et al.] // World Electr. Veh. J. – 2022. – Vol. 13, №12:233. – P. 1–15.
5. Real-time 3D traffic scene understanding from moving vehicles / [L. Xu, Y. Ouyang, J. Ford et al.] // IEEE Transactions on Intelligent Transportation Systems. – 2018. – Vol. 19, № 9. – P. 2827–2838.
6. Mochurad L. Optimization of Regression Analysis by Conducting Parallel Calculations / L. Mochurad // COLINS-2021: 5th International Conference on Computational Linguistics and Intelligent Systems. – Kharkiv, Ukraine, 22–23 April, 2021. – P. 982–996.
7. Mochurad L. Canny Edge Detection Analysis Based on Parallel Algorithm, Constructed Complexity Scale and CUDA / L. Mochurad // Computing and Informatics. – 2022. – Vol. 41, № 4. – P. 957–980.
8. Li Sh. PyTorch Distributed: Experiences on Accelerating Data Parallel Training / [Shen Li, Yanli Zhao, Rohan Varma et al.] // Proceedings of the VLDB Endowment, 2020. – Vol. 13, № 12. – P. 3005–3018.
9. Petrović Đ. Traffic Accidents with Autonomous Vehicles: Type of Collisions, Manoeuvres and Errors of Conventional Vehicles' Drivers / Đ. Petrović, R. Mijailovic, D. Pešić // Transportation Research Procedia. – 2020. – Vol. 45. – P. 161–168.
10. Haris M. Navigating an Automated Driving Vehicle via the Early Fusion of Multi-Modality / M. Haris, A. Glowacz // Sensors. – 2022. – Vol. 22, № 4. – P. 1–18.
11. Hoffmann R. B. OpenMP as runtime for providing high-level stream parallelism on multi-cores / R. B. Hoffmann, J. Löff, D. Griebler et al. // J. Supercomput. – 2022. – Vol. 78. – P. 7655–7676.

DOI 10.15588/1607-3274-2023-4-11

12. Sierra-Canto X. Parallel Training of a Back-Propagation Neural Network Using CUDA / X. Sierra-Canto, F. Madera-Ramirez, V. Uc-Cetina// Ninth International Conference on Machine Learning and Applications. – Washington, DC, USA:IEEE, 2010. – P. 307–312.
13. Sewak M. Implement advanced deep learning models using Python / M. Sewak, Md. R. Karim, P. Pujari // Practical convolutional neural networks. – Birmingam-Mumbai : Packt Publishing, 2018. – 218 p.
14. Badrinarayanan V. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation / V. Badrinarayanan, A. Kendall, R. Cipolla // in IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2017. – Vol. 39, № 12. – P. 2481–2495.
15. Shijie J. Research on data augmentation for image classification based on convolution neural networks / J. Shijie, W. Ping, J. Peiyi and H. Siping // 2017 Chinese Automation Congress (CAC). – Jinan, China : IEEE, 2017. – P. 4165–4170.
16. Bjorck N. Understanding batch normalization / Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger // In Advances in Neural Information Processing Systems: 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada, 2018. – P. 7694–7705.
17. An Adaptive Kalman Filtering Approach to Sensing and Predicting Air Quality Index Values / J. Chen, K. Chen, C. Ding et al.] // in IEEE Access. – 2020. – Vol. 8. – P. 4265–4272.
18. Kang D.-J. Road lane segmentation using dynamic programming for active safety vehicles / Dong-Joong Kang and Mun-Ho Jung // Pattern Recognition Letters. – 2003. – Vol. 24, Issue 16. – P. 3177–3185.
19. Daigavane P. M. Road Lane Detection with Improved Canny Edges Using Ant Colony Optimization / P. M. Daigavane, P. R. Bajaj // 3rd International Conference on Emerging Trends in Engineering and Technology. – Goa, India, 2010. – P. 76–80.
20. Oliveira G. L. Efficient deep models for monocular road segmentation / G. L. Oliveira, W. Burgard, T. Brox // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – Daejeon, Korea (South), 2016. – P. 4885–4891.
21. Lane detection under artificial colored light in tunnels and on highways: an IoT-based framework for smart city infrastructure / [S. Ghanem, P. Kanungo, G. Panda et al.] // Complex Intell. Syst. – 2023. – Vol. 9, 2023. – P. 3601–3612.
22. Kortli Ya. Deep embedded hybrid CNN–LSTM network for lane detection on NVIDIA Jetson Xavier NX / [Yassin Kortli, Souhir Gabsi, Lew F. C. et al.], // Knowledge-Based Systems. – 2022. – Vol. 240. – P. 1–33.
23. A2-FPN: Attention Aggregation Based Feature Pyramid Network for Instance Segmentation / [Miao Hu, Yali Li, Lu Fang, Shengjin Wang] // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – 2021. – P. 15343–15352.
24. Zapata M.A.D., Erkent Ö., Laugier Ch. YOLO-based Panoptic Segmentation Network / Manuel Alejandro Diaz Zapata, Özgür Erkent, Christian Laugier // COMPSAC 2021 – Intelligent and Resilient Computing for a Collaborative World 45th Anniversary Conference. – Madrid, Spain, Jul 2021. – P. 1–5.
25. Train here, drive there: ROS based end-to-end autonomous-driving pipeline validation in CARLA simulator using the NHTSA typology / [C. Gómez-Huélamo, J. Del Egido, L. M. Bergasa et al.] // Multimed Tools Appl. – 2022. – Vol. 81. – P. 4213–4240.
26. Kostrikov I. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels / I. Kostrikov, D. Yarats, R. Fergus // arXiv preprint arXiv:2004.13649. – 2020. – P. 1–22.
27. Li Sh. Pytorch distributed: Experiences on accelerating data parallel training / Shen Li et al. // arXiv preprint arXiv:2006.15704. – 2020. – P. 1–14.
28. Severino A. Autonomous Vehicles: An Analysis Both on Their Distinctiveness and the Potential Impact on Urban Transport Systems / [A. Severino, S. Curto, S. Barberi et al.] // Appl. Sci. – 2021. – Vol. 11, № 8:3604. – P. 1–17.

DOI 10.15588/1607-3274-2023-4-11