CHAPTER 2

Solving pattern recognition problems in shipping monitoring systems based on CNN-models

Pavlo Mykhalichenko
Tetiana Cherniavska
Bohdan Cherniavskyi
Viktor Nadtochii
Anatolii Nadtochyi
Maiia Korkh

Abstract

The chapter examines modern approaches to solving pattern recognition problems in shipping monitoring systems using convolutional neural network (CNN) models. Traditional rule-based and classical classification methods remain ineffective for complex visual tasks in inland and maritime transport due to high variability of vessel shapes, viewing angles, illumination conditions, and significant intraclass differences. Deep learning architectures – particularly CNNs – are shown to provide robust and scalable solutions by automatically extracting hierarchical features and forming stable, noise-resistant image representations.

The chapter provides a comprehensive review of the fundamental principles of CNN construction, including convolutional layers, feature maps, activation functions, pooling operations, regularization techniques, batch normalization, and optimization strategies. Special attention is devoted to the mechanisms of training neural networks: gradient-based optimization, stochastic gradient descent, hyperparameter tuning, prevention of overfitting, and the use of computational graphs for backpropagation.

Experimental evaluation was conducted using a dataset comprising more than 12,000 annotated images of inland waterway vessels, including tugboats, barges, passenger ships, and dry cargo vessels, captured in real river-port conditions under varying illumination, viewpoints, partial occlusions, and noise. The proposed CNN model achieved an overall classification accuracy of 93.4%, with precision of 92.1%, recall of 91.7%, and an F1-score of 91.9%. Comparative analysis demonstrates that

 DOI: 10.21303/978-9908-845-05-0.ch2

the proposed approach outperforms classical methods such as k-nearest neighbors, linear classifiers, and feature-based support vector machines by 8–12% on average.

The practical importance of CNN-based methods for real-time monitoring of river port water areas in Ukraine is emphasized. The proposed models enable automated classification and recognition of surface vessels, enhancing the efficiency, accuracy, and adaptability of intelligent surveillance systems. The results contribute to the development of advanced decision-support tools in maritime transport, improving safety, environmental monitoring, and operational management of shipping flows.

Keywords

Convolutional neural networks, pattern recognition, vessel classification, shipping monitoring systems, deep learning, computer vision, feature extraction, optimization, batch normalization, river transport, visual information processing, machine learning, transport security, environmental monitoring.

2.1 Introduction

The novelty of this work lies not in proposing a fundamentally new convolutional neural network architecture, but in the targeted adaptation and systematic validation of CNN-based approaches for an underexplored application scenario – vision-based monitoring and classification of inland waterway vessels in river-port environments.

Unlike most existing studies that focus on open-sea maritime surveillance, satellite or SAR imagery, or multimodal fusion of AIS and visual data, this chapter addresses vessel type classification based solely on visible-spectrum images acquired from shore-based cameras operating under real port conditions.

The key novel aspects of the proposed approach include:

– its application to inland river ports characterized by limited camera elevation, variable viewpoints, dense object layouts, and complex backgrounds;

– its focus on vessel type classification rather than generic ship detection;

– a quantitative comparison with classical classifiers under identical inland waterway conditions;

– its practical orientation toward real-time port monitoring systems without reliance on AIS data.

A critical analysis of prior studies shows that while existing deep learning-based approaches demonstrate high performance in maritime and satellite scenarios, their direct applicability to inland waterways is limited due to differences in scene

geometry, data availability, and operational constraints. The proposed approach addresses these limitations, thereby extending the state of the art to a new and practically significant domain.

Within this chapter, the study is focused on the problem of visual classification of inland waterway vessels in port water areas based on images acquired from shore-based video surveillance systems operating in the visible spectrum. The primary attention is given to distinguishing the main types of river transport vessels, including tugboats, barges, passenger ships, and dry cargo vessels, under real operating conditions of port infrastructure. The study is deliberately limited to the analysis of two-dimensional static images without the use of temporal information and without incorporating auxiliary data sources such as automatic identification system (AIS) messages, radar measurements, or satellite imagery. This approach corresponds to the typical operating conditions of local port monitoring systems, where visual information from shore-based cameras is the primary available data source.

The chosen methodological approach is based on the use of convolutional neural networks, which is motivated by the limited effectiveness of classical computer vision methods in vessel recognition tasks under conditions of high intra-class variability, complex background environments, varying illumination, and constrained viewing angles. In contrast to approaches relying on handcrafted features and heuristic rules, convolutional neural networks enable automatic extraction of hierarchical visual feature representations that are robust to these factors. The focus on vessel type classification, rather than detection or tracking, is determined by the practical requirements of port monitoring systems, where the presence of an object has already been established and the key task is rapid identification to support operational decision-making. At the same time, it should be noted that the obtained results may require further adaptation when applied to other regions or in scenarios involving video streams or multimodal data sources.

2.2 Solving problems of classification of visual information

Computer vision (CV) is the analysis of visual data. The volume of this data in the world is constantly growing. About 80% of all Internet traffic is video – and this is without taking into account images and other types of visual information. Therefore, it is important to develop algorithms that can understand and process this data.

Visual information is sometimes compared to dark matter by analogy with physics. Dark matter makes up a very large fraction of the mass of the Universe, but it is not directly possible to observe it. Visual data is about the same: it contains a lot

of bits flying around the Internet. But it is very difficult for CV algorithms to understand what they actually consist of.

Visual perception affects many fields of science and technology: in physics it is important to understand the process of image formation, in biology and psychology scientists' study how people and animals perceive and process visual information. In robotics and automotive engineering, images help with orientation in the territory. To create systems that implement visual perception algorithms, knowledge in the fields of computer science, mathematics and design is required.

The main task that CV focuses on is image classification. That is, the algorithm must assign an image to one of the previously known categories. This problem is being solved both in research circles and in industry [1].

The engine of progress in solving this problem was the recently invented convolutional neural networks (CNNs). The real breakthrough came in 2012, when G. Hinton and his graduate students A. Kryzhevsky and I. Sutzkever created a seven-layer ANN AlexNet. It performed very well in the ImageNet competition. Since then, there has been a trend towards creating ever deeper networks. In 2014, the multilayer GoogLeNet and VGG appeared. At the same time, a paper was published by researchers from Microsoft Research Asia, who created a residual neural network (RN) with 152 layers.

The main breakthrough in the development of neural architectures occurred only a few years ago. This is primarily due to the emergence and rapid growth in the performance of graphics processors GPU [2]. These processors allow processing huge amounts of data in parallel, which makes them an ideal tool for large-scale computing and training RN. In addition, the amount of data available for training has increased significantly.

The CV opens up a lot of potential tasks, the main one of which is to teach a machine to see the world like a human. This goal is still unattainable, as it gives rise to a lot of related problems: for example, recognizing human activity from video, accurate 3D reconstruction of objects, semantic image segmentation, and many others. But as it is constantly moved forward and invent such amazing things as augmented and virtual reality, let's probably come up with new, interesting solutions [1].

The current state of research in vessel recognition for maritime and inland waterway monitoring is characterized by a rapid transition from traditional computer vision algorithms to deep learning-based approaches. Several recent surveys confirm that convolutional neural networks and their derivatives have become the de facto standard for ship detection and classification in satellite imagery, aerial data, and video streams. M. J. Er and Y. Zhang, for example, provide a comprehensive review of deep learning-based ship detection techniques, ranging from early CNN models

to modern one-stage detectors such as the YOLO and SSD families, and consistently demonstrate that deep architectures outperform classical methods in terms of accuracy and robustness to noise and background clutter [3].

Further progress has been driven by dedicated models for ship detection and tracking in video. B. Zhang et al. presents an extended survey of deep-learning methods for automatic ship detection and tracking, highlighting four significant challenges: environmental variability, multi-scale targets, occlusions, and the need for lightweight models capable of real-time performance [4]. For synthetic aperture radar (SAR) imagery, several surveys emphasize the specific difficulties related to speckle noise, background complexity, and limited spatial resolution, and point to an increasing adoption of advanced deep models, including transformer-based architectures [5].

In parallel, a new class of CNN-Transformer hybrid models have emerged. D. Liu [6] introduces transformer-oriented detectors (e.g. TS2Anet and DETR-like frameworks) tailored to ship detection in complex maritime environments, such as congested ports and high-clutter coastal regions [6]. More recently, Y. Wang and X. Li proposed Ship-DETR, a transformer-based detector designed for efficient ship detection in challenging maritime scenes, demonstrating the advantages of attention mechanisms in handling irregular spatial structures and densely populated scenes [7].

Another important line of work focuses on multimodal fusion, which combines visual data (such as RGB or SAR imagery) with Automatic Identification System (AIS) messages and other sensor sources. Studies by A. Galdelli et al. demonstrate that integrating heterogeneous data streams can significantly enhance the reliability of ship detection and classification, notably when individual channels are partially missing or severely degraded [8]. However, most of these approaches are tailored explicitly to open-sea scenarios and satellite or airborne platforms, where the observation scale and typical traffic patterns differ substantially from those in inland waterways and river ports.

In contrast to the majority of existing research that focuses on maritime domains, a more petite but growing body of work addresses inland waterways. M. Salem et al. proposes a CNN-based method for classifying vessel types on inland waterways. Yet, their experiments rely on relatively standardized imagery and do not fully reflect the complexity of port environments [9]. Studies by G. Agorku emphasize the significance of deep learning for barge load estimation, autonomous vessel control, and performance assessment of inland waterway freight corridors, primarily in the context of AIS-based trajectory analysis rather than vision-based recognition [10].

Within this landscape, the approach presented in this chapter focuses specifically on **vision-based classification of inland waterway vessels in real river-port**

conditions in Ukraine, using images captured by shore-based cameras operating in the visible spectrum. Unlike most prior work, which targets satellite surveillance or generic maritime scenes, the proposed CNN model is explicitly optimized for:

– varying viewpoints typical of shore-side infrastructure;
– complex and cluttered backgrounds (piers, port facilities, other vessels);
– limited sensor height and constrained fields of view;
– the characteristic mix of inland and mixed navigation vessel types.

Consequently, this chapter advances the state of the art in two key directions:

1. It demonstrates the practical applicability of CNN-based classification to **river-port monitoring on inland waterways**, a scenario that remains relatively underexplored in literature.

2. It provides a **quantitative evaluation** on a real dataset collected from Ukrainian river ports, enabling a direct comparison between CNN-based methods and classical classifiers discussed earlier in the chapter.

The experimental study described in this chapter is based on a combination of publicly available datasets and a proprietary image collection obtained under real river-port operating conditions. For preliminary testing and architectural validation, well-known open benchmark datasets for image classification were used, including CIFAR-10, which provides standardized RGB images and is commonly applied for verifying the learning capability of convolutional neural networks. The use of such benchmark datasets ensures methodological consistency with prior studies and enables reproducibility of baseline experiments.

The primary dataset for vessel classification was formed from images acquired by shore-based video surveillance cameras installed in river-port areas. This dataset includes visual samples of inland waterway vessels such as tugboats, barges, passenger ships, and dry cargo vessels captured under varying illumination conditions, viewing angles, background clutter, and partial occlusions. Due to operational and security constraints, this dataset is not publicly available; however, its structure, class composition, and acquisition conditions are described in sufficient detail to enable replication using similar port monitoring systems.

All computational experiments were conducted using widely adopted open-source software tools. The implementation of neural network models was performed using the Python programming language and deep learning frameworks such as PyTorch, which provide transparent model definitions, reproducible training procedures, and extensive community support. Image preprocessing and augmentation operations were carried out using standard computer vision libraries, including OpenCV and NumPy. The use of open-source tools ensures transparency of the experimental pipeline and facilitates independent reproduction of the proposed approach.

When the system receives an input image, it already knows a fixed set of categories or labels. These can be any objects: "tugboats", "barges", "passenger ships", "dry cargo ship". The computer must look at the image and assign it one of the labels.

From the outside, the task seems simple, since most of our visual system is programmed to recognize objects. But for a machine, it is not so simple. Especially in such a specific field as water transport, since the outlines of vessels are similar and it is necessary to determine certain classification features of each type of vessel.

When a computer analyzes an image, it doesn't "see" the whole picture of a cat or, for example, a tugboat. It only "sees" a giant grid of numbers. For example, if the image size is 800 by 600 and each pixel is represented by three numbers for the red, green, and blue channels, the resulting grid is $800 \times 600 \times 3 = 1{,}440{,}000$ numbers. It is very difficult to distinguish any particular object represented in the image from this grid.

This problem is called the "semantic gap" – a misunderstanding of the information contained in the data. For example, if to photograph a dry cargo ship from a different angle or in different lighting, the entire grid of numbers will look completely different. In addition, the photograph may only show part of the ship, for example the stern. Recognition algorithms must be resistant to such changes.

In addition to these difficulties, there is the problem of intraclass variation, where a single concept encompasses a multitude of visual manifestations. For example, passenger ships can be of different types, ages, and sizes. And recognition methods must handle all possible variations.

The first thing that comes to mind is to create reference rules. It is known that ships have certain deck equipment. In photos of them, it is possible to detect edges and then classify different angles and boundaries: for example, determine how the lines of the hull and bow of a ship meet.

But explicit rule sets don't work very well: any deviation can break everything, and new objects will have to be created with new conditions. Therefore, this approach is not scalable.

Instead of trying to manually create a set of rules, it is possible to open the Internet and collect a large dataset with photos of different types of river and sea vessels. For this, Google image search or a ready-made dataset is suitable. Then it is necessary to train the classifier by sending all the collected images to it. At the output, let's get a model that generalizes the knowledge of recognizing different objects. After that, it can work on new images and distinguish tugboats from dry docks.

So, instead of one function that simply recognizes an object in an input image, there are two: the first is called "training" – this is the process of processing images and creating a model. The second function – "prediction" – recognizes new photos. Together, they form the basis for CNN and DNN in general.

To begin, let's look at the simplest classifier, which is called the "nearest neighbor method". During the training process, it remembers all the original data, and during the prediction stage, it tries to find the most similar new images.

There are many different ways to compare two images. In the example below, let's use the $L_{1\ distance}$, also known as the Manhattan distance. It simply compares the pixels of the images. Let's say that there is a test sample of 4 × 4 pixels. Let's take one of the training images and calculate the absolute difference between the colors of the pixels of the training and test samples, the $L_{1\ distance}$

$$d_1(I_1, I_2) = \sum_p \left| I_1^p - I_2^p \right|, \tag{2.1}$$

and then sum up the obtained values.

If the training set contains N examples, then training will always be performed in constant time $p(1)$ [12], and prediction will be performed in linear time $p(N)$ because the test image is compared to each training example. Linear prediction time is not very good. In reality, classifiers should be trained slowly and tested quickly. More advanced algorithms work in this way: they can be trained for a long time in a data center or on cloud servers, and then run on mobile phones.

The l_2 *(Euclidean)* distance is also often used in classification problems. It looks like the square root of the sum of the squared differences between pixel colors.

Distance metrics make different assumptions about the expected geometry or topology of the space. l_1 forms a square region, while l_2 creates a circle. When the coordinate system is rotated, the Euclidean distance l_2 will not change, while the Manhattan (MN) l_1 will give a different result. It is important to take this spatial effect into account and choose the metric according to the original task.

The process of choosing values that affect the performance of any method, such as the number of neighbors k and the distance metric, is called hyperparameter tuning. Hyperparameters cannot be explicitly extracted from the training data and depend only on the algorithm itself, so there are no clear recommendations for their selection. Most often, it is possible to find values by trial and error, finding out which ones work best.

Let's figure out what "best" is:

1. Choose parameters that will give the highest accuracy on the training data. And this is a very bad idea. In the case of the nearest neighbor method, at $k = 1$, almost perfect accuracy is achieved during training, but the algorithm does extremely poorly on the test data. This is called "overtraining".

2. Split the population into training and testing sets and find hyperparameters that will make the algorithm perform better on the test samples. This strategy looks smarter, but in reality, it is also very bad. The main idea of L_1 MN is that it is possible

to know how the method will perform. And if to choose parameters that achieve good results on known images, then there is no guarantee that they will be achieved on other unknown images.

3. Split the dataset into training, testing, and validation sets; select hyperparameters for the evaluation data and test them on the test data. And that's a good idea. First, the classifier is trained with different parameter options. Then, the values that work best on the evaluation data are selected. After that, the model processes the test set only once. The accuracy achieved in this case shows the true efficiency of the classifier.

4. Divide the training data into many small subsamples, then cross-validate using different subsamples as the evaluation data, and average the results.

The method is called "cross-validation" and works well on small datasets, but requires a lot of computational resources for huge datasets.

The *k*-nearest neighbors' method is never applied to photographs because: it is very slow on test data; the metric of distances between pixel colors does not indicate the similarity of images.

Another problem is the so-called curse of dimensionality, which is associated with the exponential growth of the amount of data as the dimensionality of the space increases. Therefore, algorithms based on brute force become inefficient as the dimensionality of the system increases.

Linear classification algorithms are quite simple. However, they are used to create full-fledged neural networks. It is similar to Lego: it is possible to put different components together and build a "tower" convolutional neural network (CNN).

Let's use the CIFAR-10 dataset. To classify an image, it is possible to create a simple parametric model with two components. The first is the input data, usually denoted as *X*, and the second is a set of parameters or weights *W*. Now let's write a function that takes in the data *X* and the parameters *W* (**Fig. 2.1**) and then outputs 10 numbers describing the scores for each of the 10 categories in CIFAR-10.

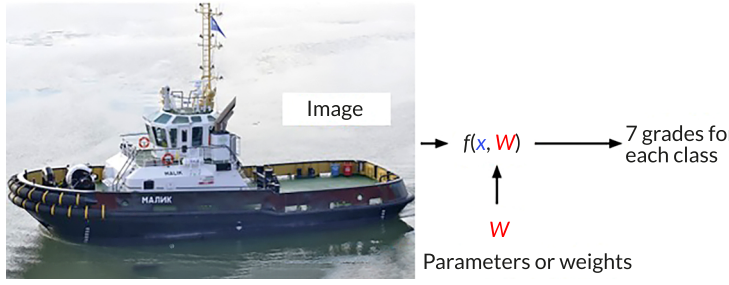

Fig. 2.1 Creating a linear estimate

A properly working model in the example above will give the highest score for the cat class. In this parametric approach, unlike the nearest neighbor method, let's generalize knowledge about the training data and use it to create parameters. Therefore, during testing, it was not possible to refer to the training sample every time.

To get a function that combines weights and data, the easiest way is to simply multiply them. This will be a linear classifier

$$f(x, W) = Wx + b, \tag{2.2}$$

where b – a linear vector, whose size is equal to the number of classes (in this case 10). It does not interact with the training data and produces independent predictions for the classes. For example, if the set contains many more images of tugboats than of dry cargo ships, then the elements of the bias vector corresponding to the cat class will be higher than the others.

The score for any class is the product of the image pixels and the corresponding rows of the weight matrix, with an offset added. This is similar to a pattern match: each row corresponds to some template image, and the score indicates its similarity to the original photo.

If to try to fold the rows of the weight matrix back into the image, it is actually possible to get patterns collected from the data. A linear classifier uses only one pattern for each class. By creating more complex functions that relate the data and parameters, it is possible to learn more patterns and achieve better accuracy.

In the decision domain, each image is represented as a point in a multidimensional space, which the linear classifier tries to fit within the boundaries of a linear solution (**Fig. 2.2**). In other words, it will separate the categories from each other by straight lines.

Another challenge is the phenomenon of multimodality, which occurs in unevenly distributed data. For example, photographs of ships may show them from different sides [12]. In this case, isolated islands with ships facing right, left, and other directions appear in the decision-making area.

Despite the challenges, linear classification is a simple algorithm that is easy to interpret and implement.

The main result that the loss function gives is an estimate of how well the classifier works on the sample. But it is possible to remember that the main goal of MN is to make the algorithm work correctly on the test sample. Therefore, a method that copes well with the training data may not work at all on new objects. This is called "overtraining". Suppose that our "dataset" consists of some points. If to force the algorithm to adapt perfectly to each point with zero losses, then the classification

graph will turn into a winding curve (**Fig. 2.3**, curve 1). But this is a bad result, because it is about accuracy on the test sample, not on the training one. If to check the work on the test data (marked by the green squares in **Fig. 2.3**), then the blue curve will become completely wrong. Most likely, the classifier should find some curve (**Fig. 2.3**, curve 2) that would approximately correspond to both those and other data.

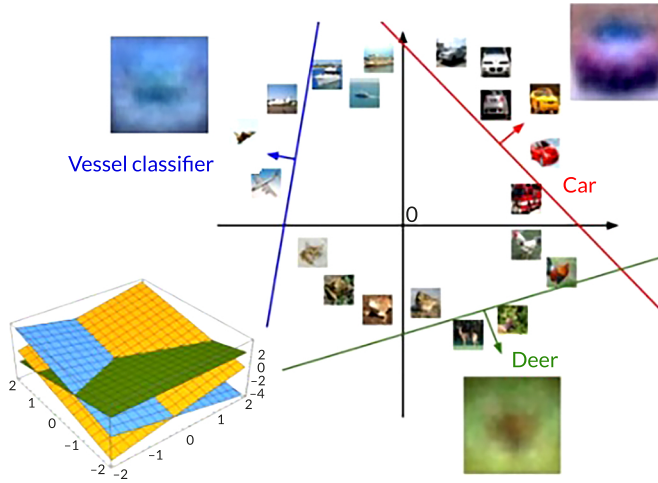

Fig. 2.2 Linear classification of a dataset

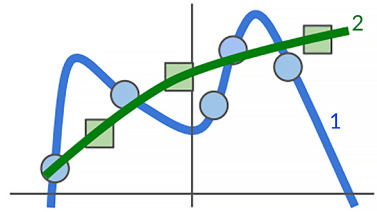

Fig. 2.3 Adding a test sample to the retrained model

This is the fundamental problem of MN. Various regularization methods are usually used to solve it.

It was said above that there are many options for optimal weights *W*. A regularization parameter is introduced specifically for this – it forces the model to choose such weights *W*, at which the solution will be the simplest. The concept of simplicity depends on the problem that the algorithm solves.

This idea is related to the principle of Occam's razor – if an observation can be explained by several hypotheses, the simplest one should be chosen. In the case of classification, the simplest partition of the data should be chosen – in the example above, this should be curve 2, not curve 1 of order *N*. So, our function now consists of two components: the training data loss (the sum of all L_i) and the regularization loss $R(W)$

$$L(W)=\frac{1}{N}\sum_{i=1}^{N}L_i\left(f\left(x_i,W\right),y_i\right)+\lambda R(W), \tag{2.3}$$

where λ – a hyperparameter that determines how strongly regularization will affect the model.

Now let's see what $R(W)$ functions are and how they work. In practice, quite a few types of regularization are used, but for the purposes of this chapter, let's consider only the most popular ones:

1. Regularization L_2.

It is used quite often in MN. It is the normal Euclidean norm of the parameters *W* (sometimes the square or half-square of the norm is taken). The idea is to simply add a "penalty" to the weights so that their values do not turn out to be too large. The function $R(W)$ looks like this

$$R(W)=\sum_k\sum_l W_{k,l}^2. \tag{2.4}$$

2. Regularization L_1.

L_1 uses the Manhattan norm *W* and also adds a penalty to the weights. With regularization the L_1 parameter matrix looks sparser

$$R(W)=\sum_k\sum_l\left|W_{k,l}\right|. \tag{2.5}$$

3. Elastic network.

This is a combination of regularization L_1 and L_2

$$R(W)=\sum_k\sum_l\beta W_{k,l}^2+\left|W_{k,l}\right|. \tag{2.6}$$

There are other types of regularizations, such as maximum norm, dropout, batch normalization, and stochastic depth. They are used in narrower GN problems, which will be explored in the following sections.

Let's imagine that we are walking through a large valley among mountains, fields and rivers. The height of each object in this landscape corresponds to the number of losses that appear at a certain setting *W*. Since the task is to achieve the

lowest losses, it is necessary to somehow find the lowest point in the valley. In practice, the optimization process looks like a set of iterative operations, where a random solution is taken as the starting point, which, using certain mathematical tools, is gradually improved.

Strategy 1. The first solution is to generate random values of *W* and see how well they work. But this is a bad idea and should not be used in practice. For example, for 10 classes of the CIFAR-10 "dataset", the probability of finding the corresponding *W* parameters will be 10%. This is a very bad indicator.

Strategy 2. Standing on a slope, it is possible to look for a descent. In mathematics, the analogue of finding a descent is the derivative. By taking a one-dimensional function *f*(*x*) and calculating its derivative at any point, it is possible to determine whether the function is increasing or decreasing at that point. But in MN, *X* is usually a vector, so the generalized analogue of descent is the vector of partial derivatives or gradient. The gradient indicates the direction of increase of a function, to find its decrease, a negative gradient is used.

So, now there is a parameter vector *W*, and our goal is to calculate the gradient vector ∇W. It is simply possible to take the derivatives of each individual element. This way finds out how much the losses will change if to move by an infinitesimal amount in one of the coordinate directions. This approach is called "numerical gradient". In most real cases, these calculations will be too slow. After all, the training data sometimes contains millions of examples, and the algorithms for processing them can be much more complex. To solve the problem, there is the so-called "analytic gradient". From the course of mathematical analysis, it is simple to write down the expression for the losses, and then use differential calculus and immediately find the necessary gradient. This becomes possible thanks to the analytical derivative, which does not require substitution of values, but works with the entire function at once [13]

$$\hat{f}(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} = \lim_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta x}. \quad (2.7)$$

It's actually better to use the analytical gradient, but check the solution numerically. This is called gradient checking.

First, let's initialize *W* with random values, calculate the losses and gradient, and then update the weights according to the negative direction of the gradient (**Fig. 2.4**).

"Step size" is a hyperparameter that specifies the step size moved with each new gradient calculation. It is sometimes called the Learning Rate (LR). This is one of the most important settings in MN.

```
while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad
```

Fig. 2.4 Listing of implemented gradient descent in Python

Let's see how it looks visually. In the graph below, the large multi-colored area is our loss function (**Fig. 2.5**). The red area is the minimum values that is possible to achieve, and the blue and green indicate high losses.

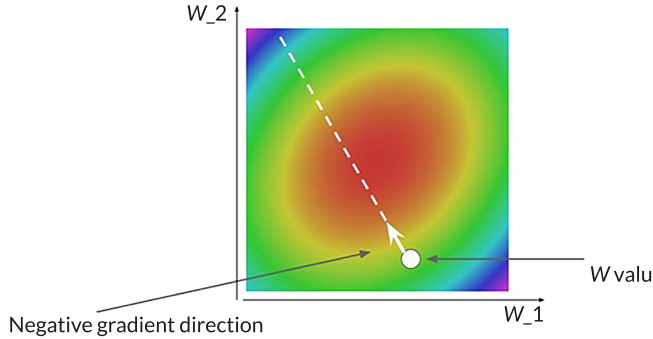

Fig. 2.5 Physical understanding of gradient descent

Let's start with the initial W at a random point and calculate the negative gradient direction that should lead step by step to the minimum loss. This is the most obvious example of stepwise gradient descent.

In the case of calculating the gradient W at each step, it is possible to consider all the training examples in the dataset. Because ∇W is the sum of the individual gradients caused by each training element. But their number can reach a million, and in this case, ordinary gradient descent will work very slowly. Therefore, stochastic gradient descent is often used in practice:

$$L(W)=\frac{1}{N}\sum_{i=1}^{N}L_i(x_i,y_i,W)+\lambda R(W); \tag{2.8}$$

$$\nabla_W L(W)=\frac{1}{N}\sum_{i=1}^{N}\nabla_W L_i(x_i,y_i,W)+\lambda\nabla_W R(W). \tag{2.9}$$

Its difference is that at each step the losses and gradient are calculated not on the entire "dataset", but on a small set of examples – a mini-package (and sometimes

only on one example). That is, the *W* parameters are updated after processing several objects, and not after passing through the entire dataset. This significantly speeds up the optimization process, and only one line is added to the code (**Fig. 2.6**):

```
while True:
  data_batch = sample_training_data(data, 256)
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad
```

Fig. 2.6 Listing implementation of stochastic gradient descent in Python

The function *f* takes data *x* and parameter *W*, as input and outputs a vector of scores *s* for each of the categories to be classified. The loss function *L* (for example, SVM) determines how correct the scores are, with its help it is possible to calculate the data loss. Regularization helps to learn about the simplicity of the model. The goal is to find the parameters *W* that correspond to the smallest losses. Let's use the negative direction of the gradient of the function *L*, which indicates the path to its minimum. There are two main ways to calculate the gradient: analytical and numerical. The numerical method is quite simple, but works very slowly and gives approximate values [14]. The analytical gradient is more accurate and faster, but it is easy to make mistakes when calculating it. To avoid this and easily calculate the gradient even for complex functions, it is better to use computational graphs.

2.3 Development of a neural network for classifying visual information

In order to ensure the possibility of independent reproduction of the obtained results, this chapter provides a detailed description of the main stages involved in the construction and training of the convolutional neural network model for vessel classification. The overall solution pipeline follows a sequential structure that includes input image preprocessing, formation of training and test datasets, definition of the neural network architecture, model training using gradient-based optimization methods, and quantitative evaluation of classification performance.

During image preprocessing stage, input images are resized to a fixed spatial resolution, pixel intensity values are normalized, and basic data augmentation techniques are applied, when necessary, in order to improve robustness to illumination changes and viewpoint variations. The convolutional neural network architecture is designed according to a classical layered scheme that alternates convolutional

layers, nonlinear activation functions, and dimensionality reduction layers, followed by a fully connected classification block. Model training is performed using the backpropagation algorithm in combination with stochastic gradient descent and its commonly used variants.

The algorithmic sequence of model training and inference can be represented in the form of generalized pseudocode, reflecting the key computational steps of the process, including network parameter initialization, forward propagation, loss function computation, weight updates, and accuracy evaluation on a validation dataset. Such a representation enables the proposed approach to be reproduced using any modern deep learning framework.

The structural diagram of the convolutional neural network presented in this chapter illustrates the general logic of model construction and the interactions between its main components, serving as a reference template for implementing similar classification systems in water area monitoring applications. Taken together, the descriptions of the network architecture, training procedure, and experimental setup provide a sufficient level of detail to allow independent verification and replication of the reported results.

A computational graph is an illustrated representation of any function consisting of vertices (sometimes called nodes) and edges. Vertices are computational operations to be performed, and edges connect them in a certain sequence. **Fig. 2.7** shows an example of a graph with a classifier.

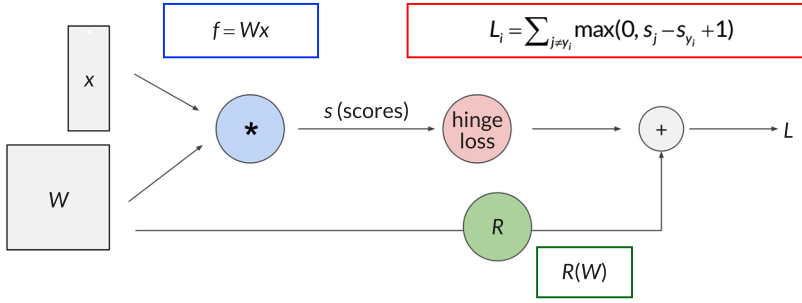

Fig. 2.7 Artificial neuron as a graph

A node with an operation (*) means the multiplication of the parameter matrices *W* and the data *x*, which results in a weight vector *s*. The next dependent loss node (hinge loss) defines the data loss *L*. Node *R* computes the regularization. Finally, it is possible to obtain the total loss by summing the regularization and data loss.

The advantage of graphs is that they allow to use the so-called backpropagation method. This algorithm recursively uses the differentiation rule of a complex function to calculate the gradient of each variable in the graph [15]. The method becomes very useful for really complex functions, such as those used in convolutional neural networks (CNNs).

Let's recall the linear classifier function. If to "transform" it on a neural network (NN), it is necessary to split the parameters W into two parts: W_1 and W_2 and apply one linear transformation on top of the other (**Fig. 2.8**).

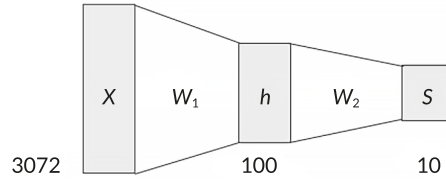

Fig. 2.8 Simple two-layer neural network

A simple two-layer NM with two linear layers was obtained

$$f = W_2 \max(0, W_1 x).$$

In **Fig. 2.8** above, x – the input data, h – the intermediate nonlinearity, and s – the output vector of estimates. In a broader sense, neural networks are complex functions made up of simple ones.

It was previously noted that each row of the weight matrix W is a template of one of the classes. These templates looked like an average object. It was also found that there is a problem with such single templates: for example, the class of ships in the datasets can be colored in different colors. Multilayer networks solve this problem: W_1 contains the single templates themselves, but now the estimates for them are stored in an intermediate nonlinear variable h. The next layer W_2 will combine the templates using a weighted sum, which will allow more accurate estimates of cars of other colors and other various objects.

By the way, nothing prevents from adding another layer to improve recognition accuracy

$$f = W_3 \max(0, W_2 \max(0, W_1 x)). \tag{2.10}$$

This is how deep NMs appear.

Convolutional neural networks (CNNs) handle large data sets well and are trained efficiently on GPUs using parallel computing. These features have become crucial to the fact that AI is now used almost everywhere. It solves problems such as image classification and search, object detection, segmentation, and is also used in more specialized fields of science and technology.

Let's assume that there is a 32 × 32 × 3 original 3D image. Let's stretch it into one long vector 3072 × 1 and multiply it with a weight matrix of size, for example, 10 × 3072. As a result, it is necessary to get an activation (an output with class scores) – to do this, let's take each of the 10 rows of the matrix and perform a scalar product with the original vector (**Fig. 2.9**).

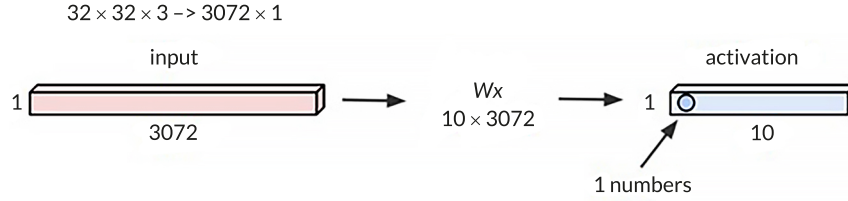

Fig. 2.9 Unfolding an image into a one-dimensional vector

The number 1 is the result of the dot product between one row of the weights *W* and the output vector. As a result, let's get a number that can be compared to the value of a neuron. In our case, let's get ten values. Fully connected layers work on this principle.

The main difference between convolutional layers is that they preserve the spatial structure of the image. Now it is possible to use weights in the form of small filters – spatial matrices that pass through the entire image and perform a dot product on each of its sections **Fig. 2.10**. In this case, the size of the filter always corresponds to the dimensions of the original image.

As a result of the image pass, let's obtain an activation map, also known as a feature map. This process is called spatial convolution [16, 17].

A large number of filters can be applied to an image and different activation maps can be obtained at the output. This way forms one convolutional layer. To create a whole NM, the layers are alternated one after the other, and activation functions (for example, ReLU [17]) and special pooling layers are added between them, which reduce the size of the feature maps.

In the first layers, convolutional filters are usually associated with low-level image features, such as edges and boundaries. In the middle, there are more complex

features, such as corners and circles. And in the final layers, the filters are more like some specific features that can be interpreted more broadly.

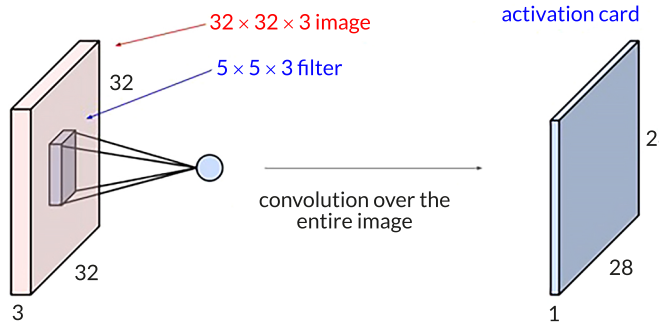

Fig. 2.10 Collapse of the image

Fig. 2.11 below shows examples of 5 × 5 filters and the resulting activation maps when applied to the original image (top left). The first filter (circled in red) looks like a small section of a border tilted to the right. If to apply it to a photograph, the highest values (white) will be where there are edges with roughly the same orientation. It is possible to see this by looking at the first activation map.

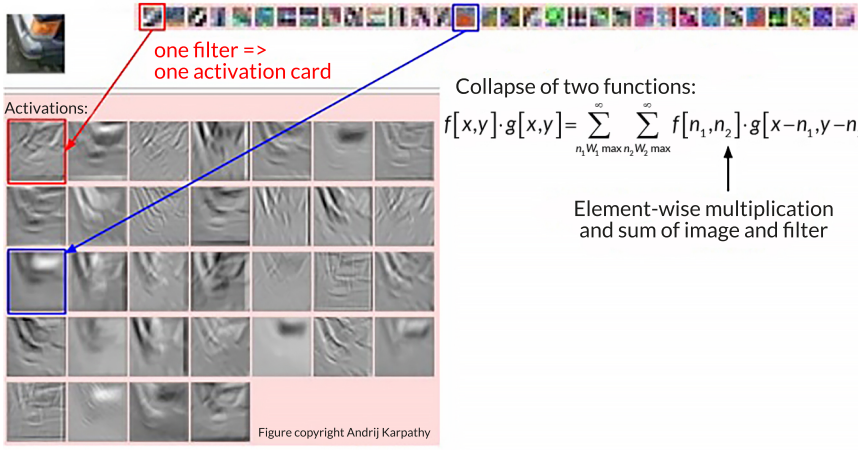

Fig. 2.11 Example of 5 × 5 filter operation

Thus, one layer of the NM finds the areas of the image that are most similar to the given filters. This process is very similar to the usual convolution of two functions. It shows how objects correlate with each other. Putting everything together, let's approximately get the following picture: taking the original photo, let's pass it through alternating convolutional layers, activation functions and pooling layers. At the end, let's use the usual fully connected layer, connected to all the outputs, which shows the final estimates of each class (**Fig. 2.12**).

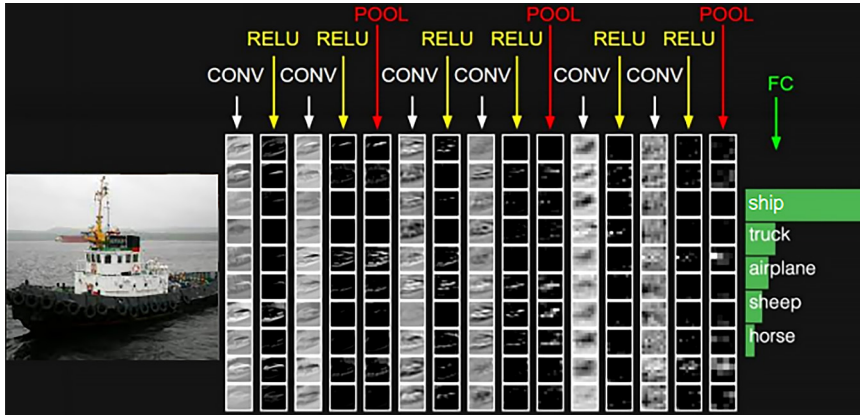

Fig. 2.12 Scheme of the convolutional operation neural networks

This is the principle that modern ZNMs work on. So, let's look at the full set of functions of the convolutional layer:

1. Accepts an original image of a certain dimension $W_1 \times H_1 \times D_1$.

2. Uses four hyperparameters for filters: the number of filters K; their size F; step S; number of extra zeros P (used to fill in the "lost" areas of the image after applying convolution).

3. Outputs an activation map of size $W_2 \times H_2 \times D_2$, where:

$$W_2 = (W_1 - F + 2P) / S + 1; \quad (2.11)$$

$$H_2 = (H_1 - F + 2P) / S + 1; \quad (2.12)$$

$$D_2 = K. \quad (2.13)$$

4. Uses $F \times F \times D_1$ weights per filter, for a total of $(F \times F \times D_1) \times K$ weights and K offsets.

Learning NM is an unpredictable and exciting process, which, however, requires careful preparation. In general, it can be divided into three main stages:

– **one-time setup**. Activation function selection, data preprocessing, weight initialization, regularization, gradient testing;

– **learning dynamics**. Tracking the learning process, optimizing and updating hyperparameters;

– **evaluation**. Using ensemble methods [18].

Activation function selection. Earlier it was found out that each layer of the NM receives input data. They are multiplied by the weights of the fully connected or convolutional layer, and the result is passed to the activation function or nonlinearity. Let's also talk about sigmoid and ReLU, which are often used as such functions. But the list of possible options is not limited to them. The task is to choose the activation function.

Data preparation. There are three most common methods of data preprocessing. Let's assume that the data X is a matrix of size $[N \times D]$:

1. *Subtracting the mean*. To avoid skewing the data and make it symmetric about zero, the mean value is subtracted from each element. This helps prevent the situation where all the original numbers turn out to be only positive or negative. In *NumPy*, the operation has the form $X = np.mean(X, axis = 0)$. In particular, when processing images, it is possible to subtract one value from all pixels (for example, $X = np.mean(X)$) or do it separately for each of the three-color channels.

2. *Normalization*. Transforming the data so that they are approximately the same scale. One option is to divide each dimension by its standard deviation: $(X = np.std(X, axis = 0))$. Another option is to normalize each value so that min and max are −1 and 1, respectively. Normalization should only be used if the original data has different formats or units. Pixel values do not fall outside the range of 0 to 255, so there is no need to perform normalization for them.

Initializing the weights. So, the neural network architecture was built and the data were prepared. Before starting training, it is necessary to initialize the parameters (weights).

What not to do: set the weights to zero. This will cause all neurons to behave the same way – not what we want. The NM should learn different features.

Small random variables. The most convenient option is to assign small values to the weights. Then all neurons will be unique and will gradually integrate into different parts of the network during the learning process. The implementation can look like this: $W = 0.01 \times np.random.randn(D, H)$. The $randn(n)$ method forms an array of size $n \times n$, the elements of which are random variables distributed according to the normal law with a mathematical expectation of 0 and a standard deviation

of 1 (Gaussian distribution [19]). The disadvantage of this is that it works well for small architectures, but copes much worse with bulky NMs.

Calibration using 1 / *sqrt*(*n*). The problem with the above method is that the variance of the random variables increases with the number of neurons. To avoid this, it is possible to scale the weights by dividing them by the square root of the number of inputs: *W* = *np.random.randn*(*n*) / *sqrt* (*n*). This ensures that all neurons in the network initially have approximately the same output distribution.

It is also possible to use the option *W* = *np.random.randn*(*n*) × *sqrt*(2.0 / *n*), which was proposed in one of the studies [20, 21]. It leads to the most successful distribution of neurons, so in practice it is possible to recommend using it.

Batch normalization. A method also known as batch normalization solves many initialization problems by forcing all activations to adopt a unit Gaussian distribution at the beginning of training.

Let's consider a small number of neuron activations on a layer. Let the activation function be a vector of dimension *D*: *x* = (*x*(1), ..., *x*(*D*)). Let's normalize it by each dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - M\left(x^{(k)}\right)}{\sqrt{D\left(x^{(k)}\right)}}, \tag{2.14}$$

where *M*(*x*) – the mathematical expectation; *D*(*x*) – the variance, which is calculated over the entire training sample.

So, instead of initializing the weights, it is possible to use this simple differentiable function and get a normal distribution at each layer.

Batch normalization is usually applied between layers (fully connected or convolutional) and activation functions.

This is a very useful algorithm that is often used in modern MN. NMs that use batch normalization, are much more resistant to bad initialization.

Hyperparameter optimization. As is seen, training neural networks involves many stages of hyperparameter tuning. The most common are:

– initial SHN;

– the attenuation graph of the SH (for example, constant attenuation);

– regularization power.

If desired, it is even possible to modernize the network architecture if to suspect that it was not chosen very well.

The graphical representations of the experimental results are based on the quantitative performance indicators summarized in **Table 2.1**. As shown by the obtained

data, the proposed convolutional neural network–based model demonstrates a clear advantage over classical classification methods across all major evaluation metrics. In particular, the overall classification accuracy of the CNN approach reaches 93.4%, exceeding the performance of *k*-nearest neighbors, linear classifiers, and support vector machines with handcrafted features by approximately 7–12%.

Table 2.1 Quantitative performance metrics for vessel classification

Classification method	Overall accuracy, %	Precision, %	Recall, %	F1-score, %
k-nearest neighbors	81.2	78.5	76.9	77.7
Linear classification	84.6	83.1	82.4	82.7
SVM (handcrafted features)	86.1	84.8	83.9	84.3
CNN (proposed approach)	93.4	92.1	91.7	91.9

The precision and recall values indicate that the proposed model is capable of reliably distinguishing different vessel types under conditions of complex background scenes, varying illumination, and changing viewing angles that are typical for inland river-port environments. The aggregated F1-score confirms the balanced performance of the classifier and indicates the absence of a significant trade-off between false-positive and false-negative errors.

Thus, the presented numerical results directly substantiate the trends illustrated in the corresponding graphs and provide a quantitative basis for the conclusions regarding the improved accuracy and effectiveness of convolutional neural networks in shipping monitoring and vessel classification tasks.

Learning rate is one of the most important values. **Fig. 2.13** on the left shows the effects that occur when the SH changes.

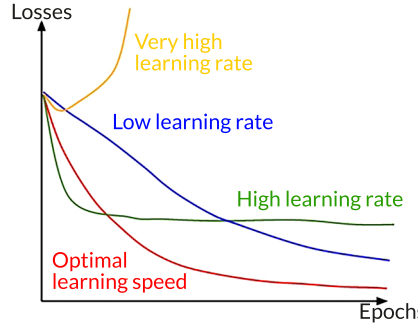

Fig. 2.13 Effect of learning rate on its accuracy

The second important thing to monitor is the accuracy of the network on the training and evaluation data. If to put them on the same graph, it is possible to assess the presence of overfitting, as evidenced by the diverging curves (**Fig. 2.14**).

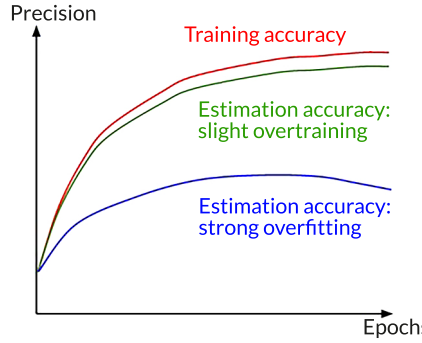

Fig. 2.14 Dependence of estimation accuracy on retraining

To find the optimal hyperparameters, it is possible to write a separate function that will independently select them and perform the optimization. In this case, it is better to use a non-uniform search (also known as "grid search"), but a random search, which often gives much better results.

2.4 Conclusion

The conducted research demonstrates that modern tasks of visual information recognition in water-area monitoring systems and inland waterway transport far exceed the capabilities of traditional computer vision algorithms. The analysis of theoretical and applied aspects of vessel-classification systems shows that methods based on heuristic rules, classical classifiers, or simple similarity metrics cannot ensure the required scalability or accuracy under real operating conditions of port infrastructure [21]. High variability in hull shapes, viewing angles, illumination, intraclass diversity, and the so-called *semantic gap* significantly complicate the construction of reliable models capable of generalizing information based on primitive features.

Based on theoretical justification and comparison of approaches, it has been established that only deep learning – and specifically convolutional neural networks (CNNs) – provides the necessary foundation for automatically extracting

relevant features from large sets of images [22]. CNN architectures demonstrate the ability to form multilayered representations, where initial layers respond to basic gradients and contours, and deeper layers extract complex spatial structures characteristic of particular vessel types. This approach ensures robustness to rotations, perspective changes, partial visibility, and various noise effects that are inevitable in video streams from river-port surveillance systems [23, 24].

Special attention was given to the training process of neural models, including the choice of loss functions, regularization techniques (l_1, l_2, elastic net), the use of batch normalization, optimization strategies, and the application of derivatives, gradient descent, and stochastic gradient descent. The findings show that training effectiveness largely depends on proper weight initialization, correct data preprocessing, and optimal hyperparameter tuning. The use of randomized search, cross-validation, and regularization significantly reduces overfitting and improves overall classification accuracy [25–27].

Thus, the results confirm that CNN-based models represent the most promising and effective technological foundation for developing intelligent monitoring systems for water transport. Their application provides:

– improved accuracy of vessel classification in port water areas;
– the capability for automated processing of large video streams in real time;
– adaptability to variable external conditions;
– reduced need for manual programming of rules and features;
– robustness to noise and incomplete data [28].

In the long term, implementation of the described models can ensure substantial progress in developing autonomous monitoring systems, enhance the efficiency of transport-flow management, optimize the operation of Ukrainian river ports, and strengthen safety mechanisms in the water-transport sector. At the same time, it can contribute to broader methodological application in areas such as organizational management, ecological analysis, engineering cybernetics, and simulation modeling, even including tasks related to the environmental protection of water resources [29]. The obtained results create a solid scientific basis for further advancement of intelligent image-analysis methods, integration of deep-learning models into complex surveillance systems, and development of new algorithms for vessel recognition in challenging environments.

Further research within the direction considered may focus on extending and refining the proposed approach. In particular, the use of continuous video streams instead of individual static images would allow temporal information to be incorporated and could improve classification reliability in complex scenarios involving object occlusions or visually ambiguous cases. Another important direction is the expansion

of the dataset to include images collected in different regions and under diverse operating conditions, which would enhance the generalization capability of the models.

A promising avenue for future work is the integration of visual data with additional information sources, such as navigation or sensor-based systems, in order to increase robustness and reduce the impact of noise and incomplete observations. Moreover, further studies may address optimization of convolutional neural network architectures with respect to computational complexity, inference latency, and hardware constraints, which are critical factors for practical deployment in real-time port monitoring systems.

Thus, the results obtained in this study provide a foundation for continued development of intelligent shipping monitoring systems and can serve as a starting point for expanding the functionality of decision-support and traffic management solutions for inland waterway transport.

Conflict of interest

The authors declare that there is no conflict of interest in relation to this paper, as well as the published research results, including the financial aspects of conducting the research, obtaining and using its results, as well as any non-financial personal relationships.

Financing

The study was performed without financial support.

Data availability

The data that support the findings of this study will be made available by the authors on reasonable request.

Use of artificial intelligence statement

The authors have used artificial intelligence technologies solely to verify the correctness and consistency of the English translation of the manuscript in an

academic style. In particular, the authors used the generative language model ChatGPT (OpenAI, GPT-5.1 Thinking) for language checking and minor stylistic refinement. The authors bear full responsibility for the content of the final manuscript; the AI tool is not credited as an author and is not responsible for the reported results.

Acknowledgments

The authors received no specific support from individuals or organizations that should be acknowledged beyond their institutional affiliations stated in the manuscript.

Authors' contributions

Pavlo Mykhalichenko: Conceptualization, Methodology, Supervision, Validation, Writing – review & editing.

Tetiana Cherniavska: Conceptualization, Theoretical framework, Formal analysis, Writing – original draft, Writing – review & editing.

Bohdan Cherniavskyi: Methodology, Data curation, Software, Investigation (experiments), Visualization, Writing – review & editing.

Viktor Nadtochii: Conceptualization, Domain expertise (water transport and monitoring systems), Validation, Formal analysis, Writing – review & editing.

Anatolii Nadtochyi: Methodology, Modeling and algorithmic development, Formal analysis, Interpretation of results, Writing – review & editing.

Maiia Korkh: Literature review, Investigation, Integration of results, Project administration, Writing – review & editing.

All authors contributed equally to the scientific content of this chapter, jointly developed the conceptual idea of the work, and approved the final version of the manuscript.

References

1. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S. et al. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115 (3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

2. Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097–1105.
3. Er, M. J., Zhang, Y. (2023). Ship detection with deep learning: a survey. Artificial Intelligence Review, 56 (10), 11825-11865. https://doi.org/10.1007/s10462-023-10455-x
4. Zhang, B., Liu, J., Liu, R. W., Huang, Y. (2025). Deep-learning-empowered visual ship detection and tracking: Literature review and future direction. Engineering Applications of Artificial Intelligence, 141, 109754. https://doi.org/10.1016/j.engappai.2024.109754
5. Awais, C. M., Reggiannini, M., Moroni, D., Salerno, E. (2025). A Survey on SAR ship classification using Deep Learning. arXiv preprint arXiv:2503.11906. https://doi.org/10.48550/arXiv.2503.11906
6. Liu, D. (2023). TS2Anet: Ship detection network based on transformer. Journal of Sea Research, 195, 102415. https://doi.org/10.1016/j.seares.2023.102415
7. Wang, Y., Li, X. (2025). Ship-DETR: A Transformer-Based Model for EfficientShip Detection in Complex Maritime Environments. IEEE Access, 13, 66031–66039. https://doi.org/10.1109/access.2025.3559107
8. Galdelli, A., Narang, G., Pietrini, R., Zazzarini, M., Fiorani, A., Tassetti, A. N. (2025). Multimodal AI-enhanced ship detection for mapping fishing vessels and informing on suspicious activities. Pattern Recognition Letters, 191, 15–22. https://doi.org/10.1016/j.patrec.2025.02.022
9. Salem, M. H., Li, Y., Liu, Z., AbdelTawab, A. M. (2023). A Transfer Learning and Optimized CNN Based Maritime Vessel Classification System. Applied Sciences, 13 (3), 1912. https://doi.org/10.3390/app13031912
10. Agorku, G., Hernandez, S., Falquez, M., Poddar, S., Amankwah-Nkyi, K. (2024). Real-Time Barge Detection Using Traffic Cameras and Deep Learning on Inland Waterways. Transportation Research Record: Journal of the Transportation Research Board, 2679 (2), 703–720. https://doi.org/10.1177/03611981241263574
11. Simonyan, K., Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. https://doi.org/10.48550/arXiv.1409.1556
12. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D. et al. (2015). Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1–9. https://doi.org/10.1109/cvpr.2015.7298594

13. He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 770–778. https://doi.org/10.1109/cvpr.2016.90
14. LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. Nature, 521 (7553), 436–444. https://doi.org/10.1038/nature14539
15. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86 (11), 2278–2324. https://doi.org/10.1109/5.726791
16. Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. Cambridge: MIT Press, 775.
17. Ioffe, S., Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. Proceedings of the 32nd International Conference on Machine Learning, 448–456.
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15, 1929–1958.
19. Kingma, D. P., Ba, J. (2014). Adam: a method for stochastic optimization. arXiv:1412.6980. https://doi.org/10.48550/arXiv.1412.6980
20. Hastie, T., Tibshirani, R., Friedman, J. (2009). The Elements of Statistical Learning. New York: Springer, 745.
21. Huang, I.-L., Lee, M.-C., Nieh, C.-Y., Huang, J.-C. (2023). Ship Classification Based on AIS Data and Machine Learning Methods. Electronics, 13 (1), 98. https://doi.org/10.3390/electronics13010098
22. Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28, 91–99.
23. Leonidas, L. A., Jie, Y. (2021). Ship Classification Based on Improved Convolutional Neural Network Architecture for Intelligent Transport Systems. Information, 12 (8), 302. https://doi.org/10.3390/info12080302
24. Gallego, A.-J., Pertusa, A., Gil, P. (2018). Automatic Ship Classification from Optical Aerial Images with Convolutional Neural Networks. Remote Sensing, 10 (4), 511. https://doi.org/10.3390/rs10040511
25. Zhao, T., Wang, Y., Li, Z., Gao, Y., Chen, C., Feng, H., Zhao, Z. (2024). Ship Detection with Deep Learning in Optical Remote-Sensing Images: A Survey of Challenges and Advances. Remote Sensing, 16 (7), 1145. https://doi.org/10.3390/rs16071145
26. Redmon, J., Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, 6517–6525. https://doi.org/10.1109/cvpr.2017.690

27. Girshick, R. (2015). Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV). IEEE, 1440–1448. https://doi.org/10.1109/iccv.2015.169
28. Niranjan, A., Patial, S., Aryan, A., Mittal, A., Choudhury, T., Rabiei-Dastjerdi, H., Kumar, P. (2024). A Deep Learning Approach for Ship Detection Using Satellite Imagery. EAI Endorsed Transactions on Internet of Things, 10. https://doi.org/10.4108/eetiot.5435
29. Cherniavska, T., Nadtochii, V., Nadtochyi, A., Lomonosov, D., Cieślak, R., Cherniavskyi, B. et al.; Cherniavska, T. (Ed.) (2025). Organizational and structural modeling of the integration of marine robotics into multilevel environmental and ecological monitoring systems. Ecological systems modeling. Tallinn: Scientific Route OÜ, 169–195. https://doi.org/10.21303/978-9908-9706-6-0.ch8