Electronics and Information Technologies, 2026, 34, 113–132
http://publications.lnu.edu.ua/collections/index.php/electronics/index

UDC 004.8; 004.932.4; 623.4

DEEP LEARNING ARCHITECTURES FOR VISUAL RECOGNITION OF SELECTED TYPES OF EXPLOSIVE ORDNANCE: COMPARATIVE ANALYSIS AND SYSTEM IMPLEMENTATION

Volodymyr Hrabovskyi* & ***Oleksii Hryhorashyk***
Department of Optoelectronics and Information Technologies
Ivan Franko National University of Lviv
107 Gen. Tarnavsky Str., Lviv, 79017, Ukraine

Hrabovskyi, V. A., & Hryhorashyk, O. D. (2026). Deep Learning Architectures for Visual Recognition of Selected Types of Explosive Ordnance: Comparative Analysis and System Implementations. *Electronics and Information Technologies*, 34, 113–132. https://doi.org/10.30970/eli.34.7

ABSTRACT

Background. In the context of significant contamination of Ukraine's territory with explosive ordnance (EO) resulting from the Russian Federation's aggression and over 11 years of ongoing hostilities, coupled with limited resources for detection and demining, the implementation of innovative technologies for identifying and recognizing such objects is critical. Therefore, the application of artificial intelligence-based methods for the detection and recognition of explosive devices is both relevant and essential.

Materials and Methods. This article explores the potential of Deep Neural Networks (DNNs) and computer vision for automating EO detection. The study reviews the theoretical foundations and practical applications of Convolutional Neural Networks (CNNs) in object recognition tasks. The primary focus is on utilizing modern deep learning methods to identify explosive devices, specifically technologies employing single-stage (SSD – Single Shot MultiBox Detector, and YOLO – You Only Look Once) and two-stage (R-CNN, Regions-based Convolutional Neural Networks) approaches for object detection and identification. Key aspects of using single- and two-stage DNN architectures for visual object recognition are investigated, comparing the strengths and weaknesses of both approaches and examining ways to improve them to enhance detection efficiency.

Results and Discussion. Based on the conducted analysis, models were developed using single-stage (SSD and YOLOv8) and two-stage (Faster R-CNN) object identification technologies. These models were trained on an original dataset created from internet-sourced images, which included eight common types of EO found in combat zones and de-occupied territories. Testing results of the trained models demonstrated the superiority of the YOLOv8 architecture for explosive device detection and recognition, which was subsequently used to develop a web server for explosive ordnance recognition.

Conclusions. A project was developed to train EO recognition model architectures based on two-stage and single-stage approaches. Using these models, a web server with a REST API (Representational State Transfer Application Programming Interface) was created for EO recognition in both static images and real-time video streams. The proposed system can facilitate demining operations and reduce risks to civilian populations.

Keywords: deep learning, visual recognition, explosive ordnance, SSD (Single Shot MultiBox Detector), YOLOv8, Faster R-CNN

INTRODUCTION

In the context of modern military conflicts, the problem of detecting explosive devices (EDs) has become particularly urgent. For Ukraine, which finds itself at the epicenter of the largest armed confrontation in Europe in recent decades, the timely detection of such objects is critical for both military units and the civilian population. Traditional search methods, which rely on metal detectors and manual operator work, often fail to provide the necessary speed, accuracy, and adaptability required on the modern battlefield.

According to the State Emergency Service of Ukraine, as of late 2024, up to 25% of the country's territory is contaminated with explosive hazards [1]. The frontline and de-occupied areas remain the most dangerous. In recent years alone, Ukrainian sappers have detected and neutralized about 780,000 explosive items [2], yet approximately 144,000 km^2 of territory remains potentially hazardous. Preliminary estimates show that complete clearance will require around 37 billion USD and the involvement of over 10,000 specialists, whereas only about 3,000 sappers currently operate in Ukraine [3]. The scale of the problem is so vast that relying solely on traditional methods could mean that complete clearance takes hundreds of years [4].

One of the most promising avenues for accelerating ED detection is the implementation of computer vision technologies [5], where the vital task is the automatic detection, classification, and localization of objects within images. The primary performance metrics of such systems are recognition accuracy and data processing throughput. According to research [6], the evolution of object detection methods can be broadly divided into two periods: the traditional methods era (before 2014) and the deep learning-based systems era (after 2014) (**Fig. 1**). The rapid advancement of deep learning methods [7] has driven a qualitative breakthrough in the field, transitioning object detection technologies from a primarily research-oriented domain into widespread practical application [8].

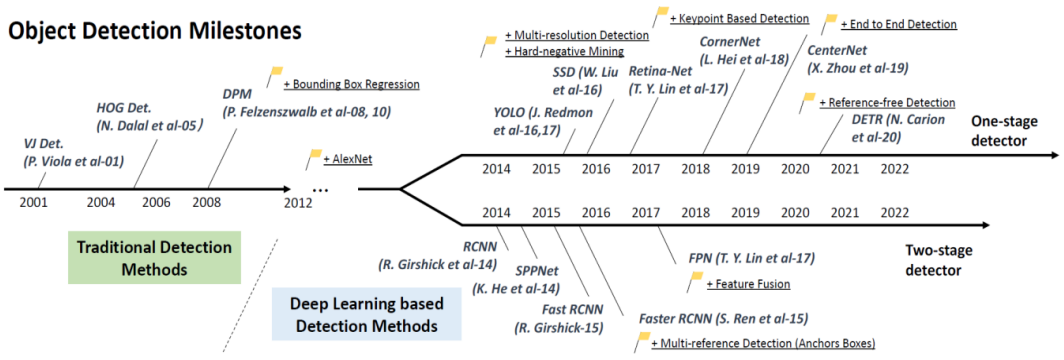

Fig. 1. Object detection "roadmap" [6]

Unlike classical computer vision approaches that require manual feature engineering, deep neural networks can automatically extract informative features directly from raw images. Convolutional neural networks (CNNs) [9, 10] play a pivotal role in this process, demonstrating high efficiency in visual recognition tasks. Their capacity to construct multi-level data representations and identify complex patterns makes these models a promising foundation for automated explosive device detection systems.

The integration of deep learning technologies has led to the formation of two main approaches to object detection: two-stage and one-stage frameworks. Two-stage methods implement a "coarse-to-fine" principle, first generating region proposals and subsequently refining the classification and localization results. One-stage methods execute these operations within a single unified process, offering significantly higher processing speeds and unlocking the potential for real-time applications.

MATERIALS AND METHODS

Two-Stage CNN-Based Recognition Systems

The emergence of two-stage object detection systems was a direct response to the limitations of traditional computer vision methods and early convolutional neural network architectures. Throughout the 2000s and 2010s, CNNs were primarily used for image classification–determining whether an object of a certain class was present without establishing its precise location or quantity. In contrast, the task of object detection requires the simultaneous execution of two interconnected functions: object localization using bounding boxes and its subsequent classification.

A significant challenge was also the handling of objects varying in scale and shape. Traditional approaches often relied on multi-scale analysis or image pyramids, which drastically increased computational costs. The sliding window method [11, 12], where a classifier was repeatedly applied to a vast number of image patches, was particularly resource-intensive. Consequently, these systems could not provide sufficient processing speed for practical use, especially in real-time environments. Additional drawbacks included a heavy reliance on manual feature engineering, insufficient robustness to changing observation conditions, and low efficiency in detecting small or partially occluded objects.

To overcome these constraints, a two-stage approach was proposed, dividing the detection process into two sequential phases. In the first stage, region proposals are generated–potential areas of the image that are likely to contain objects. Fast algorithms, such as Selective Search [13], are utilized here to significantly reduce the number of candidate regions requiring further analysis.

In the second stage, the selected regions are processed by a convolutional neural network, which performs object classification and refines the coordinates of their bounding boxes. By decoupling the localization and recognition tasks, two-stage architectures delivered a substantial leap in detection accuracy and established a foundation for the further evolution of modern computer vision systems.

R-CNN

The development of two-stage object detectors based on convolutional neural networks began with the introduction of the R-CNN (Regions with Convolutional Neural Networks) architecture, proposed by R. Girshick et al. in 2013 [14]. Prior to this, most object detection systems relied on manual feature engineering and the sliding-window method, which required significant computational resources and provided limited accuracy.

The key idea behind R-CNN was to combine region proposal algorithms with the automated feature extraction capabilities of CNNs. The model's workflow consisted of three main stages. First, using the Selective Search algorithm [13], approximately 2,000 candidate regions that could potentially contain objects were generated. Next, each region was scaled to a fixed size, which was required for input into the pre-trained network. In the final stage, the CNN extracted a feature vector, which was then used to determine the presence of an object, classify it, and refine its bounding box coordinates using a regression model.

The introduction of R-CNN delivered a substantial increase in detection accuracy compared to previous approaches and marked a major step forward in the evolution of modern computer vision. At the same time, the architecture suffered from several drawbacks. The most significant of these was the necessity to process each region’s proposal individually. Since a large portion of the candidate regions overlapped, identical features were repeatedly recomputed, which drastically reduced the system's processing speed. Furthermore, the use of fully connected layers required a fixed input size, complicating the handling of regions with different scales and aspect ratios.

To overcome these limitations, the SPPNet (Spatial Pyramid Pooling Network) architecture was proposed [15]. Its primary advantage was that feature maps were computed for the entire image only once and were subsequently shared across all candidate regions. The key element of this model was the Spatial Pyramid Pooling (SPP) layer, which ensured the generation of fixed-size features regardless of the input region's dimensions. This eliminated the need to scale each proposal and significantly reduced processing time compared to R-CNN.

Fast R-CNN and Faster R-CNN

The evolution of two-stage detectors advanced significantly with the introduction of Fast R-CNN [16] (2015) and Faster R-CNN [17] (2015), which brought substantial improvements in both speed and accuracy.

- Fast R-CNN unified the training of all components (feature extraction, classification, and bounding box regression) into a single network. It utilized RoI Pooling (Region of Interest Pooling), a simplified version of SPP, to extract features from object proposals on a feature map generated from the entire image and represent them as a single fixed-length vector.

According to [16], the RoI Pooling mechanism operates by first locating the region on the feature map generated by the backbone CNN that corresponds to each region proposal (RoI). This region, regardless of its arbitrary size, is then divided into a fixed grid (e.g., 7×7 cells). Next, a pooling operation (typically max pooling) is performed on each cell to select the maximum feature value within it. The result of this process is a fixed-size feature vector for each RoI, which can then be fed into fully connected layers for subsequent classification and regression. This approach significantly accelerated both training and inference compared to R-CNN.

- Faster R-CNN marked the next breakthrough by integrating object proposal generation directly into the network using the Region Proposal Network (RPN). The RPN is a fully convolutional network that operates on top of the feature map generated by a backbone convolutional network (such as VGG or ResNet).

One of the key advantages of the RPN is its feature-sharing mechanism, where the features generated by the convolutional layers are simultaneously used by both the RPN and the detection network (Fast R-CNN). This eliminates the need for redundant computations, drastically boosting computational efficiency.

Another critical component is the anchor mechanism, which involves placing a set of predefined bounding boxes (anchor boxes) of various scales and aspect ratios at each spatial position of the feature map. For each anchor, the network performs two tasks: it estimates an objectness score (the probability that an object is present) and performs bounding box regression offsets to refine the position of the bounding box.

Due to its fully convolutional architecture, the RPN can efficiently process images of arbitrary sizes, generating high-quality region proposals based on local features. Furthermore, the capacity for joint training with the Fast R-CNN detector within an end-to-end optimization framework allows for a coordinated improvement in the performance of both system components.

Combined, these characteristics ensure high throughput and efficiency for the RPN, substantially cutting down the time required to generate region proposals and enabling the real-time application of two-stage object detectors. The integration of the RPN into Faster R-CNN resolved the primary bottleneck that hindered the speed and efficiency of previous methods–namely, computationally expensive region proposal generation–and transformed Faster R-CNN into a completely end-to-end system, resulting in unprecedented acceleration and accuracy enhancements.

Feature Pyramid Networks

A significant improvement for two-stage detectors, especially in detecting objects of varying scales, was the introduction of Feature Pyramid Networks (FPN, 2017) [18].

Traditional CNNs extract features from the upper layers, which possess rich semantics but low spatial resolution, making the detection of small objects difficult. FPN addresses this by constructing a multi-level feature pyramid where each layer combines semantically strong features from the upper layers with spatially precise features from the lower layers. This is achieved through:

- Bottom-up pathway: A standard CNN generates a hierarchy of feature maps where, with each subsequent layer, spatial resolution decreases while semantic information becomes more abstract.
- Top-down pathway: Using upsampling operations, features from higher levels are enlarged and merged with the corresponding feature maps from lower levels.
- Lateral connections: These connections transfer information between layers of the same scale, enhancing the overall informativeness of the features.

Thanks to FPN, detectors like Faster R-CNN can efficiently identify objects of any size. An extension of this architecture for instance segmentation tasks is Mask R-CNN [19]. By adding a parallel branch for predicting an object mask alongside the standard bounding box branch, it simultaneously localizes the object and generates its high-quality pixel-level mask.

Single-Stage CNN-Based Recognition Systems

Single-stage detectors based on convolutional neural networks have become a powerful alternative to two-stage systems and are currently one of the key directions in computer vision. This approach to visual object detection is characterized by the ability to perform object detection in a single pass of the network, unlike two-stage architectures that first generate region proposals and then classify them. This approach provides a significant increase in recognition speed, which is critical for real-time applications.

The idea of single-stage detection arose from the need to overcome the computational overhead of two-stage systems. The first breakthrough in this direction was the YOLO (You Only Look Once) model, introduced by Joseph Redmon and colleagues in 2016 [20]. YOLO redefined the object detection task as a regression problem, where a single CNN directly predicts bounding boxes and corresponding class probabilities for all objects in an image by dividing the input image into a grid. This allowed for unprecedented inference speeds, albeit with some loss of accuracy compared to the two-stage models of that time, particularly for small and closely packed objects.

YOLO: Architectures and Applications

YOLO (You Only Look Once) is an object detection approach that fundamentally differs from two-stage methods by formulating detection as a single regression problem [20]. This allows the entire detection pipeline to be performed in a single forward pass of a neural network, ensuring high image processing speed.

The main idea of YOLO is to divide the input image into an S×S grid (e.g., 7×7), where each grid cell is responsible for detecting objects whose centers fall within it. For each cell, the model predicts B bounding boxes (typically 2), each described by coordinates (x, y), dimensions (w, h), and a confidence score indicating both the probability of an object being present and the accuracy of localization. In addition, each cell estimates conditional class probabilities for C classes. As a result, the output tensor has a size of S×S×(B×5+C). For example, with S=7, B=2, and C=20, the output becomes 7×7×30.

The YOLOv1 architecture consists of 24 convolutional layers followed by 2 fully connected layers, enabling effective feature extraction from images. Despite its high speed, early versions had certain limitations: each grid cell could predict only one object, which made it difficult to detect multiple small objects within the same region. Moreover, the use of a single-scale representation and the absence of anchor boxes reduced accuracy for small objects and objects with unusual aspect ratios.

Further development of YOLO significantly improved its performance. YOLOv2 (YOLO9000) [21] introduced anchor boxes, K-means clustering for their selection, batch normalization, and a stronger backbone network, Darknet-19. YOLOv3 [22] added multi-scale predictions and the Darknet-53 architecture with residual connections, improving detection accuracy, especially for small objects.

Subsequent versions focused on balancing speed and accuracy: YOLOv4 [23] integrated modern optimization techniques (CSPDarknet53, PANet, SPP, Mosaic augmentation, CIoU loss); YOLOv5 [24] gained popularity due to its PyTorch implementation and ease of deployment; YOLOv7 [25] introduced E-ELAN and improved scaling strategies; YOLOv8 [26] shifted to an anchor-free approach and supports multi-task learning (detection, segmentation, classification); and YOLOv9 [27] focuses on efficiency improvements through Programmable Gradient Information (PGI) and the GELAN architecture, achieving higher accuracy with reduced computational cost.

SSD and Other Single-Stage Recognition Architectures

Almost simultaneously with YOLO, the SSD (Single Shot MultiBox Detector) architecture was proposed in 2016 [28], further advancing the one-stage detection approach. SSD utilizes multi-scale predictions by adding auxiliary convolutional layers to the backbone network (e.g., VGG-16 or ResNet). This enables object detection at various scales by leveraging feature maps from different levels of the network. The application of default (anchor) boxes with diverse aspect ratios at each feature level substantially enhanced accuracy, particularly for small objects, while maintaining high processing speeds.

Subsequent development of one-stage detectors focused on improving accuracy and training stability without causing a significant drop in performance. RetinaNet [29] (2017) addressed the extreme imbalance between background and foreground (object) examples by introducing Focal Loss, which down-weights the loss assigned to well-classified examples. This allowed the model to achieve accuracy comparable to two-stage systems while maintaining high inference speeds.

EfficientDet [30] (2019) focused on efficiency and scalability, introducing the BiFPN (Bi-directional Feature Pyramid Network) for enhanced multi-scale feature fusion, alongside a compound scaling method to uniformly scale the network's depth, width, and input resolution.

FCOS (Fully Convolutional One-Stage Object Detector) [31], also introduced in 2019, moved away from anchor boxes entirely. Instead, it directly predicts the distance from a given pixel to the boundaries of an object, which simplifies the architecture and increases model flexibility.

In conclusion, despite certain limitations in specific scenarios, one-stage detectors have achieved high accuracy and have become a versatile tool for fast and efficient object detection.

Comparison of Key Characteristics of Two-Stage and Single-Stage Recognition Systems

Modern object recognition systems based on Convolutional Neural Networks (CNNs) have achieved significant progress in solving computer vision tasks. They are generally divided into two main categories: two-stage and one-stage detectors, which differ in architecture, speed, and accuracy.

Two-stage models, such as R-CNN, Fast R-CNN, and Faster R-CNN, first generate region proposals where objects are likely to be located and then perform classification and bounding box refinement. This approach provides high localization accuracy but is computationally more complex and slower.

One-stage detectors, such as YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), and RetinaNet, perform detection in a single forward pass without a

separate proposal generation stage. They directly predict object coordinates and class labels, enabling significantly higher speed and making them suitable for real-time applications, although sometimes at a slight cost in accuracy compared to the best two-stage models.

At the same time, a convergence of the two paradigms can be observed. Modern one-stage models (YOLOv7, YOLOv8, RetinaNet with Focal Loss) have significantly improved accuracy, while two-stage systems continue to be optimized for speed. Techniques such as Feature Pyramid Networks (FPN), anchor boxes, improved loss functions, and advanced backbone networks are widely used in both approaches, allowing models to be adapted to specific task requirements.

Based on the conducted analysis, YOLOv8, SSD, and Faster R-CNN were selected for studying the recognition of explosive devices from images and video streams. These models were fine-tuned using a specially prepared dataset. The most effective system will be used for further evaluation of automated explosive device detection capabilities.

IMPLEMENTATION

The practical component of the project involves three sequential stages aimed at developing an explosive device recognition system.

- Stage I: Curation of the Labeled Image Dataset. This stage encompasses gathering images of the eight primary types of explosive devices, annotating them to generate baseline ground-truth data, and subsequently exporting and structuring the dataset into formats compatible with the selected model architectures.
- Stage II: Training of the Detection Models. This phase focuses on training the YOLOv8, Faster R-CNN, and SSD models. It involves environment configuration, loading pre-trained weights, and fine-tuning them for the specific task of explosive device detection. The models are then trained on the compiled dataset to optimize detection accuracy, followed by validation and testing on independent data to evaluate precision and inference speed.
- Stage III: Comparative Performance Analysis. The final stage entails a comprehensive evaluation of the model benchmarks to select the most effective architecture for the deployment of the recognition system.

Dataset Development for Training Explosive Object Detection Models

For the effective development of visual object detection systems using deep neural networks (DNNs), the availability of a representative and high-quality annotated training dataset is paramount [32, 33, 34]. Data quality has a critical impact on deep learning outcomes; low-quality data leads to degraded model performance even when using optimal network architectures and advanced training algorithms.

A high-quality dataset provides several vital advantages:

- Accuracy and Strong Generalization Capability: It allows the network to learn true underlying patterns in the data, ensuring high accuracy on new, unseen samples. Conversely, noisy or incorrectly annotated data forces the network to learn false correlations or random noise, leading to overfitting.
- Robustness to Noise and Anomalies: Incorporating diverse conditions (varying lighting, aspect angles, and partial occlusions) makes the model robust against real-world environmental interference.
- Speed and Stability of Convergence: It minimizes the risk of oscillations or a complete failure to converge during training.
- Mitigation of Bias: It prevents systematic errors from reflecting in predictions, which is critical for safety-critical and security applications.
- Optimization of Computational Costs: It reduces the training time required to reach target performance metrics.

However, standardized, publicly accessible image datasets of explosive devices (EDs) are non-existent due to several factors:

- Information Sensitivity: Data regarding EDs is highly critical to national security, meaning most collected repositories remain strictly confidential.
- Complexity and Danger of Data Collection: Documenting EDs in field environments is hazardous, expensive, and requires specialized permits and explosive ordnance disposal (EOD) expertise.
- Object Diversity and Specificity: Significant variations in device type, physical condition, viewing angles, and background clutter demand vast amounts of data to achieve proper generalization.

This forces developers to adopt custom approaches: scraping open-source data (frequently limited by low quality), generating synthetic data via 3D modeling (resource-intensive), or utilizing proprietary corporate datasets. Consequently, custom curation and meticulous annotation of a training dataset is the only viable path to successfully engineering effective DNN-based ED detection systems. This ensures that the training data remains directly relevant to the specific characteristics of the target objects.

Since no ready-made datasets exist for detecting Soviet- and Russian-designed explosive devices, creating a proprietary dataset became necessary. Eight of the most prevalent ED types frequently encountered in combat zones and de-occupied territories of Ukraine were selected:

1. PFM-1 ("Lepestok" / "Petal") – an anti-personnel blast mine;
2. PMN-2 ("Black Widow") – an anti-personnel blast mine;
3. F-1 ("Limonka" / "Lemon") – a defensive hand fragmentation grenade;
4. RGD-5 – an offensive hand fragmentation grenade;
5. RKG-3 – a hand anti-tank grenade;
6. TM-62 – a heavy high-explosive anti-tank blast mine;
7. OZM-72 ("Vidhma" / "Witch" or "Bounding Mine") – a bounding fragmentation anti-personnel mine;
8. MON-50 – a directional fragmentation anti-personnel mine.

Images were collected from open sources using search engines. For each device type, 100 unique samples were downloaded. In total, 800 images were selected for primary training and 200 for hyperparameter experimentation. During collection, care was taken to approximate a normal distribution of image quality: roughly two-thirds consisted of clear photos of objects under various natural conditions to facilitate stable model convergence, while the remaining third comprised low-quality samples with unusual angles, partial occlusions, or sensor noise (an example for the PFM-1 is shown in **Fig. 2**). This distribution ensures the system's resilience to real-world visual interference. All gathered images were standardized to the JPEG format, and file naming conventions were restricted to alphanumeric characters, excluding Cyrillic letters and special symbols.

Annotation and data structuring were carried out using the online service CVAT.io (Computer Vision Annotation Tool) [35] – a widely recognized tool for image and video labeling (**Fig. 3**).

A total of 992 images were manually annotated in CVAT. The workflow included creating a project, defining 8 target class labels, uploading the imagery, tight labeling of each ED with bounding boxes while assigning its respective class, and exporting the annotated data in YOLO and COCO formats.

The data structures of these formats differ substantially in how they encode annotations and bind them to image files:

- COCO Format: Labels for all data splits (train, valid, test) are consolidated into a single *_annotations.coco.json file*. This file contains complete metadata regarding the classes; bounding box coordinates, and references to the corresponding image files.

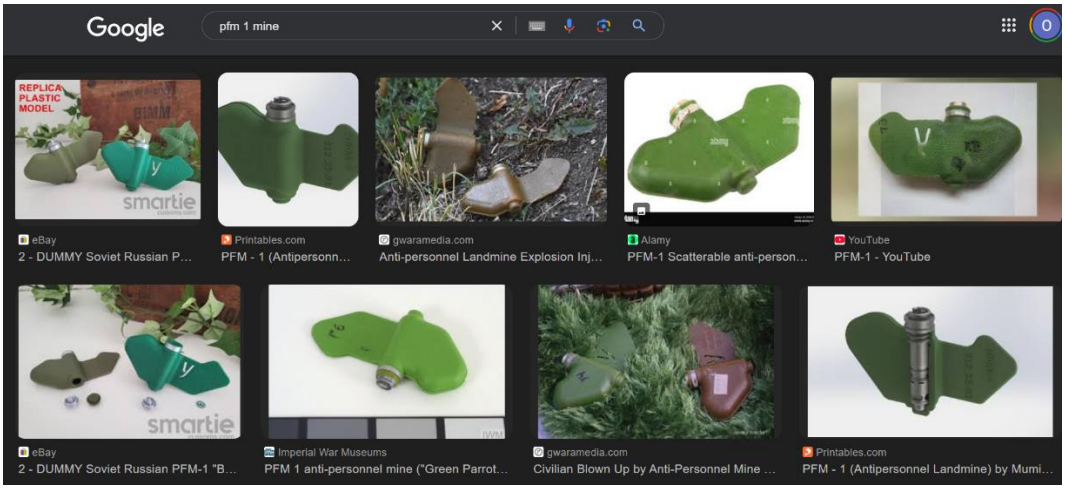

Fig. 2. Visual representation of PFM-1 "Petal" anti-personnel mine images used for dataset creation

Fig. 3. An example of the image annotation process in CVAT

- YOLO Format: Each directory split (train, valid, test) contains two mirrored subdirectories: *images* and *labels*. Each image file maps to a text file with an identical name containing the class indices and normalized bounding box coordinates bounded in the range [0, 1]. Overall metadata, including class names and relative directory paths, is defined in a separate *data.yml* configuration file.

The annotated images were split into two primary datasets:

- explosives_dataset (approximately 800 images) – used for baseline model training.
- explosives_small_dataset (approximately 200 images) – used for rapid hyperparameter optimization, as the smaller data volume allows for faster convergence and accelerated iterative testing.

Each dataset was partitioned into three subsets following an 80% / 10% / 10% ratio:

- Training (train) – 80%: Fed directly into the network to optimize and update the model weights.
- Validation (valid) – 10%: Utilized to monitor performance during training, calculate intermediate evaluation metrics, and guard against overfitting.
- Testing (test) – 10%: Maintained strictly for final benchmarking to evaluate the detector's true performance and generalization capabilities on completely unseen data.

Software Implementation of the Explosive Device Detection System

Structure of the Model Training Project

A software suite was developed for training and evaluating the performance of deep learning models – specifically YOLOv8, SSD+VGG, and Faster R-CNN–to solve the task of explosive device detection in images.

The project architecture includes the following directories:

- explosives_dataset – a directory containing the labeled set of images classified into eight types of explosive devices, divided into training, validation, and test subsets.
- explosives_small_dataset – a reduced version of the main dataset including 200 images; used to verify model convergence and investigate the impact of hyperparameters on the training process.
- trained_models – a directory containing pre-optimized models adapted for the explosive device recognition task, which includes:
  - *exp_yolo.pt* – an optimized model based on YOLOv8;
  - *exp_ssd.pth* – a configuration file for the optimized ssdlite320_mobilenet_v3_large model from the *torchvision* library;
  - *exp_faster_rcnn.pth* – a configuration file for the optimized *fasterrcnn_resnet50_fpn_v2* model from *torchvision*.
- YOLO – a module for training, validation, and practical application of the YOLOv8-based model:
  - yolo_model.ipynb – a Jupyter Notebook for model training and validation;
  - runs – a folder containing training and validation results;
  - *yolo_test.py* – a script for analyzing static images;
  - *webcam_detection.py* – a real-time module for processing video from a webcam.
- R-CNN and SSD – a set of components for training and testing Faster R-CNN and SSD models from the *torchvision* library:
  - RCNN_notebook and SSD_notebook – interactive Jupyter Notebooks for testing the respective models;
  - *pytorch_detection_tools.py* – methods for object detection using these models in PyTorch;
  - *training_tools.py* – tools for training, optimization, and saving models;
  - *faster_rcnn_train.py, ssd_train.py* – scripts for training models on a custom dataset.

Model Training Based on YOLOv8

The script for training and validating the model is contained within the *yolo_model.ipynb* file.

First, we create a model object pre-trained on the COCO dataset:

```
model = YOLO("yolov8n.pt").
```

Alternatively, we load the weights of a saved model that has already been trained on our custom dataset:

```
model_path = "../trained_models/exp_yolo.pt"
model = YOLO(model_path) # using pre-trained model
```

Next, we specify the path to the `data.yaml` configuration file from our dataset:

```
dataset_config = "../explosives_dataset/yolov8/data.yaml"
```

We conduct the model training for 10 epochs, specifying the requirement for input image augmentation:

```
results = model.train(data=dataset_config, epochs=10, augment=True)
```

Finally, we export the model:

```
model.export()
```

After loading the trained model and specifying the path to the validation dataset:

```
model_name = "exp_yolo"
model = YOLO(f"../trained_models/{model_name}.pt")
dataset_config = "../explosives_dataset/yolov8/data.yaml"
```

We perform validation, specifying the relevant metrics:

```
metrics = model.val(data=dataset_config)
metrics.box.map # mAP50-95
metrics.box.map50 # mAP50
metrics.box.map75 # mAP75
metrics.box.maps # a list containing mAP50-95 of each category
```

In addition to the aforementioned mAP50 and mAP50-95, the following metrics are used to evaluate model accuracy: precision (the ratio of correctly identified objects to all identified objects), recall (the ratio of correctly identified objects to all objects that should have been identified), and F1-score (reflecting the balance between precision and recall; used to find the optimal IoU (Intersection over Union) confidence threshold at which the minimum number of Type I and Type II errors is achieved) [37].

The recognition results for all eight types of explosive objects in the validation set are shown in **Table 1.**

Table 1. **Recognition results of the YOLOv8-based model on the validation set objects**

Class	Images	Instances	P	R	mAP50	mAP50-95
all	730	938	0.969	0.956	0.984	0.889
PFM 1	730	191	0.948	0.921	0.978	0.819
PMN 2	730	106	1	0.962	0.994	0.922
F1	730	127	0.956	0.976	0.990	0.867
RGD 5	730	115	0.949	0.974	0.992	0.930
RKG 3	730	44	0.995	0.886	0.947	0.835
TM 62	730	125	0.978	0.968	0.992	0.934
OZM 72	730	127	0.934	0.969	0.987	0.880
MON 50	730	103	0.993	0.990	0.995	0.927

The obtained results demonstrate that the trained YOLOv8 model achieves a mean average precision mAP50 of 0.984. However, the mAP50-95 metric appears more realistic, indicating that in our case, the detected object matches the identified class with an average accuracy of 89%.

A recall (R) value of 0.956 reflects that 95.6% of all objects in the validation dataset were correctly recognized, while a precision (P) of 0.969 indicates that nearly 97% of all recognized objects actually corresponded to the identified class.

Model Training Based on Faster R-CNN

To create the recognition model, the *fasterrcnn_resnet50_fpn_v2* architecture with pre-trained convolutional layer weights was utilized. The modification of the model

involved replacing the existing logistic regressor layers with new ones that provide a number of model outputs corresponding to the number of explosive device types + 1.

A training script, *fast_rcnn_train.py*, was developed, which handles loading the saved model from a configuration file along with optimizer parameters, training epochs, and current error rates, and executes the model training for a specified number of epochs.

To load our dataset, we use a DataLoader for the MS COCO format:

```
train_data_dir = "../explosives_dataset/coco/train"
my_dataset = get_custom_dataset(train_data_dir)
print('Number of samples: ', len(my_dataset))

# DataLoader instance
data_loader = DataLoader(my_dataset,
                         batch_size=2,
                         shuffle=True,
                         collate_fn=collate_fn)
```

We load the *fasterrcnn_resnet50_fpn_v2* model with pre-trained convolutional weights:

```
model = fasterrcnn_resnet50_fpn_v2(pretrained=True)
```

The model modification involved replacing the existing logistic regressor layers at its head with new ones that define the number of model outputs to match the number of detected explosive device types + 1:

```
num_classes = 9  # 8 types of explosive devices + 1 background class
# Update the classifier head
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = torchvision.models.detection.faster_rcnn.
FastRCNNPredictor(in_features, num_classes)
```

The training process can be executed either on a GPU using CUDA or on a CPU:

```
device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')
model.to(device)
```

SGD (Stochastic Gradient Descent) is used as the optimizer, and StepLR is employed as the learning rate scheduler:

```
params = [p for p in model.parameters() if p.requires_grad]
optimizer = optim.SGD(params, lr=0.001, momentum=0.7, weight_decay=0.004)
lr_scheduler = StepLR(optimizer, step_size=3, gamma=0.1)
```

The model was trained incrementally, with the learning rate decreasing every three scheduler steps:

```
for images, targets in data_loader:
    images = list(image.to(device) for image in images)
    targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

    optimizer.zero_grad()
    losses = model(images, targets)
    loss = sum(loss for loss in losses.values())

    loss.backward()
    optimizer.step()

    if not i % 2:
        print(f"Epoch: {epoch}, Iteration: {i}, Loss: {loss.item()}")
```

```
        i += 1
    lr_scheduler.step()
    epoch += 1
    save_model(model, model_state_path, optimizer, epoch, loss)
```

Additionally, the configuration allows for the use of the Adam optimizer and the ReduceLROnPlateau scheduler:

```
optimizer = optim.Adam(params, lr=0.001)
lr_scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=2,
factor=0.1)
```

SSD Model Training

The model training is implemented in the *ssd_train.py* script. The model is built upon the ssd300_vgg16 architecture, utilizing VGG16 as the backbone convolutional network:

```
def create_ssd_model(num_classes=8, size=640):
    # Load the Torchvision pretrained model.
    model = ssd.ssd300_vgg16()

    # Retrieve the list of input channels.
    in_channels = _utils.retrieve_out_channels(model.backbone, (size, size))

    # List containing number of anchors based on aspect ratios.
    num_anchors = model.anchor_generator.num_anchors_per_location()
```

The modification of this architecture involved replacing the final classification layers as well as the additional SSD convolutional layers directly connected to the logistic regressor:

```
    # The classification head.
    model.head.classification_head = ssd.SSDClassificationHead(
        in_channels=in_channels,
        num_anchors=num_anchors,
        num_classes=num_classes,
    )

    # Image size for transforms.
    model.transform.min_size = (size,)
    model.transform.max_size = size

    return model
```

Adam was used as the optimizer, and ReduceLROnPlateau was employed as the learning rate scheduler:

```
optimizer = optim.Adam(params, lr=0.001)
lr_scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=2,
factor=0.1, verbose=True)
```

Analysis of the Obtained Recognition Results on Validation and Test Sets

Three different types of object detection models were trained on the custom-made explosive device dataset. To evaluate the accuracy of the models, the following metrics were used: mAP50, mAP50-95, precision, and recall [36]. The absolute values of these metrics are a function of the IoU (Intersection over Union) confidence threshold. Therefore, it is necessary to find a threshold value that achieves a satisfactory balance between the precision and recall metrics.

In the case of the YOLOv8-based model, after only 10 training epochs, the values of the corresponding metrics were: precision = 0.950 and recall = 0.951. The F1-curve

demonstrates that the maximum F1-score is reached at a confidence threshold of 0.765 (**Fig. 4**). The obtained values for the accuracy metrics mAP50 and mAP50-95 for the validation and test sets of our dataset are as follows:

- Validation set: mAP50 – 0.984 and mAP50-95 – 0.889.
- Test set: mAP50 – 0.943 and mAP50-95 – 0.795.

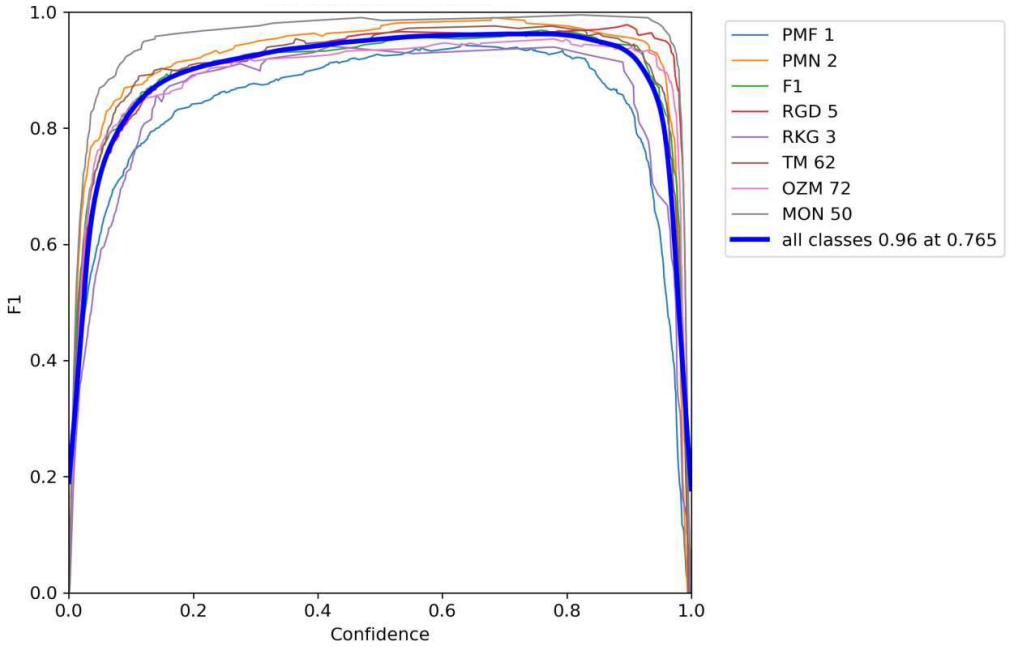

Fig. 4. F1-confidence curve for the YOLOv8 model.

The higher metric values for the validation set in the YOLOv8 model can be explained by the fact that the test set contained more images with interference (noise). Therefore, the test set results are clearly more representative for evaluating the real-world recognition performance of the specified explosive devices by this model.

At the same time, the SSD and Faster R-CNN models showed significantly worse convergence than the YOLOv8-type model. Adjusting the training hyperparameters (switching the optimizer from SGD to Adam and the scheduler from StepLR to ReduceLROnPlateau, which should have provided a more adaptive reduction in the learning rate) did not allow for accuracy comparable to the YOLO model. Specifically, the Faster R-CNN-based model, after a total of approximately 20 training epochs and various manipulations with training parameters and optimizers, was able to reach an mAP50 of 0.724 and an mAP50-95 of 0.421. The SSD-based model encountered similar issues as Faster R-CNN, achieving an mAP50 of 0.630 and an mAP50-95 of 0.261. It was established that under the given conditions, after the third epoch, the SSD and Faster R-CNN models became "stuck" in a local minimum plateau and, regardless of the optimizer parameters or even its type, failed to achieve sufficient accuracy.

Although all three models were trained on the same dataset, it appears that the error function landscape (loss landscape) for them turned out to be different. While the YOLO architecture managed to reach a satisfactory minimum error providing approximately 95% accuracy, the models of the other two architectures failed to achieve a satisfactory result.

These results were somewhat unexpected. According to the data presented in [37], these three models achieve nearly comparable accuracy levels on popular general-purpose datasets, such as COCO and Pascal VOC. However, in our case, when using a

custom-made dataset, the training results for the Faster R-CNN and SSD models failed to achieve satisfactory accuracy that would allow for their integration into an explosive device recognition server. Based on our findings and considering the conclusions of [37], it can be assumed that the training performance of the models depends not only on their architecture but is also significantly influenced by the structure, quality, and specific characteristics of the training dataset, which, in turn, are also determined by the application domain.

Web Server Development for Trained Models

The web server was developed using the Flask framework for Python, a powerful tool for building web applications and APIs. The server's endpoints are Image Detection (*/image-detection*) for recognition in static images and Video Feed (*/video-feed*) for real-time video stream recognition.

Based on the obtained results, the trained YOLOv8 model was selected for the explosive device recognition functionality within the web server. The use of this model for recognition, especially for real-time applications, is recommended by [37].

Experimental deployment of the server was performed on an Apple M4 Max (36 GB RAM) hardware platform with an integrated graphics system lacking CUDA architecture support. Under these conditions, the developed web server delivers an average processing speed of over 30 FPS. Utilizing hardware with CUDA technology support, however, will significantly optimize computational resources and enable system load scaling.

The project consists of:

- three saved model configuration files;
- *detection.py* – containing the detector descriptions and logic;
- *server.py* – defining the endpoints and rendering the user interface and API documentation;
- *swagger.py* – containing the documentation for the endpoints;
- *image.html* and *video.html* – defining the user interface pages, along with *style.css* for the stylesheet.

The "/image-detection" endpoint accepts an image, detects explosive devices within it, overlays bounding boxes and class names, and returns the processed image to the client (**Fig. 5**).

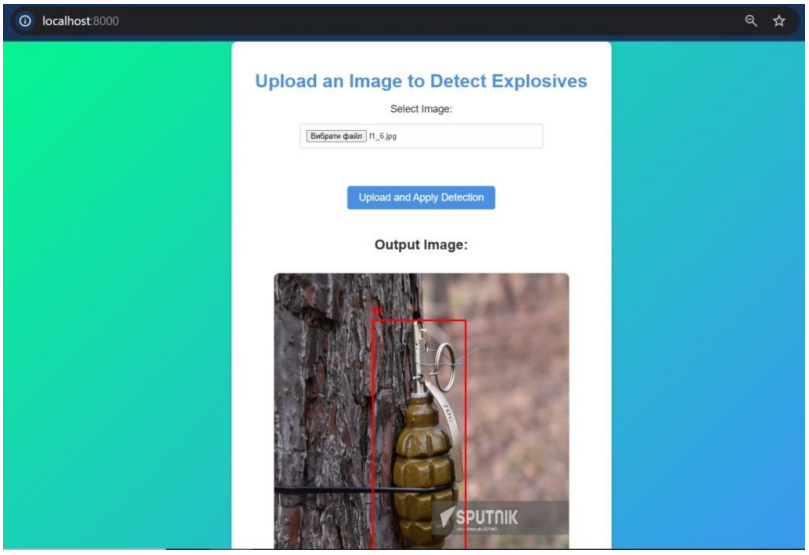

Fig. 5. View of the explosive device recognition window for images.

The "/video-feed" endpoint receives a video stream, detects explosive devices in its frames, overlays bounding boxes and class names, and returns the processed video frames to the client in real-time (**Fig. 6**).

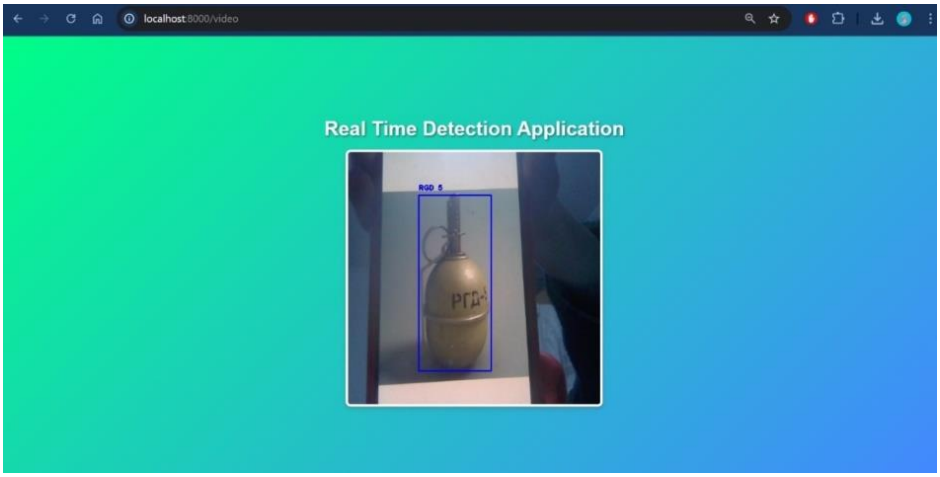

Fig. 6. Real-time recognition result from a video stream.

CONCLUSIONS

This study confirms the effectiveness of applying deep neural networks (DNNs) to the practical task of automated explosive device detection in both static images and video streams. The developed web server with a REST API ensures the integration of trained deep learning models into application systems, enabling the detection of hazardous objects in uploaded images as well as in real-time mode.

In the course of this work, a theoretical analysis of modern image processing methods and deep neural network architectures was conducted, specifically focusing on single-stage and two-stage approaches (YOLOv8, SSD, Faster R-CNN). To train the models, a proprietary dataset was formed containing images of the eight most common types of explosive devices found in Ukraine. The created dataset facilitated proper training and enabled a comparative analysis of the selected architectures.

Experimental results demonstrated that:

- The YOLOv8 model exhibited the highest performance metrics (mAP50 = 0.984 on validation data and 0.943 on test data; maximum F1-score reached at a threshold of 0.765), indicating its suitability for the stated task.
- The Faster R-CNN and SSD models proved to be less effective when trained on our custom dataset (mAP50-95 = 0.421 and 0.261, respectively). This could be attributed to architectural limitations, specific characteristics of the created dataset, and the insufficient robustness of these models to data variability.

Based on the most productive model under our conditions (YOLOv8), a software suite was implemented capable of providing efficient and rapid explosive device detection across various use cases. The interface, built in accordance with the OpenAPI standard, offers users flexible options for system integration and practical application.

The results obtained confirm the high potential of deep neural networks for enhancing security levels and demonstrate the feasibility of developing robust practical solutions based on modern deep learning methods. Future research should focus on expanding the dataset, optimizing models for operation on resource-constrained devices, and integrating the system with other security monitoring technologies.

ACKNOWLEDGMENTS AND FUNDING SOURCES

The authors received no financial support for the research, writing, and/or publication of this article.

COMPLIANCE WITH ETHICAL STANDARDS

The authors declare that the research was conducted in the absence of any conflict of interest.

AUTHOR CONTRIBUTIONS

Conceptualization, [V.H.]; methodology, [H.V, O.H.]; validation, [O.H.]; writing – original draft preparation, [V.H.]; writing – review and editing [H.V, O.H.]; supervision, [V.H.].

All authors have read and agreed to the published version of the manuscript.

REFERENCES

[1] Interactive map of areas potentially contaminated with explosives. (ukr) URL: https://mine.dsns.gov.ua/

[2] What percentage of Ukraine's territories are mined? (ukr). URL: https://tsn.ua/ukrayina/skilki-vidsotkiv-teritoriy-ukrayini-zaminovani-klimenko-prigolomshiv-cifroyu-2550241.html

[3] Over 144,000 sq km of Ukraine are considered potentially mined – new data from the Ministry of Internal Affairs. (ukr). URL: https://suspilne.media/781039-ponad-144-tisaci-kv-km-ukraini-vvazautsa-potencijno-zaminovanimi-novi-dani-mvs/

[4] Demining Ua – Rozminuvannya Ukraiyy (ukr). URL: https://demininguа.com/rozminuvannya-ukrayini

[5] Szeliski, R. (2022). *Computer vision: Algorithms and applications* (2nd ed.). Springer. https://doi.org/10.1007/978-3-030-34372-9

[6] Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023). Object detection in 20 years: A survey. *Proceedings of the IEEE,* 111(3), 257–276. https://doi.org/10.48550/arxiv.1905.05055

[7] Lamichhane, B. R., Srijuntongsiri, G., & Horanont, T. (2025). CNN based 2D object detection techniques: A review. *Frontiers in Computer Science*, 7, 1437664. https://doi.org/10.3389/fcomp.2025.1437664

[8] Edozie, E., et al. (2025). Comprehensive review of recent developments in visual object detection based on deep learning. *Artificial Intelligence Review*, 58, 277. https://doi.org/10.1007/s10462-025-11284-w

[9] Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018, Article 7068349. https://doi.org/10.1155/2018/7068349

[10] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM,* 60(6), 84–90. https://doi.org/10.1145/3065386

[11] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2014). OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *In International Conference on Learning Representations (ICLR 2014).* https://arxiv.org/abs/1312.6229

[12] Szegedy, C., Toshev, A., & Erhan, D. (2013). *Deep neural networks for object detection*. In C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 26). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/f7cade80b7cc92b991cf4d2806d6bd78-Paper.pdf.

[13] Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., & Smeulders, A. W. M. (2013). Selective search for object recognition. *International Journal of Computer Vision*, 104(2), 154–171. https://doi.org/10.1007/s11263-013-0620-5
[14] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587). https://doi.org/10.1109/CVPR.2014.81
[15] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, *37*(9), 1904-1916. https://doi.org/10.1109/TPAMI.2015.2389824
[16] Girshick, R. (2015). Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448). https://doi.org/10.1109/ICCV.2015.169
[17] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, *39*(6), 1137-1149. https://doi.org/10.1109/TPAMI.2016.2577031
[18] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2117–2125). https://doi.org/10.1109/CVPR.2017.106
[19] He, K.M., Gkioxari, G., Dollár, P. and Girshick, R. (2017) Mask R-CNN. *IEEE International Conference on Computer Vision (ICCV)*, Venice, 22-29 October 2017, 386-397. https://doi.org/10.1109/ICCV.2017.322
[20] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779–788). IEEE. https://doi.org/10.1109/CVPR.2016.91
[21] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7263–7271). IEEE. https://doi.org/10.1109/CVPR.2017.690
[22] Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv preprint* arXiv:1804.02767. https://doi.org/10.48550/arXiv.1804.02767
[23] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint* arXiv:2004.10934. https://arxiv.org/abs/2004.10934
[24] Jocher, G. (2020). YOLOv5 by Ultralytics *(Version 7.0*) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3908559
[25] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2023). YOLOv7: Trainable bag-of-freebies for real-time object detection. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7701–7711). https://doi.org/10.48550/arXiv.2207.02696
[26] Ultralytics. (2023). YOLOv8 Official Documentation. Retrieved from https://docs.ultralytics.com/yolov8/
[27] Wang, C. Y., Liao, H. Y. M., & Yeh, I. H. (2024). YOLOv9: Learning what you want to learn using programmable gradient information. *arXiv preprint* arXiv:2402.13616. https://arxiv.org/abs/2402.13616
[28] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – ECCV 2016*. Lecture Notes in Computer Science (Vol. 9905, pp. 21–37). Springer, Cham. https://doi.org/10.1007/978-3-319-46448-0_2

[29] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (pp. 2977–2985). IEEE. https://doi.org/10.48550/arXiv.1708.02002
[30] Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 10781–10790). IEEE. https://doi.org/10.48550/arXiv.1911.09070
[31] Tian, Z., Shen, C., Chen, H., & He, T. (2019). FCOS: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 9627–9636). IEEE. https://doi.org/10.48550/arXiv.1904.01355
[32] Domingos, P. (2015). *The master algorithm: How the quest for the ultimate learning machine will remake our world.* Basic Books.
[33] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
[34] Bishop, C. M. (2006). *Pattern recognition and machine learning.* Springer.
[35] CVAT.io – Open Data Annotation Platform. URL: https://www.cvat.ai/
[36] mAP (mean Average Precision) for Object Detection, URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173
[37] Shobaki, W. A., & Milanova, M. (2025). A comparative study of YOLO, SSD, Faster R-CNN, and more for optimized eye-gaze writing. *Sci*, 7(2), 47. https://doi.org/10.3390/sci7020047

АРХІТЕКТУРИ ГЛИБОКОГО НАВЧАННЯ ДЛЯ ВІЗУАЛЬНОГО РОЗПІЗНАВАННЯ ВИБРАНИХ ТИПІВ ВИБУХОНЕБЕЗПЕЧНИХ ПРЕДМЕТІВ: ПОРІВНЯЛЬНИЙ АНАЛІЗ ТА СИСТЕМНІ ВПРОВАДЖЕННЯ.

Володимир Грабовський*, Олексій Григорашик
Кафедра оптоелектроніки та інформаційних технологій,
Львівський національний університет імені Івана Франка,
вул. Ген. Тарнавського 107, Львів, 79017, Україна

АНОТАЦІЯ

Вступ. У контексті значного забруднення території України вибухонебезпечними пристроями (ВНП), що сталося внаслідок агресії Російської Федерації та понад 11 років бойових дій, а також обмежених ресурсів для виявлення та розмінування, впровадження інноваційних технологій для ідентифікації та розпізнавання таких об'єктів є критично важливим. Тому застосування технологій на основі штучного інтелекту для виявлення та розпізнавання вибухових пристроїв є актуальним та необхідним.

Матеріали та методи. У цій статті досліджується потенціал глибоких нейронних мереж (ГНМ) та комп'ютерного зору для автоматизації виявлення ВНП. У дослідженні розглядаються теоретичні основи та практичне застосування згорткових нейронних мереж (ЗНМ) у задачах розпізнавання об'єктів. Основна увага приділяється використанню сучасних методів глибокого навчання для ідентифікації вибухових пристроїв, зокрема технологій, що використовують одноступінчасті (однопрохідний детектор об'єктів із множинними обмежувальними рамками SSD – Single Shot MultiBox Detector, та однопрохідний метод виявлення об'єктів YOLO – You Only Look Once,) та двоступінчасті (регіонально-орієнтована згорткова нейронна мережа R-RCN) підходи для виявлення та ідентифікації об'єктів. Досліджено ключові аспекти використання одно- та двоступінчастих архітектур ГНМ для візуального

розпізнавання об'єктів, порівняння сильних та слабких сторін обох підходів, а також вивчення шляхів їх удосконалення для підвищення ефективності виявлення.

Результати. На основі проведеного аналізу було розроблено моделі з використанням одноступінчастої (однопрохідний детектор із множинними обмежувальними рамками SSD, та одноступінчастий детектор виявлення об’єктів, восьма версія архітектури You Only Look Once YOLOv8) та двоступінчастої (двоступінчастий регіональний детектор об’єктів на основі ЗНМ Faster R-CNN) технологій ідентифікації об'єктів. Ці моделі були навчені на оригінальному наборі даних, створеному з зображень з Інтернету, який включав вісім поширених типів ВНП, що зустрічаються в зонах бойових дій та на деокупованих територіях. Результати тестування навчених моделей продемонстрували перевагу архітектури YOLOv8 для виявлення та розпізнавання вибухових пристроїв, що було використано для розробки веб-сервера для розпізнавання вибухонебезпечних пристроїв.

Висновки. Розроблено проєкт для навчання моделей розпізнавання ВНП на основі двоступінчастого та одноступінчастого підходів. Використовуючи ці моделі, створено веб-сервер з REST API (інтерфейсу прикладного програмування на основі архітектури Representational State Transfer) для розпізнавання ВНП як на статичних зображеннях, так і у відеопотоках у реальному часі. Запропонована система може сприяти прискоренню процесів розмінування та зменшенню ризиків для цивільного населення.

Ключові слова: глибоке навчання, візуальне розпізнавання, вибухонебезпечні предмети, SSD (Single Shot MultiBox Detector), YOLOv8, Faster R-CNN

Received / Одержано	Revised / Доопрацьовано	Accepted / Прийнято	Published / Опубліковано
28 May, 2026	02 Juny, 2026	08 Juny, 2026	29 Juny, 2026