scientific reports

OPEN

Comparison classification algorithms and the YOLO method for video analysis and object detection

Martin Magdin[1,3]✉ & Zoltán Balogh[1,2]

This article focus on designing and programming an application for implementing the YOLO method v8 in the detection and subsequent classification of objects in video recordings. The application, developed in the Python programming language, allows the insertion of video recordings in mp4 format. It then divides the frames, enabling the use of the YOLO method for object detection, classification, and simultaneous determination of the time and frame at which the identified object is located. Compared to previous versions, YOLOv8 from 2023 allows the use of up to 53 convolutional layers. As part of the YOLO method implementation, therefore was devised a convolutional network with 5 layers and 5 object classes. Throughout the training process, was configured a total of 10 epochs, resulting in an accuracy of 94.79%. Notably, after the 7th epoch, the error rate exhibited a declining trend, reaching a value of 0.15. These values signify a sufficiently trained network without the need for further retraining.

Keywords Neural network, Object detection, Classification of objects, YOLO method, CNN

In contemporary industries, the need to scrutinize video records for object identification has become imperative. Given the prolonged duration of some videos and the inherent limitations of human attention, the likelihood of overlooking errors or missing crucial details is substantial. Videos pose a challenge for comprehensive manual review, prompting the adoption of machine learning-based detection methodologies. Many existing solutions are geared towards real-time applications, where human observers must meticulously scrutinize videos at a slow pace to discern the contained objects simultaneously.

Implementing algorithms for video analysis significantly amplifies computational complexity, as it concurrently addresses all three phases of the recognition process: object localization/detection within the image, extraction of regions of interest, and subsequent classification. The right object localization in video recordings is currently an important field within informatics, predominantly relying on deep learning methodologies as part of the broader machine learning paradigm. The object detection problem entails two fundamental steps:

- Determining where the object is located in the given image (object localization),
- Classification of the object after localization into a category (classification of the object).

It is most often used in applications for:

- object tracking,
- human detection,
- anomaly detection,
- counting people,
- in cars with autonomous driving,
- face recognition,
- video surveillance.

[1]Department of Informatics, Faculty of Natural Sciences and Informatics, Constantine the Philosopher University in Nitra, Nitra, Slovakia. [2]Obuda Univ, Kando Kalman Fac Elect Engn, Budapest, Hungary. [3]Faculty of Economics, Department of Applied Mathematics and Informatics, University of South Bohemia in České Budějovice, České Budějovice, Czechia. ✉email: mmagdin@ukf.sk

To alleviate undue strain on the processor and memory caused by computational complexity, our focus centered on designing a streamlined application that operates on the principle of frame-based video search. This application employs a neural network for object detection and subsequent classification. Upon localizing objects, users can designate the specific object of interest, and the application provides results, such as the timestamp and frame number where the identified object is situated. This approach serves to minimize the likelihood of errors attributable to human factors. The paper is structured into several interconnected sections. The research-related section is characterized by the research papers that have most significantly influenced the current state of the research problem. The methodology section describes the design and individual features of the application. The discussion section compares the results with those of other researchers.

Related work

One of the research methods for detecting an object in a video recording can be the application of the so-called histograms. Histograms began to be used much earlier than neural networks themselves. However, the histogram method of oriented graphs (HOG) is possible for object detection in videos. The method is based on the evaluation of appropriately normalized local histograms of image gradient orientation in a dense grid. The basic idea is that an object can be characterized relatively well by the distribution of local intensity gradients or edge directions, even without knowing the specific positions of the gradients or edges[1–3]. This method we cannot use for real-time image detection because it is too complex and it is not real to perform as many algorithm operations in as short a time as necessary. These problems were gradually eliminated by deep learning methods of convolutional neural networks (CNN). Deep learning models can also be referred to as neural networks with deep structures. The history of neural networks dates back to the 1940s. With the batch normalization method, the training of very deep neural networks became effective. At the same time, network structures based on convolutional neural networks (CNNs) such as AlexNet, Overfeat, VGG and ResNet have been extensively studied to improve their performance[4]. Convolutional neural networks are very similar to classical neural networks. The main difference is that a neuron in a hidden layer is directly connected to a subset of neurons in the previous layer[5–8]. As with any neural network, it is also necessary to train the convolutional network. It is one of the most important network activities. If we do not have a sufficiently well-trained neural network, then its results will not be sufficient, or they will be very bad. In order to have a successfully trained neural network, it is necessary to find a suitable dataset. Due to the fact that the training time is very long for more complex networks and high technical requirements are required for real-time applications, cloud platforms are often used to speed up the computing process[9,10]. Typical known problem of convolutional neural networks is the training domain, where a large number of parameters need to be tuned to make the network architecture suitable for use[11]. An example is the so-called conflicting image to avoid differential CNN classification when the network gives the wrong answer with high confidence[12]. In the application of convolutional networks, in terms of speed and success rate, the method Single Shot MultiBox Detector (SSD) is considered to be one of the most effective[13–15]. The SSD method is based on a forward convolutional network. It creates a collection of fixed-size frames on the objects and calculates the occurrence score of instances of the object class for each frame. It then evaluates the final detections in a maximum suppression step. The basic part of the network is based on the principle of VGG-16 which is a convolutional neural network, to which three additional features have been added (multidimensional maps, convolutional predictors and default fields with aspect ratios)[13].

A very similar method (but with a completely different principle) to SSD is the so-called YOLO (You Only Look Once) method currently in use. Existing various versions of the YOLO method. First method YOLO as single-stage object detector that uses a convolutional neural network (CNN) was developed by Joseph Redmon and Ali Farhadi in 2015. This method was used as predictor for the bounding boxes and class probabilities of objects in input images. In the following years, various improvements came gradually:

- YOLO V2: Anchor with K-means added, two-stage training, full convolutional network;
- YOLO V3: Multi-scale detection by using FPN;
- YOLO V4: SPP, MISH activation function, data enhancement Mosaic/Mixup, GIOU(Generalized Intersection over Union) loss function;
- YOLO V5: Flexible control of model size, application of Hardswish activation function, and data enhancement.

The latest versions, YOLOv9[16] and YOLOv10[17], were released in 2024 but have not been sufficiently researched or widely adopted by researchers and users[18]. Therefore, YOLOv8, released in 2023, remains the most commonly used version[19–23]. YOLOv8 excels in versatility and ease of use, making it a robust general-purpose model. YOLOv9 and Yolov10, with its innovative architecture, offers enhanced efficiency and accuracy, particularly beneficial for lightweight models and resource-limited environments. In generally, while YOLOv8 excels in correctly identifying objects with a higher true positive count, it also exhibits a higher false positive count, indicating a potential for over-detection. On the other hand, YOLOv9 and YOLOv10 demonstrates a lower false positive count but may miss some instances of objects due to its higher false negative count[24,25]. It was for the sake of versatility and simplicity that we decided to use the YOLOv8 version.

Using YOLOv8 is possible achieving 209 FPS on YOLOv8s (small version) and 525 FPS on YOLOv8n (nano version)—a 10x speedup over PyTorch and ONNX Runtime (cross-platform inference and training machine-learning accelerator)[26]. YOLO sees the whole picture during the training and testing process, so it implicitly encodes information about the classes and their appearance. The problem is that it lags slightly behind in detection accuracy, especially for small objects[27].

Currently, in addition to CNN, various other methods are used to detect objects in video, for example, the TensorFlow method[28,29]. The TensorFlow method is used in most cases as a real-time detection method.

It was created in 2015 by Google. The curiosity of this method is that it works with variable data graphs. If nodes represent mathematical operations, edges represent multidimensional data fields between them used for communication[30]. This method can be implemented in several ways: locally[28,31,32] in various devices[33,34]; distributed interface[32,35].

Object detection in computer vision is a difficult research topic. Currently, there are still no programs that can handle object detection 100%. In most cases, this is about 80–90% (depending on the dataset) of the success rate of such applications.

The aim of this paper is:

(1) Description of the methodology for designing and implementing an application for segmenting video frames for further analysis.
(2) Detection and recognition of objects in images-determining the frame on which the object is located.
(3) Summary comparison of classification algorithms and the YOLOv8 method in terms of accuracy and F1 score.
(4) Discussion of the current capabilities of YOLOv8 in relation to large learning models.

Methodology

Object detection and localization is one of the basic predispositions for object recognition which we can perform using various methods. All these methods belong to the machine learning or deep learning methods. In areas of computer vision, where it is necessary to avoid errors, local algorithms based on the recognition of object properties are used for feature extraction. These algorithms are more robust and resistant to changes such as lighting changes, scale changes, etc. Properties that such an algorithm discovers are called key points. Such key points can be, for example, patches, edges, corners, and others. The objective is to streamline object recognition by discarding irrelevant image information. Local or global descriptors are then calculated from the key points and compared to the similarity in the database. The degree of similarity between descriptors depends on the type of data. After assigning the descriptors representing the features of the object, the closest matches are determined using the method of nearest neighbours (FLANN). Another used comparison algorithm is the so-called Brute-Force. The latter finds the closest descriptors by performing all matching combinations between the database and the retrieved object descriptors. The whole process of object recognition can be seen in Fig. 1.

Local 3D descriptors include the following methods[37]:

- Signature of Histograms of OrienTations (SHOT).
- Point Feature Histogram (PFH).
- Point Feature Histogram RGB (PFH-RGB).
- Fast Point Feature Histogram (FPFH).

Among the global 3D descriptors are the following methods[37]:

- Ensemble of Shape Functions (ESF).
- Viewpoint Feature Histogram (VFH).
- Clustered Viewpoint Feature Histogram (CVFH).

In some cases, it is possible to use both variants, global and local methods. In this case, there are two methods used-stacking and classification hierarchy. Both of these methods significantly improve results compared to using only global or local features.

For our solution, we used the YOLO method because of its good speed to detection accuracy ratio which has already been subjected to several benchmark tests[4,26]. The YOLO method was designed for the first time in

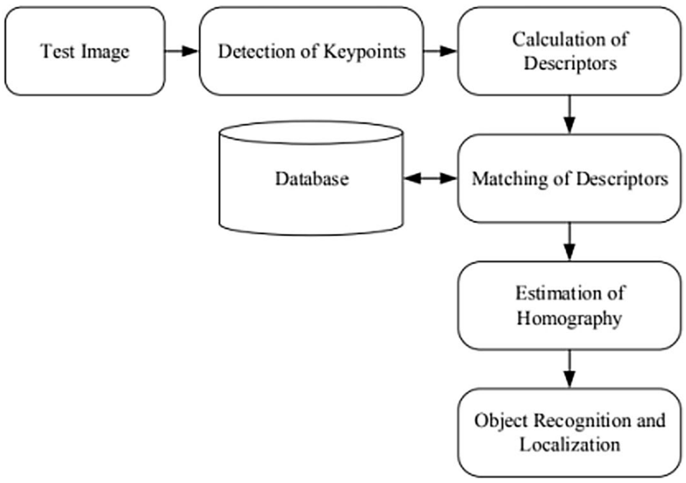


Fig. 1. Block diagram of the object recognition principle[36].

2015 and since then there have been several modifications of it. The latest version is YOLOv8. Our GUI (Fig. 2) was created in Python using the tkinter v8.5 library. The size of the application window is constant and has a dimension of 370 × 380 pixels. It contains 4 basic buttons-*Browse Video*, *Open*, *Quit*, *Show Info*.

Button Browse Video–serves to obtain the path to the video file. There are several other methods on it that we will describe in the next part of the work, namely frame_split and listbox filling.

Button Open–is used to open a specific frame from the video that we select in the given listbox menu.

Button Quit–is used to exit the entire application and clean up the directory data.

Button Show Info–shows information about the selected frame. First of all, it lists all the objects that are in the video and then it lists the time in the video where this particular frame can be found.

In standard, it is not possible to input a video into a neural network without watching its entire progress in real time. We solved this problem by gradually dividing the video into individual frames to make detection as simple and fast as possible.

The application works very simply. After clicking the *Browse Video* button, a file browser opens with a predefined path and displaying files in *.mp4* format, or all files. To prevent the program from crashing, a data directory is created using button *Browse Video* (in case it is missing in the program folder). If the program is not run for the first time, this directory already exists and will not be created again. We get the path to the file from the button and it is sent to the *Frame_split* class.

The *Frame_split* class includes the import of several Python libraries:

- cv2–OpenCV library used to work with video and image,
- os–library used for working with the system (creating, saving files, etc.),
- shutil–library used for working with the system, especially for deleting directories and their contents and similar actions,
- moviepy–library used for working with video. We used it because of the functions that the cv2 library does not contain,
- GUI–our custom created class that we imported for future use in the second method of the class.

Frame_split contains 2 different methods. First-*splitter*, it is used for splitting frames of video (frame by frame in format *.jpg* to the folder *data*). So that, for example, an 11-second video does not have more than 1000 frames (it would take up a lot of space and take a long time to take sampling), we only use every 200 milliseconds of video which is the ideal spacing so that the difference between the frames is not too big, but not too small either. In this way, approximately 125 frames will be created from a 20-second video.

This approach however, can evoke 2 potential problems:

(1) Case of uneven frame rate: selecting frames based on time intervals (e.g., one frame every 200 milliseconds) may cause some video frames to be lost because the frame rates of different videos may be different. For example, some videos may have a frame rate of 30 FPS while others may have a frame rate of 60 FPS, and extracting frames using a fixed time interval does not ensure that the distribution of frames is uniform. This can lead to loss of information, especially in scenes with fast-changing motion.

(2) Creating too many images: Extracting a frame every 200 milliseconds from a video may seem to feel to reduce the computational burden, but the reality is that this approach may still result in too many frames, especially in long videos. For long videos, extracting a frame every 200 milliseconds may generate a large

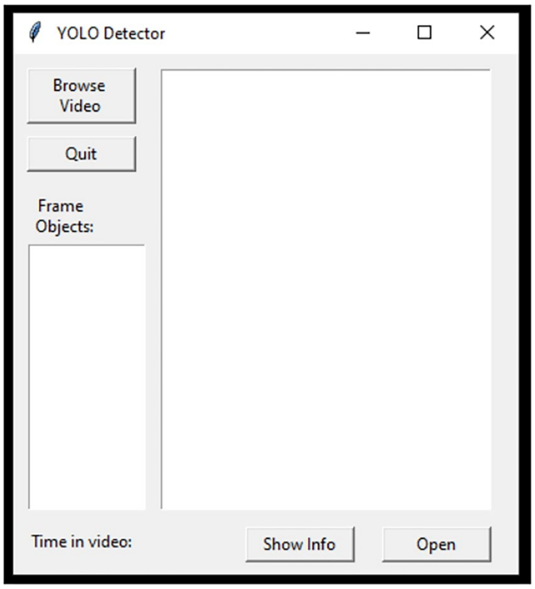


Fig. 2. GUI after starting application.

number of image files, which not only takes up storage space but may also cause performance bottlenecks when processing these frames.

Therefore, we modified the code by adding a function to remove redundant identical (duplicate) images. These frames occur in the video when there is no change in the storyline (still image-no change in motion or new object). Determining a change in the storyline (motion or appearance of a new object in the scene) consists of 4 basic steps by default:

(1) Computing the grayscale Frame Difference.
(2) Thresholding the Frame Difference to get the Motion Mask.
(3) Finding of Contours in the Motion Mask and get their Bounding Boxes.
(4) Performing Non-Maximal Suppression on the detected Bounding Boxes.

Is very important understand that they should be removed and then replaced by Interpolated frames between the previous frame and the next frame. This is correct way for avoid "speeding up the output video" and removing stutter in the video.

Each image is at the same time adjusted to a size of 416 × 416 pixels due to capacity and also due to YOLO detection. After saving the last frame, the video is released from the RAM which also ends the method. In button *Browse Video* exist part of code which will fill the ListBox with the names of these *.jpg* images (Fig. 3).

A slight drawback of *ListBox* in Python is the unarranged of names. There is no option in it to arrange the names so that they follow each other as needed, but it always takes them by numbers. Although it is possible to eliminate this shortcoming with other programming languages, such as Java or C#, they are not suitable for the use of neural networks. The second important method of the *Frame_split* class is *setVideoTime*, with which we get the total time of the video.

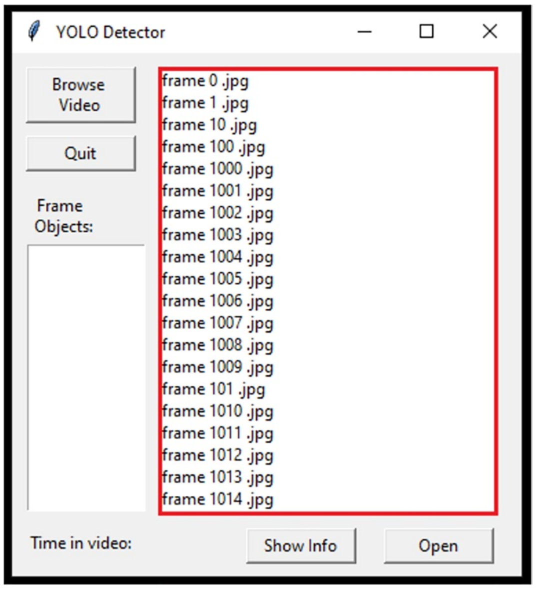
In order for us to be able to perform YOLOv8 detection, three files are required:

- Yolov8 configuration file (yolov8.cfg)-contains a defined 53 convolutional layers and another layers for tracing, desampling.
- class names contained in the COCO dataset (*coco.names*)-up to 80 classes that the detection can recognize and classify, such as cars, people, bicycles, trains, different types of animals, up to objects such as a toothbrush.
- file with the weights of the neural network itself (*yolov8.weights*). These files can be obtained from the official website of the YOLOv8 method. The file contains individual weights of the pre-trained network. It serves so that we do not have to train the entire model anew, but only to load the values into the network. If we wanted to create our own model, we would use that instead of this weight file.

The principle of the detection using the YOLO detector class

The *YOLO_Detector* class comprises two methods: *Detect* and *getInfo*. The *Detect* method is linked to the *Open* button to retrieve the name of the specific image selected in the *ListBox*. The file path is then generated from this name. Within the detect method, the network is loaded from the yolov8.weights and *yolov8.cfg* configuration files.

Values determining the object type are assigned to individual YOLO objects, and weak or inaccurate detections are filtered out. Names are assigned to these objects, and coordinates for the bounding boxes are



frame 0 .jpg
frame 1 .jpg
frame 10 .jpg
frame 100 .jpg
frame 1000 .jpg
frame 1001 .jpg
frame 1002 .jpg
frame 1003 .jpg
frame 1004 .jpg
frame 1005 .jpg
frame 1006 .jpg
frame 1007 .jpg
frame 1008 .jpg
frame 1009 .jpg
frame 101 .jpg
frame 1010 .jpg
frame 1011 .jpg
frame 1012 .jpg
frame 1013 .jpg
frame 1014 .jpg

Fig. 3. GUI after complete preparation of the video.

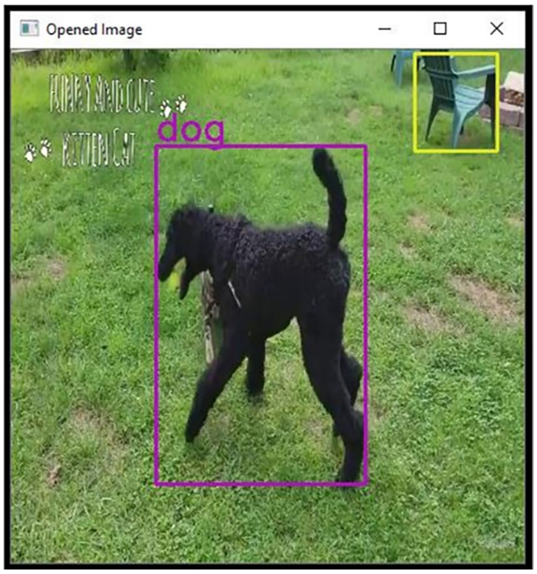
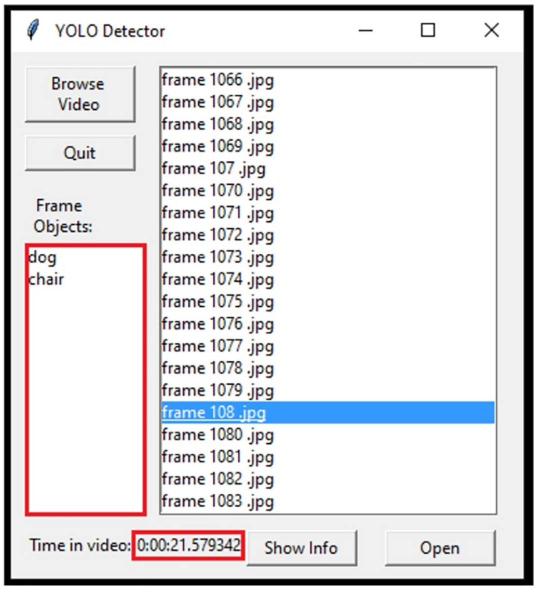


Fig. 4. Correct object detection.



Frame Objects:
dog
chair

frame 1066 .jpg
frame 1067 .jpg
frame 1068 .jpg
frame 1069 .jpg
frame 107 .jpg
frame 1070 .jpg
frame 1071 .jpg
frame 1072 .jpg
frame 1073 .jpg
frame 1074 .jpg
frame 1075 .jpg
frame 1076 .jpg
frame 1077 .jpg
frame 1078 .jpg
frame 1079 .jpg
frame 108 .jpg
frame 1080 .jpg
frame 1081 .jpg
frame 1082 .jpg
frame 1083 .jpg

Time in video: 0:00:21.579342

Fig. 5. GUI after clicking the Show Info button–listed objects and time of the frame.

calculated. In the final iteration, individual boxes and class names are drawn onto the image (Fig. 4). To enhance clarity, the *numpy* library is employed to display them in distinct colors.

The *getInfo* method is activated by the *Show Info* button. It operates similarly to the previous detect method, with the distinction that it does not render class names and bounding boxes, nor does it display the image. During the cycle, all object names are sequentially written from this variable to the *listbox*.

To enhance program completeness and user-friendliness, the *Show Info* button reveals the time at which the frame in the video is located. We determine the exact time value using *getTime* by dividing the total number of acquired frames by the elapsed time. This value is then converted into hours, minutes, and seconds using the *datetime* library (Fig. 5).

CNN detection-comparison of classification networks

To compare classification networks, we constructed our own convolutional neural network. This network operates on the same principle as the one in the YOLO detector. However, it is not as robust due to the challenges of creating a structure with 53 neural layers, requiring a powerful machine for training. Configuring such an extensive network for satisfactory detection and classification is also highly challenging. Consequently, we developed a more compact sequential convolutional neural network tailored for specific object detection. Training on the desired objects necessitates the use of an appropriate dataset.

Our designed convolutional neural network comprises five convolutional layers, each with specific parameters—some common and some different. Notably, the number of filters varies across layers: 32 for the first layer, 64 for the second and third layers, 128 for the fourth, and 256 for the fifth. The input shape features three channels and a size of 64×64 pixels.

Following the convolutional layers, we incorporated pooling layers, including three *MaxPooling* layers and one *AveragePooling* layer, to reduce spatial size in network representation. Batch normalization follows, enhancing network stability, performance, and speed. Additionally, we implemented dropout as a training tool.

For effective data representation, a flatten layer converts the data into a 1-dimensional matrix, forwarding it to the dense layer. The dense layer, with 256 hidden neurons, precedes the output dense layer. The output layer has five classes and utilizes a softmax activation function suitable for multiple classes. To compile the network model, we employed the *Adam optimizer* and categorical cross-entropy to express error. We used a standardised datasets; each dataset included varying conditions such as lighting variations, occlusions, or different object sizes. Our dataset comprises 20,000 images across five classes, with 4000 images allocated for training per class and 1,000 images each for validation and testing:

- Cats–contains 4000 pictures of cats for training and 1000 pictures for testing[38],
- Dogs–contains 4500 pictures of dogs for training and 1000 pictures for testing[38],
- Airplanes–contains 4000 pictures of airplanes for training and 1000 pictures for testing[39],
- Peoples–contains 2300 pictures of human for training and 1000 pictures for testing[40],
- Cars–contain 4000 pictures for training a 1000 pictures for testing[41].

To systematically fill the network during training, we employed a generator, ensuring that all images from the designated folder were input into the network. Similarly, for the validation part, a separate generator was created, utilizing images from the folder dedicated to validation. The data distribution was set with 80% for training and 20% for testing.

We trained a model on a dataset using various image recognition algorithms and then tested the individual image recognition algorithms on the second half of the dataset. During testing, we recorded the following metrics and characteristics:

- *Accuracy*-to measure ohow well a machine learning model or algorithm can correctly classify.
- *Precision*-the proportion of true positive predictions among all positive predictions made by the model. A high precision value indicates that the positive predictions of the model are mostly correct, while a low precision value indicates that many of the positive predictions are incorrect.
- *Recall*-a metric that measures the proportion of actual positive results correctly identified by the model.
- *F1 score*-a measure of predictive performance, calculated from the precision and recall of the test, where precision is the number of true positive results divided by the number of all samples predicted to be positive, including those that were not correctly identified.
- *Validation accuracy*-a metric used to interpret the results obtained when training the model on unclassified data. High validation accuracy can be a good sign that the model will perform well on new data. On the other hand, very high validation accuracy in the first epochs may indicate overfitting of the model.
- *Validation loss*-a metric used to evaluate the performance of a deep learning model on a dataset. Validation loss is calculated as the sum of errors for each example in the dataset.

To output the metrics of the classification algorithm, we imported the methods accuracy_score, precision_score, recall_score, and f1_score from the sklearn library. The model returned predicted probabilities for each class, and the variable y contained one-hot encoded values of the classes. Using the average recall helped us account for different class sizes in the dataset. Thanks to this data, we obtained a detailed evaluation of the effectiveness of our model in classifying data.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Class prediction
prediction = model.predict(final_properties)

# Convert prediction and real values to index class
prediction_labels = np.argmax(prediction, axis = 1)
real_labels = np.argmax(y, axis = 1) # y je one-hot encoded

# Metrics calculations
accuracy = accuracy_score(real_labels, prediction_labels)
precision = precision_score(real_labels, prediction_labels, average = 'weighted')
recall = recall_score(real_labels, prediction_labels, average = 'weighted')
f1 = f1_score(real_labels, prediction_labels, average = 'weighted')
print(f'Paccuracy: {accuracy}')
print(f'precision: {precision}')
print(f'recal: {recal}')
print(f'F1 Score: {f1}')
```

Considering that the YOLO method is primarily designed for real-time object classification, we were interested in evaluating the parameters of accuracy, precision, and error that can be achieved when classifying objects from video footage parsed into individual frames. For comparison with the YOLO method, we selected the NasNetLarge, InceptionResNetV2, Xception, and InceptionV3 algorithms. These algorithms are among the most commonly used in terms of recognition and classification.

Each image recognition algorithm used its own preprocessor for feature extraction, imported from the Keras library, which prepared the data for feature extraction. In Fig. 6, we can see an example of feature extraction for InceptionV3, Xception, NasNetLarge, and InceptionResNetV2, performed on the training set of the dogs and their breed dataset. The number of extracted features varies, except for the Xception and InceptionV3 algorithms, where the number of extracted features is the same. The most features (4075) were extracted using the YOLO method and NasNetLarge with 4032 features, whereas InceptionResNetV2 extracted the fewest features, with only 1536.

The YOLO method achieved the highest accuracy and F1 score at 94.79% (Fig. 7). Similar values were also achieved by the NasNetLarge image recognition algorithm, with an accuracy of 94.60% and an F1 score of

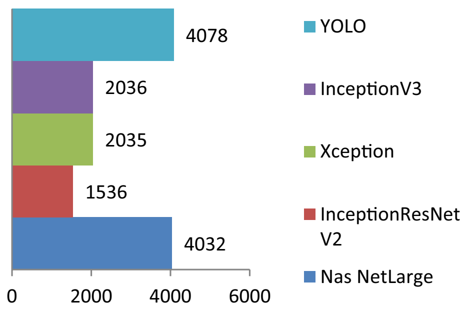


Fig. 6. Number of extracted features by image recognition algorithms.

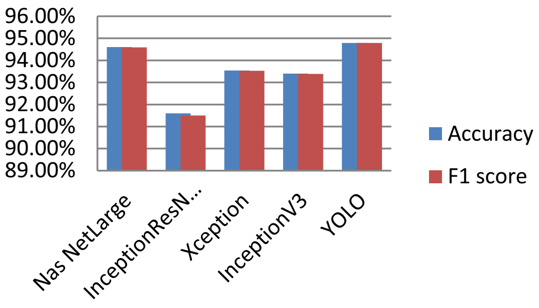


Fig. 7. Comparison of image recognition algorithms by accuracy and F1 score.

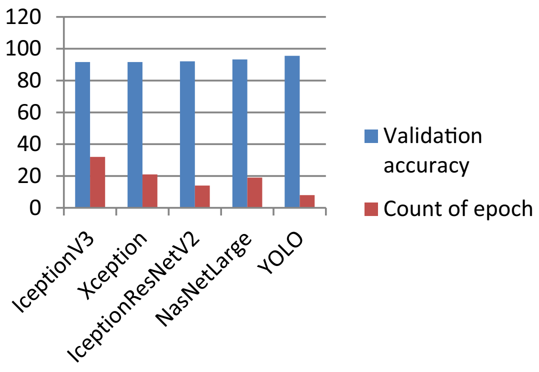


Fig. 8. Comparison of image recognition algorithms by epochs and validation accuracy.

94.59%. While other image recognition algorithms like InceptionV3 and Xception were close to the NasNetLarge algorithm, for the InceptionResNetV2 algorithm, we measured an accuracy of 91.6% and an F1 score of 91.5%. Although the use of a dropout layer with a value of 0.7, which randomly disables 70% of neurons, causes the values to vary in repeated tests, we did not observe significant changes in the ranking of the results of the individual image recognition algorithms during the tests.

Apart from accuracy, training time is also important. For the dog recognition model, we set the maximum number of epochs to 50, but this many were not needed because the early stopping function stopped the model training sooner. In Fig. 8, we can see that the number of epochs varied significantly among the different image recognition algorithms. While only 14 epochs were needed for the InceptionResNetV2 algorithm, InceptionV3 required 32 epochs. Nevertheless, the validation accuracy was similar across all algorithms.

Discussion

Optimal results were obtained with YOLO method by training the network with a batch size of 32 over 10 epochs. With this configuration and a substantial amount of training data, we achieved a success rate of 94.79%, indicative of a proficient neural network (Fig. 9). The trained model was saved as a .h5 file, encapsulating both the weights and the architecture of the neural network. This file can be utilized at any time for further neural network training.

Figure 10 illustrates the training progress of the network over the entire 10 epochs. Following the first epoch, the neural network achieved an 86% success rate, already considered successful. The network's success exhibited an upward trend throughout most of the training, culminating in the final success rate. Notably, the value of 94.79% was surpassed in one epoch, exceeding 95%. In Fig. 7, we observe the evolution of the network error during training. The error demonstrates a downward trend throughout the training process. During validation, the error evolution initially fluctuates, but after the 7th epoch, it also follows a consistent downward trajectory, reaching approximately 0.15, which is a very result.

The YOLOv8 detector is designed to recognize 1000 pre-defined classes and comprises 53 convolutional layers. Configuring a network of this scale is challenging, so a weights file and a network design file are commonly used. While training the YOLO network for specific purposes is possible, it is more complex than training ordinary convolutional neural networks. YOLO is often utilized for the general classification of various objects encountered in daily life, as it is trained on several million images, achieving high accuracy. In contrast, our convolutional neural network consists of 5 layers and can determine 5 classes. Trained on approximately 20,000 images, it is more compact and suitable for specific object detection. To expand the network, adding new classes and retraining suffice. Although we can design and scale this network to match others, adding unique layers not commonly used in networks like YOLO. Every detector, regardless of its success, encounters issues such as misclassification or errors (Fig. 11) or failing to detect objects at their actual locations. Improving detectors

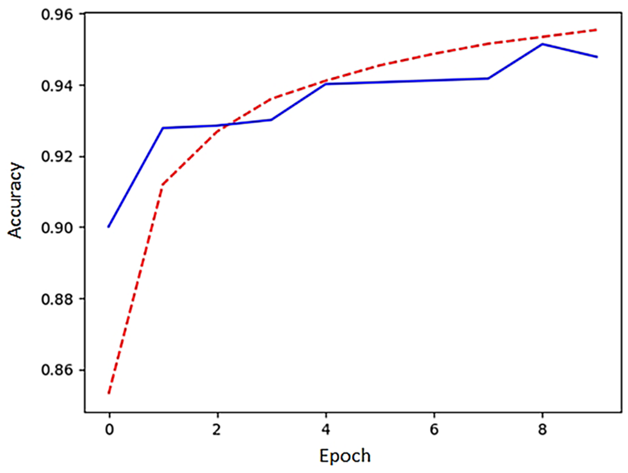


Fig. 9. Graph depicting the evolution of neural network accuracy (red–training set, blue–validation set).

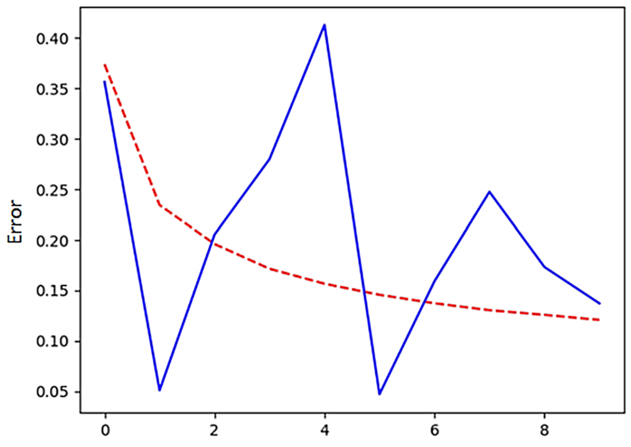


Fig. 10. Graph depicting the evolution of neural network error (red–training set, blue–validation set).

involves training, modifying the neural network's structure, and refining the detection and classification accuracy rates.

In Fig. 11 we can see examples of bad detection and classification. On the left part we can see objects that are too near to each other and on the right part we can see a wrong classification of the object, where the algorithm evaluated a dog as a cat.

If we compared other authors, we achieved comparable accuracy and error rate. The results are primarily dependent on how the network is trained and how many layers it contains. For example Shen et al.[42]. created a detection system to detect fire, did not detect other objects, only created a simple neural network containing 12 convolutional layers. The first 9 layers were pre-trained and the other layers use prediction to estimate bounding boxes, use pre-validation to calculate accuracy, and use validation to calculate formal training accuracy. During the training phase, they used 2 methods: pre-training and formal training. During the training phase, the two methods alternated–20 epochs of pre-training and 40 epochs of formal training. They used pre-training to reduce the loss function and increase the accuracy of the prediction. Formal training focused more on bounding boxes. In the paper, they report a accuracy rate of pre-training of almost 100% and formal training of 76%, which means that we can consider the detector to be reliable[42].

Li et al.. created a detector for detecting defects on the surface of steel belts. n production, materials can be damaged in various ways, such as scratches, pollution, and others. Such detection is very demanding in terms of the speed of movement of such a belt. In their project, they created a network with 27 convolutional layers (YOLO uses 24 layers by default). The first 25 layers were used to detect defects and the last two layers to categorize and label them. They replaced the pooling layers with two-step convolutional layers. With the network trained in this way, they achieved an accuracy of roughly 60%[43].

Conclusion

In this paper, we present a straightforward method for analysing video recordings using a pre-trained object network. From results we can see that the YOLO detector, along with its diverse set of classes, provides an

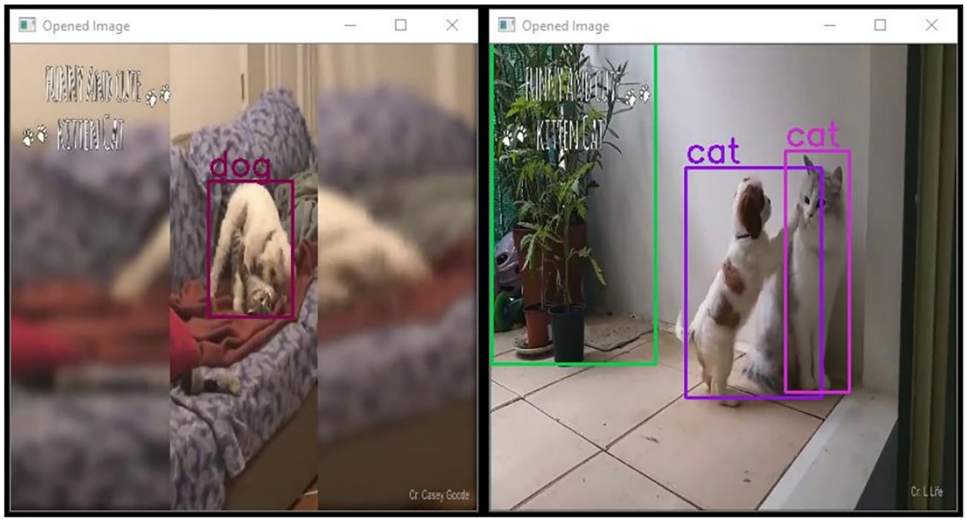


Fig. 11. Example of bad classification.

advantage in detecting and classifying various objects. If the YOLO detector lacks specific classes, they can be supplemented by adding a suitable dataset and retraining the network. The detection and classification of objects are increasingly utilized in various applications today. It serves purposes such as identifying individuals during attendance checks at facilities like factories or schools, enhancing security. Additionally, it functions as a method of classification in airport baggage control, counting different types of vehicles in traffic (e.g., passenger cars or lorries), checking for traffic violations, or aiding in the detection and classification of objects for individuals with visual impairments.

There are a large number of algorithms that detect motion and classify an object in real time. For us was not a priority for us to use the app in real time. For us, it was important to determine the following based on training the model: what object it is and where (within the time record) it is located. The advantage of this solution is that we have a record that we can revisit at any time and search for the object we are interested in (of course if it has been trained through the model). By inserting a new dataset and re-training the model, we can classify essentially any object. In our future investigations, we focus on the detection and classification of brown bears, determining the time and manner of their appearance near beehives.

Data availability

Application is available from the corresponding author on data repository (Magdin, M., & Kasan, M. (2025). YOLO Detector. Zenodo. https://doi.org/10.5281/zenodo.15771554).

Received: 18 February 2025; Accepted: 30 June 2025
Published online: 14 July 2025

References

1. Iqbal N, Saad Missen MM, Salamat N, Prasath VBS (2019) On video based human abnormal activity detection with histogram of oriented gradients. Handbook of Multimedia Information Security: Techniques and Applications 431–448. https://doi.org/10.1007/978-3-030-15887-3_21/TABLES/1
2. Ghasemi, M., Varshosaz, M., Pirasteh, S. & Shamsipour, G. Optimizing sector ring histogram of oriented gradients for human injured detection from drone images. *Geomat. Nat. Haz. Risk* **12**, 581–604. https://doi.org/10.1080/19475705.2021.1884608 (2021).
3. Ghasemi M, Varshosaz M, Pirasteh S (2020) Evaluating sector ring histogram of oriented gradients filter in locating humans within UAV images. The international archives of the photogrammetry, remote sensing and spatial information sciences XLIII-B2-2:23–27. https://doi.org/10.5194/ISPRS-ARCHIVES-XLIII-B2-2020-23-2020
4. Zhao, Z. Q., Zheng, P., Xu, S. T. & Wu, X. Object detection with deep learning: a review. *IEEE Trans. Neural Netw. Learn. Syst.* **30**, 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865 (2019).
5. Khan, A., Sohail, A., Zahoora, U. & Qureshi, A. S. A survey of the recent architectures of deep convolutional neural networks. *Artif. Intell. Rev.* **53**, 5455–5516. https://doi.org/10.1007/S10462-020-09825-6 (2020).
6. Bhatt, D. et al. CNN variants for computer vision: history, architecture, application. *Chall. Future Scope. Electron.* **10**, 2470. https://doi.org/10.3390/ELECTRONICS10202470 (2021).
7. Bhat, S. S., Ananth, A. & S VP,. Design and evolution of deep convolutional neural networks in image classification – a review. *Int. J. Integr. Eng.* **15**, 213–225. https://doi.org/10.30880/ijie.2023.15.01.019 (2023).
8. Owais, M. et al. Prioritizing rear-end crash explanatory factors for injury severity level using deep learning and global sensitivity analysis. *Expert Syst. Appl.* **245**, 123114. https://doi.org/10.1016/J.ESWA.2023.123114 (2024).
9. Liu T, Fang S, Zhao Y, et al (2015) Implementation of training convolutional neural networks
10. Bommareddy S, Samyal T, Dahiya S (2023) Implementation of a deepfake detection system using convolutional neural networks and adversarial training. In: 2023 3rd international conference on intelligent technologies, CONIT 2023. https://doi.org/10.1109/CONIT59222.2023.10205614
11. Aloysius N, Geetha M (2018) A review on deep convolutional neural networks. In: proceedings of the 2017 IEEE international conference on communication and signal processing, ICCSP 2017 2018 pp. 588–592. https://doi.org/10.1109/ICCSP.2017.8286426

12. Goodfellow IJ, Shlens J, Szegedy C (2014) Explaining and harnessing adversarial examples. In: 3rd international conference on learning representations, ICLR 2015 - Conference Track Proceedings
13. Liu W, Anguelov D, Erhan D, et al (2016) SSD: Single shot multibox detector. Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) 9905 LNCS:21–37. https://doi.org/10.1007/978-3-319-46448-0_2/FIGURES/5
14. Huang, C. et al. Trident SSD: a trident single-shot multibox object detector with deconvolution. *J. Phys. Conf. Ser.* **1631**, 012182. https://doi.org/10.1088/1742-6596/1631/1/012182 (2020).
15. Mukti BK, Novita M, Amirudin M, et al (2023) A comprehensive analysis of mask detection using convolutional neural networks (CNN) and single shot multibox detector (SSD) approach. 2023 international conference on sustainable emerging innovations in engineering and technology, ICSEIET 2023 512–516. https://doi.org/10.1109/ICSEIET58677.2023.10303567
16. Youwai, S., Chaiyaphat, A. & Chaipetch, P. YOLO9tr: a lightweight model for pavement damage detection utilizing a generalized efficient layer aggregation network and attention mechanism. *J. Real-Time Image Proc.* **21**, 542. https://doi.org/10.1007/s11554-024-01545-2 (2024).
17. Mao, M., Lee, A. & Hong, M. Efficient fabric classification and object detection using YOLOv10. *Electronics (Switzerland)* **13**, 93840. https://doi.org/10.3390/electronics13193840 (2024).
18. Alif MAR, Hussain M (2024) YOLOv1 to YOLOv10: A comprehensive review of YOLO variants and their application in the agricultural domain
19. Geetha, A. S., Alif, M. A. R., Hussain, M. & Allen, P. Comparative analysis of YOLOv8 and YOLOv10 in vehicle detection: Performance metrics and model efficacy. *Vehicles* **6**, 1364–1382. https://doi.org/10.3390/VEHICLES6030065 (2024).
20. Sapkota, R., Ahmed, D. & Karkee, M. Comparing YOLOv8 and Mask R-CNN for instance segmentation in complex orchard environments. *Artif. Intell. Agric.* **13**, 84–99. https://doi.org/10.1016/J.AIIA.2024.07.001 (2024).
21. Wang, S., Lin, S., Sun, F. & Li, X. Enhanced YOLOv8 with attention mechanisms for accurate detection of colorectal polyps. *Biomed. Signal Proc. Control* **100**, 106942. https://doi.org/10.1016/j.bspc.2024.106942 (2025).
22. Niu Y, Cao J, Wang Y (2025) A Road Defect Detection Algorithm Based on Improved YOLOv8
23. Daniel E, Kathiresan V, Priyadarshini C, et al (2025) Real Time Sign Recognition using YOLOv8 Object Detection Algorithm for Malayalam Sign Language. Fusion: Practice and Applications 17: 135–145. https://doi.org/10.54216/FPA.170110
24. Meng, Z. et al. YOLOv10-pose and YOLOv9-pose: Real-time strawberry stalk pose detection models. *Comput. Ind.* **165**, 104231. https://doi.org/10.1016/j.compind.2024.104231 (2025).
25. Li X, Cai C, Song B, Yang Y (2025) YOLOV8-MR: An Improved Lightweight YOLOv8 Algorithm for Tomato Fruit Detection (April 2024). IEEE Access 1–1. https://doi.org/10.1109/ACCESS.2025.3533489
26. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition. pp 779–788.
27. Bhat GP, Cholli NG (2021) Effective object detection using Tensorflow facilitated YOLOv3 model. In: CSITSS 2021 - 2021 5th international conference on computational systems and information technology for sustainable solutions, proceedings
28. Abadi M, Barham P, Chen J, et al (2016) {TensorFlow}: a system for {large-scale} machine learning
29. Magdin, M. et al. Comparison of multilayer neural network models in terms of success of classifications based on EmguCV MLNET and TensorflowNet. *Appl. Sci.* **12**, 3730. https://doi.org/10.3390/APP12083730 (2022).
30. Han, S. et al. Using the tensorflow deep neural network to classify mainland china visitor behaviours in hong kong from check-in data. *ISPRS Int. J. Geo Inform.* **7**, 158. https://doi.org/10.3390/IJGI7040158 (2018).
31. Verma A, Pedrosa L, Korupolu M, et al (2015) Large-scale cluster management at Google with Borg. In: proceedings of the 10th european conference on computer systems, EuroSys 2015. https://doi.org/10.1145/2741948.2741964
32. Yu Y, Abadi Google Brain M, Barham Google Brain P, et al (2018) Dynamic control flow in large-scale machine learning. In: proceedings of the 13th eurosys conference, EuroSys 2018 2018-January:15. https://doi.org/10.1145/3190508.3190551
33. Li L, Fang J, Fu H, et al (2018) SwCaffe: a parallel framework for accelerating deep learning applications on sunway taihulight. Proceedings - IEEE international conference on cluster computing, ICCC 2018-September: pp. 413–422. https://doi.org/10.1109/CLUSTER.2018.00087
34. Li, M. et al. Towards optimized tensor code generation for deep learning on sunway many-core processor. *Front. Comp. Sci.* **18**, 1–15. https://doi.org/10.1007/S11704-022-2440-7/METRICS (2024).
35. Li, M. et al. The deep learning compiler: a comprehensive survey. *IEEE Trans. Parallel Distrib. Syst.* **32**, 708–727. https://doi.org/10.1109/tpds.2020.3030548 (2021).
36. Kaymak C, Ucar A (2019) Implementation of object detection and recognition algorithms on a robotic arm platform using raspberry Pi. In: 2018 international conference on artificial intelligence and data processing, IDAP 2018
37. Alhamzi KM, Elmogy M, Alhamzi K, Barakat S (2015) 3D object recognition based on local and global features using point cloud library SEE PROFILE 3D object recognition based on local and global features using point cloud library
38. Cat and Dog. https://www.kaggle.com/datasets/tongpython/cat-and-dog. Accessed 1 Feb 2024
39. Maji S, Chicago T, Rahtu E, et al (2013) Fine-grained visual classification of aircraft
40. Men-Women Classification. https://www.kaggle.com/datasets/saadpd/menwomen-classification. Accessed 1 Feb 2024
41. Krause J, Stark M, Deng J, Fei-Fei L (2013) 3D object representations for fine-grained categorization. In: proceedings of the IEEE international conference on computer vision. pp 554–561
42. Shen D, Chen X, Nguyen M, Yan WQ (2018) Flame detection using deep learning. In: Proceedings - 2018 4th International Conference on Control, Automation and Robotics, ICCAR 2018. pp 416–420
43. Li, J., Su, Z., Geng, J. & Yin, Y. Real-time detection of steel strip surface defects based on improved YOLO detection network. *IFAC-PapersOnLine.* **51**(21), 76–81 (2018).

Acknowledgements
This work was supported by the Scientific Grant Agency of the Ministry of Education of the Slovak Republic (ME SR) and of Slovak Academy of Sciences (SAS) under the contract No. VEGA 1/0385/23.

Author contributions
All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by M.M, Z.B. programming and experiment preparation M.M. The first draft of the manuscript was written by M.M. and all authors commented on previous versions of the manuscript.

Funding
This work was supported by the Scientific Grant Agency of the Ministry of Education of the Slovak Republic (ME SR) and of Slovak Academy of Sciences (SAS) under the contract No. VEGA 1/0385/23.

Declarations

Competing interests
Each named author has substantially contributed to conducting the underlying research and drafting this manuscript. Additionally, to the best of our knowledge, the named authors have no conflict of interest, financial or otherwise.

Additional information
Correspondence and requests for materials should be addressed to M.M.

Reprints and permissions information is available at www.nature.com/reprints.