Computer vision methods and algorithms for automatic detection and classification of objects in decision support systems in agriculture

Alena Yablokova[1,2*], *Igor* Kovalev[2,3,4,5]*, Dmitry* Kovalev[1,6]*, Valeria* Podoplelova[2,7], and *Aziza* Kobilova[8]

[1]Krasnoyarsk Science and Technology City Hall of the Russian Union of Scientific and Engineering Public Associations, Krasnoyarsk, Russia
[2]Krasnoyarsk State Agrarian University, Krasnoyarsk, Russia
[3]Siberian Federal University, Krasnoyarsk, Russia
[4]Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia
[5]Navoi State University of Mining and Technology, Navoi, Uzbekistan
[6]National Research University «Tashkent Institute of Irrigation and Agricultural Mechanization Engineers", Tashkent, Uzbekistan
[7]Sochi State University, Sochi, Russia
[8]Bukhara State University, Bukhara, Uzbekistan

Abstract. The paper examines aspects of developing and formalizing the task of applying computer vision methods and algorithms using OpenCV (implemented in Python version 3.13 notation) for automatic detection and classification of objects in decision support systems. A software implementation of a modular example is provided, enabling automatic detection and classification for the detection of plant diseases based on their external characteristics in decision support systems in agriculture. This approach will facilitate prompt response to plant diseases and the implementation of necessary measures for their treatment.

1 Introduction

Scientific and technological progress, as a key aspect of the fourth industrial revolution, sets the direction for the internal policy of implementing information and communication technologies into the activities of key sectors of the state as fundamental foundations for sustainable development [1-4]. The widespread digitization of processes is accompanied by an exponential increase in data volumes, making the question of processing and analytics relevant. Quality characteristics of input data (completeness and volume, accuracy, validity, relevance, timeliness, variability, and verifiability) can ensure the correctness of conclusions and decisions for effective decision support system functioning. In this context, data processing and analysis technologies become an integral part of the digitization process, providing opportunities for automation and optimization of decision-making processes, creating a need for the use of specialized methods and algorithms for their analysis. The task

* Corresponding author: alena.yabl@yandex.ru

of automatic object detection and classification becomes crucial in solving applied problems where visual data analysis is required to make informed actions based on analysis. In the modern world, such systems are increasingly in demand due to their ability to process and analyze large volumes of data and assist in making informed decisions in various fields of activity. In industry, these systems are used for quality control of production, defect detection on material surfaces, equipment monitoring, and preventing emergencies. In the field of transportation and logistics, systems are applied for managing transportation flows, optimizing delivery routes, automatic license plate recognition, and managing warehouse inventory. In the security sphere, automatic detection and classification systems are used for face recognition, access control, video surveillance analysis, anomaly detection, and early warning of possible threats, etc. However, in some cases, problems arise when applying methods and algorithms for automatic object detection and classification in decision support systems. One of the key problems is the selection and application of patterns, algorithms, and methods for processing visual information that do not take into account operating conditions and properties of the decision support system. In this context, the Python programming language and the OpenCV computer vision library are powerful tools for solving the aforementioned problems. Python is one of the most popular and flexible programming languages, which has a vast ecosystem of libraries and tools for data processing, including visual information. The OpenCV library, in turn, provides a wide range of computer vision functions and algorithms, allowing for solving various tasks related to visual information processing. Choosing Python and OpenCV for solving the task of automatic object detection and classification is justified by their wide functionality, ease of use, high efficiency, and the possibility of integration with existing patterns. They enable rapid prototyping and experimentation and efficiently address complex visual data analysis and processing tasks [5-8].

2 Materials and methods

2.1 Methods and algorithms for automatic detection and classification of objects in medium decision support

Decision support systems use various methods and algorithms for automatic detection and classification of objects to analyze visual data and make decisions. The most common approaches include machine learning, deep learning, and computer vision [9-10].

2.2 Machine learning

Machine learning is a branch of artificial intelligence that studies the use of algorithms and methods (based on mathematical models of data) to train systems without direct instructions. In machine learning, patterns are identified using algorithms and methods, based on which a data model is created for prediction.

Support Vector Machine (SVM) is a machine learning algorithm that is used for classification and regression problems. In the context of automatic object detection and classification in decision support systems, SVM is used to classify objects into different classes based on their features. The main idea is to search for a hyperplane that maximally separates objects of different classes in the feature space. To do this, SVM constructs an optimal separating hyperplane in such a way that the distance from it to the nearest objects of each class, called the gap, is maximum. This approach allows SVM to build a linear or nonlinear classifier that can generalize well from data and detect complex patterns. The advantages of the method include efficiency in high-dimensional spaces, the ability to process data with different types of features, as well as generalization ability, which allows you to avoid overfitting on training data [11-12].

Ensemble learning methods combine the predictions of individual models to produce more accurate and robust results than individual models could achieve on their own. In the context of automatic object detection and classification in decision support systems, ensemble learning methods can be applied to improve classification accuracy and reliability. The main methods of ensemble learning include random forest (the method builds several decision trees during training and uses them to predict the class of an object by voting or averaging the predictions of individual trees), gradient boosting (step-by-step construction of an ensemble of models, each of which compensates for the errors of the previous one, model predictions are combined to produce the final prediction), the bagging method creates several subsamples from the training dataset using the bootstrap method and trains a separate model on each subsample, and the predictions of all models are averaged to obtain the final result. Ensemble learning methods are suitable for processing complex data with a large number of features and nonlinear dependencies [13-14].

2.3 Deep learning

Deep learning is a branch of machine learning that uses neural networks with multiple layers to extract high-level features from input data. Unlike classical machine learning algorithms, which require manual feature generation, deep learning allows the model to extract hierarchical features automatically at different levels of abstraction. The main feature is the use of deep neural networks, such as deep convolutional neural networks (CNN), recurrent neural networks (RNN) and their various combinations, capable of learning from large amounts of data and automatically identifying complex patterns and patterns, which allows them to solve detection problems and classification of objects [15-16].

Convolutional neural networks (ConvNet/CNN) are a class of deep neural networks that specialize in processing and analyzing visual data. CNNs are based on the principle of convolution, which allows automatic extraction of hierarchical features from input data with minimal preprocessing. CNNs consist of the following layers: Convolutional layers are used to process input images: the image is scanned using filters (kernels) to extract key features. Subsampling layers reduce the size of the image while preserving key features, which reduces the number of model parameters and improves its generalization ability. The subsampling results are fed to fully connected layers, which perform object classification based on the extracted features. These layers are used to combine information from all previous layers and make a final decision about the class of the object. Advantages of CNNs include automatic feature extraction from input data without the need for manual definition, hierarchical processing that allows CNNs to adapt to complex and diverse object structures, high accuracy in object classification, and applicability to various visual data types [17-18].

2.4 Computer vision

Computer vision is a branch of artificial intelligence that develops algorithms for analyzing, modifying and processing visual data. The main goal of computer vision in such systems is to provide automatic detection, recognition and classification of objects in order to provide information for decision making. The main computer vision methods include feature and edge-based object detectors, gradient descriptors, clustering algorithms, and Haar cascades [19-21].

Feature-based object detectors are a computer vision method that detects objects in images by analyzing their characteristic features. Instead of using patterns and trained models, the method focuses on identifying unique visual characteristics.

Edge-based detectors rely on identifying sharp changes in brightness or color that correspond to transitions between different regions of objects in an image.

Gradient descriptors analyze local regions of an image and describe them using luminance gradients. Gradient descriptors are used to describe the textural characteristics of objects in images, which allows objects to be classified based on their texture.

Clustering algorithms are used to group similar objects in an image into different classes or clusters. Clustering algorithms can help separate objects into groups based on their similarities, making subsequent classification easier.

Haar cascades use sets of features (cascades) to find objects in images using a cascade approach, allowing you to quickly weed out inappropriate areas of the image and focus on areas where the object is likely to be present.

3 Results and discussion

3.1 Statement and formalization of the problem

The task of applying computer vision methods and algorithms for automatic detection and classification of objects in an abstract decision support system has the following structure:

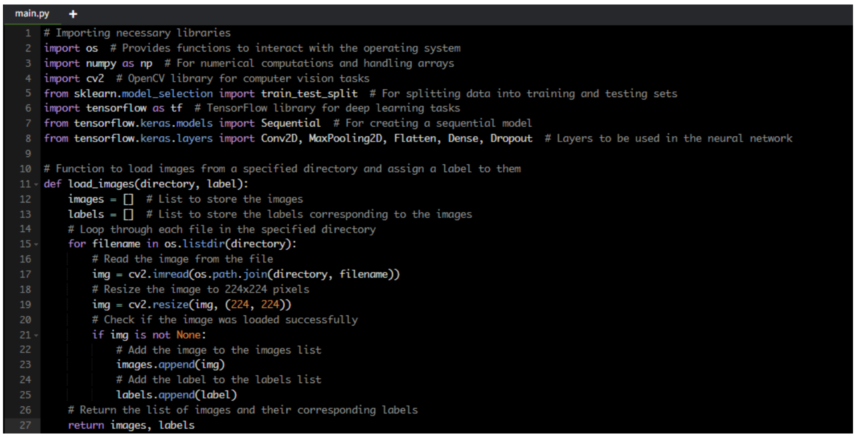
1. Determining system requirements: determining the types of objects to be detected and classified, setting criteria for the accuracy and speed of detection and classification.
2. Selection of the most suitable computer vision methods and algorithms, taking into account system requirements and selection of machine and deep learning patterns for integration.
3. Data preparation: collection and preparation of a training data set, including images of objects and their classification labels, dividing the data into training and test samples.
4. Model training: implementation of selected algorithms and methods using the Python programming language and the OpenCV library, training the model on a training data set for the purpose of detecting and classifying objects.
5. Model evaluation: assessment of the quality of the trained model on a test sample, iterative improvement of the model by changing the parameters and architecture of the algorithms.
6. Integration with a decision-making system: development of an interface or API for integrating the developed model with a decision support system, integration of the model into the decision-making process for automatic analysis of visual data and issuing recommendations.
7. Testing and optimization: testing the integrated system on various input data and use scenarios.
8. Optimization of system operation in order to increase the speed and accuracy of detection and classification.
9. Ensuring reliability and security: developing mechanisms for error handling and failure recovery, ensuring data protection and privacy when working with visual information.

3.2 Modular example and software implementation

A modular example is automatic detection and classification for detecting plant diseases based on their external signs in a decision support system. As a result, the system must effectively and accurately detect both plants and diseases on them, with high accuracy, in order to ensure a prompt response to diseases and the adoption of the necessary measures for their treatment.

Example code implementation:

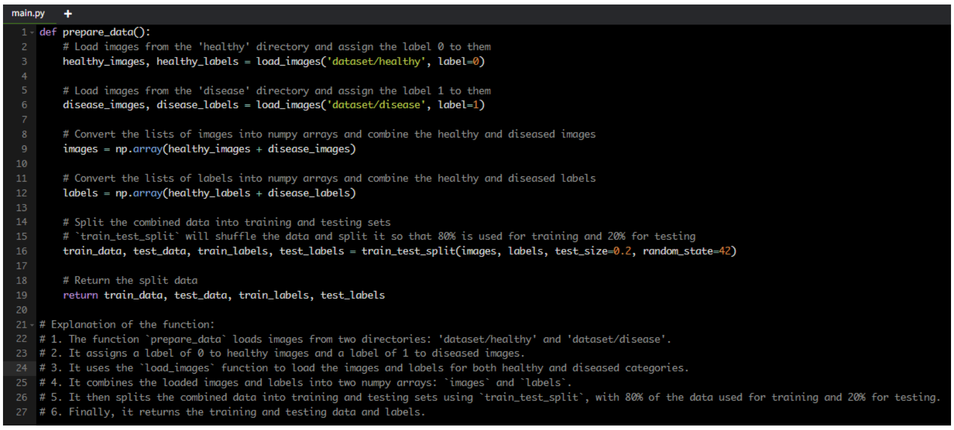
1. Loading and Preprocessing Images: The load_images function loads images from the directory and resizes them to 224x224 pixels.



```python
# Importing necessary libraries
import os  # Provides functions to interact with the operating system
import numpy as np  # For numerical computations and handling arrays
import cv2  # OpenCV library for computer vision tasks
from sklearn.model_selection import train_test_split  # For splitting data into training and testing sets
import tensorflow as tf  # TensorFlow library for deep learning tasks
from tensorflow.keras.models import Sequential  # For creating a sequential model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  # Layers to be used in the neural network

# Function to load images from a specified directory and assign a label to them
def load_images(directory, label):
    images = []  # List to store the images
    labels = []  # List to store the labels corresponding to the images
    # Loop through each file in the specified directory
    for filename in os.listdir(directory):
        # Read the image from the file
        img = cv2.imread(os.path.join(directory, filename))
        # Resize the image to 224x224 pixels
        img = cv2.resize(img, (224, 224))
        # Check if the image was loaded successfully
        if img is not None:
            # Add the image to the images list
            images.append(img)
            # Add the label to the labels list
            labels.append(label)
    # Return the list of images and their corresponding labels
    return images, labels
```

Fig. 1. Loading and Preprocessing Images.

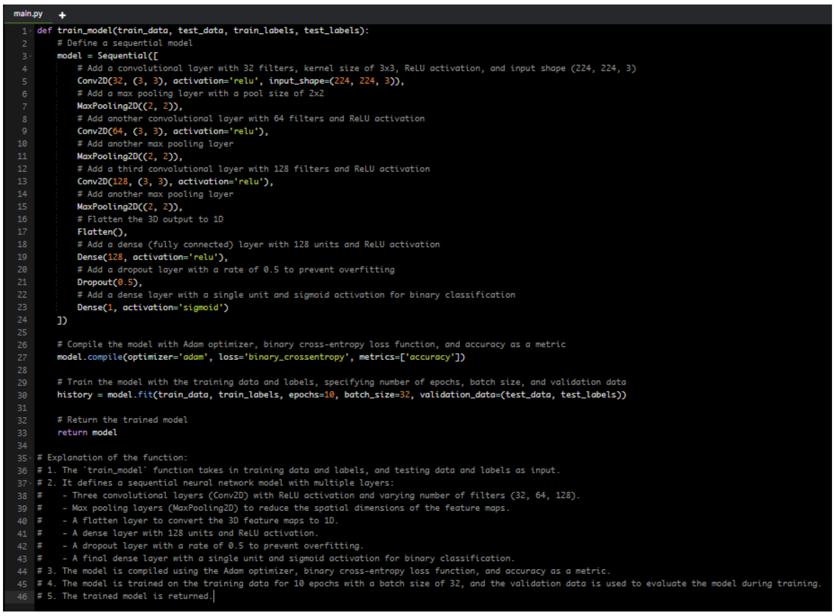
2. Data preparation: The prepre_data() function collects data from two catalogs (healthy and diseased plants), creates a common set of data and labels, and then splits it into training and test sets.



```python
def prepare_data():
    # Load images from the 'healthy' directory and assign the label 0 to them
    healthy_images, healthy_labels = load_images('dataset/healthy', label=0)

    # Load images from the 'disease' directory and assign the label 1 to them
    disease_images, disease_labels = load_images('dataset/disease', label=1)

    # Convert the lists of images into numpy arrays and combine the healthy and diseased images
    images = np.array(healthy_images + disease_images)

    # Convert the lists of labels into numpy arrays and combine the healthy and diseased labels
    labels = np.array(healthy_labels + disease_labels)

    # Split the combined data into training and testing sets
    # `train_test_split` will shuffle the data and split it so that 80% is used for training and 20% for testing
    train_data, test_data, train_labels, test_labels = train_test_split(images, labels, test_size=0.2, random_state=42)

    # Return the split data
    return train_data, test_data, train_labels, test_labels

# Explanation of the function:
# 1. The function `prepare_data` loads images from two directories: 'dataset/healthy' and 'dataset/disease'.
# 2. It assigns a label of 0 to healthy images and a label of 1 to diseased images.
# 3. It uses the `load_images` function to load the images and labels for both healthy and diseased categories.
# 4. It combines the loaded images and labels into two numpy arrays: `images` and `labels`.
# 5. It then splits the combined data into training and testing sets using `train_test_split`, with 80% of the data used for training and 20% for testing.
# 6. Finally, it returns the training and testing data and labels.
```

Fig. 2. Data preparation.

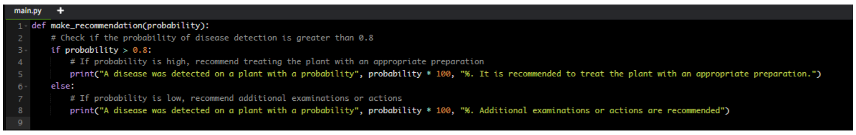
3. Model creation and training: The train_model function creates and compiles a convolutional neural network (CNN) model and then trains it on the training dataset.



```python
def train_model(train_data, test_data, train_labels, test_labels):
    # Define a sequential model
    model = Sequential([
        # Add a convolutional layer with 32 filters, kernel size of 3x3, ReLU activation, and input shape (224, 224, 3)
        Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
        # Add a max pooling layer with a pool size of 2x2
        MaxPooling2D((2, 2)),
        # Add another convolutional layer with 64 filters and ReLU activation
        Conv2D(64, (3, 3), activation='relu'),
        # Add another max pooling layer
        MaxPooling2D((2, 2)),
        # Add a third convolutional layer with 128 filters and ReLU activation
        Conv2D(128, (3, 3), activation='relu'),
        # Add another max pooling layer
        MaxPooling2D((2, 2)),
        # Flatten the 3D output to 1D
        Flatten(),
        # Add a dense (fully connected) layer with 128 units and ReLU activation
        Dense(128, activation='relu'),
        # Add a dropout layer with a rate of 0.5 to prevent overfitting
        Dropout(0.5),
        # Add a dense layer with a single unit and sigmoid activation for binary classification
        Dense(1, activation='sigmoid')
    ])

    # Compile the model with Adam optimizer, binary cross-entropy loss function, and accuracy as a metric
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model with the training data and labels, specifying number of epochs, batch size, and validation data
    history = model.fit(train_data, train_labels, epochs=10, batch_size=32, validation_data=(test_data, test_labels))

    # Return the trained model
    return model

# Explanation of the function:
# 1. The `train_model` function takes in training data and labels, and testing data and labels as input.
# 2. It defines a sequential neural network model with multiple layers:
#    - Three convolutional layers (Conv2D) with ReLU activation and varying number of filters (32, 64, 128).
#    - Max pooling layers (MaxPooling2D) to reduce the spatial dimensions of the feature maps.
#    - A flatten layer to convert the 3D feature maps to 1D.
#    - A dense layer with 128 units and ReLU activation.
#    - A dropout layer with a rate of 0.5 to prevent overfitting.
#    - A final dense layer with a single unit and sigmoid activation for binary classification.
# 3. The model is compiled using the Adam optimizer, binary cross-entropy loss function, and accuracy as a metric.
# 4. The model is trained on the training data for 10 epochs with a batch size of 32, and the validation data is used to evaluate the model during training.
# 5. The trained model is returned.
```

Fig. 3. Model creation and training.

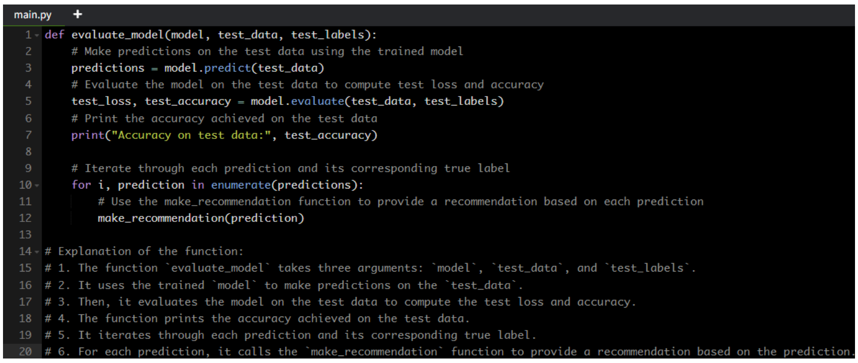
4. The make_recommendation() function takes the probability of detecting a disease on a plant and outputs the appropriate recommendation.



```python
def make_recommendation(probability):
    # Check if the probability of disease detection is greater than 0.8
    if probability > 0.8:
        # If probability is high, recommend treating the plant with an appropriate preparation
        print("A disease was detected on a plant with a probability", probability * 100, "%. It is recommended to treat the plant with an appropriate preparation.")
    else:
        # If probability is low, recommend additional examinations or actions
        print("A disease was detected on a plant with a probability", probability * 100, "%. Additional examinations or actions are recommended")

```

Fig. 4. Model creation and training.

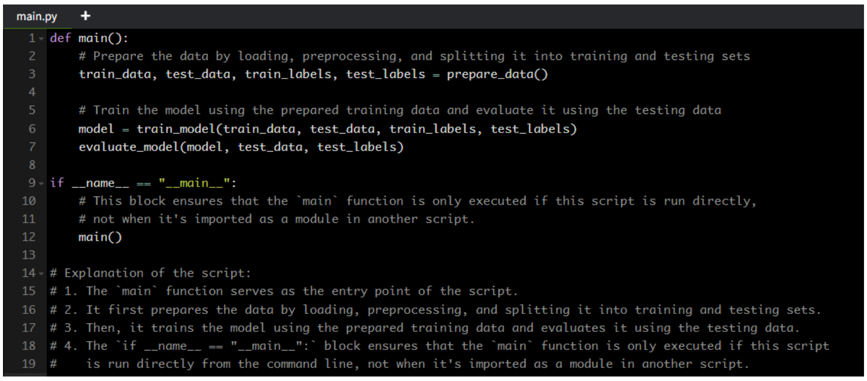
5. Model quality evaluation and recommendations: The evaluate_model() function evaluates the quality of the model on the test set, and also makes recommendations based on the predicted probabilities of detecting a disease.



```python
def evaluate_model(model, test_data, test_labels):
    # Make predictions on the test data using the trained model
    predictions = model.predict(test_data)
    # Evaluate the model on the test data to compute test loss and accuracy
    test_loss, test_accuracy = model.evaluate(test_data, test_labels)
    # Print the accuracy achieved on the test data
    print("Accuracy on test data:", test_accuracy)

    # Iterate through each prediction and its corresponding true label
    for i, prediction in enumerate(predictions):
        # Use the make_recommendation function to provide a recommendation based on each prediction
        make_recommendation(prediction)

# Explanation of the function:
# 1. The function `evaluate_model` takes three arguments: `model`, `test_data`, and `test_labels`.
# 2. It uses the trained `model` to make predictions on the `test_data`.
# 3. Then, it evaluates the model on the test data to compute the test loss and accuracy.
# 4. The function prints the accuracy achieved on the test data.
# 5. It iterates through each prediction and its corresponding true label.
# 6. For each prediction, it calls the `make_recommendation` function to provide a recommendation based on the prediction.
```

Fig. 5. Model quality evaluation and recommendations.

6. Running. The main function main() performs the above steps: preparing data, training the model and assessing the quality of the model with subsequent output of recommendations.



```python
def main():
    # Prepare the data by loading, preprocessing, and splitting it into training and testing sets
    train_data, test_data, train_labels, test_labels = prepare_data()

    # Train the model using the prepared training data and evaluate it using the testing data
    model = train_model(train_data, test_data, train_labels, test_labels)
    evaluate_model(model, test_data, test_labels)

if __name__ == "__main__":
    # This block ensures that the `main` function is only executed if this script is run directly,
    # not when it's imported as a module in another script.
    main()

# Explanation of the script:
# 1. The `main` function serves as the entry point of the script.
# 2. It first prepares the data by loading, preprocessing, and splitting it into training and testing sets.
# 3. Then, it trains the model using the prepared training data and evaluates it using the testing data.
# 4. The `if __name__ == "__main__":` block ensures that the `main` function is only executed if this script
#    is run directly from the command line, not when it's imported as a module in another script.
```

Fig. 6. Running.

4 Conclusion

Digital image processing is becoming increasingly important in agriculture, making it possible to effectively detect plant diseases, monitor their condition and take appropriate action. The use of computer vision methods, such as convolutional neural networks, significantly improves diagnostic and monitoring processes in the agricultural sector. Automatic detection and classification of objects allows you to quickly respond to possible problems, minimize damage from plant diseases and increase production levels. However, it must be taken into account that the accuracy and efficiency of the algorithms depend on the quality of the input data, the training sample and the correct choice of model. It is important to continue research in this area to further improve computer vision methods and their successful integration into agricultural practice.

References

1. K. Schwab, *Shaping the Fouth Industrial Revolution* (PENGUIN GROUP, 2018)
2. C.F. Machado, J.P. Davim, *Industry 5.0: Creative and Innovative Organizations* (Springer International Publishing, 2023)
3. A. Yablokova, D. Kovalev, I. Kovalev, V. Podoplelova, K. Astanakulov, E3S Web of Conferences **471** 04018 (2024)
4. A. Yablokova, D. Kovalev, I. Kovalev, V. Podoplelova, E3S Web of Conferences **486** 03025 (2024)
5. R. Abbasi, P. Martinez, R. Ahmad. Smart Agricultural Technology **2** 100042 (2022)
6. R.S. Sandhya Devi, V.R. Vijay Kumar; P. Sivakumar, *A Review of image Classification and Object Detection on Machine learning and Deep Learning Techniques,* in Proceedings of the 5th International Conference on Electronics, Communication and Aerospace Technology, 2-4 December 2021, Coimbatore, India (2021)
7. D. Racheed, R. Muin, A. Jaylan, International Journal of Information Technology and Applied Sciences **2(3)**, 21-29 (2020)
8. E.V. Khrol, K.S. Sharonova, Modern Innovations, Systems and Technologies **3(4)**, 0311-0321 (2023)
9. A. Rahman, H. Anwar, N. Shah, Kh. Sulaiman, H.U. Khan. Applied Sciences **13(22)**, 12426 (2023)
10. I.H. Sarker, SN Computer Science **2(160)** 2021

11. J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, A. Lopez, Neurocomputing **408**, 189-215 (2020)
12. Zh. Jun. J. Phys.: Conf. Ser. **1748** 052006 (2021)
13. A. Mahammed, R. Kora, Journal of King Saud University - Computer and Information Sciences **35(2),** 757-774 (2023)
14. D.I. Mienye, Ya. Sun, IEEE Access **10**, 99129-99149 (2022)
15. L. Alzubaidi et al., Journal of Big Data **8**, 53(2021)
16. Zh. Hao, MATEC Web of Conferences **277**, 02035 (2010)
17. J. Ayeni. Applied Journal of Physical Science **4(4),** 42-50 (2022)
18. M. M. Taye, Jordan Computation **11(3)**, 52 (2023)
19. F. Huang, J. Phys.: Conf. Ser. **1648**, 042065 (2020)
20. Allashyam Charan et al., IOP Conf. Ser.: Mater. Sci. Eng. **1224**, 012027 (2022)
21. F. Ke et al., Modern Innovations, Systems and Technologies **3(1)**, 0401-0410 (2023)