Implementation of a Programming Environment for Recognition of Metallurgical Production Objects in Real Time

2025 International Russian Smart Industry Conference (SmartIndustryCon) | 979-8-3315-1124-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/SMARTINDUSTRYCON65166.2025.10986209

Anatolii V. Lednov
Novotroitsk branch of the National Research Technological University MISIS
Novotroitsk, Russia
alednov@mail.ru

Vitalii P. Oleinichenko
Novotroitsk branch of the National Research Technological University MISIS
Novotroitsk, Russia
volieinichienko@bk.ru

Danila D. Prokhorov
Novotroitsk branch of the National Research Technological University MISIS
Novotroitsk, Russia
prokhorov.9898@mail.ru

***Abstract*—The article is devoted to the creation and testing of a programming environment for an automatic identification system for steel sheets in metallurgical production based on the analysis of unique biometric features. The proposed solution ensures detection of a representative surface area and its subsequent recognition as the object moves along the process line. The proposed approach eliminates dependence on external markers, such as QR codes or tags, due to a detailed study of the surface texture, defect distribution and rolling patterns. To optimize the neural network model and simulate real operating conditions, a production line model was used that takes into account variable parameters such as lighting, object movement and image noise level. An important aspect of the study is a solution that allows performing recognition functions in real time and provides end-to-end integration of the system with adjacent industrial MES and SCADA platforms via the REST API.**

Keywords—neural network technologies, machine learning, convolutional neural networks, object modeling, digital twin

I. Introduction

Modern metallurgical production faces the need to implement high-tech solutions to improve efficiency, reduce costs and minimize the human factor. One of the key tasks in this area is the automation of identification and tracking of objects, such as steel sheets moving along production lines [1]. The use of computer vision technologies [2] and neural networks opens up new possibilities for solving this problem, providing tools for determining and identifying objects in real time.

The aim of the study is to configure and test a software environment for the development of an integrated sys-tem for dynamic identification of steel sheets, combining deep learning and computer vision methods. The work solves the following problems: detection of a representative area by marking in real time; determination of the location of steel sheets in the production line; integration with related industrial systems through standardized exchange protocols.

The areas of application of artificial intelligence technologies are currently formalized [3,4]. Industrial solutions remain open to exploration due to the complexity of obtaining primary data and selecting criteria for deep learning models [5-6]. Developing an automatic sheet metal identification system in metallurgical production requires a comprehensive approach, including both the selection of optimal hardware and the configuration of software components for data processing, model training, and its integration into industrial processes.

II. Methodology

In the context of computer vision and deep learning tasks, to ensure high accuracy and speed of the system, it is necessary to take into account modern requirements for computing resources. Let's consider the key stages of system development, starting with the selection of hardware that can effectively cope with the loads during model training and ending with the integration of the system with industrial platforms. The methodology is based on recommendations from the official NVIDIA documentation [7], which ensures its compliance with modern standards in the field of deep learning. To train and operate the automatic identification model of steel sheets, it is necessary to provide appropriate hardware. As a graphics processing unit (GPU), it is recommended to use NVIDIA devices with CUDA support, such as the RTX 3090 or A100, which provide significant acceleration of computing due to optimized cores for deep learning tasks. According to NVIDIA documentation, the Ampere architecture used in the RTX 30xx series demonstrates high performance in processing neural networks [7]. The central processing unit (CPU) should be multi-core, such as Intel Core or AMD Ryzen, which allows for efficient pre- and post-processing of data. To speed up matrix operations, it is recommended to use processors with AVX-512 support neural network and testing the operation of the program. RAM of at least 32 GB is required to work with large amounts of data, such as images and video streams. To store datasets and models, an SSD drive with a capacity of 1 TB is required, which provides high read and write speeds, which are critical for working with large amounts of information.

The work used hardware consisting of: graphics processor (GPU) - NVIDIA RTX 3070 video card, which sup-ports CUDA technology and provides significant acceleration of calculations due to optimized cores for deep learning tasks; central processing unit (CPU) - AMD Ryzen 7 5800X with 8 cores and 16 threads; RAM 32 GB; SSD drive - 1 TB.

The selected configuration provides a balance between performance and cost, allowing you to effectively solve the

problems of training models and processing data in conditions close to real industrial requirements. To capture im-ages of steel sheets on a production line, it is necessary to use high-resolution cameras, preferably with a resolution of at least Full HD. This ensures high image detail, which is a key factor for accurate identification and analysis of surface texture, defects and rolling patterns. To implement the system, powerful hardware alone is not enough. It is also necessary to organize clear interaction between software components that provide data processing, model training and its integration into industrial processes. The paper proposes the following structure for the interaction of frameworks and libraries (Fig.1).

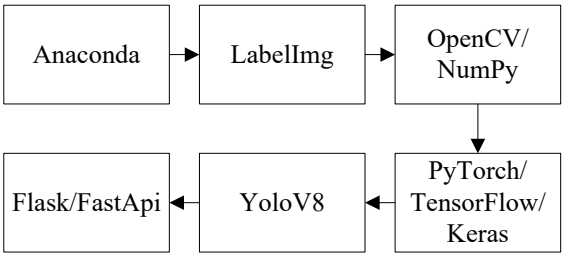


Fig. 1. Software structure.

Anaconda is used as an operating system for artificial intelligence, which provides convenient tools for manag-ing packages and creating isolated virtual environments, which simplifies development and deployment [7]. The Labe-lImg tool is used to mark up images and create annotations in YOLO or COCO formats, which allows you to quickly and accurately prepare data for training models [8]. Data processing is carried out using the OpenCV and NumPy libra-ries. OpenCV provides reading, transformation and visualization of images, including lighting correction, resizing and normalization. NumPy is used to work with data arrays and perform mathematical operations, which is especially important for augmentation and preprocessing tasks. The PyTorch, TensorFlow and Keras frameworks are used to develop and train deep learning models. PyTorch provides flexibility and high performance when working with neural networks [9-11], and TensorFlow and Keras allow you to create and optimize complex model architectures [12-14].

To detect objects in images of steel sheets, the YOLOv8 model from Ultralytics is used, which combines high accuracy and speed, making it an ideal choice for real-time computer vision tasks. The system is integrated with industrial platforms such as MES and SCADA via a REST API developed on the basis of Flask or FastAPI. This allows you to transfer object detection results in real time and automate the processes of accounting and quality control.

To understand the role of each technology in the system, let's consider them in more detail. This will allow you not only to evaluate their functionality, but also to determine the optimal approaches to their use within the framework of the task.

Anaconda is an integrated development environment. That is, all the packages and tools included in it are configured to work together. Its main difference from other package managers, such as pip, is that the installation of libraries occurs taking into account the already installed versions and their features.

The environment is an isolated environment (virtual machine) that simulates the operation of a full-fledged physical PC. The virtual environment manager built into Anaconda is responsible for its creation and operation, allocating a portion of the PC's physical resources (RAM, disk space, processor power, etc.) for this purpose. This tool can be used to simultaneously create several working environments. This feature simplifies the work, since for each specific task you can create your own environment with specific libraries. The working environment was installed using Anaconda. The virtual environment with the necessary libraries for model training and its command line are shown in Figure 2.

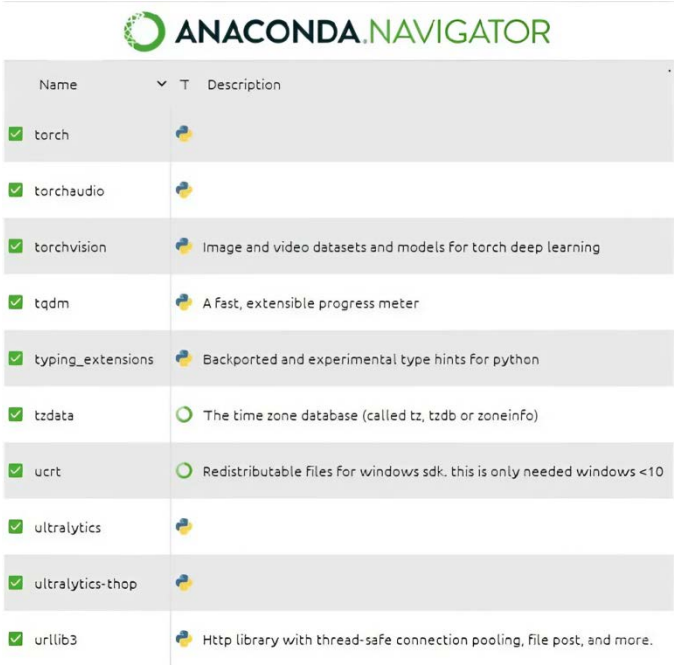


Fig. 2. Virtual environment.

The Ultralytics and PyTorch libraries are installed in the “yolov8_custom” virtual environment, in addition to them, numerous necessary libraries for training the model are also installed, which work simultaneously.

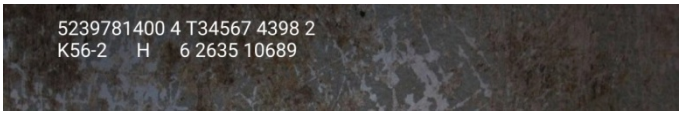
To train the model for the detection task, it is necessary to form a high-quality dataset of experiments without interfering with the production process. . Creating a dataset when conducting experiments directly on the production line is associated with a number of limitations, such as the need to install specialized cameras, the complexity of reproducing various situations when shooting conditions change - lighting, speed of the object, contrast and brightness of the marking. Carrying out installation work during the implementation of the pilot system requires multiple shutdowns of production, which is unacceptable. In this regard, it was decided to use a fairly simple digital twin, which creates conditions as close as possible to real ones.

The digital twin simulates the movement of sheets along the roller table, taking into account various parame-ters, such as

lighting, conveyor speed and the presence of defects. By specially distorting images, blurring them, and covering up inscriptions (simulating sheet cleaning in real conditions), it is possible to experiment with images to simulate real conditions and understand the operation of the model as a whole. Using images generated by the digital twin, an animation of sheet movement was created, on the basis of which the required dataset was formed and the model operation was tested. The appearance of the digital twin is shown in Figure 3. When creating a digital twin of a technological process, the key emphasis is not on an idealized model, but on the maximum approximation to real production conditions (Fig. 3a). Factors such as image sharpness variability, dynamic contrast, and uneven illumination are taken into account — parameters typical of an industrial environment. In practical implementation, the system is trained on data that is as close as possible to production conditions, including sharpness, contrast, and illumination; however, to demonstrate the conceptual basis of the solution, the article uses data reduced to a reference form (Fig. 3b). It is important to emphasize: all graphs and illustrations are of a demonstration nature and serve solely to visualize the methodology, while the working version of the solution is adapted to chaotic production scenarios

The LabelImg graphical tool is used to mark data. LabelImg is a graphical tool for annotating images that al-lows the user to manually mark bounding boxes around objects and assign classes to them. The annotation process via LabelImg typically includes the following steps: loading an image, creating a bounding box, assigning a class, and sav-ing annotations. An example of label placement in LabelImg is shown in Figure 4.

a)



b)

Fig. 3. Digital twin, a) real, b)prepared for article.

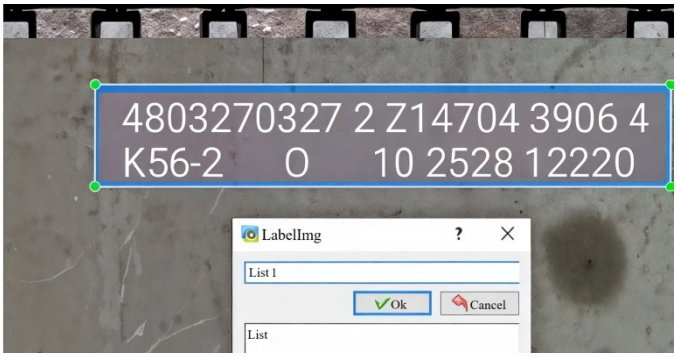


Fig. 4. Producing tag in LabelImg.

The class name chosen is List, which is also used in the python script to access the datasets. After all the labels have been placed and the classes have been specified, the coordinates of the frames and the classes of the objects are saved in text files, through which the model is provided with information about where the objects are in the image and what classes these objects belong to. Then we determine the size of the dataset and select the number of epochs for training.

The choice of the optimal combination of the number of images and the number of epochs depends on the specific conditions and goals of the project. Ideally, you want to strive for a balance that provides sufficient training without overfitting, given the constraints of time and resources. Here are some strategies for achieving this balance:

1. Applying regularization methods (e.g., Dropout, L1/L2 regularization) can help prevent overfitting when working with a small dataset.

2. Using data augmentation can effectively increase the size and diversity of the training set without requiring additional images.

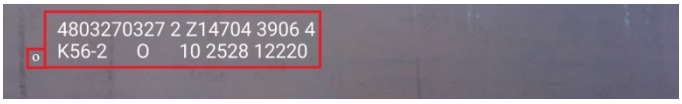
3. Monitoring the model performance on the validation dataset and stopping training when improvements are no longer observed helps avoid overfitting.

4. Using cross-validation to evaluate the model performance on different subsets of the data can help determine the best trade-off between the dataset size and the number of epochs.

It is worth noting that the Ultralytics library has a built-in early stopping feature when the model performance on the validation dataset stops improving over a specified number of epochs, which helps avoid overfitting.

After determining the size and number of epochs, the YOLOv8 model was retrained for the production task of real-time tile marking detection (Fig. 5a,b).

a)



b)

Fig. 5. Marking detection.

Since the recognition of the text itself is impossible within the production framework (due to various external factors), it was decided to make an imprint of the marking (similar to human biometrics) and identify the sheet based on it. Accordingly, a convolutional neural network was trained to identify signs of similarity and difference in pairs of images with sheet markings.

The Tensorflow and Keras libraries were used to train this neural network. TensorFlow is a powerful open-source library developed by Google that is used to create and execute computational graphs, especially in machine learning and deep learning tasks. It allows you to develop and train machine learning models. TensorFlow provides everything you need to

create complex models, including convolutional and recurrent neural networks.

Keras is a high-level library that runs on top of TensorFlow and provides a simple interface for creating and training neural networks. It is designed to quickly prototype and simplify working with deep models, especially for those without deep knowledge of low-level computation. Keras makes it easy to combine layers, activation functions, optimizers, and other components to create custom architectures.

TensorFlow provides a powerful infrastructure for computing, and Keras provides a convenient interface for building models. Together, they create an optimal ecosystem that is suitable for both research and industrial use. TensorFlow is responsible for performance, scalability, and compatibility with various devices, and Keras speeds up the development process with an intuitive API.

To train the convolutional neural network, as for the previous training, a dataset was compiled, all images were converted to a fixed size to bring them to a single format suitable for training the model. Based on the file name, a numeric label was extracted, which determines the class of the image. The data was also converted to NumPy arrays for further work with them in Keras.

Next, the model architecture was created and its hyperparameters were tuned for training.

Model architecture is the structure or design of a neural network that determines how input data is processed to obtain output results. It describes what components (layers) are used in the model, how these components are connected to each other, and how data flows through the model during the forward pass. The model architecture consists of: an input layer that defines the format and size of the input data, such as image dimensions (width, height, number of channels); hidden layers between the input and output layers that perform data processing; an output layer that is responsible for generating the final results; activation functions that introduce nonlinearity into the model so that it can learn complex relationships; architectural blocks. Some models use standard blocks, such as ResNet blocks or Inception modules, which consist of combinations of layers.

The architecture determines how the model will solve the problem, process data, and learn. It directly affects the efficiency of the model, its performance, ability to generalize data, and learning speed. The code for setting up the model architecture is shown in Figure 6.

```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
    input_shape=(140, 608, 5)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(30, activation='softmax')
])
```

Fig. 6. Setting up the model architecture.

Model hyperparameters are parameters that are manually set before training the model and determine its behavior. Unlike parameters (weights and biases), which the model learns itself, hyperparameters require tuning by the developer. Hyperparameters work together to ensure the efficiency and accuracy of the model, its ability to learn and successfully adapt to new data.

The hyperparameters were tuned accordingly, 20 training epochs were chosen during which the model was passed through the entire training dataset, the Adam optimizer was specified to update the model weights during train-ing, activation functions were used, namely ReLU for nonlinear transformations in hidden layers and Softmax to form a probability distribution at the output.

During the training process, convolutional layers gradually learn various image features at different levels, starting from lower-level features such as lines, angles, and textures, and ending with higher-level features such as shapes and objects. Each subsequent layer concentrates on more abstract and complex features. During the training process, the model adjusts the weights of its layers to represent important features of the images that allow predictions to be made about their class. After training, these features can be used to compare images and determine their similarities or differences, for example, by analyzing the activations of internal layers or by the distances between the feature vectors obtained at the output of the layers.

After completing the training of the model, a trained model of the convolutional neural network was obtained.

All created models were tested accordingly. For the trained YOLOv8 model, the predict function was used to check its performance and visually evaluate the results.

The convolutional neural network was tested using test datasets, which allowed us to evaluate its accuracy and generalization ability on previously unseen examples.

III. Test Results

In this paper, two models were implemented and tested: an object detection model based on YOLOv8 and a feature comparison model based on a convolutional neural network. Both models demonstrated high efficiency in solving the tasks, which is confirmed by the training and testing results.

Results of the object detection model. The test/box_loss, test/cls_loss and test/dfl_loss relations for the object detection model reflect the quality of its operation on test data that were not used in training. These metrics show how well the model copes with predicting the bounding box (box_loss), classifying objects (cls_loss) and, if distributed regression is used to predict coordinates, with the corresponding losses (dfl_loss) (Fig. 7). The loss value (loss function) is a dimensionless scalar indicator that quantitatively characterizes the degree of discrepancy between the values predicted by the model and the true target labels.

Analysis of the dynamics of the loss functions (test/box_loss, test/cls_loss, test/dfl_loss) demonstrates a stable tendency for their values to decrease with an increase in the

number of training epochs, which indicates correct optimization of the object detection model. A decrease in test/box_loss reflects a progressive improvement in the accuracy of object localization through regression of the coordinates of the bounding boxes. A decrease in test/cls_loss confirms an increase in the classification quality, expressed in a decrease in errors in determining the belonging of objects to target classes. If the model uses distributed regression (DFL), a stable drop in test/dfl_loss indicates the efficiency of the algorithm in predicting object positions through optimization of probability distributions. The observed simultaneous de-crease in all three loss components confirms the absence of signs of training divergence. For a more complete assess-ment of the generalizing ability of the model, it is also necessary to consider the dynamics of losses on the validation data: val/box_loss, val/cls_loss and val/dfl_loss (Fig. 8). These graph will help you assess how well the model adapts to data that was not used in training and identify possible signs of overfitting or insufficient generalization ability. Comparing test and validation losses will help you clarify how consistently the model improves its performance on new data.

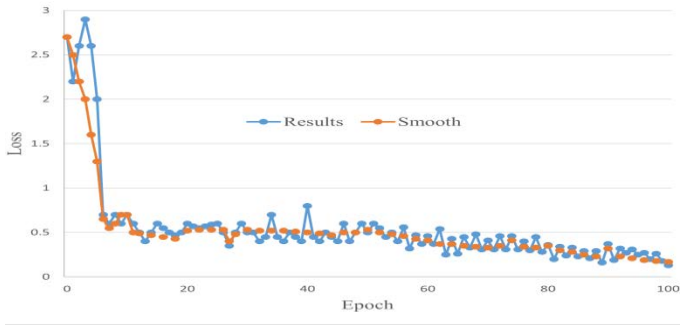


Fig. 7. Losses of the traning dataset.

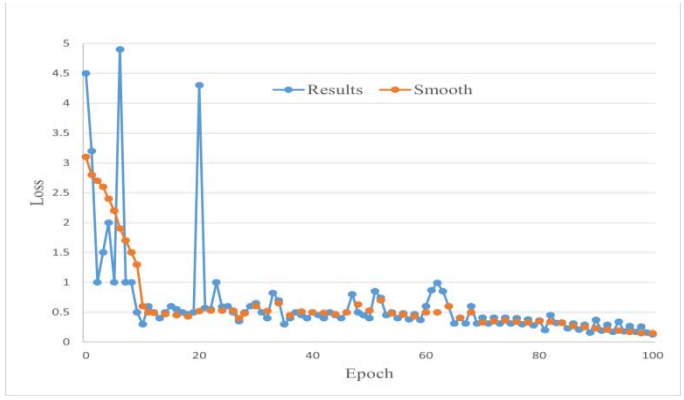


Fig. 8. Losses of the validation dataset

Analysis of the dynamics of the loss functions on the validation data (val/box_loss, val/cls_loss, val/dfl_loss) allows us to evaluate the generalization ability of the object detection model. A steady decrease in val/box_loss indicates a progressive improvement in the accuracy of object localization on data that was not involved in training, which confirms the ability of the model to correctly predict the coordinates of the bounding boxes. A decrease in val/cls_loss indicates an increase in the quality of object classification, demonstrating a decrease in errors in determining their belonging to the target classes. If the model uses distributed regression (DFL), a decrease in val/dfl_loss reflects an improvement in the accuracy of predicting object positions through optimization of probability distributions. A consistent decrease in all three loss components on the validation data confirms the absence of signs of overfitting and indicates a stable convergence of the training process. Thus, the analysis of the data loss graphs confirms the effectiveness of the selected architecture and training parameters of the YOLOv8 model. The model demonstrates high accuracy of localization, classification and processing of features, which makes it suitable for solving problems of dynamic identification of objects in metallurgical production.

Feature comparison model based on a convolutional neural network. The model accuracy graph (Fig. 9) shows a stable increase in accuracy throughout training, which indicates successful training of the model and its ability to correctly compare and contrast features. This is especially important for the problems of identifying markings on steel sheets, where it is necessary to accurately determine the similarities and differences between images.

The loss graph (Fig. 10) of the feature comparison model shows a gradual decrease in losses, which confirms the improvement in the quality of training and the ability of the model to minimize errors when comparing features. This allows the model to work effectively in conditions where high accuracy and reliability are required.

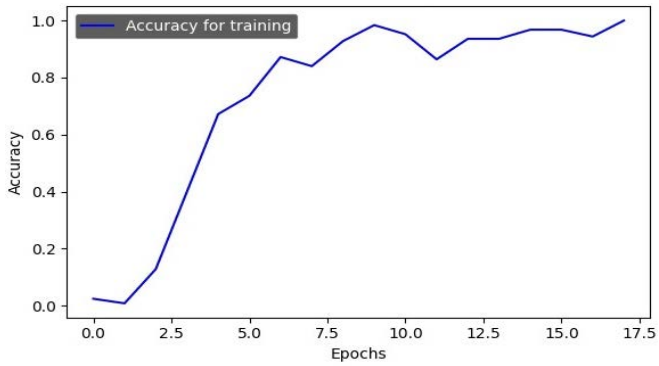


Fig. 9. Model accuracy.

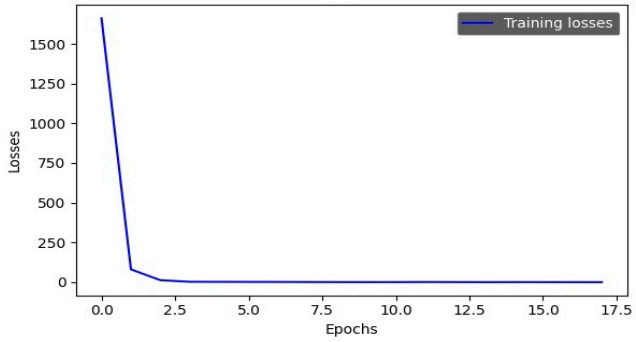


Fig. 10. Losses of the model.

To integrate trained models into production [15], it is necessary to perform several consecutive steps, starting with

setting up the software environment and ending with integrating the model into real production processes. First of all, it is necessary to prepare a working environment in Python, which includes installing all the necessary libraries and frameworks. To work with neural networks, it is recommended to use Anaconda, which provides a convenient package management system and creating isolated virtual environments. After that, you should install libraries such as PyTorch, OpenCV, NumPy, TensorFlow and other dependencies necessary for image processing, as well as for training and inference of the model. Then it is necessary to prepare a dataset including images of steel sheets with various defects and textures. For this, the LabelImg markup tool is used, which allows you to create annotations in formats compatible with YOLOv8, for example, in the TXT format for YOLO. The next step is training the neural network model. To do this, you need to prepare the data, including image augmentation to increase the diversity of the training set, which improves the model's ability to generalize in real-world conditions. The process of training a YOLOv8-based model requires tuning hyperparameters such as learning rate, number of epochs, and mini-batch size. After training the model, it should be tested on a set of lags to evaluate its accuracy, speed, and ability to detect objects in real time. To improve the quality of the model, it is important to integrate post-processing algorithms using OpenCV to correct shooting artifacts such as blur or glare. Once the model is ready and tested, it needs to be prepared for real-world conditions. This includes setting up hardware such as high-resolution cameras for image capture and GPUs for accelerated data processing. It is also important to integrate the system with adjacent systems via a REST API.

IV. Conclusions

The In the course of the work, modern technologies and tools such as PyTorch, Ultralytics, TensorFlow and Keras libraries were considered and applied, which ensured high performance and accuracy of the models. The use of a digital twin allowed testing in conditions close to real ones, which confirmed the operability of the developed algorithms.

To achieve this goal, the following stages were completed. First, a digital twin of the production line was de-veloped, simulating the conditions of sheet movement, lighting variations and background noise. Secondly, an aug-mented dataset of images created using a digital twin was formed. Thirdly, the YOLOv8 architecture was optimized for working with a resolution of 416 × 416 pixels, which ensured the detection of objects from 2 mm in size.

The scientific novelty of the work lies in the combination of methods aimed at overcoming the key limitations of existing solutions. Unlike previous studies that focused exclusively on the algorithmic component, the proposed approach integrates: deep convolutional networks (YOLOv8 with the CSPDarknet53 backbone) adapted for processing high-frequency video streams; dynamic data augmentation, including the generation of realistic defects via Generative Adversarial Networks (GAN); post-processing algorithms based on OpenCV for correcting shooting artifacts (blur, glare).

Thus, the conducted study demonstrates that the introduction of neural network technologies in metallurgical processes opens up broad prospects for improving productivity and control quality. The developed system is not only a prototype, but also a basis for further adaptation and expansion of functionality in real production conditions. The work was carried out taking into account modern approaches and technologies, which ensures its relevance and significance for enterprises of the metallurgical industry.

References

[1] A. V. Lednov, D. D. Prokhorov, and A. V. Shvaleva, “Identification of metal sheets in the flow, based on the marking imprint, using neural networks,” in Advances in Automation VI, vol. 1324, A. A. Radionov and V. R. Gasiyarov, Eds. Springer Nature Switzerland AG, 2024, pp. 85–95, doi: 10.1007/978-3-031-82494-4_9.

[2] “Video Analytics (Terms, Scopes of Application, Technologies),” TAdviser State Business IT. Accessed: Jun. 5, 2024. [Online]. Available: https://tadviser.com/index.php/

[3] “Artificial Intelligence (AI),” TAdviser State Business IT. Accessed: Jun. 5, 2024. [Online]. Available: https://tadviser.com/index.php/Product:Artificial_intelligence_(AI,_Artificial_intelligence,_AI)

[4] “The Most Unusual Tasks That Artificial Intelligence Can Solve,” National Association of Robotics Market Participants. Accessed: Jun. 5, 2024. [Online]. Available: https://robotunion.ru/glavnaya/tpost/t34ieblae1-samie-neobichnie-zadachi-kotorie-mozhet

[5] R. E. Shilov, O. S. Logunova, and A. V. Lednov, “Methods and algorithms for segmentation of the image in control flotation process,” 2018 International Russian Automation Conference (RusAutoCon), Sochi, Russia, 2018, pp. 1–4, doi: 10.1109/RUSAUTOCON.2018.8501678.

[6] B. N. Parsunkin, G. F. Obukhov, A. V. Lednov, et al., “Formation of testing signals to identify heat and power management objects,” Energy: Proceedings of the Universities, vol. 6, no. 6, pp. 65–70, 1988.

[7] NVIDIA, “NVIDIA Deep Learning Performance Guide,” NVIDIA Corporation. Accessed: Jun. 5, 2024. [Online]. Available: https://docs.nvidia.com/deeplearning/performance/

[8] “Anaconda: Operating System for Artificial Intelligence,” Anaconda, Inc. Accessed: Jun. 5, 2024. [Online]. Available: https://www.anaconda.com/

[9] “LabelImg: Image Markup Tool,” HumanSignal. Accessed: Jun. 5, 2024. [Online]. Available: https://github.com/HumanSignal/labelImg

[10] “PyTorch: Machine Learning Framework,” PyTorch Foundation. Accessed: Jun. 5, 2024. [Online]. Available: https://pytorch.org/

[11] “Complete YOLO v8 Custom Object Detection Tutorial Windows & Linux,” YouTube. Accessed: Jun. 5, 2024. [Online]. Available: https://www.youtube.com/watch?v=gRAyOPjQ9_s

[12] “Ultralytics: AI Platform for Building, Training, and Deploying Machine Learning Models,” Ultralytics. Accessed: Jun. 5, 2024. [Online]. Available: https://www.ultralytics.com/ru#

[13] “TensorFlow: Comprehensive Platform for Machine Learning,” TensorFlow. Accessed: Jun. 5, 2024. [Online]. Available: https://www.tensorflow.org/?hl=ru

[14] “Keras: Machine Learning Library,” Keras. Accessed: Jun. 5, 2024. [Online]. Available: https://keras.io/

[15] E. L. Itskovich, Methods for Rational Automation of Production, (in Russian). Infra, 2009, 255 p.