

Cuprins

1. Rezumat în limba engleză.....	3
1.1 State of the Art.....	3
1.1.1 Machine Learning	3
1.1.2 Deep Learning.....	3
1.1.3 Neural Networks	3
1.1.4 Layers of an Neural Network.....	4
1.2 Theoretical Fundamentals.....	4
1.2.1 YOLOv5 model.....	4
1.2.2 Faster R-CNN model.....	4
1.2.3 Arguments for using YOLOv5.....	4
1.2.4 Necessary notions and concepts for using YOLOv5.....	4
1.2.5 Data Input	4
1.2.6 Data Preprocessing.....	5
1.2.7 Forward Propagation	5
1.2.8 Backward Propagation.....	5
1.2.9 Training Process.....	5
1.2.10 Model Evaluation	5
1.2.11 Monitoring durring training	6
1.3 Implementation	6
1.3.1 Working environment.....	6
1.3.2 Training a YOLOv5 model.....	6
1.3.3 Model training and evaluation with <i>DatasetNr1</i>	6
1.3.4 Visualisation of the model's performance metrics	7
1.3.5 Model training and evaluation with <i>DatasetNr2</i>	7
1.3.6 Generating the LTspice netlist	8
1.4 Experimental results.....	8
1.4.1 Evaluation of the results for <i>DatasetNr1</i>	8
1.4.2 Evaluation of the results for <i>DatasetNr2</i>	9
1.4.3 The functionality of the <i>generate_netlist.py</i> script.....	9
1.5 Conclusions.....	10
2. Planificarea activității.....	11
3. Stadiul actual.....	12
3.1 Învățarea automată (Machine Learning – ML).....	12
3.2 Învățarea profundă (Deep Learning – DL)	13
3.3 Rețele neuronale (Neural Network – NN).....	13

3.4 Straturile unei rețele neuronale	14
4. Fundamentarea Teoretică.....	16
4.1 Modele de detecție a obiectelor cu arhitecturi CNN	16
4.1.2 Modelul YOLOv5	16
4.1.3 Modelul Faster R-CNN	18
4.2 Argumente pentru utilizarea YOLOv5.....	19
4.3 Noțiuni și concepe necesare pentru utilizarea YOLOv5	20
4.3.1 Introducerea datelor de intrare (Data Input)	20
4.3.2 Preprocesarea datelor (Data Preprocessing)	21
4.3.3 Propagarea înainte (Forward Propagation)	21
4.3.4 Propagarea înapoi (Backward Propagation)	22
4.3.5 Procesul de antrenare (Training Process)	24
4.3.6 Evaluarea modelului (Model Evaluation).....	24
4.3.7 Monitorizarea în timpul antrenării	26
5. Implementarea soluției adoptate	27
5.1 Mediul de lucru.....	27
5.2 Antrenare model YOLOv5	27
5.3 Antrenarea și evaluarea modelului cu setul de date <i>DatasetNr1</i>	29
5.3.1 Vizualizarea metricilor de performanță a modelului	31
5.4 Antrenarea și evaluarea modelului cu setul de date <i>DatasetNr2</i>	35
5.5 Generare netlist LTspice.....	37
6. Rezultate experimentale	42
6.1 Rezultatele antrenării modelului cu <i>DatasetNr1</i>	42
6.1.2 Evaluarea rezultatelor obținute pentru <i>DatasetNr1</i>	45
6.2 Rezultatele antrenării modelului cu <i>DatasetNr2</i>	46
6.2.1 Evaluarea rezultatelor obținute pentru <i>DatasetNr2</i>	50
6.3 Funcționalitatea scriptului <i>generate_netlist.py</i>	51
7. Concluzii.....	56
8. Bibliografie.....	57
9. Anexe	59
9.1 <i>train_yolov5.py</i>	59
9.2 <i>generate_netlist.py</i>	59
	61

1. Rezumat în limba engleză

1.1 State of the Art

Artificial intelligence (AI) in computer science focuses on creating systems and algorithms to perform tasks requiring human intelligence, such as learning, reasoning, and pattern recognition.

The work aims to develop an automated system for recognizing and converting electronic circuit components from images into files compatible with the LTSpice application. This involves training a YOLOv5 model for accurate detection of components in images and developing an algorithm to translate these detections into a format usable in LTSpice. The primary objective is to fully automate the generation of SPICE netlist files, reducing the time and effort required to transform graphical documentation into functional models.

1.1.1 Machine Learning

Machine Learning is a subfield of AI that involves developing algorithms that allow computers to learn from and make predictions or decisions based on data. Rather than being explicitly programmed to perform a task, a machine learning model learns patterns from data, enabling it to improve its performance over time. ML is used in various applications, such as recommendation systems, fraud detection, and predictive analytics.

1.1.2 Deep Learning

Deep Learning (DL) is a specialized subset of machine learning that focuses on using neural networks with many layers. These deep neural networks can model complex patterns in large datasets and are particularly effective for tasks such as image and speech recognition, natural language processing, and autonomous driving. DL models require large amounts of data and computational power to train effectively.

1.1.3 Neural Networks

Neural Networks (NN) are a core technology in deep learning and machine learning. Inspired by the human brain, NN consist of interconnected layers of nodes (also called neurons) that process data in a hierarchical manner. Each neuron takes input, processes it using a mathematical function, and passes the output to the next layer. NN are highly versatile and can be used for a wide range of tasks, including classification, regression, and pattern recognition.

The perceptron is the simplest type of neural network and serves as the foundation for more complex networks. It consists of a single layer of neurons and can be used for binary classification tasks. The perceptron receives multiple inputs, applies weights to each input, sums them, and passes the result through an activation function to produce an output. If the output is above a certain threshold, the perceptron classifies the input into one category; otherwise, it classifies it into another.

1.1.4 Layers of an Neural Network

Neural networks are composed of multiple layers, each with a specific function within the overall structure. The first layer, known as the input layer, is where the raw data is initially received. In this layer, each neuron corresponds to a specific feature of the input data. Following the input layer are the hidden layers, which are positioned between the input and output layers. These hidden layers are responsible for performing intermediate computations. Each hidden layer typically applies a nonlinear activation function to the data, enabling the network to capture and model more intricate patterns.

1.2 Theoretical Fundamentals

1.2.1 YOLOv5 model

YOLOv5 (You Only Look Once version 5) is part of the YOLO family, known for its speed and efficiency in detecting objects in real-time. YOLOv5 frames object detection as a single regression problem, directly predicting bounding boxes and class probabilities from images in a single network pass. This architecture makes YOLOv5 incredibly fast and suitable for real-time applications.

1.2.2 Faster R-CNN model

Faster R-CNN (Region-based Convolutional Neural Networks) is a two-stage object detection framework. It first generates region proposals using a Region Proposal Network (RPN) and then refines these proposals and classifies objects in the second stage. Faster R-CNN is known for its high accuracy but is generally slower than YOLO models due to its more complex architecture.

1.2.3 Arguments for using YOLOv5

1.2.4 Necessary notions and concepts for using YOLOv5

YOLOv5 is designed to be extremely fast, making it ideal for processing large datasets quickly. This is particularly advantageous when real-time detection is required or when working with large-scale datasets where training time is a critical factor. The single-stage architecture of this model allows for end-to-end training, simplifying the process and reducing the complexity compared to Faster R-CNN, which involves multiple stages.

1.2.5 Data Input

When working with the model, data input begins with providing the model with annotated images that include both the images themselves and labels specifying the objects' classes and bounding boxes. These annotations are essential as they guide the model in learning to identify and locate objects within images. The dataset is typically organized into three subdirectories: train, valid, and test. Before this data can be used for training, it undergoes preprocessing.

1.2.6 Data Preprocessing

Preprocessing includes resizing images to a consistent size, typically 640x640 pixels, to ensure uniformity across the dataset. Additionally, pixel values are normalized, often scaled to a range between 0 and 1, which helps stabilize the training process and speeds up learning. Data augmentation techniques such as flipping, scaling, rotation, and color adjustments are also applied to increase the diversity of the training data, which in turn enhances the model's ability to generalize to new, unseen data. During this step, the bounding boxes are also adjusted to ensure they correctly correspond to the objects after resizing.

1.2.7 Forward Propagation

In the forward propagation phase, the input image passes through the network layers, where various filters and operations are applied to detect features and predict bounding boxes and class probabilities. YOLOv5's architecture, composed of convolutional layers, works by extracting features from the input image, then applying these features to predict potential bounding boxes for objects. Each bounding box is associated with a class label and a confidence score, which indicates the likelihood that the detected object belongs to a certain class.

1.2.8 Backward Propagation

Backward propagation is the process by which model learns by adjusting its weights based on the error between the predicted outputs and the actual annotations. The loss function, which combines localization loss, classification loss, and confidence loss, is calculated to measure how far off the predictions are from the actual labels. Gradients of this loss function are then computed with respect to the model's parameters, indicating how the weights need to be adjusted to reduce the error. These weights are updated using an optimization algorithm like Stochastic Gradient Descent (SGD) or Adam, iterating through this process across many batches of data until the model's performance converges.

1.2.9 Training Process

The training process involves initializing the model's weights, often using pre-trained weights from a previous model to accelerate learning and improve performance. The training dataset is processed in batches, with the model refining its weights through forward and backward propagation in each batch. This process is repeated across multiple epochs, where the entire dataset is passed through the network several times to allow the model to learn effectively. After each epoch, the model's performance is evaluated on a separate validation dataset to ensure that it is not overfitting and to allow for any necessary adjustments to hyperparameters.

1.2.10 Model Evaluation

Evaluating the model involves testing its performance on a dataset it has not seen during training. The mean average precision (mAP) metric is commonly used to assess accuracy, considering both precision and recall across different intersection-over-union (IoU) thresholds.

Precision measures how many of the objects identified by the model are correctly detected, while recall measures how many of the actual objects in the image the model successfully identifies.

Recall is a key metric used to evaluate the performance of a model, particularly in classification and object detection tasks. It measures the proportion of actual positive instances that the model correctly identifies. In other words, recall quantifies the model's ability to detect all relevant instances in the data.

1.2.11 Monitoring during training

During training, it is important to monitor the model's performance to avoid underfitting and overfitting. Underfitting occurs when the model is too simple to capture patterns in the data, leading to poor performance on both training and validation sets. Overfitting arises when the model becomes overly complex, memorizing the training data and thus performing poorly on unseen data. To mitigate overfitting, Dropout and L2 regularization are commonly employed.

1.3 Implementation

1.3.1 Working environment

To set up the necessary dependencies for YOLOv5, the process begins with the installation of Python 3.8, ensuring compatibility with the required deep learning libraries. Following this, Anaconda is installed to facilitate the management of isolated environments, which is important for avoiding conflicts between different project dependencies. A dedicated virtual environment named *Yolov5Train_3.8* is then created using Anaconda, ensuring a clean and organized workspace for the project. With the environment in place, Git is installed to handle version control and enable the cloning of the YOLOv5 repository from GitHub. The next step involves installing all required Python dependencies specified in the requirements.txt file within the repository.

1.3.2 Training a YOLOv5 model

The project was initially created in PyCharm with the name *YoloV5Train*. A script named *train_yolov5.py* was developed to facilitate the training of the YOLOv5 model. This script was integrated with TensorBoard to allow real-time visualization of training metrics.

TensorBoard provides a comprehensive interface to monitor various performance indicators, such as loss, accuracy, and other relevant metrics during the training process. This integration was particularly useful for implementing early stopping techniques, which help to prevent overfitting by halting the training process when the model's performance on the validation set no longer improves.

1.3.3 Model training and evaluation with *DatasetNr1*

The dataset contains 1,284 images and is organized into three subdirectories: *train* - a subdirectory used for training the model, *valid* - a validation subdirectory used during training to prevent overfitting, and *test* - a subdirectory used to evaluate the model after training is successfully completed.

Each of these subdirectories contains an *images* folder with JPG images and a *labels* folder with TXT files that include information about the classes of electronic components and their bounding boxes. For training with *DatasetNr1*, real-time visualization of performance metrics was utilized by integrating TensorBoard and running the training process from the Anaconda Prompt.

1.3.4 Visualization of the model's performance metrics

The performance metrics of YOLOv5 trained on the *DatasetNr1* are represented in the Table 2.

Class	Images	Instance s	<i>precision</i>	<i>recall</i>	<i>mAP_0.5</i>	<i>mAP_0.5:0.95</i>
All	133	720	95.7%	82.8%	82.6%	58.0%
<i>ACV</i> (alternating voltage source)	133	55	94.8%	99.1%	97.6%	76.3%
<i>ARR</i> (arrow indicating voltage drop)	133	1	0.0%	0.0%	0.0%	0.0%
<i>C</i> (capacitor)	133	62	93.3%	96.8%	94.6%	55.4%
<i>I</i> (current source)	133	61	99.4%	99.4%	99.4%	80.1%
<i>L</i> (inductor)	133	71	99.7%	99.7%	99.5%	69.9%
<i>R</i> (resistor)	133	419	99.5%	99.5%	99.2%	67.0%

Tabel 2. Performance metrics of YOLOv5 trained on the *DatasetNr1*

1.3.5 Model training and evaluation with *DatasetNr2*

In the study, two datasets were used, with the first being unsuitable for the initial goal of generating NET files for LTspice from hand-sketched electronic circuit images. Consequently, the YOLOv5 model was also trained on *DatasetNr2*, which includes 2,703 images of hand-drawn electronic circuits. Like the first dataset, *DatasetNr2* is structured into train, valid, and test subdirectories for training, validation, and model evaluation. The performance metrics of YOLOv5 trained on the *DatasetNr1* are represented in the Table 4.

Class	Images (<i>valid</i>)	Instance s	<i>precision</i>	<i>recall</i>	<i>mAP_0.5</i>	<i>mAP_0.5:0.95</i>
All	99	385	96.1%	97.1%	96.5%	74.5%
<i>ac</i> (alternating voltage source)	99	44	95.0%	95.5%	96.0%	84.1%
<i>capacitor</i>	99	47	95.8%	97.4%	96.1%	60.8%
<i>dc</i> (direct voltage source)	99	35	96.5%	100%	99.5%	85.4%
<i>diode</i>	99	63	93.8%	95.8%	92.4%	73.4%
<i>inductor</i>	99	79	97.2%	97.5%	97.6%	72.4%
<i>resistor</i>	99	117	98.1%	96.6%	97.6%	70.8%

Table 4. Performance metrics of YOLOv5 trained on the *DatasetNr2*

The results from training with *DatasetNr1* and *DatasetNr2* were compared, and it was decided to use the model trained with *DatasetNr2* in the application, as it demonstrated better performance metrics. The final model utilized the *best.pt* file, which contains the neural weights obtained from the training with *DatasetNr2*.

1.3.6 Generating the LTspice netlist

In the project *YoloV5Train*, a script named *generate_netlist.py* was created to generate LTspice netlists for images included in *DatasetNr2*. Using the predictions from the YOLO model, functions for recognition, classification, and mapping of electronic components were integrated to generate a NET file compatible with LTspice. The script is fully functional and adheres to the netlisting rules of LTspice.

1.4 Experimental Results

The experimental results obtained from testing the YOLOv5 model for generating LTspice netlist files were presented using two distinct datasets, *DatasetNr1* and *DatasetNr2*. These results included various metrics and visualizations, such as confusion matrices, precision-confidence curves, recall-confidence curves, class distribution and bounding box analysis which provided insights into the model's performance in detecting and classifying electronic circuit components from images.

1.4.1 Evaluation of the results for *DatasetNr1*

The model trained on *DatasetNr1* demonstrated varying performance across different classes of electronic components. Overall, the model achieved an average precision of 95.7% and a recall of 82.8%, with a *mAP_0.5* of 82.6% and a *mAP_0.5:0.95* of 58.0%.

However, the model struggled with the *ARR* class (voltage drop indicator), where it failed to detect any instances, likely due to the limited representation of this class in the validation set. Despite this, the model showed excellent performance for other classes such as *ACV* (AC voltage source), *L* (inductor), and *R* (resistor), with near-perfect precision and recall, indicating its strong ability to accurately recognize components in diverse contexts. Overall, while the model shows promise, it is not fully optimized for the application due to its inconsistencies with certain classes. The confusion matrix for the model trained with *DatasetNr1* is represented in Figure 11.

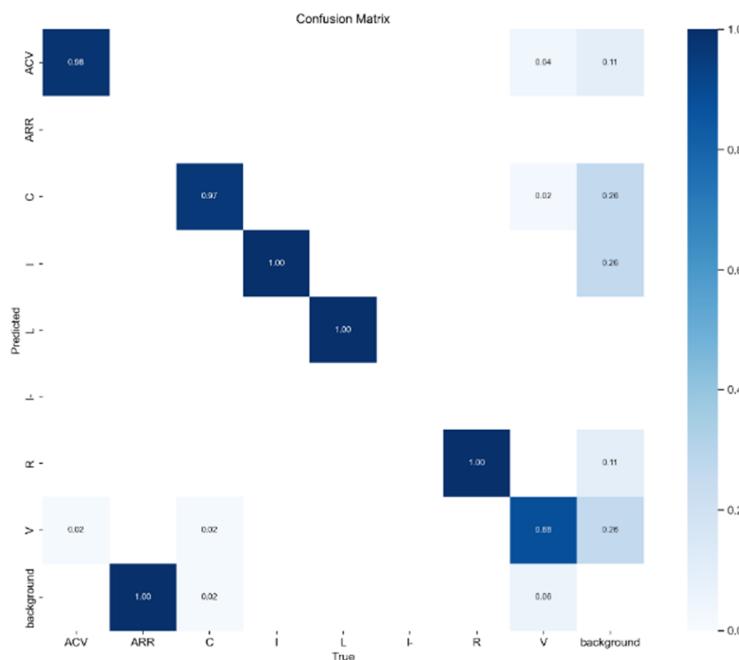


Figure 11. Confusion matrix for the model trained with *DatasetNr1*.

1.4.2 Evaluation of the results for *DatasetNr2*

The YOLOv5 model trained on *DatasetNr2* demonstrated strong performance, achieving an *mAP_0.5* of 96.5%, indicating high precision in detecting electronic circuit components. While most classes performed well, the diode class exhibited slightly lower accuracy, likely due to dataset limitations. Recommendations for improving the model include rebalancing the dataset, enhancing labeling, and incorporating SPICE model names and component specifications to further automate the application. The confusion matrix for the model trained with *DatasetNr2* is represented in Figure 17.

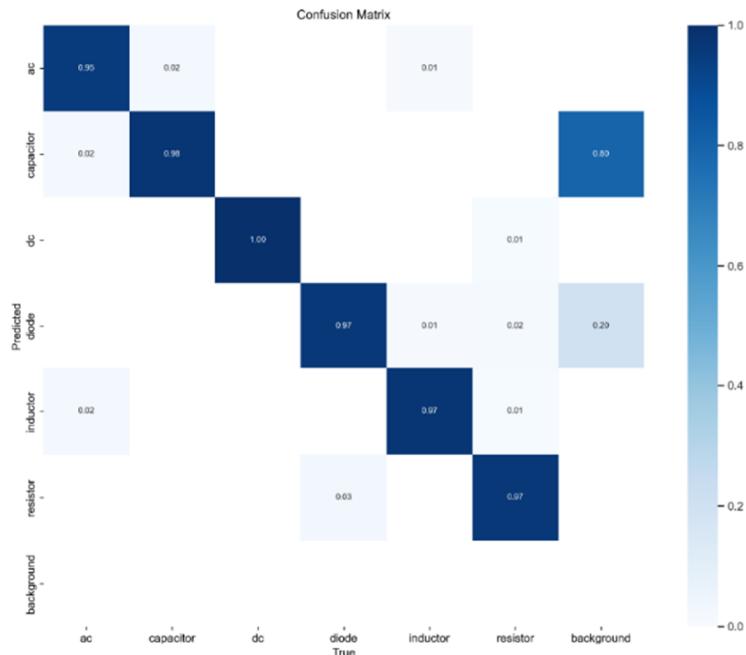


Figure 17. Confusion matrix for the model trained with *DatasetNr2*

1.4.3 The functionality of the *generate_netlist.py* script

To validate the LTspice netlist generation script, a set of test images from the validation subset of *DatasetNr2* was used. The script accurately maps electronic components detected by YOLOv5 into LTspice-compatible NET files, adhering to netlisting rules.

Additionally, the script includes functionality for assigning standard component values, enabling automatic DC operating point simulations in LTspice. This functionality was further enhanced by adding directives and specifying library paths to ensure accurate simulation results. An example of the functionality of the *generate_netlist.py* file can be seen in Figures 25. and 26.

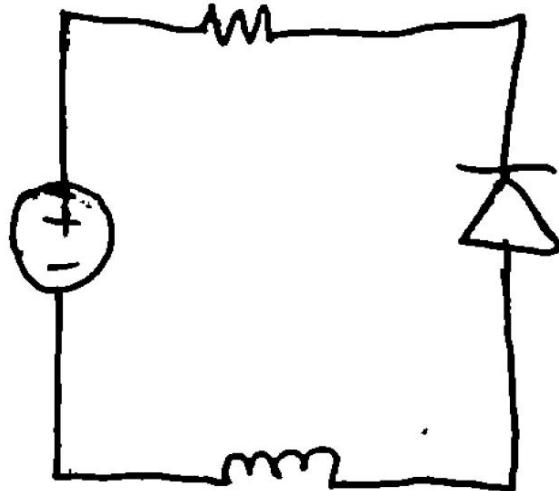


Figure 25. Image *Test2.png*

```
* C:\Users\Username\Desktop\Validation netlist\Draft1.asc
V1 N001 0 5
R1 N002 N001 1k
D1 N002 N003 D
L1 N003 0 10m
.model D D
.lib C:\Users\Username\Documents\LTspiceXVII\lib\cmp\standard.dio
.op
.backanno
.end

--- Operating Point ---

V(n001):      5          voltage
V(n002):      0.692893   voltage
V(n003):      4.30711e-006 voltage
I(D1):        0.00430711 device_current
I(L1):        0.00430711 device_current
I(R1):       -0.00430711 device_current
I(V1):       -0.00430711 device_current
```

Figure 26. Netlist file and *.op* simulation for image *Test2.png*

1.5 Conclusions

The work presents a solution for recognizing electronic components in circuits using CNNs, aiming to automate and speed up circuit design and simulation. The application generates netlist files directly from images, reducing manual effort. YOLOv5 was employed to identify components and their positions within circuits. The model was trained and evaluated using precision, recall, and mAP metrics, with Python used for implementation and LTspice for netlist generation.

The prototype struggles with distinguishing similar components and recognizing terminals, which impacts simulation realism and accuracy. Future improvements could include adapting the model for new circuit types, enhancing terminal recognition, and integrating detailed LTspice data for better netlist generation.

2. Planificarea activității

Nr.	Obiective planificate	Data începerii	Data finalizării
1	Alegerea temei de lucrare de licență	12/11/2023	16/11/2023
2	Documentare biliografică	16/11/2023	02/02/2024
3	Documentare despre elaborarea aplicației	02/02/2024	12/04/2024
4	Alegerea setului de date pentru antrenare	12/04/2024	4/05/2024
5	Alegerea modelului pentru implementarea aplicației	4/05/2024	11/06/2024
6	Antrenarea modelului	11/06/2024	14/06/2024
7	Crearea aplicatiei	14/06/2024	19/07/2024
8	Testarea aplicației	19/07/2024	28/07/2024
9	Colectarea rezultatelor experimentale	28/07/2024	03/08/2024
10	Redactarea implementării soluției adoptate	03/08/2024	14/08/2024
11	Finalizarea documentației	14/08/2024	25/08/2024

3. Stadiul actual

Într-un domeniu interdisciplinar al științei computerelor, inteligența artificială (IA) se ocupă de crearea de sisteme și algoritmi care pot îndeplini sarcini ce necesită inteligență umană, cum ar fi învățarea, raționamentul, procesarea limbajului natural, luarea deciziilor și recunoașterea tiparelor. Puterea computațională (Computing Power) în contextul unui sistem de IA reprezintă capacitatea efectuării calculelor complexe necesare pentru antrenarea și rularea modelelor de IA.

Aceasta include procesarea volumelor mari de date, realizarea de operații matematice avansate și optimizarea modelelor pentru a obține predicții precise [1].

Puterea de calcul este adesea exprimată în FLOPS (Floating Point Operations Per Second), deoarece antrenarea modelelor implică un număr foarte mare de operații în virgulă mobilă. În unele cazuri, sistemele de IA necesită TFLOPS (teraflops) sau chiar PFLOPS (petaflops) pentru antrenare [2]. În mod evident, puterea de calcul a unui sistem de IA este incomparabilă cu cea a unei persoane, astfel în procesele ce implică un volum mare de date, activitatea umană este treptat înlocuită cu utilizarea IA.

Scopul lucrării este dezvoltarea și implementarea unui sistem automatizat de recunoaștere și conversie a componentelor electronice din imagini de circuite în fișiere de tip SPICE netlist, compatibile cu aplicația LTSpice. Proiectul implică antrenarea unui model YOLOv5 pentru detectarea precisă a componentelor electronice din imagini și dezvoltarea unui algoritm eficient care să traducă aceste detectii într-un format utilizabil pentru vizionări de circuite în LTSpice.

Obiectivul principal fiind facilitarea procesului de proiectare și simulare a circuitelor electronice prin automatizarea completă a generării fișierelor netlist direct din imagini, reducând astfel timpul și efortul necesar inginerilor pentru a transforma documentația grafică în modele de simulare funcționale. Lucrarea de față va aborda aspectele tehnice ale antrenării modelului de detectie și integrarea acestuia cu un sistem de generare a fișierelor SPICE netlist.

3.1 Învățare automată (Machine Learning – ML)

Învățarea automată este un domeniu al inteligenței artificiale care dezvoltă algoritmi capabili să învețe și să facă predicții din date, adaptându-se și îmbunătățindu-și performanța pe baza experienței. Învățarea automată este împărțită în mod tradițional în trei categorii principale: învățarea supervizată, învățarea nesupervizată și învățarea prin consolidare. Învățarea supervizată (Supervised Learning) are la bază folosirea unui set de date etichetat, unde modelul pe parcursul învățării creează o funcție care mapează datele de intrare la ieșiri dorite. Învățarea nesupervizată (Unsupervised Learning) se caracterizează prin tipul datelor de intrare neetichetate, astfel modelul trebuie să descopere singur structuri sau tipare în date. Acest tip de învățare este puțin aplicabil pentru dezvoltarea proiectului de față, însă poate fi uzat pentru analiza clusterelor sau reducerea dimensiunilor de date. Învățare prin consolidare (Reinforcement Learning) implică existența unui agent care interacționează cu un mediu și învăță să ia decizii optime folosind metoda trial and error, maximizând o recompensă cumulată. Principalele scenarii în care se aplică acest tip de învățare sunt cele în care există un set de acțiuni și rezultate, dar nu este aplicabil în mod direct pentru detectarea obiectelor.

3.2 Învățare profundă (Deep Learning – DL)

Învățarea profundă (DL) este alt domeniu al inteligenței artificiale, care se concentrează pe arhitecturi și algoritmi inspirate de rețelele neuronale biologice. Descoperirea modelelor computaționale capabile să învețe reprezentări complexe și ierarhice ale datelor prin intermediul mai multor straturi de procesare este scopul principal al învățării profunde. Aceste straturi ascunse sunt cunoscute și sub numele de straturi ascunse. Modelul poate extrage și reprezenta caracteristici la diferite niveluri de abstractizare datorită acestor straturi. Învățarea profundă se bazează pe utilizarea unor structuri ce sunt numite rețele neuronale.

3.3 Rețele neuronale (Neural Network – NN)

Rețelele neuronale artificiale sunt modele matematice inspirate din structura și funcționarea creierului uman, având ca unități fundamentale neuronii artificiali (Nodes/Units) [3].

Un neuron artificial primește mai multe semnale de intrare, le procesează printr-o funcție de activare și generează un semnal de ieșire. Primul și unele din cele mai simple modele de neuron artificial este perceptronul, introdus de Frank Rosenblatt în 1958.

Perceptronul este un model clasificator liniar, care funcționează prin combinarea liniară a intrărilor ponderate și aplicarea unei funcții de activare pentru a lua decizii asupra clasei de ieșire. Datele de intrare a unui perceptron sunt numite intrări (Inputs) (x_1, x_2, \dots, x_n), fiecare intrare fiind asociată cu un coeficient de pondere (w_1, w_2, \dots, w_{nj}). Intrările sunt măritate cu o pondere (Weights), care reflectă importanța intrării respective pentru predicția finală. Toate inputurile ponderate sunt adunate pentru a produce un semnal de activare. Acest semnal este numit funcție de transfer (Σ) (Transfer Function) (1), iar matematic procesul poate fi exprimat astfel:

$$Z = \sum_{n=1}^k w_{nj} \cdot x_n + b \quad (1)$$

Unde: b este bias-ul, termen adăugat pentru facilitarea adaptării modelului la date. Funcția de activare (φ) (Activation Function) decide dacă neuronul se activează sau nu, bazându-se pe valoarea lui Z . În perceptronul simplu, funcția de activare (2) este de obicei o funcție treaptă (Step Function).

$$f(Z) = \begin{cases} 1, & \text{pentru } Z \geq \theta_j \\ 0, & \text{pentru } Z < \theta_j \end{cases} \quad (2)$$

Pragul de activare (θ_j) (Activation Threshold) este valoarea la care se decide dacă neuronul se activează sau nu, astfel influențând ieșirea. Ieșirea (o_j) (3) este valoarea finală calculată după aplicarea funcției de activare:

$$o_j = f(Z) \quad (3)$$

Perceptronul, structura cătuia se regăsește în Figura 1., poate clasifica datele liniar separabile, adică datele care pot fi separate de o hiperplană într-un spațiu de dimensiuni n . Deși perceptronul este limitat la clasificarea liniar separabilă, el a constituit baza dezvoltării ulterioare a rețelelor neuronale artificiale. Prin adăugarea unor straturi suplimentare de perceptri, s-a reușit crearea rețelelor neuronale multilayer perceptron (MLP), care sunt capabile să rezolve probleme mai complexe, inclusiv cele non-liniare. Fiind precursorul rețelelor neuronale mai complexe,

perceptronul a fost esențial pentru dezvoltarea inițială a inteligenței artificiale. Acesta a introdus ideea de ajustare a ponderilor în funcție de eroarea comisă, un mecanism esențial pentru învățarea supravegheată, pe lângă capacitatea sa de a clasifica datele liniar separabile. Acest proces, numit regula de învățare sau regula de ajustare, implică ajustarea greutăților astfel încât să reducă eroarea de clasificare. În cazul unui perceptron simplu, procesul de învățare este formulat (4) [4]:

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = \eta \cdot (y - o_j) \cdot x_i \quad (4)$$

Unde: η - rata de învățare, y - ieșirea dorită, o_j - ieșirea perceptronului.

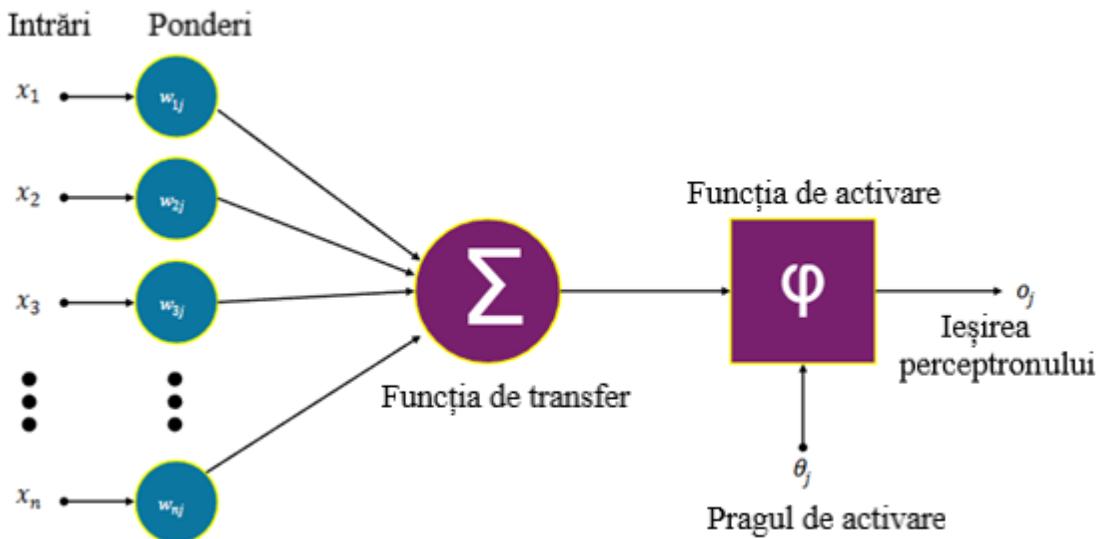


Figura 1. Structura perceptronului

3.4 Straturile unei rețele neuronale

Stratul de intrare (Input Layer) este responsabil pentru colectarea datelor brute din surse externe. Fiecare neuron din stratul de intrare este asociat cu un anumit tip de date de intrare. De exemplu, un neuron în stratul de intrare al unui sistem de recunoaștere a imaginii poate reprezenta fiecare pixel al imaginii. Prin intermediul ponderilor care sunt legate de fiecare conexiune, datele sunt transferate la nivelul următor [5].

Straturile ascunse (Hidden Layers) se află între stratul de intrare și cel de ieșire. Denumirea specifică alor provine de la faptul nu pot fi văzute în mod direct sau găsite din afara rețelei. Fiecare strat ascuns transformă datele cu o transformare liniară sau neliniară. Capturarea complexității și a relațiilor non-liniare din date necesită prezența straturilor ascunse. Rețeaua este adâncită de mai multe straturi ascunse, ceea ce le permite să învețe reprezentări mai complexe și abstrakte ale datelor de intrare. Un număr mare de straturi ascunse poate permite rețelei să învețe modele complexe, dar poate duce și la probleme precum supraînvățarea (overfitting) [6].

Rezultatul final al modelului este generat de stratul de ieșire (Output Layer), care este ultimul strat al rețelei neuronale. În cazul apariției problemelelor de clasificare, fiecarui neuron o să-i corespundă o clasă de ieșire, iar în cazul apariției problemelor de regresie neuronilor vor corespunde valori numerice. Activarea neuronilor din stratul respectiv este determinată de activările din straturile anterioare, modulate de funcțiile de activare și ponderi.

Metode de optimizare precum algoritmul de învățare pe bază de gradient (Gradient Descent) sunt utilizate pentru a ajusta ponderile în timpul antrenării rețelei. În scopul minimizării erorii, procesul de actualizare a ponderilor este esențial pentru ca rețeaua să învețe din datele de intrare și să facă predicții cât mai precise.

Rețelele neuronale pot capta și învăța relațiile complexe dintre intrări și ieșiri, datorită acestor funcții de activare, care la rândul său introduc non-liniaritatea în model. Funcția sigmoidă, funcția ReLU (Rectified Linear Unit) și funcția tangenta hiperbolică sunt exemple comune de funcții de activare [7]. În Figura 2., poate fi examinată o rețea neuronală artificială cu straturile descrise anterior, unde fiecare conexiune între neuroni este ponderată și influențează rezultatul final.

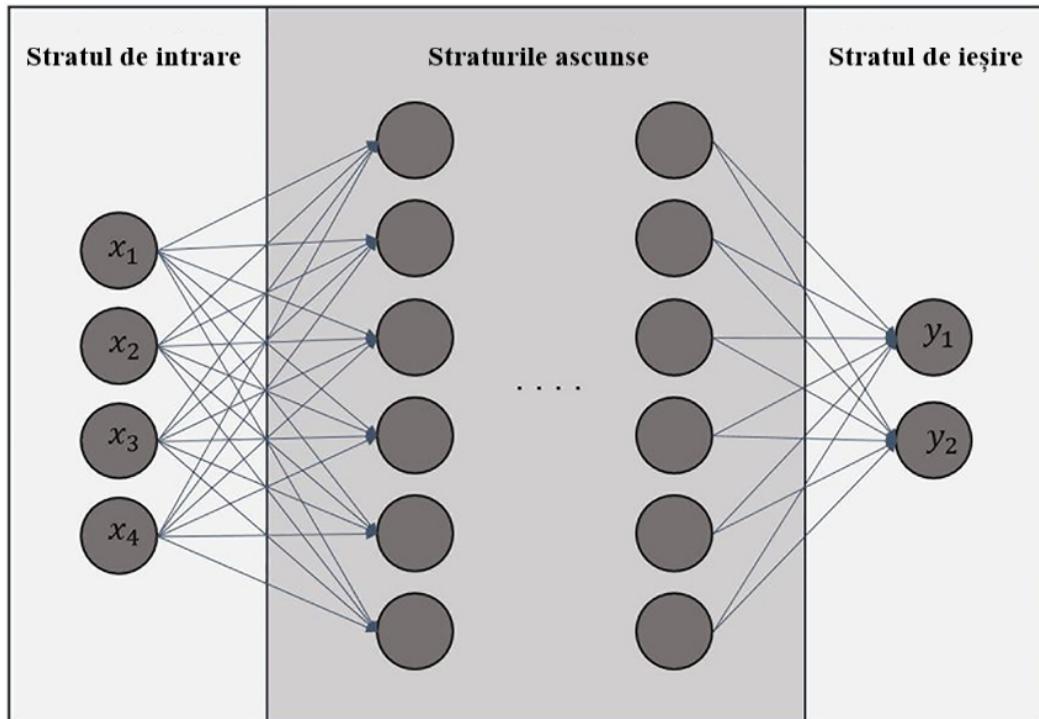


Figura 2. Straturile unei rețele neuronale

4. Fundamentare Teoretica

Pentru a lua decizia în vederea alegerii unui model potrivit și efectiv lucrării respective, avem nevoie de informații despre tipurile existente ale rețelelor neuronale și ale aplicațiilor în care acestea sunt folosite.

Rețele neuronale convoluționale (Convolutional Neural Networks - CNNs) sunt un tip specific de rețele neuronale, acestea fiind destinate analizei datelor care au structuri de rețea locală, cum ar fi imaginile. Operațiile de conoluție le permit să extragă automat caracteristici din imagini prin aplicarea unor filtre la diferite subsecțiuni ale acesteia.

Aplicații: Procesare de limbaj natural (NLP), traducere automată, recunoaștere de vorbire, modelare secvențială [3].

Rețele neuronale recurente (Recurrent Neural Networks - RNNs) sunt folosite în special pentru date secvențiale, în care ordinea datelor este importantă, cum ar fi textul sau seria temporală. Ele au conexiuni ciclice care le permit să păstreze o “memorie” a intrărilor anterioare.

Aplicații: Crearea conținutului audio sau video, augmentarea datelor, generare de imagini realiste [8].

Rețele de memorie pe termen lung scurt (Long Short-Term Memory Networks - LSTMs) sunt un tip de RNN care pot învăța să păstreze informații relevante pentru o perioadă lungă de timp. Acest lucru rezolvă problema gradientului evanescent. Când vine vorba de captarea dependentelor pe termen lung în datele secvențiale, acestea demonstrează eficiență.

Aplicații: Compoziție muzicală automată, traducere de text, recunoaștere a vorbirii [9].

O altă categorie de arhitecturi cunoscută sub numele de rețele neuronale de tip transformer a revoluționat procesarea secvențială, în special în procesarea limbajului natural. Ele procesează datele în paralel folosind mecanisme de atenție (attention mechanisms). Caracteristica dată le face extrem de eficiente pentru sarcini de traducere automată, sumarizare și alte aplicații în NLP (Natural Language Processing).

Aplicații: Analiza sentimentului, Modele de limbaj avansate (GPT, BERT, etc.), traducere automată [10].

Autoencodere (Autoencoders) sunt rețele neuronale care sunt utilizate pentru învățarea neasistată a reprezentărilor eficiente de date. Ele sunt alcătuite dintr-un encoder, care comprimă datele într-un spațiu latent și un decoder care reconstruiește datele din această reprezentare.

Aplicații: Denoising, generarea de date, detectarea anomaliei, Reducerea dimensionalității [11].

Având în vedere scopul proiectului respectiv care este detecția componentelor electronice din imagini, am decis să folosesc un model ce utilizează o arhitectură de rețele convoluționale neuronale, în următoarele subcapitole sunt descrise câteva variante de modele open source bazate pe CNN.

4.1 Modele de detecție a obiectelor cu arhitecturi CNN

4.1.2 Modelul YOLOv5

YOLOv5 (You Only Look Once version 5) utilizează o arhitectură de rețele neuronale convoluționale pentru a realiza detectarea obiectelor într-o imagine. Modelul YOLOv5 este bazat pe o arhitectură de tip Single-Stage Object Detector, caracteristica principală a acestei arhitecturi fiind realizarea detecției obiectelor într-o singură etapă, combinând atât localizarea obiectelor (determinarea bounding box-urilor) cât și clasificarea lor (determinarea clasei la care aparține obiectul) într-un singur pas, fără a avea nevoie de o etapă intermedieră, cum ar fi propunerea de regiuni de interes [12]. Un bounding box este o cutie de delimitare, un dreptunghi care încadrează un obiect detectat într-o imagine, reprezentând zona specifică unde se află obiectul în imagine.

Acest dreptunghi este definit prin coordonatele colțului său superior stânga și ale colțului său inferior dreapta. Clasele obiectelor fiind parametri a setului de date, acestea definesc categoriile obiectelor pe care modelul a fost antrenat să le recunoască și să le clasifice. Clasele funcționează ca etichete pentru obiectele care sunt încadrate în setul de date.

YOLOv5 a fost lansat pentru prima dată în anul 2020 de către organizația Ultralytics. Modelul folosește o rețea neuronală convezională profundă (Deep Convolutional Neural Network - DCNN) care este compusă dintr-un Backbone, un Neck și un Head.

Backbone-ul se definește ca partea principală a rețelei, are rolul de extragere a caracteristicilor principale din imagine, în această parte au loc majoritatea operațiilor de procesare a imaginii. YOLOv5 utilizează arhitectura CSPDarknet53 ca Backbone, care este o variantă a Darknet53 optimizată pentru a reduce numărul de parametri, menținând în același timp performanța ridicată. CSP (Cross Stage Partial) este o metodă care împarte rețeaua în două fluxuri de date pentru a reduce complexitatea și a îmbunătăți eficiența de calcul, păstrând în același timp puterea de reprezentare [13].

Neck-ul în YOLOv5 utilizează PANet (Path Aggregation Network) ce este o rețea de agregare a căii, care combină informațiile extrase la diferite niveluri de la backbone pentru a îmbunătăți performanța în detectarea obiectelor la diferite scări. Neck-ul ajută la fuziunea caracteristicilor extrase din straturile diferite ale backbone-ului, permitând modelului să detecteze obiecte cu diferite dimensiuni și niveluri de granularitate. PANet asigură generalizarea modelului în diverse situații prin eficiența detecției a obiectelor mici, cât și pentru cele mari [14].

Head-ul este partea finală a modelului, cea care efectuează predicția finală a cutiilor de delimitare (bounding boxes) și a etichetelor claselor pentru fiecare obiect detectat în imagine. Aceasta utilizează informațiile procesate de părțile anterioare ale modelului pentru a realiza predicții precise privind locația și clasa obiectelor din imagine [15].

YOLOv5 a fost antrenat pe diverse seturi de date, printre cele mai utilizate fiind COCO (Common Objects in Context), care conține aproximativ 330.000 de imagini, dintre care peste 200.000 sunt anotate cu mai mult de 1,5 milioane de instanțe de obiecte. Aceste imagini acoperă 80 de clase de obiecte. Setul de date COCO este unul dintre cele mai cuprinzătoare și utilizate dataseturi pentru detectarea obiectelor, oferind un mix divers de contexte și obiecte [16]. Antrenarea pe COCO permite modelului YOLOv5 să fie robust în detectarea unei varietăți de obiecte și scenarii din viața reală. Pentru funcționare, modelul are nevoie de utilitare, precum biblioteci Python.

Pascal VOC (Visual Object Class) a fost unul dintre primele seturi de date pe care modelele YOLO au fost antrenate și evaluate. Versiunile anterioare ale YOLO, cum ar fi YOLOv1 și YOLOv2, au fost testate extensiv pe Pascal VOC, iar rezultatele obținute au demonstrat capacitatea modelului de a combina precizia ridicată cu viteza mare de inferență. Pascal VOC include 20 de clase de obiecte.

O bibliotecă Python este definită ca un set organizat de module și funcții predefinite care sunt disponibile pentru utilizare în limbajul de programare Python. Aceste biblioteci oferă funcționalități specializate, care pot fi utilizate pentru a îndeplini sarcini specifice, economisind timp și efort. Ele sunt esențiale pentru folosirea eficientă a modelelor de IA, permitând dezvoltatorilor să se concentreze pe implementarea funcționalităților specifice aplicației lor.

YOLOv5 folosește o serie de biblioteci Python care acopera o gamă largă de funcționalități, începând de la procesarea imaginilor și calculul numeric, până la gestionarea datelor și vizualizarea rezultatelor. Următoarea listă detaliază principalele biblioteci utilizate de modelul respectiv:

PyTorch: Este biblioteca principală folosită pentru crearea și antrenarea rețelelor neuronale. PyTorch oferă suport pentru calcule tensoriale și învățare profundă, fiind baza pe care este construit YOLOv5.

Torchvision: O extensie a PyTorch, utilizată pentru sarcini de viziune computațională. Include funcții pentru manipularea imaginilor și modele pre-antrenate.

OpenCV: O bibliotecă robustă pentru procesarea imaginilor și viziune artificială. Este utilizată pentru citirea, manipularea și vizualizarea imaginilor în YOLOv5.

Numpy: Biblioteca fundamentală pentru calcul numeric în Python, utilizată pentru manipularea și analiza datelor numerice.

Pandas: O bibliotecă pentru manipularea și analiza datelor, care oferă structuri de date rapide și flexibile. YOLOv5 folosește Pandas pentru gestionarea seturilor de date.

Matplotlib: Utilizată pentru vizualizarea datelor, Matplotlib este esențială în YOLOv5 pentru crearea de grafice și diagrame care ilustrează performanțele modelului.

Seaborn: O bibliotecă bazată pe Matplotlib, utilizată pentru vizualizarea statisticilor și crearea de diagrame mai complexe.

Requests: Utilizată pentru realizarea de cereri HTTP în scopul descărcării modelelor pre-antrenate și a altor resurse.

Tqdm: O bibliotecă folosită pentru a afișa bare de progres în timpul execuției codului, utilă pentru monitorizarea progresului în timpul antrenării modelului.

YAML: Folosită pentru încărcarea și manipularea fișierelor de configurare în format YAML, care sunt utilizate pentru a defini parametrii și setările modelului YOLOv5 [17][18][19].

YOLOv5 este un model open-source, ceea ce înseamnă că oricine îl poate folosi, modifica și răspândi gratuit. Această accesibilitate face parte dintr-o mișcare mai largă în domeniul inteligenței artificiale care promovează inovația și colaborarea prin partajarea și distribuirea liberă a codului sursă.

Modelul YOLOv5 este disponibil pe GitHub, o platformă recunoscută pentru dezvoltarea și distribuirea software-ului open-source [20]. O altă caracteristică a modelului este abordarea sa prietenoasă pentru utilizator, atât din perspectiva dezvoltatorilor experimentați, cât și a celor care abia încep să exploreze domeniul computer vision. Modelul facilitează integrarea rapidă în diverse proiecte prin documentație bine structurată și exemple clare. Stilul său modular și ușor de înțeles permite personalizarea codului fără a face modificări complicate. Parametrii de antrenament pot fi ușor ajustați de utilizatori sau pot fi adăugate noi funcționalități [21]. Instalarea și utilizarea YOLOv5 sunt ușurate datorită integrării cu PyTorch și faptului că este disponibil ca repozitoriu pe GitHub.

Utilizatorii trebuie doar să cloneze repository-ul folosind limbajul de scripting și instrumentul Powershell, apoi rămâne instalarea dependențelor necesare pentru folosirea modelului. Acest proces simplu permite o configurare rapidă și elimină obstacolele tehnice [22]. Modelul suportă năiv mai multe formate de date, prin intermediul fișierelor de configurare de tip YAML, permite ajustarea simplă a hiperparametrilor. Această flexibilitate este esențială pentru utilizatorii care experimentează cu diferite arhitecturi de rețea neuronală sau doresc să adapteze modelul la seturi de date specifice [23].

Modelul poate fi folosit ca bază pentru soluții de recunoaștere a obiectelor noi sau poate fi integrat ușor în aplicații existente. YOLOv5 permite exportarea modelului în mai multe formate, cum ar fi ONNX și CoreML, ceea ce determină implementarea să ușoară și eficientă pe diferite platforme [24].

4.1.3 Modelul Faster R-CNN

Faster R-CNN (Region-based Convolutional Neural Networks) este un model de detecție a obiectelor introdus în 2015 de Shaoqing Ren, Kaiming He, Ross B. Girshick și Jian Sun. Predecesorii modelului respectiv aveau integrat algoritmul Selective Search, scopul principal al acestuia fiind crearea unui set de zone posibile (bounding boxes) ce sunt susceptibile să conțină obiecte.

Modelul respectiv folosește metoda Two-Stage Detection, aceasta implică două etape în procesul de detectare a obiectelor din imagini. Prima etapă, numită generarea regiunilor de interes (Regional Proposal Stage), este caracterizată de utilizarea sub-rețelei Region Proposal Network

(RPN) rolul căreia este generarea unui set de propunerii de regiuni (region proposals) în urma analizei imaginii, de asemenea sub-rețea produce pentru fiecare zonă un bounding box și un scor de probabilitate, care indică prezența obiectului în zona analizată.

A doua etapă, poartă denumirea de clasificare și reglare finală a regiunilor (Region Classification and Refinement Stage), etapa se definește prin extragerea propunerilor de regiuni generate în etapa precedentă de RPN și transmiterea acestora către o rețea neuronală complet conectată (Fully Connected Network).

Rețeaua complet conectată este responsabilă pentru clasificarea fiecărei regiuni propuse în una din clasele de obiecte existente și ajustarea precisă a coordonatelor bounding box-urilor [25]. Faster R-CNN reprezintă o îmbunătățire semnificativă a versiunilor anterioare R-CNN și Fast R-CNN, datorită introducerii conceptului de Region Proposal Network (RPN), care permite generarea dinamică a propunerilor de regiuni, eliminând necesitatea utilizării algoritmului Selective Search [26].

Modelul are o arhitectură asemănătoare cu cea implementată în YOLOv5, fiind compus din trei părți principale: Backbone, RPN și Head. Backbone-ul este reprezentat de rețeaua neuronală convezională care extrage caracteristici din imagine. Rețelele populare utilizate pentru acest scop includ VGG16, ResNet50, și ResNet101, acestea la rândul lor fiind pre-antrenate pe seturi de date mari precum ImageNet.

Backbone-ul modelului generează o hartă de caracteristici care reține informații spațiale esențiale pentru detecția obiectelor. RPN-ul este o componentă inovatoare care generează propunerii de regiuni direct din harta de caracteristici furnizată de Backbone. Rețeaua respectivă modifică coordonatele casetelor de delimitare (bounding boxes) după ce parcurge harta de caracteristici și determină dacă un obiect se află în regiunile propuse. RPN-ul este antrenat cu restul rețelei, făcându-l foarte eficient și integrat.

Head-ul este partea rețelei de clasificare a regiunilor propuse și de ajustare a casetelor de delimitare (bounding boxes). Arhitectura Head-ului include de obicei straturi complet conectate care generează probabilitățile de clasă și ajustările casetelor de delimitare pentru fiecare regiune propusă de RPN [27]. Faster R-CNN a fost antrenat pe mai multe seturi de date populare, precum:

PASCAL VOC: Set de date ce include imagini cu obiecte annotate în 20 de categorii. Este un benchmark standard pentru modelele de detecție a obiectelor.

COCO: Dataset deja descris anterior, ce oferă anotări complexe cu un număr mare de imagini și categorii de obiecte, fiind provocator pentru detecția obiectelor.

ImageNet: Deși este utilizat în principal pentru clasificarea imaginilor, anumite versiuni ale setului de date respectiv includ anotări cu casete de delimitare care pot fi utilizate pentru antrenarea modelelor de detecție a obiectelor.

Faster R-CNN suportă diverse formate de fișiere pentru seturi de date, inclusiv:

VOC XML: Utilizat în dataseturile PASCAL VOC.

COCO JSON: Formatul utilizat de datasetul COCO pentru anotări.

CSV: Uneori utilizat în dataseturi personalizate [28].

Dependențele și bibliotecile folosite de model:

PyTorch sau TensorFlow: Aceste framework-uri de învățare profundă sunt utilizate pentru implementarea și antrenarea modelelor Faster R-CNN.

OpenCV: Folosit pentru sarcini de procesare a imaginilor.

NumPy: O bibliotecă fundamentală pentru operații numerice în Python.

Matplotlib: Utilizată pentru vizualizarea rezultatelor antrenării [26][27].

4.2 Argumente pentru utilizarea YOLOv5

În urma analizei modelelor descrise anterior, am ales utilizarea modelului YOLOv5 în cadrul acestui proiect.

Primul motiv al acestei decizii este viteza de procesare ridicată și latența scăzută a modelului respectiv [30]. Aplicația dezvoltată în aceasta lucrare necesită antrenarea modelului folosind un set de date ce conține peste 1000 de imagini, în cazul de față viteza de procesare este o caracteristică importantă. Deși modelul Faster R-CNN este foarte precis, arhitectura Two-Stage Detection îl face considerabil mai lent ca YOLOv5, fapt pentru care utilizarea acestuia este o variantă nepotrivită pentru proiectul respectiv. Cu toate că există versiuni mai noi ale YOLO, precum YOLOv7 sau YOLOv8, versiunea 5 este bine maturizată, stabilă, și extrem de bine documentată, ceea ce ușurează implementarea și integrarea în proiectul prezentat în lucrare. Versiunea 5 și în prezent este competitivă cu modele lansate recent în ceea ce privește precizia medie ponderată (Mean Average Precision - mAP) și alte metriki de performanță. Un alt punct forte al utilizării modelului respectiv este suportul pentru hardware variat, modelul dat fiind optimizat pe o gamă largă de GPU-uri.

Antrenarea modelului în lucrarea respectivă este realizată folosind un GPU de model NVIDIA GeForce GTX 1050 TI, un model vechi lansat în octombrie 2016 care beneficiază de compatibilitatea cu YOLOv5 folosind platforma CUDA (Compute Unified Device Architecture) [30]. CUDA se definește ca un suport software și un model de programare dezvoltat de NVIDIA care permite utilizarea unităților de procesare grafică pentru calcule generale. Cu alte cuvinte, CUDA permite dezvoltatorilor să utilizeze puterea considerabilă de calcul a GPU-urilor pentru a accelera operațiuni care în mod tradițional ar fi fost executate pe procesoarele unităților [32].

Antrenarea modelelor de IA implică execuția paralelă a calculelor complexe, cum ar fi multiplicarea matricilor și operațiuni necesare pentru propagarea înainte și înapoi (forward and backward propagation). YOLOv5 implică operațiuni intensive din punct de vedere computațional, cum ar fi convoluțiile, care sunt ideale pentru execuția paralelă, iar CUDA oferă posibilitatea de execuție a acestor operațiuni pe mii de nuclee GPU, accelerând semnificativ procesul de antrenare comparativ cu un CPU. De asemenea, framework-urile necesare în implementarea proiectului precum TensorFlow și PyTorch sunt complet compatibile cu CUDA, ceea ce permite integrarea fără probleme a resurselor GPU-ului în fluxul antrenării și inferenței modelului [33].

4.3 Noțiuni și concepte necesare pentru utilizarea YOLOv5

4.3.1 Introducerea datelor de intrare (Data Input)

Setul de date (Dataset) este un ansamblu de date structurate organizat, utilizat în antrenarea și evaluarea modelelor de IA. În cazul proiectului dezvoltat, dataset-ul conține imagini și etichete asociate fiecărei imagini de circuite electronice, fiecare circuit conține la rândul lui componente electronice precum condensatoare, diode, surse de tensiune, rezistoare, bobine și altele. Setul de date este organizat în mod tipic în trei directoare principale:

- Antrenare (Train): Conține imaginile și etichetele asociate, informațiile extrase din acestea sunt utilizate pentru antrenarea modelului.
- Validare (Validation): Conține un set de imagini și etichete utilizate pentru a valida performanța modelului în timpul antrenării, cea ce previne învățarea excesivă a detaliilor (Overfitting). Overfitting-ul apare când modelul învață prea bine detaliile și zgromotul din setul de antrenament, acesta reprezintă un obstacol major în antrenarea modelelor de învățare profundă, deoarece compromite capacitatea modelului de a face predicții corecte pe date noi [34].

- Testare (Test): Este un director separat pentru testare, care conține imagini și etichete folosite doar pentru evaluarea finală a modelului, după ce antrenarea a fost realizată cu succes.

Fiecare director principal al setului de date conține două subdirectoare, unul de imagini (Images) și celălalt de etichete (Labels), pentru stabilirea corespondenței între imagine și etichetă. Denumirea etichetei este echivalentă cu cea a imaginii. Cel mai des formatul imaginilor este JPEG sau PNG. YOLOv5 utilizează etichete de format TXT, aceste fișiere conțin informații despre bounding boxes și clasele obiectelor ce se regăsesc în imaginile corespunzătoare.

Structura unui fișier de etichete fiind alcătuită astfel: *class_id x_center y_center width height*, fiecare linie corespunde unui obiect din imagine, *class_id* - id-ul clasei căreia îi aparține obiectul (spre exemplu, 0 pentru condensator, 1 pentru rezistor, 2 pentru sursă de tensiune, etc.), *x_center y_center* - coordonatele centrului bounding box-ului, normalizate în raport cu dimensiunile imaginii, iar *width height* reprezintă lățimea și înălțimea bounding box-ului, similar celorlalți parametri normalizate în raport cu dimensiunile imaginii. Coordonatele bounding boxes, lățimea și înălțimea sunt exprimate în valori float între 0 și 1, astfel permitând modelului generalizarea la imagini de dimensiuni diferite [35].

4.3.2 Preprocesarea datelor (Data Preprocessing)

Pregătirea imaginilor pentru utilizarea acestora de către model este definită ca preprocesarea datelor. Aceasta include redimensionarea imaginilor la dimensiunea cerută de model (cum ar fi 640x640 pixeli), normalizarea valorilor pixelilor și augmentarea datelor, care poate include operațiuni precum rotații, schimbări de scală și flipări orizontale și verticale pentru a îmbunătăți robustețea modelului. Modelul YOLOv5 știe să aplique automat augmentarea datelor pentru a crea variații suplimentare ale imaginilor de antrenament, astfel contribuind la prevenirea învățării excesive a detaliilor și majorând diversitatea datelor [36].

4.3.3 Propagarea înainte (Forward Propagation)

Fiecare strat al rețelei neuronale efectuează calcule și transformări asupra datelor și le transmite stratului următor până când se ajunge la stratul final, care generează predicțiile modelului, acest proces poartă denumirea de propagare înainte. Transformările includ operații precum:

- Convoluțiile ce extrag informații și caracteristici locale din imagini, cum ar fi forme, marginile și texturile.
- Funcții de activare, cum ar fi ReLU (Rectified Linear Unit), au ca scop modificarea datelor după ce acestea trec prin operații de conlovuție. ReLU introduce non-linearități în rețea, permitând înțelegerea relațiilor mai complexe dintre date.
- Pooling, straturile de pooling (spre exemplu max-pooling) reduc dimensiunea datelor, agregând informațiile importante și reducând complexitatea calculului.

În YOLOv5, primele straturi pot extrage caracteristici simple, cum ar fi linii și colțuri, în timp ce straturile mai profunde pot recunoaște structuri mai complexe, cum ar fi contururile întregilor componente electronice, ceea ce permite combinarea și agregarea caracteristicilor extrase la diferite niveluri. Stratul final al rețelei este caracterizat de producerea predicțiilor finale, precum coordonatele care delimită poziția obiectelor detectate, probabilitățile asociate fiecărei clase

posibile și probabilitățile că un bounding box conține un obiect ce nu este încurcat cu un zgromot sau eroare (Confidence Score) [37].

4.3.4 Propagarea înapoi (Backward Propagation)

O tehnică esențială pentru antrenarea rețelelor neuronale este propagarea înapoi, cunoscută și sub numele de backpropagation. Aceasta reprezintă procesul prin care rețeaua ajustează biasurile și ponderile fiecărui neuron pentru a reduce eroarea dintre predicțiile create și valorile reale (ground truth). În esență, backward propagation permite modelului să-și îmbunătățească performanța pe măsură ce face greșeli. Metoda respectivă include următoarele etape:

- Calcularea erorii la ieșire: O predicție a ieșirii este obținută după procesarea unei date de intrare prin propagare înainte. Această predicție este comparată cu valoarea reală (ground truth), iar diferența dintre ele este calculată folosind o funcție de pierdere (Loss Function), cum ar fi o eroare medie pătratică (Mean Squared Error) sau cross-entropy. Funcția de pierdere are ca scop cuantificarea diferenței între predicția modelului și valoarea reală. În YOLOv5, funcția de pierdere este compusă din trei componente. Prima funcție se numește pierdere pentru bounding boxes (5) (Complete Intersection over Union - CIoU Loss), având următoarea formulă:

$$L_{LCIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v \quad (5)$$

unde: IoU - suprapunerea dintre box-uri (Intersection Over Union), $\rho(b, b^{gt})$ - distanța euclidiană dintre centrele bounding box-ului prezis și a celui real, c - diagonala bounding box-ului minim care le încadrează pe ambele, α - factor de echilibrare, v - măsoară consistența raportului de aspect între box-uri. Următoarea funcție poartă denumirea de pierdere pentru clasificare (6) (Classification Loss), YOLO folosește o funcție de pierdere bazată pe entropie încruțiată binară (Binary Cross-Entropy - BCE), deoarece fiecare bounding box poate fi asociat cu o probabilitate pentru fiecare clasă existentă. Relația pentru pierderea de clasificare fiind:

$$L_{cls} = - \sum_{i=1}^C [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (6)$$

unde, C – numărul de clase, y_i - eticheta reală (0 sau 1) pentru clasa i , \hat{y}_i - probabilitatea prezisă pentru clasa i .

Cea de-a treia componentă a funcției de pierderi se numește pierdere pentru obiectivitate (7) (Objectness Loss), funcția respectivă măsoară cât de sigur este modelul că un anumit bounding box conține efectiv un obiect. Este de asemenea bazată pe BCE. Pierderea pentru obiectivitate se definește cu următoarea relație:

$$L_{obj} = -[t \log(p) + (1 - t) \log(1 - p)] \quad (7)$$

unde: t - eticheta obiectivului (1 dacă box-ul conține un obiect, 0 altfel), p – probabilitatea prezisă că boxul conține un obiect. Funcția de pierdere totală (8) (Total Loss Function), este suma ponderată a tutror funcțiilor de pierdere:

$$L_{total} = \lambda_{box} \cdot L_{CIOU} + \lambda_{cls} \cdot L_{cls} + \lambda_{obj} \cdot L_{obj} \quad (8)$$

unde: λ_{box} , λ_{cls} , λ_{obj} - sunt hiperparametri care controlează influența fiecărei componente asupra pierderii totale. Această relație complexă permănd modelului simultan să învețe localizarea obiectului cu precizie, clasificarea lor corectă și luarea deciziei în privința existenței obiectului într-o anumită zonă a imaginii [36].

- Gradientul funcției de pierdere totală (9) este necesar pentru ajustarea corectă a parametrilor de pondere - w_{nj} și bias - b_j . Gradientul se calculează folosind regula lanțului din calcul diferențial, aceasta la rândul său permite propagarea gradientului înapoi prin rețea. Gradientul funcției de pierdere totală (8) față de o pondere - w_{nj} este dat de relația:

$$\frac{\partial L}{\partial w_{nj}} = \frac{\partial L}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{nj}} \quad (9)$$

unde: z_j - activarea netă a neuronului respectiv în stratul j , dată de (10):

$$z_j = \sum_{n=1}^k w_{nj} \cdot x_n + b_j \quad (10)$$

unde: x_n - activarea neuronului din stratul anterior, iar $\partial L / \partial w_{nj} = x_n$, deoarece activarea netă depinde liniar de w_{nj} .

Gradientul funcției de pierdere totală (10), poate fi detaliat prin următoarea relație (11):

$$\frac{\partial L}{\partial w_{nj}} = \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{nj}} \quad (11)$$

unde: $\partial L / \partial y_j$ - gradientul funcției de pierdere față de ieșirea y_j a neuronului j , $\partial y_j / \partial z_j$ - derivata funcției de activare (spre exemplu ReLU, sigmoidă, treaptă (2)) în raport cu activarea netă z_j , $\partial z_j / \partial w_{nj} = x_n$. Pentru fiecare strat de rețea, relațiile descrise anterior se repetă, permănd astfel propagarea gradientului înapoi prin straturi. Gradientul față de biasul b_j (12), se calculează similar, fiind definit astfel:

$$\frac{\partial L}{\partial b_j} = \frac{\partial L}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_j} \quad (12)$$

unde: $\partial z_j / \partial b_j = 1$, deoarece biasul b_j este adăugat direct la activarea netă z_j . Diferența dintre valoarea prezisă de neuronul j și valoarea țintă (ground truth) y_j^{true} este indicată de derivata funcției MSE (Mean Squared Error) (13), funcția fiind dată de următoarea relație:

$$L = \frac{1}{2} \sum_j (y_j - y_j^{true})^2 \quad (13)$$

Derivata funcției MSE (14) în raport cu ieșirea y_j fiind:

$$\frac{\partial L}{\partial y_j} = y_j - y_j^{true} \quad (14)$$

După ce au fost calculate toate gradiențele pentru fiecare dintre valorile parametrilor de pondere și bias, parametrii respectivi sunt actualizați folosind un algoritm de optimizare Stochastic Gradient Descent (SGD) (15):

$$w_{nj}^{nou} = w_{nj}^{vechi} - \eta \cdot \frac{\partial L}{\partial w_{nj}}$$

$$b_j^{nou} = b_j^{vechi} - \eta \cdot \frac{\partial L}{\partial b_j} \quad (15)$$

unde: η - rata de învățare care are rolul de control a mărimei ajustării. Propagarea gradientului actualizat începând de la stratul de ieșire și terminând cu cel de intrare permite fiecărui strat să-și ajusteze parametrii în funcție de impactul său asupra erorii totale [34].

4.3.5 Procesul de antrenare (Training Process)

Antrenarea modelului YOLOv5 inițial începe cu setarea hiperparametrilor. Hiperparametrii includ dimensiunea batch-ului (batch size), rata de învățare (learning rate) și numărul de epoci (epochs). Rata de învățare controlează magnitudinea ajustărilor greutăților la fiecare pas de antrenare. Dimensiunea batch-ului determină numărul de imagini utilizate pentru calculul gradientului în fiecare etapă. Setarea hiperparametrilor este un pas important pentru obținerea unei performanțe optime a modelului. Următorul pas include rularea modelului pe un set de date pe multiple epoci (epochs), fiecare epocă reprezentând o trecere completă prin setul de date. Procesul antrenării determină modelul să învețe să-și minimizeze funcția de pierdere prin ajustări iterative ale ponderilor neuronale, îmbunătățind astfel precizia predicțiilor [3].

4.3.6 Evaluarea modelului (Model Evaluation)

Evaluarea YOLOv5 se realizează folosind date independente, în cazul proiectului respectiv se va realiza pe datele din subdirectorul *test* al datasetului pentru a verifica capacitatele de generalizare a modelului. Pentru evaluarea modelului se folosesc mai multe metrii de performanță. Precizia (precision) (16) este raportul dintre numărul de predicții corecte pozitive și numarul total de predicții pozitive (greșite și corecte):

$$precision = \frac{TP}{TP+TF} \quad (16)$$

unde: TP – predicții corecte pentru obiectele detectate (True Positives), TF - predicții greșite, cazuri când modelul detectează un obiect inexistent (True Negatives) [34].

Recall-ul (17) este o altă metrică care măsoară capacitatea YOLOv5 de a detecta toate obiectele prezente în imagine, cu alte cuvinte proporția între predicțiile corecte pozitive TP și

totalul obiectelor reale din imagine [3]:

$$Recall = \frac{TP}{FN+TP} \quad (17)$$

FN fiind numărul de obiecte reale din imagini ce nu au fost detectate de model (False Negatives). F1-Score (18) este definită ca media armonică între precizie și recall, aceasta fiind utilă în cazurile în care este importantă menținerea echilibrului între identificarea corectă a obiectelor și evitarea predicțiilor false [3]:

$$F1 = 2 \cdot \frac{precision \cdot Recall}{precision + Recall} \quad (18)$$

Media preciziei pentru toate clasele (Mean Average Precision – mAP) (19) este una dintre cele mai importante metriki pentru evaluare modelelor de detecție a obiectelor din imagini, se măsoară la diferite praguri a suprapunerii între bounding box-uri (Intersection over Union - IoU) și este dată de relația [40]:

$$mAP = \frac{1}{C} \sum_{c=1}^C AP_c \quad (19)$$

unde: C – numărul total de clase, AP – aria sub curba precizie-recall la diferite praguri de IoU.

Suprapunerea dintre bounding box-ul prezis și cel real (ground truth) este definită de metrica IoU (20), aceasta fiind esențială pentru a determina dacă o predicție este considerată corectă sau greșită în funcție de pragul specificat (de obicei setat la 0.5 sau 0.75):

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (20)$$

unde: *Area of Overlap* - aria comună între box-ul prezis și cel real, *Area of Union* - suma ariilor celor 2 box-uri minus aria de suprapunere.

Curba precizie-recall este un grafic ce reprezintă relația între metricile respective la diferite praguri de clasificare, oferind o înțelegere clară a performanței YOLOv5 la diverse setări de sensibilitate. Axa x fiind de fapt recall, iar axa y de precizie [40].

Un alt instrument de evaluare a modelului este matricea de confuzie (Confusion Matrix), aceasta afișează informații despre predicțiile pentru fiecare clasă oferite de model:

- Numărul de obiecte (entități pe care modelul este antrenat să le detecteze și să le clasifice) corect identificate (True Positives - TP);
- Numărul de non-obiecte (orice altă regiune a imaginii care nu conține entități de interes) corect identificate ca fiind absente (True Negatives - TN);
- Numărul de non-obiecte incorect identificate ca obiecte (False Positives – FP);
- Numărul de obiecte reale care nu au fost detectate de model (False Negatives – FN) [34].

4.3.7 Monitorizarea în timpul antrenării

Pe parcursul antrenării YOLOv5, se urmăresc diferite metriki pentru a preveni supraînvățarea (overfitting) și underfitting-ul. Underfitting-ul se descrie ca fenomenul când modelul nu este capabil să captureze relațiile complexe din date și, prin urmare, performează slab atât pe setul de antrenament, cât și pe setul de testare, cauzele aparenței problemei respective pot fi: arhitectura prea simplă a modelului, antrenament insuficient (au fost setate prea puține epoci), regularizarea excesivă, date insuficiente pentru antrenament sau care conțin erori. Metrica de monitorizare care oferă informații pe parcursul antrenării se numește curba de învățare (Learning Curve), aceasta ilustrând pe un grafic performanțele de antrenament în funcție de epoci de antrenare.

Eroarea de antrenament reprezintă eroare calculată după fiecare epocă parcursă de model. Eroarea de validare reprezintă eroarea calculată pe setul de date de validare. În condiții ideale, eroarea de validare scade împreună cu eroarea de antrenament. În caz contrar, este semnalizat overfitting-ul modelului [34]. Pentru prevenirea overfitting-ului se utilizează tehnica de oprire anticipată (Early Stopping), mecanismul acesta fiind descris de monitorizarea performanțelor modelului pe setul de validare și oprirea antrenării atunci când nu se mai observă îmbunătățiri semnificative a erorilor pe parcursul la un anumit număr de epoci consecutive. Tehnica oferă beneficiul de prevenire a memorării zgmotului sau datelor irelevante din setul de antrenament [34]. De asemenea, pe baza monitorizării continue pot fi luate decizii de ajustare fină (Fine-Tuning).

Dacă se observă o stagnare sau o scădere bruscă a performanțelor modelului, rata de învățare poate fi micșorată, acest fapt permite ajustarea fină a ponderilor în etapele finale ale antrenării. În cazul în care se detectează overfitting-ul, se pot adăuga tehnici de regulazare suplimentară. Una din aceste tehnici se numește Dropout și funcționează prin oprirea aleatorie a unor neuroni în timpul antrenării (cu alte cuvinte setarea temporală a ieșirilor lor la 0), tehnica respectivă împiedică modelul să devină dependent de anumiți neuroni și încurajează rețeaua să dezvolte reprezentări robuste și generalizabile ale datelor[41]. O altă tehnică de ajustare fină este penalizarea normei L2 (21) (L2 Regularisation/ridge regression), de asemenea folosită pentru prevenirea overfitting-ului.

Metoda respectivă funcționează prin adăugarea unei penalizări la funcția de pierdere proporțională cu suma pătratelor ponderilor modelului, având ca efect deplasarea valorilor ponderilor către zero fără a le seta valoarea efectivă la zero, ceea ce reduce complexitatea modelului. Funcția de pierdere ajustată de tehnica L2 Regularisation este dată de relația:

$$L_{total} = L_{original} + \lambda \sum w_{nj}^2 \quad (21)$$

unde: L_{total} - funcția de pierderi totală, $L_{original}$ - funcția de pierderi originală, λ - hiperparametru care controlează importanța termenului de penalizare, w_{nj} - ponderile rețelei neuronale[42].

O opțiune pentru salvarea periodică a setărilor modelului pe parcursul antrenamentului este tehnica Checkpointing, folosită pentru a putea reveni la o versiune anterioară a modelului în caz de necesitate. După ce procesul antrenării s-a sfârșit, modelul care a obținut cele mai bune performanțe pe setul de date de validare este considerat versiunea finală care poate fi utilizată pentru testare și aplicații[43].

5. Implementarea soluției adoptate

5.1 Mediul de lucru

Primul pas pentru pregătirea modelului de antrenament, este instalarea Anaconda, aceasta fiind o distribuție a limbajului de Programare Python, care include numeroase pachete și biblioteci utile pentru învățarea profundă. Un alt punct forte al Anaconda este gestionarea ușoară a mediilor virtuale. După descarcarea și instalarea Anaconda de pe site-ul oficial [43], urmează crearea mediului virtual în Anaconda.

Un mediu virtual este un este un spațiu izolat în care pot fi instalate specific pachetele necesare unui anumit proiect, fără a afecta alte proiecte Python.

Pentru crearea și activarea mediului virtual, se accesează linia de comandă *Anaconda Prompt* din meniul *Start*. se tastează comenziile specifice:

```
conda create -n Yolov5Train_3.8 python=3.8  
conda activate Yolov5Train_3.8
```

Unde: *Yolov5Train_3.8* este numele mediului virtual, *python=3.8* este versiunea specifică proiectului. Versiunea 3.8 a fost aleasă datorită compatibilității cu CUDA și suportului pentru bibliotecile esențiale Python (cum ar fi NumPy, Pandas, SciPy, TensorFlow, PyTorch).

Următorul pas este crearea și salvarea proiectului PyCharm, versiunea folosită în acest proiect este *PyCharm 2024.1.4* fiind descarcată de pe site-ul oficial [44]. În mod evident se selectează ca interpretor Python mediul virtual creat anterior *Yolov5Train_3.8*.

Este necesară instalarea CUDA pentru folosirea resurselor GPU la antrenarea modelului, aplicația fiind disponibilă pe site-ul oficial NVIDIA [45]. CUDA are nevoie și de o librerie cuDNN (CUDA Deep Neural Network library) pentru funcționarea corectă, aceasta fiind de asemenea disponibilă pe site-ul oficial NVIDIA [46].

YOLOv5 este un model open-source disponibil pe GitHub [20], astfel descărcarea codului necesar pentru a rula YOLO este simplificată la executarea următoarei comenzi:

```
git clone https://github.com/ultralytics/yolov5
```

Apoi se navighează în directorul *yolov5* pentru a descărca lista de pachete și dependențe necesare modelului, pentru instalarea acestora este utilizat comanda *pip*.

```
cd yolov5  
yolov5>pip install -r requirements.txt
```

Verificarea instalării pachetelor necesare se face cu comanda *pip list*, dacă *Anaconda Prompt* nu returnează erori, toate dependențele au fost instalate cu succes.

5.2 Antrenare model YOLOv5

În mod evident este necesară crearea scriptului de antrenare, pentru asta se accesează din meniul din stânga a aplicației PyCharm denumirea proiectului *YoloV5Train*, se alege opțiunea ‘*New>Python File*’ și se salvează scriptul cu denumirea *train_yolov5.py*.

Scriptul trebuie să conțină: calea către directorul unde a fost descărcat modelul YOLO, calea către fișierul de configurare YAML a setului de date și comanda de rulare a antrenării.

```

yolov5_dir = "C:/Users/Username/PycharmProjects/Yolov5Train/yolov5"
yaml_path = "E:/DatasetNr1/data.yaml"
os.system(f"python {yolov5_dir}/train.py --img 640 --batch 16 --epochs 100 --data {yaml_path} - -weights yolov5s.pt")

```

- *import os* - permite scriptului să ruleze comenzi externe din PowerShell;
- *yolov5_dir* – este o variabilă careia î se atribuie calea către directorul cu modelul;
- *yaml_path* – variabila care conține calea către fișierul YAML care specifică configurația a setului de date;
- *os_system(...)* - este o funcție ce execută o comandă în PowerShell, totul ce se află înăuntrul parantezelor funcției se trimit ca o comandă sistemului de operare;
- *python {yolov5_dir}/train.py* - rulează scriptul *train.py* din directorul YOLOv5 folosind interpretul *Yolov5Train_3.8* ;
- *--img 640* – specifică dimensiunea imaginilor în pixeli (pentru setul de date respectiv 640x640 pixeli);
- *--batch 16* – numărul de imagini procesate simultan de model într-un singur pas de antrenament (dimensiunea batch-ului);
- *--epochs 100* – Numărul de epoci de antrenament, în cazul proiectului respectiv modelul va trece întregul set de 100 de ori;
- *--data {yaml_path}* - conține calea către setul de date *DatasetNr1*;
- *--weights yolov5s.pt* - conține calea către fișierul de ponderi pre-antrenate, *yolov5s.pt* este un model preantrenat utilizat ca punct de plecare pentru fine-tuning.

Pentru a putea vizualiza metricile de evaluare a antrenamentului în timp real, este nevoie de instalarea instrumentului *tensorboard*. Se accesează *Anaconda Prompt*, se accesează mediul virtual *Yolov5Train_3.8*, iar apoi se introduce comanda:

```
pip install tensorboard
```

După ce instrumentul a fost instalat cu succes, urmează introducerea unor schimbări în fișierul *train_yolov5.py* pentru inițializarea *tensorboard*.

```

From torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter(log_dir=os.path.join(yolov5_dir, 'runs', 'train'))
writer.close()

```

- *from torch.utils.tensorboard import SummaryWriter* - importă instrumentul “*SummaryWriter*” care permite scrierea datelor (precum grafice, scalarii, imagini) pe care *tensorboard* le poate vizualiza;
- *writer = SummaryWriter(...)* - inițializează un obiect “*SummaryWriter*”, specificând locația *yolov5/runs/train* în care se stochează fișiere *log*, un *log* fiind un fișier de înregistrare a evenimentelor, în cazul modelului YOLOv5 aceste fișiere se actualizează după fiecare epocă de antrenament;
- *writer_close()* - linia respectivă închide “*SummaryWriter*” la sfârșitul procesului de antrenare, asigurând că toate logurile au fost înregistrate corect și că nu există probleme de corupere a datelor.

Se salvează codul actualizat al fișierului *train_yolov5.py* și se introduce în *Anaconda prompt* următoarea linie, pentru a porni procesul de antrenare a modelului:

```
(Yolov5Train_3.8) C:\Users\Username\PyCharmProjects\Yolov5Train>python train_yolov5.py
```

Pentru vizualizarea în timp real a metricilor de performanță, se accesează o nouă fereastră de *Anaconda Prompt*. Se navighează în directorul în care este salvat proiectul PyCharm și se introduce următoarea comandă:

```
C:\Users\Username\PyCharmProjects\Yolov5Train>tensorboard --logdir yolov5/runs/
```

Comanda respectivă specifică directorul în care *tensorboard* va căuta log-urile, după ce instrumentul de vizualizare a fost pornit, *Anaconda Prompt* returnează mesajul:

```
"Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all  
TensorBoard 2.17.0 at http://localhost:6006/ (Press CTRL+C to quit)".
```

Care indică faptul că trebuie deschisă o pagină de web browser pe accesând *http://localhost:6006/*, pentru a vizualiza metricile și graficele evaluării antrenării în timp real prin interfața *tensorboard*. Pentru închiderea instrumentului se tastează CTRL+C.

5.3 Antrenarea și evaluarea modelului cu setul de date *DatasetNr1*

Setul de date folosit pentru antrenarea modelului, a fost descărcat de pe *roboflow*, care este un open-source ce oferă acces pentru o mulțime de seturi de date pentru diferite aplicații de învățare automată și învățare profundă [47].

Setul de date conține 1284 de imagini și este alcătuit din trei subdirectoare: *train* – subdirector folosit la antrenarea modelului, *valid* – subdirector de validare, folosit pe parcursul antrenamentului pentru evitarea antrenării excesive, și subdirectorul *test* – folosit pentru evaluarea modelului după ce antrenarea s-a sfârșit cu succes. Fiecare din subdirectoarele respective, conține câte un director *images* care conține imagini de extensie JPG și câte un director de etichete *labels* care conține fișiere de format TXT cu informațiile despre clasele componentelor electronice și bounding box-urile lor. Fișierul de configurare a datasetului *data.yaml* conține următoarele informații:

```
train: E:/DatasetNr1/train/images  
val: E:/DatasetNr1/valid/images  
test: E:/DatasetNr1/test/images  
  
nc: 8  
names: ['ACV', 'ARR', 'C', 'I', 'L', 'l-', 'R', 'V']
```

- *train*, *val*, *test* - specifică căile către subdirectoarele de imagini pentru antrenare, validare și testare;
- *nc* - reprezintă numărul de clase, 8 clase pentru setul respectiv;
- *names* - denumirile claselor;
- *ACV* - sursă de tensiune alternativă (clasa 0);
- *ARR* - săgeată ce indică direcția căderii de tensiune (*arrow* din engleză, clasa 1);
- *C* - condensator (clasa 2);
- *I* - sursă de curent (clasa 3);
- *L* - bobină (clasa 4);
- *l-* - După o inșeptare a setului de date și a bounding box-urilor am ajuns la concluzia că este o greșeală a autorului setului de date (clasa 5);
- *R* - rezistor (clasa 6);
- *V* - sursă de tensiune DC (clasa 7);

Un exemplu de imagine din subdirectorul *train* a setului *DatasetNr1* este reprezentat în Figura 3.

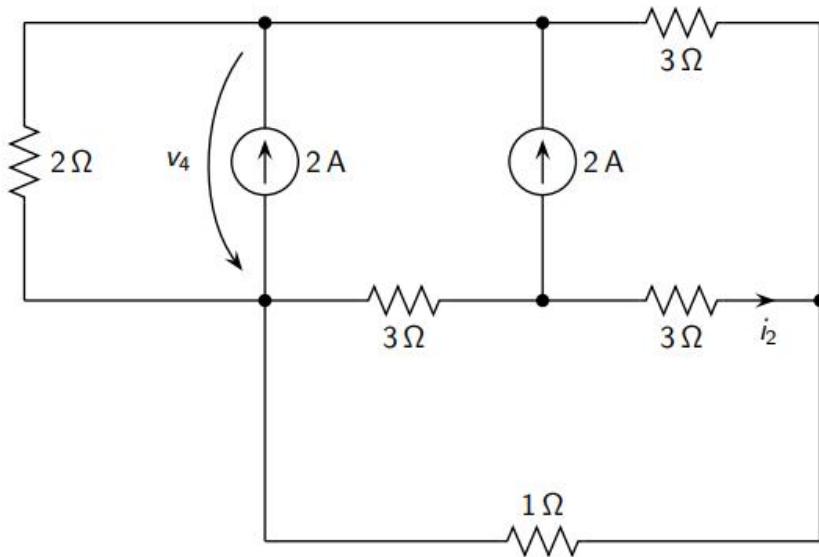


Figura 3. Imagine a unui circuit din setul de date *DatasetNr1*

Este binevenită și examinarea fișierului de etichete (labels) din Tabelul 1., ce este asociat imaginii respective, pentru o înțelegere mai profundă a structurii unui bounding box:

Clasă	X_center	Y_center	lățime	înălțime
6 (<i>resistor</i>)	0.082866043613 70717	0.319760191846 52277	0.048395386292 832849	0.168321342925 65948
6 (<i>resistor</i>)	0.739423676012 4611	0.319760191846 52277	0.110950155763 23988	0.082374100719 42447
6 (<i>resistor</i>)	0.462757009345 79433	0.521894844124 7	0.094672897196 26168	0.072494004796 16307
6 (<i>resistor</i>)	0.740436137071 6511	0.521055155875 2997	0.092118380623 2053	0.079520838693 04555
6 (<i>resistor</i>)	0.602336448598 1308	0.898657074340 5276	0.092118380623 2053	0.080407673860 91127
3 (<i>I</i>)	0.322087227414 3302	0.312925659472 4207	0.079548286604 36138	0.147422062350 12
3 (<i>I</i>)	0.599485981308 4112	0.315611510791 36696	0.079283489096 5732	0.147422062350 12

Tabelul 1. Structura fișierului *label* asociat imaginii din Figura 3.

Fiecare instanță de obiect este încadrată într-un dreptunghi de delimitare numit bounding box caracterizat cu parametrii observați în Tabelul 1., toți parametrii sunt normalizați la dimensiunile imaginii. Modelul YOLOv5 analizează bounding box-urile din imagini conform schemei din Figura 4.

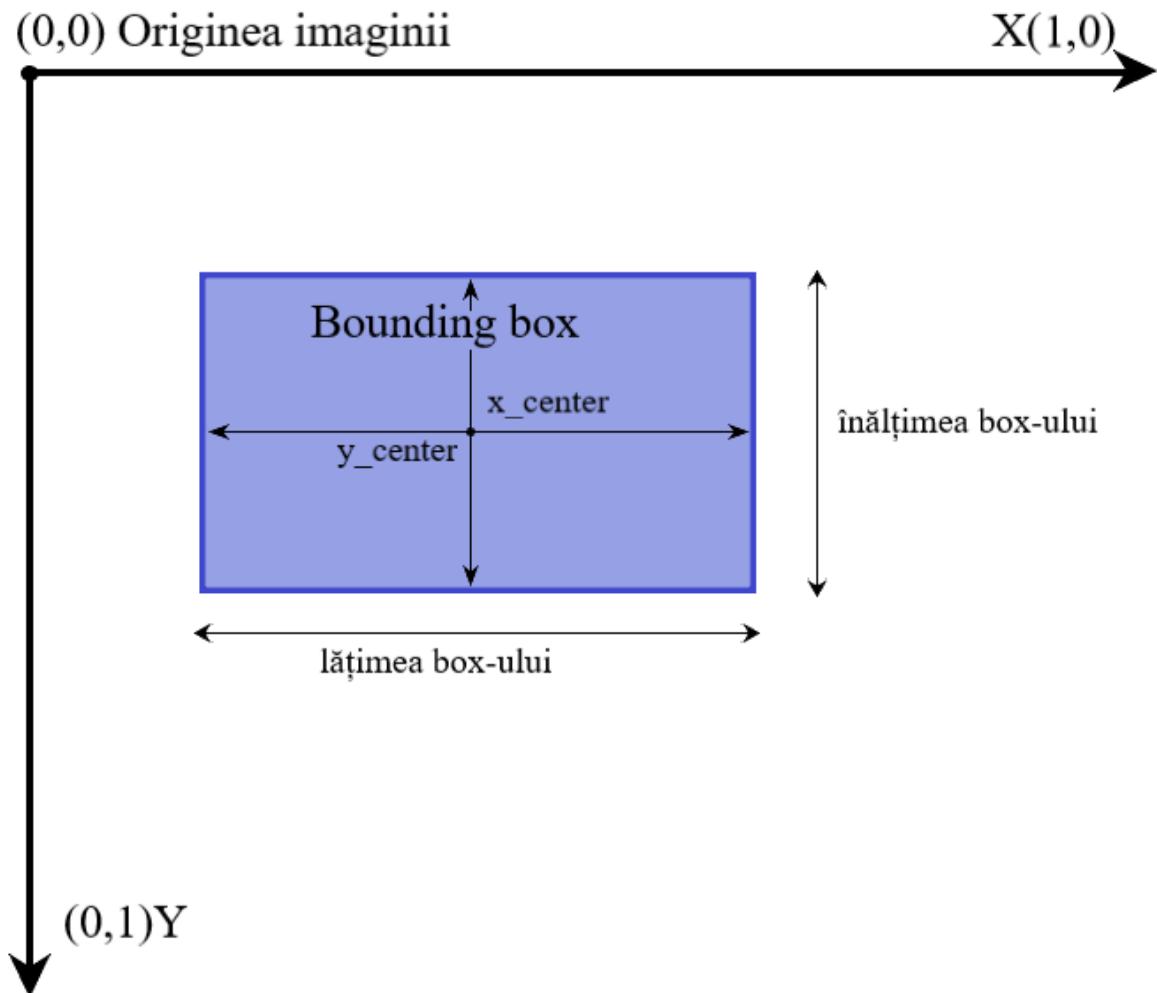


Figura 4. Parametrii bounding box-ului încadrat într-o imagine

5.3.1 Vizualizarea metricilor de performanță a modelului

Accesând link-ul <http://localhost:6006/> pe o pagină de browser web se poate vizualiza evoluția în timp real a metricilor de performanță a modelului pe parcursul antrenării. Axa Ox a graficelor reprezintă epociile antrenării care s-au finalizat, iar axa Oy reprezintă valori de la 0 la 1 [0...100%]. Capturile graficelor s-au efectuat după primele 30 epoci, prin apăsarea refresh a paginii web datele se actualizează.

mAP_0.5 (Mean Average Precision at Intersection over Union) se definește ca media preciziei a suprapunerii casetelor de delimitare (bounding la boxes) la o valoare fixă de 0.5, cu alte cuvinte aceasta măsoară cât de bine sunt detectate componente din imagini, considerând o suprapunere de 50% între box-ul prezis și cel real ca o detecție corectă.

Pe graficul din Figura 5. stânga, se observă o creștere rapidă a *mAP_0.5* în primele câteva epoci, urmată de o stabilizare. Această evoluție sugerează că modelul învață rapid la început, dar îmbunătățirile devin mai lente pe măsură ce antrenamentul progresează.

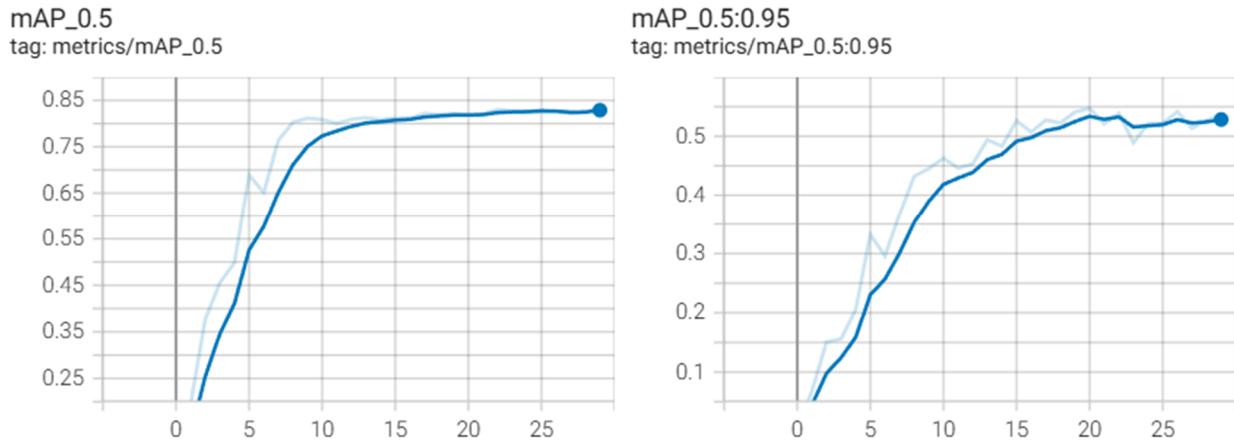


Figura 5. Graficele $mAP_0.5$ (Mean Average Precision at IoU 0.5) și $mAP_0.5:0.95$ (Mean Average Precision across IoU thresholds from 0.5 to 0.95)

$mAP_0.5:0.95$ (Mean Average Precision across IoU thresholds from 0.5 to 0.95) - reprezintă media preciziei a suprapunerii casetelor de delimitare pentru gama valorilor [0.5;0.95], cu pași de 0.05, aceasta fiind o metrică mai strictă comparativ cu $mAP_0.5$, deoarece necesită o suprapunere mai mare între box-uri pentru a considera o detecție corectă. În graficul din dreapta a Figurii 6., observăm o creștere a $mAP_0.5:0.95$, cea ce indică că modelul devine mai precis la identificarea obiectelor.

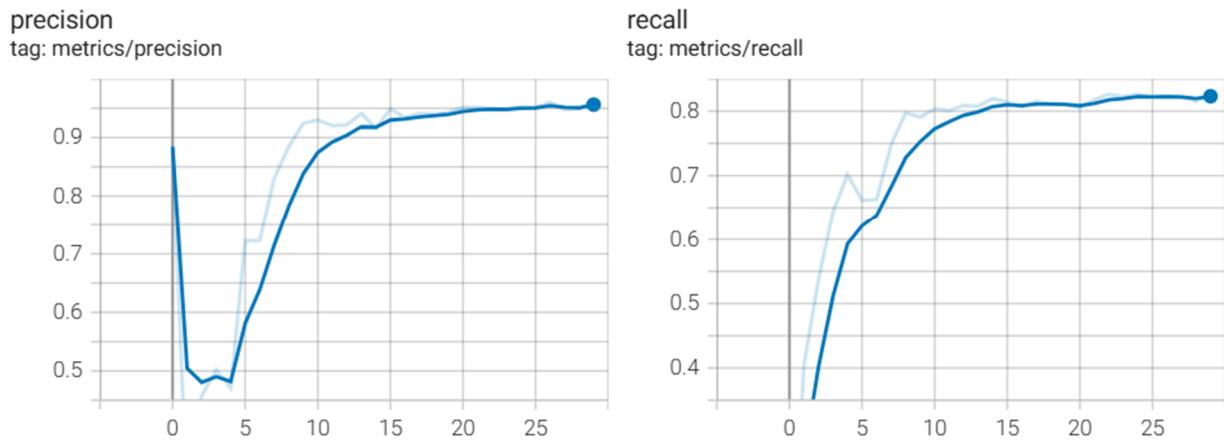


Figura 6. Graficele $precision$ și $recall$

$precision$ - reprezintă precizia proporția detecțiilor corecte (adevărate pozitive) din totalul detecțiilor făcute (adevărate pozitive + false pozitive). Graficul reprezentat în partea dreaptă a Figurii 6. , indică o precizie scăzută la primele epoci, după care precizia se stabilizează la valori relativ ridicate. O fluctuație semnificativă la început poate indica o perioadă inițială de ajustare a modelului.

$recall$ - acuratețea sau recuperarea, reprezintă proporția detecțiilor corecte (adevărate pozitive) din totalul obiectelor care ar fi trebuit detectate (adevărate pozitive + false negative). Un recall mai mare înseamnă că modelul identifică corect mai multe dintre obiectele prezente în imagini. Graficul din stânga Figurii 7. indică faptul că modelul învață să identifice corect majoritatea obiectelor, dar probabil atinge o limită a capacitatei sale de generalizare pe acest set de date.

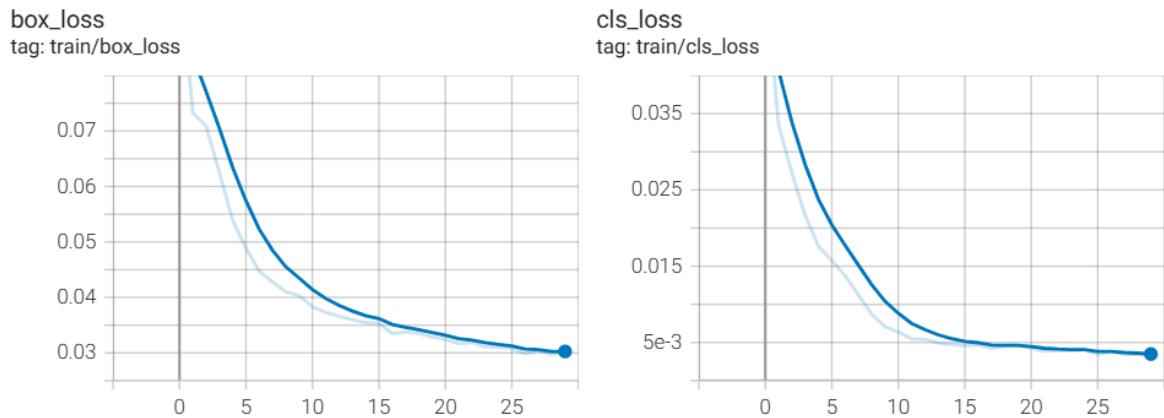


Figura 7. Graficele *box_loss* și *cls_loss*

box_loss - este definită ca pierderea asociată cu predicția corectă a coordonatelor casetelor de delimitare (bounding boxes), aceasta măsurând cât de bine se aliniază casetele de delimitare prezise de model cu cele reale (ground truth). Graficul din stânga reprezentat în Figura 7. Oferă informații că metrica *box_loss* scade rapid în primele epoci, ceea ce indică faptul că modelul își îmbunătășește rapid capacitatea de a prezice corect poziția și dimensiunea casetelor de delimitare. O scădere constantă a *box_loss* este un semn pozitiv și indică faptul că modelul învață să se ajusteze mai bine la datele de antrenament.

cls_loss - (Classification Loss) se definește ca pierderea asociată cu clasificarea corectă a obiectelor din imagini, cu alte cuvinte măsoară cât de bine modelul atribuie etichetele corecte obiectelor detectate. În graficul din dreapta reprezentat în Figura 7., *cls_loss* scade semnificativ la începutul antrenamentului, indicând faptul că modelul învață rapid să clasifice corect obiectele detectate. La fel ca în cazul metricii *box_loss*, scăderea devine mai lentă pe măsură ce modelul se antrenează și performanța se stabilizează.

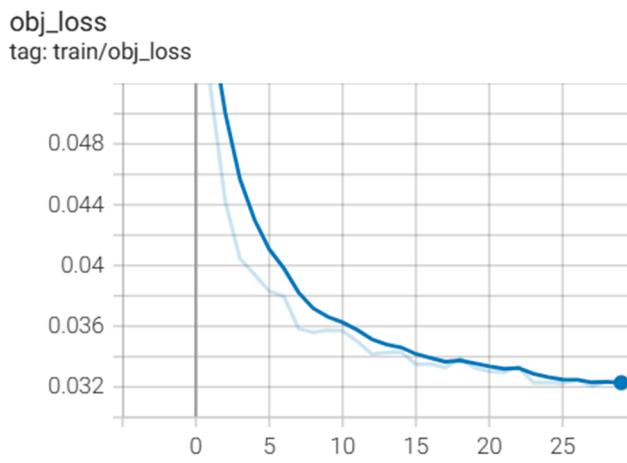


Figura 8. Graficul *obj_loss*

obj_loss (Objectness Loss) - graficul căreia se poate vizualiza în Figura 8. este componenta funcției de pierderi care măsoară capacitatea modelului de a prezice prezența sau absența unui obiect în fiecare celulă de detecție. În primele epoci ale antrenamentului, graficul arată o scădere rapidă a *obj_loss*, ceea ce indică faptul că modelul învață repede să identifice regiunile din imagini care conțin obiecte versus cele care nu conțin. După aproximativ 10-15 epoci, graficul treptat se stabilizează, iar scăderea devine mai graduală. Astfel modelul se apropie de performanță sa optimă în identificarea prezenței obiectelor. Graficele metricilor de performanță afișate și actualizate pe

parcursul antrenării. Acestea sunt utile pentru detectarea timpurie a simptomelor învățării excesive (overfitting), dacă din informațiile afișate de *tensorboard* după un anumit număr de epoci se observă scăderea performanțelor, se recomandă finazilarea antrenării tastând CTRL+C în *Anaconda Prompt*.

O altă opțiune pentru vizualizarea metricilor de performanță și a progresului de antrenare este afișată în *Anaconda Prompt* după ce se finalizează fiecare epocă de antrenare, un exemplu fiind reprezentat în Figura 9. :

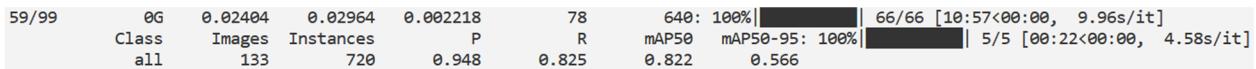


Figura 9. Metricile unei epoci a procesului de antrenare afișate în *Anaconda Prompt*

- *59/99 (epoch)* - indică că modelul este la epoca 59 din 100 epoci de antrenament (0.99);
- *0G* – indică utilizarea memoriei GPU;
- *0.02404 (box_loss)* - pierdere asociată cu predicția corectă a casetelor de delimitare (bounding boxes);
- *0.02964 (obj_loss)* - pierdere asociată cu prezicerea prezenței sau absenței unui obiect într-o celulă de detecție;
- *0.002218 (cls_loss)* - pierdere asociată cu clasificarea corectă a obiectelor în imagini.
- *78* - Numărul de pași de antrenament finalizați în această epocă, pe parcursul fiecarui pas se procesează un batch de imagini;
- *640* – dimensiunile imaginilor utilizate în antrenare (640x640 pixeli);
- *100%* - progresul indică că epoca curentă a fost finalizată;
- *all* – aceste valori sunt medii pentru toate clasele (Class) din dataset;
- *133* – numărul total de imagini (Images) evaluate în etapa de validare respectivă;
- *720* - numărul total de instanțe (Instances) de obiecte detectate în setul de validare;
- *0.948* - Precizia (*precision* - P) modelului pentru predicția corectă a clasei unui obiect;
- *0.825* - Proporția (*recall* - R) a obiectelor detectate în raport cu toate obiectele ce trebuiau să fie detectate ;
- *0.822* - Media preciziei pentru toate clasele la un prag IoU de 0.5 (*mAP_0.5* - mean Average Precision at IoU 0.5);
- *0.566* - Media preciziei pentru toate clasele, măsurată pe gama de valori IoU [0.5; 0.95] (*mAP_0.5:0.95* - mean Average Precision across IoU thresholds from 0.5 to 0.95);

După ce procesul de antrenare s-a sfârșit, *Anaconda Prompt* afișează un mesaj cu metrii de performanță a modelului, acestea sunt expuse în Tabelul 2. :

Clasă	Imagini	instanțe	<i>precision</i>	<i>recall</i>	<i>mAP_0.5</i>	<i>mAP_0.5:0.95</i>
toate	133	720	95.7%	82.8%	82.6%	58.0%
<i>ACV</i> (sursă de tensiune alternativă)	133	55	94.8%	99.1%	97.6%	76.3%
<i>ARR</i> (săgeata ce indică căderea de tensiune)	133	1	0.0%	0.0%	0.0%	0.0%
<i>C</i> (condensator)	133	62	93.3%	96.8%	94.6%	55.4%
<i>I</i> (sursă de curent)	133	61	99.4%	99.4%	99.4%	80.1%
<i>L</i> (bobină)	133	71	99.7%	99.7%	99.5%	69.9%
<i>R</i> (rezistor)	133	419	99.5%	99.5%	99.2%	67.0%

Tabel 2. Metrii de performanță a YOLOv5 antrenat pe setul de date *DatasetNr1*

Precizia generală (*precision*) indică că din toate componentelete pe care le-a detectat modelul 95.7% au fost identificate corect. Rata de detecțare a modelului (*recall*) indică că modelul a detectat 82.8% de obiecte reale din imagini, însă există 17.2% de obiecte reale nedetectate de model.

Media preciziei pentru toate clasele la un prag IoU de 0.5 (*MAP_0.5*) are valoarea de 82.6%, sugerând faptul că modelul are o acuratețe moderată în localizarea și clasificarea componentelor de circuit.

Media preciziei pentru clasele pentru un interval IoU de [0.5; 0.95] (*MAP_0.5:0.95*) are o valoare scăzută de 58.0%, acest lucru sugerând ca modelul întâmpină dificultăți în detectarea precisă a obiectelor atunci când se impun criterii mai stricte de localizare.

Clasa *ARR* care reprezintă săgețile de cădere a tensiunii, nu a fost invățată deloc de către model având un procentaj de 0 la fiecare metrică, cauza principală a acestui rezultat fiind numărul insuficient de instanțieri a obiectelor clasei *ARR* în setul de date respectiv.

În concluzie rezultatele performanțelor sunt nesatisfăcătoare pentru funcționarea eficientă a scriptului de generare fișierelor de extensie NET. Ca soluție este aleasă antrenarea YOLO pe alt set de date.

5.4 Antrenarea și evaluarea modelului cu setul de date *DatasetNr2*

În lucrarea respectivă s-au folosit două seturi de date, primul fiind nepotrivit scopului lucrării inițial care propunea generearea fișierelor de extensie NET pentru aplicația LTspice din imagini cu circuite electronice schițate de mână. Astfel a fost aleasă soluția de a antrena modelul YOLOv5 și pe un set de date care include doar circuite electronice desenate de mână. Setul de imagini *DatasetNr2* este ca și în cazul precedent descărcat de pe site-ul open-source *roboflow* [48]. Aceasta la rândul său conține 2703 imagini și etichete de circuite electronice de structură simplă. Structura directorului setului respectiv este identică ca în cazul precedent: *valid* – subdirector de validare, *train* – subdirector folosit la antrenarea modelului, *test* – folosit doar pentru evaluarea modelului. Cu siguranță fiecare subdirector conține imagini de tip PNG, iar fiecărei imagini îi este asociat prin duplicarea denumirii un fișier de etichete (*labels*) de tip TXT care conține informații despre clasele obiectelor și coordonatele bounding box-urilor.

Fișierul de configurare *data.yaml* din setul de date oferă următoarele informații:

```
train: E:/DatasetNr2/train/images
val: E:/DatasetNr2/valid/images
test: E:/DatasetNr2/test/images

nc: 6
names: ['ac', 'capacitor', 'dc', 'diode', 'inductor', 'resistor']
```

- *train*, *val*, *test* – specifică căile catre subdirectoarele de imagini pentru antrenare, validare și testare;
- *nc* – reprezintă numărul de clase, 6 clase în cazul acestui set;
- *names* – denumirile claselor;
- Fiecare clasă reprezintă următoarele componente:
- *ac* – sursă de tensiune alternativă (clasa 0);
- *capacitor* – condensator (clasa 1);
- *dc* – sursă de tensiune continuă (clasa 2);
- *diode* – diodă (clasa 3);
- *inductor* – bobină (clasa 4);
- *resistor* – rezistor (clasa 5);

Un exemplu de imagine ce face parte din subdirectorul *train* a setului respectiv îl putem vizualiza în Figura 10.

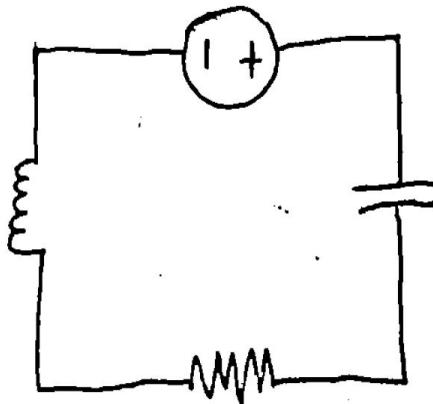


Figura 10. Un circuit electric din subdirectorul *train* a setului *DatasetNr2*

Structura fișierul de etichete (*labels*) asociat imaginii date este aceeași ca în cazul primului set de date, putem vizualiza în Figura 11.

Clasă	X_center	Y_center	lățime	înălțime
2 (<i>dc</i>)	0.51484375	0.20390625	0.2515625	0.24921875
1 (<i>capacitor</i>)	0.8609375	0.48125	0.221875	0.14921875
5 (<i>resistor</i>)	0.528125	0.8625	0.24375	0.15546875
4 (<i>inductor</i>)	0.103125	0.51640625	0.115625	0.26875

Tabelul 3. Structura fișierului de etichete *labels* asociat imaginii din Figura 10.

Pentru crearea unui script funcțional de generare a netlist-urilor LTspice avem nevoie de fișierul *best.pt* ce se salvează automat în subdirectorul *runs/train/exp11/weights* a modelului YOLO după ce se sfârșește antrenarea cu setul de date *DatasetNr2*.

Fișierul *best.pt* conține parametrii modelului YOLOv5 ce au fost învățați pe parcursul antrenării, astfel ponderile (*weights*) a fiecarei conexiuni din rețea neuronală sunt salvate în fișierul respectiv, fișierul conține informații necesare pentru a face predicții referitor la componentele reprezentate în imagini.

Metricile de performanță a modelului, validate după antrenament pe setul de date *DatasetNr2* pot fi vizualizate în Tabelul 4.

Clasă	imagini (<i>valid</i>)	instanțe	<i>precision</i>	<i>recall</i>	<i>mAP_0.5</i>	<i>mAP_0.5:0.95</i>
toate	99	385	96.1%	97.1%	96.5%	74.5%
<i>ac</i> (sursă de tensiune alternativă)	99	44	95.0%	95.5%	96.0%	84.1%
<i>capacitor</i> (condensator)	99	47	95.8%	97.4%	96.1%	60.8%
<i>dc</i> (sursă de tensiune continuă)	99	35	96.5%	100%	99.5%	85.4%
<i>diode</i> (diodă)	99	63	93.8%	95.8%	92.4%	73.4%
<i>inductor</i> (bobină)	99	79	97.2%	97.5%	97.6%	72.4%
<i>resistor</i> (rezistor)	99	117	98.1%	96.6%	97.6%	70.8%

Tabelul 4. Metrici de performanță a YOLOv5 antrenat pe setul de date *DatasetNr2*

Acum este posibilă compararea performanțelor obținute de model în urmă antrenării pe seturile de date respective, aceasta este reprezentată în Tabelul 3.

Set de date utilizat pentru antrenare	<i>precision</i>	<i>recall</i>	<i>mAP_0.5</i>	<i>mAP_0.5:0.95</i>
<i>DatasetNr1</i>	95.7%	82.2%	82.6%	58.0%
<i>DatasetNr2</i>	96.1%	97.1%	96.5%	74.5%

Tabelul 5. Compararea metricilor obținute pentru 2 seturi de date.

Având în vedere aceste rezultate, este evident că modelul antrenat pe setul *DatasetNr2* oferă performanțe mai bune, necesare pentru aplicația propusă proiectului.

5.5 Generare netlist LTspice

Primul pas este crearea unui fișier în directorul principal a proiectului PyCharm cu denumirea *generate_netlist.py*. Urmează importarea bibliotecii *torch*, aceasta fiind folosită pentru încărcarea modelului antrenat YOLOv5, de asemenea codul necesită importul bibliotecii *os* care oferă posibilitatea interacțiunii cu sistemul de fișiere, în cazul aplicației date există necesitatea salvării fișierului netlist generat.

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path =
'C:/Users/username/PycharmProjects/Yolov5Train/yolov5/runs/train/exp11/weights/best.pt')
```

Aici funcția ‘*torch.hub.load*’ realizează încărcarea modelului pre-antrenat YOLOv5 pe setul de date *DatasetNr2*. La sfârșitul procesului de antrenare cu datsetul respectiv, YOLO a salvat automat fișierul *best.pt* care conține ponderile conexiunilor neurale necesare la efectuarea predicțiilor în subdirectorul *runs*.

Se specifică calea către imaginea pentru care se va genera netlistul, iar funcția ‘*model(img)*’ realizează inferență asupra imaginii specificate, generând predicții sub forma bounding box-urilor pentru componentele detecate, împreună cu clasele componentelor și scorurile de probabilitate.

```
img = 'C:/Users/Username/Desktop/Validation netlist/Test2.jpg'
results = model(img)
```

Apoi se initializează lista ‘*components*’ care va fi populată cu dicționare ce vor conține informații despre fiecare componentă detectată. Bucla ‘*for*’ iterează prin fiecare predicție a modelului, extragând clasa prezisă pentru obiectul detectat ‘*component_class*’ și coordonatele ‘*x1, y1*’ - pentru colțul boxului de stânga jos și ‘*x2, y2*’ - pentru colțul de dreapta sus. Ulterior se calculează coordonatele centrului box-ului ca în Figura 4., informațiile respective sunt stocate în dicționare care apoi sunt adăugate în lista ‘*components*’.

```
components = []
for *box, conf, cls in results.xyxy[0]:
    x1, y1, x2, y2 = box
    component_class = model.names[int(cls)]
    x_center = (x1 + x2) / 2
    y_center = (y1 + y2) / 2
    components.append({'class': component_class, 'x': x_center, 'y': y_center})
```

Pentru a evita 2 nume identice în netlistul LTspice pentru componente ce aparțin la aceeași clasă, se utilizează dicționarul ‘*counts*’ care urmărește numărul pentru fiecare componentă

detectată în imagine, iar pe măsură ce apar componente noi de același tip, numărul asociat componentei se incrementează.

Chiar dacă sursele de tensiune alternativă și sursele de tensiune continuă aparțin la 2 clase diferite, în LTspice denumirile lor se notează cu ‘V’, astfel există necesitatea și incrementării numărului surselor în cazul în care au fost detectate 2 surse DC și AC.

```
counts = {'resistor': 1, 'inductor': 1, 'capacitor': 1, 'dc': 1, 'diode': 1}
counts['ac'] = counts['dc'] + 1
```

Funcția ‘convert_class_to_netlist’ transformă clasele detecate de YOLO în formatul specific netlistului LTspice, fiecarei componente au fost atribuite valori standard pentru a evita mesajele de erori trimise de LTspice la deschiderea fișierului netlist.

```
def convert_class_to_netlist(component_class, count):
    if component_class == 'resistor':
        return f'R{count}', '1k'
    elif component_class == 'inductor':
        return f'L{count}', '10m'
    elif component_class == 'capacitor':
        return f'C{count}', '10m'
    elif component_class in ['dc', 'ac']:
        return f'V{count}', '5' if component_class == 'dc' else 'SINE(0 1 1K)'
    elif component_class == 'diode':
        return f'D{count}', 'D'
    return None, None
```

Componentele se sortează folosind funcția standard Python ‘sorted()’, urmărind această logică: componenta ce are valoarea coordonatei X cea mai mică o să fie componenta din stânga a circuitului, componenta ce are valoare coordonatei X cea mai mare o să fie componenta din dreapta, după sortare componentele se stochează în lista ‘components’. Următorul pas este sortarea obiectelor detectate după coordonata Y similar procedurii precedente, instanțele detectate fiind stocate în lista ‘middle_components’.

```
components = sorted(components, key=lambda c: c['x'])
left_component = components[0]
right_component = components[-1]

middle_components = sorted(components[1:-1], key=lambda c: c['y'])
top_component = middle_components[0]
bottom_component = middle_components[-1]
```

Lista ‘ordered_components’ aranjează componente sortate în ordinea corectă: stânga, sus, dreapta și jos. Variabilele “ current_node = ‘N001’ ” și “ next_node = ‘N002’ ”, instanțiază nodurile inițiale ale circuitului, folosite pentru conectarea componentelor din stânga și de sus între ele.

Nodul ‘N003’ nu este inițializat la începutul codului, dar este utilizat în instrucțiunile ‘comp == right_component’ și ‘comp == bottom_component’ care conectează componenta din dreapta și componenta de jos între ele și respectiv cea de jos la ground (nodul 0).

Funcția buclei ‘for’ este de a itera prin lista ‘ordered_components’ pentru a construi netlistul pas cu pas, pentru fiecare componentă, se determină tipul componentei ‘component_type’ și o valoare implicită ‘default_value’ necesară în cazul în care utilizatorul nu dorește să seteze valori

componentelor, apoi se apelează funcția 'convert_class_to_netlist', care convertește clasa componentei într-un format compatibil cu netlistul.

```
netlist = []
ordered_components = [left_component, top_component, right_component, bottom_component]
current_node = 'N001'
next_node = 'N002'

for comp in ordered_components:
    component_type, default_value = convert_class_to_netlist(comp['class'], counts[comp['class']])
```

În funcție de poziția componentei: stânga, sus, dreapta, jos, sunt specificate nodurile între care componenta este conectată. Utilizatorului i se propune să introducă valoarea componentei pentru a schimba valoarea ei implicită, se oferă și un exemplu bazat pe valoarea care a fost setată automat.

Componenta din stânga 'left_component' – în partea asta de cod, 'node1' este setat la 'current_node', care la început este N001. 'node2' este setat la 0, indicând că această componentă este conectată la masa circuitului. Acest lucru este comun pentru componente care sunt la începutul lanțului de conexiuni în simulațiile de circuite, având un punct de referință comun (ground).

Componenta de sus 'top_component' - pentru această componentă, 'node1' este 'next_node' inițial N002, iar 'node2' este 'current_node' cu valoarea N001, pentru a avea sintaxă corectă conform regulilor ltspice de netlistare. Aceasta creează o conexiune între nodul curent și următorul nod.

Componenta din dreapta 'right_component' - aici, 'node1' este 'next_node' căruia ii este atribuit N002, iar 'node2' este N003, ultimul nod necesar pentru funcționarea corectă a netlistului.

Componenta de jos 'bottom_component' – în acest caz, 'node1' este N003, nodul creat anterior, iar 'node2' este 0, adică ground-ul. Această sintaxă este necesară pentru a închide bucla de conexiuni a circuitului.

```
if comp == left_component:
    node1, node2 = current_node, '0'
    value = input(f'{component_type} found between nodes {node1} and {node2}. Please set the
value (e.g., {default_value}): ')
elif comp == top_component:
    node1, node2 = next_node, current_node
    value = input(f'{component_type} found between nodes {node1} and {node2}. Please set the
value (e.g., {default_value}): ')
elif comp == right_component:
    node1, node2 = next_node, 'N003'
    value = input(f'{component_type} found between nodes {node1} and {node2}. Please set the
value (e.g., {default_value}): ')
elif comp == bottom_component:
    node1, node2 = 'N003', '0'
    value = input(f'{component_type} found between nodes {node1} and {node2}. Please set the
value (e.g., {default_value}): ')
```

Deoarece YOLOv5 nu știe să detecteze și să clasifice terminalele componentelor, în cod au fost adăugate tratamente speciale pentru sursele de tensiune alternativă AC și continuă DC, precum și pentru diode.

```
if comp['class'] == 'ac':
```

```

dc_offset = input(f"Alternating Voltage Source {component_type} found between nodes {node1} and {node2}. Please set the DC offset (e.g., 0V): ")
amplitude = input("Please set the Amplitude (e.g., 1V): ")
frequency = input("Please set the Frequency (e.g., 1kHz): ")
value = f"SINE({dc_offset} {amplitude} {frequency})"

```

În cazul unei surse de tensiune alternativă AC, utilizatorul este invitat să specifiche offsetul DC, amplitudinea și frecvența, iar valoarea componentei este formatată corespunzător folosind o funcție specifică aplicației LTspice 'SINE'. Pentru componente de tip AC și DC, utilizatorului i se cere să specifiche conexiunile pentru terminalele pozitive și negative la nodurile circuitului, deoarece sintaxa diferă pentru direcția conexiunii a surselor de tensiune, pentru direcția standard a conexiunii terminalul pozitiv o să fie conectat la nodul ce are valoare mai mare dintre cele 2. După ce utilizatorul setează direcția conexiunii a surselor, linia corespunzătoare se adaugă în netlist prin apelarea funcției 'netlist.append'.

```

if comp['class'] in ['ac', 'dc']:
    plus_node = input(f"Please set the plus terminal connection of {component_type} (must be N_higher for a normal direction of connection): ")
    minus_node = input(f"Please set the minus terminal connection of {component_type} (must be N_lower for a normal direction of connection): ")
    netlist.append(f"{component_type} {plus_node} {minus_node} {value}")

```

În mod similar, pentru diode, contează direcția conexiunii a anodului și catodului, specificații esențiale pentru o analiză realistă a circuitelor, astfel utilizatorului i se propune să seteze conexiunile pentru anod și catod dorite:

```

elif comp['class'] == 'diode':
    anode_node = input(f"Please set the anode connection of {component_type} (for the normal direction of connection must be N_lower): ")
    cathode_node = input(f"Please set the cathode connection of {component_type} (for the normal direction of connection must be N_higher): ")
    netlist.append(f"{component_type} {anode_node} {cathode_node} {value}")

```

După ce au fost setate valorile componentelor și conexiunea terminalelor, aplicația afișează un mesaj în care întreabă utilizatorul dacă acesta dorește să introducă o simulare de tip *.op* (operating point analysis):

```

include_op = input("Would you like to include the .op directive in the netlist? (yes/no): ")
if include_op.lower() == 'yes':
    netlist.append(".op")

```

La sfârșitul netlistului au fost adăugate liniile specifice structurii SPICE, acestea fiind reprezentate de directivele '.backanno' și '.end'. De asemenea pentru evitarea mesajelor de eroare trimise de LTspice, a fost adăugată directiva '.model D D' pentru a specifica modelul diodei, în mod evident a fost specificată și locația librăririei '.lib...' în care se conține modelul diodei.

```

netlist.append(".model D D")
netlist.append(".lib C:\\\\Users\\\\Username\\\\Documents\\\\LTspiceXVII\\\\lib\\\\cmp\\\\standard.dio")
netlist.append(".backanno")
netlist.append(".end")

```

Funcția ‘f.write’ scrie un șir de caractere ce specifică locația draftului, chiar dacă draftul este inexistent, sintaxa este necesară pentru funcționarea corectă a netlistului. După ce liniile generate anterior pentru toate componetele și directivele necesare au fost înscrise în netlist, fișierul rezultat *Generated* se salvează în locația specificată de ‘output_file =’ cu extensia NET.

```
output_file = "C:/Users/Danya/Desktop/Validation netlist/Generated.net"
with open(output_file, 'w') as f:
    f.write("* C:\\\\Users\\\\Danya\\\\Desktop\\\\Validation netlist\\\\Draft1.asc\\n")
    for line in netlist:
        f.write(line + '\\n')

print(f"Netlist saved to {output_file}")
```

La sfârșit, un mesaj de confirmare este afișat pentru a informa utilizatorul că fișierul generat netlist a fost salvat cu succes la locația specificată. Codul reprezintă un prototip funcțional de aplicație pentru simularea circuitelor LTspice recunoscute de model în imagini.

6. Rezultate experimentale

Rezultatele experimentale obținute în urma testării modelului YOLOv5 pentru aplicația de generare a fișierelor netlist LTspice vor fi prezentate pentru două seturi de date distincte, denumite *DatasetNr1* și *DatasetNr2*. Aceste rezultate includ o serie de metri și vizualizări care evaluatează performanța modelului în detectarea și clasificarea componentelor de circuite din imagini.

Matricile de Confuzie oferă o imagine de ansamblu asupra acurateței modelului în clasificarea corectă a diferitelor clase de componente, aceasta arată unde modelul a avut succese, dar și unde au apărut erori de clasificare între clasele componentelor electronice, matricile de confuzie pentru ambele seturi de date pot fi vizualizate în Figura 11. și Figura 17.

Curbele Precision-Confidence evidențiază relația dintre încrederea modelului în predicțiile sale și metricile de performanță. Curbele respective permit identificarea pragurilor de încredere la care modelul performează cel mai bine, acestea se pot vizualiza pentru ambele seturi în Figura 14. și Figura 20.

Distribuția Etichetelor oferă informații despre distribuția claselor și caracteristicile geometrice ale bounding box-urilor utilizate în detecție. Aceste vizualizări, ce sunt reprezentate în Figura 12. și Figura 18. ajută la înțelegerea modului în care datele sunt organizate și cum acestea afectează performanța modelului.

Curbele Recall-Confidence sunt utile pentru a determina echilibrul optim între recall și numărul de predicții false pozitive, în funcție de pragul de încredere, acestea pot fi vizualizate în Figura 15. și Figura 21. pentru ambele seturi de date.

Curbele de pierderi și metrii pentru antrenare și validare arată progresul modelului în timpul antrenării, evidențiind cum s-a comportat modelul pe parcursul epocilor în ceea ce privește minimizarea pierderilor și îmbunătățirea performanței, se pot vizualiza în cazul ambelor seturi de date în Figura 13. și Figura 19.

Batch-urile de imagini de validare anotate oferă un mijloc vizual de a evalua corectitudinea și precizia modelului în detectarea și clasificarea obiectelor din imagini reale, demonstrând performanța acestuia într-un context aplicabil, reprezentate în Figura 16. și Figura 22.

6.1 Rezultatele antrenării modelului cu *DatasetNr1*

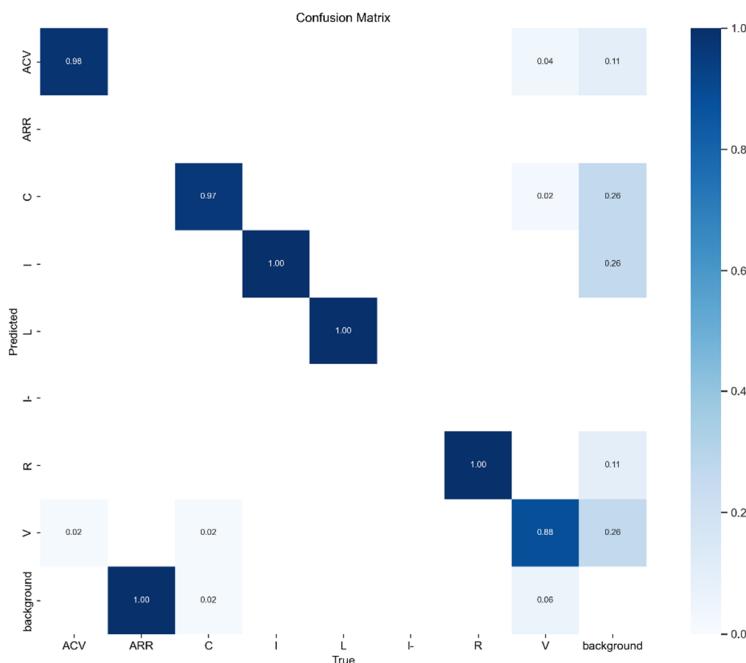


Figura 11. Matricea de confuzie pentru modelul antrenat cu *DatasetNr1*

Matricea de confuzie din Figura 11. rezultată după antrenamentul cu *DatasetNr1*, arată că modelul clasifică corect majoritatea instanțelor, cu performanțe perfecte pentru clasele *L* și *R*, fiecare având o valoare de 1.00. Totuși, există confuzii notabile între clasa *V* și alte clase, unde 26% dintre instanțele *V* au fost clasificate greșit ca *background*, iar 11% din *ACV* au fost clasificate greșit tot ca *background*. În general, modelul arată performanțe puternice, dar prezintă dificultăți în distingerea unor clase, în special *V*, care este confundată cu celelalte clase.

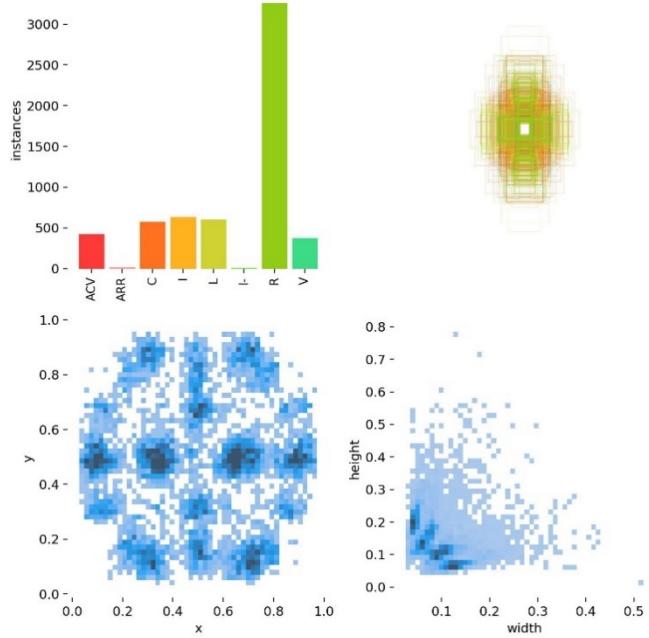


Figura 12. Distributia etichetelor a modelului anrenat cu *DatasetNr1*

Figura 12. conține în partea de stânga sus distribuția instanțelor pe clase. Pe axa X sunt reprezentate clasele obiectelor din dataset (*ACV*, *ARR*, *C*, *I*, *L*, *l-*, *R*, *V*), iar axa Y conține numărul de instanțe pentru fiecare clasă. Se observă că *DatasetNr1* are o distribuție neuniformă a instanțelor pentru fiecare clasă, spre exemplu clasa *R* are un număr de peste 3000 de instanțe, iar clasele problematice ce nu au fost învățate de model *Arr* și *l-*, au valori aproape nule. Graficul de distribuție a bounfing box-urilor, reprezentat în Figura 12. În partea dreaptă sus, reprezintă o suprapunere a bounding box-urilor în cadrul imaginii pentru a arăta cum s-au încadrat diferite dimensiuni și forme de bounding box-uri în obiectele prezente în setul de date.

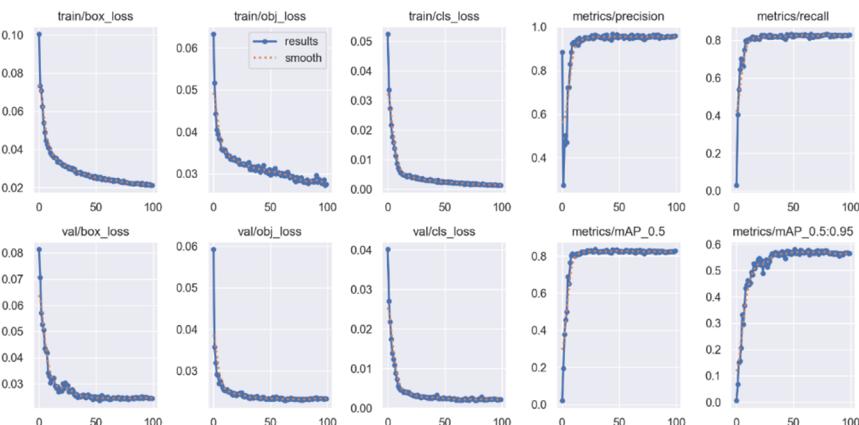


Figura 13. Curbele de pierderi și metricile antrenării și validării cu *DatasetNr1*

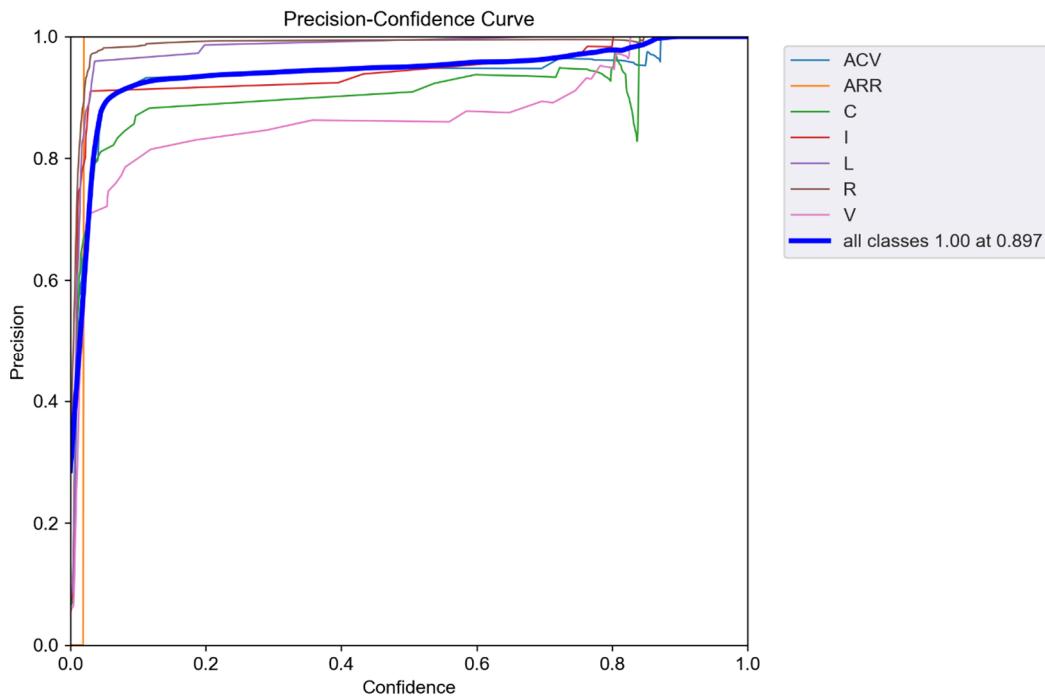


Figura 14. Curba de Precision-Confidence a antrenării cu *DatasetNrI*

Curba Precision-Confidence din Figura 14. arată că modelul YOLOv5 are o performanță ridicată pentru majoritatea claselor, cu o valoare mare de precizie pe întreaga gamă de nivele de încredere. Clasa *R* (rezistor) are o precizie de 99.5%, clasa *I* (sursă de curent) are o precizie de 99.4%, iar clasa *L* (bobină) are o precizie de 99.7%. De asemenea, clasa *ACV* (sursă de tensiune alternativă) are o precizie de 94.8%, clasa *C* (condensator) are o precizie de 93.3%, iar clasa *V* are o precizie de 89.7%. Din motivele precizate anterior, pentru clasa *ARR* (săgeata ce indică căderea de tensiune), modelul nu a reușit să facă predicții corecte, având o precizie de 0%, conform valorilor din tabel.

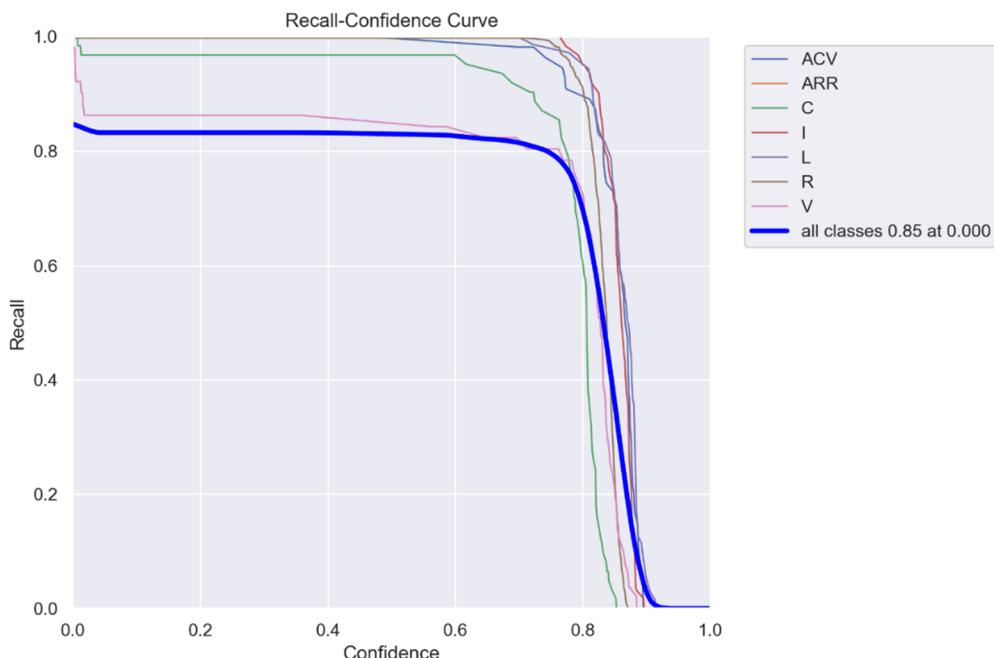


Figura 15. Curba Recall-Confidence a antrenării cu *DatasetNrI*

Graficul Recall-Confidence a antrenării cu *DatasetNr1* din Figura 15. arată că modelul YOLOv5 are o performanță excelentă în ceea ce privește acuratețea pentru majoritatea claselor, menținând valori ridicate până la un nivel de încredere destul de mare. Clasa *R* (rezistor) are un recall de 99.5%, clasa *I* (sursă de curent) are un recall de 99.4%, iar clasa *L* (bobină) are un *recall* de 99.7%. De asemenea, clasa *ACV* (sursă de tensiune alternativă) are o valoare de 99.1%, clasa *C* (condensator) are valoarea de 96.8%, iar clasa *V* prezintă rezultatul de 89.6%.

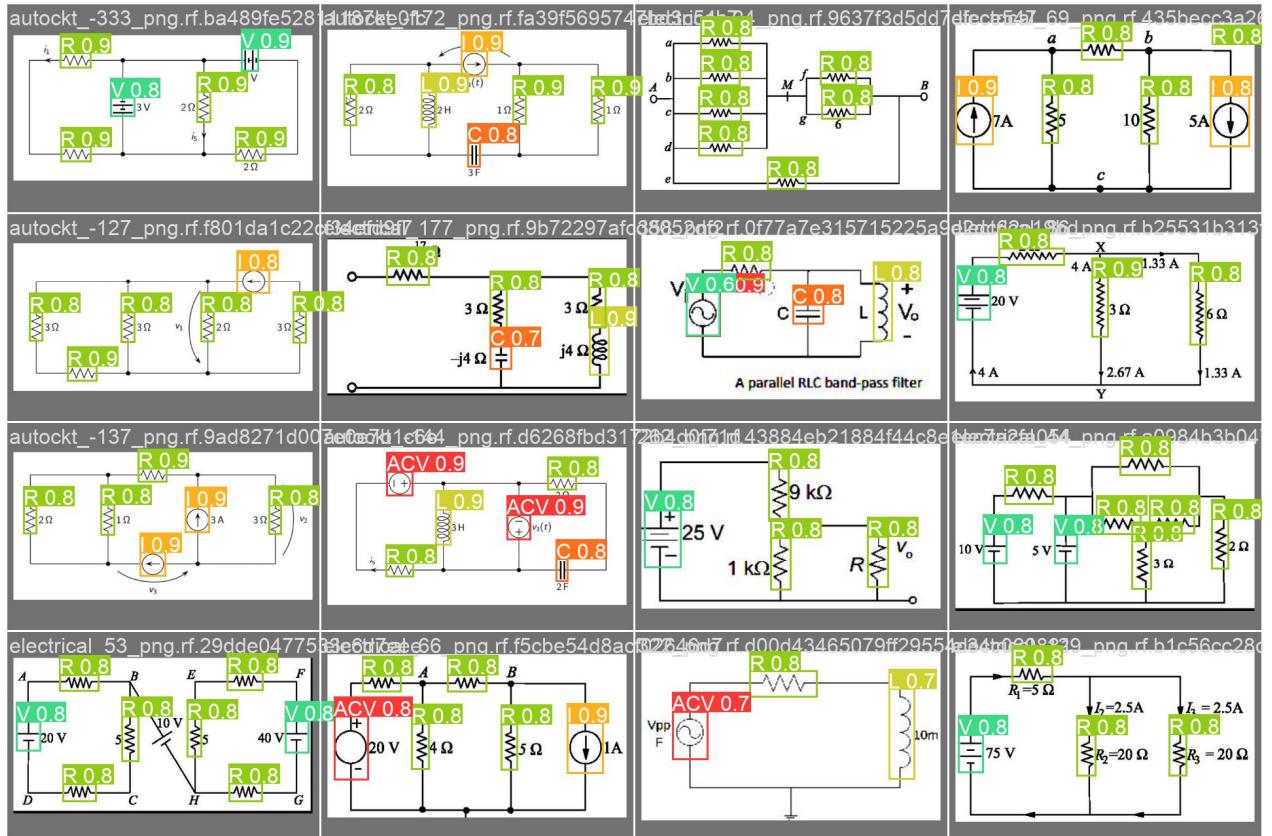


Figura 16. Predictiile modelului și clasele obiectelor pentru un batch (16 imagini) din subdirectorul de validare a setului *DatasetNr1*

6.1.2 Evaluarea rezultatelor obținute pentru *DatasetNr1*

Modelul YOLOv5 antrenat pe setul de date *DatasetNr1* a demonstrat performanțe variate în funcție de clasele de componente electronice pe care a trebuit să le detecteze. În general, modelul a atins o precizie (*precision*) medie de 95.7% și un *recall* de 82.8%, cu un *mAP_0.5* de 82.6% și un *mAP_0.5:0.95* de 58.0%. Aceste rezultate indică faptul că modelul este capabil să detecteze componentele corect în majoritatea cazurilor, dar are unele dificultăți în a captura toate instanțele corecte (reflexia în *recall*) și în a menține un nivel ridicat de performanță pe un spectru larg de scoruri de precizie.

O observație notabilă din metricile reprezentate anterior este performanța foarte scăzută a modelului pentru clasa *ARR* (sägeata ce indică căderea de tensiune), unde modelul a înregistrat toate metricile cu 0 la sută. Aceasta indică faptul că modelul nu a reușit să detecteze corect această clasă deloc, ceea ce poate sugera observația detectată în Tabelul 2., unde era prezentă doar o instanță a clasei respective în subdirectorul *valid*. Acest lucru accentuează absența unui număr suficient de exemple relevante a clasei *ARR* în setul respectiv de date.

Pentru alte clase, cum ar fi *ACV* (sursă de tensiune alternativă), *L* (bobina), și *R* (rezistorul), modelul a arătat performanțe excelente, cu precizie și *recall* apropiate de 100%, ceea ce

demonstrează capacitatea modelului de a recunoaște cu acuratețe aceste componente în contexte diverse.

În concluzie, deși antrenarea modelului YOLOv5 cu setul *DatasetNr1* arată promițător pentru recunoașterea componentelor de circuite electrice, acest set de date nu a fost optimizat corespunzător, cea ce îl face neeficient în utilizare pentru aplicația respectivă.

6.2 Rezultatele antrenării modelului cu *DatasetNr2*

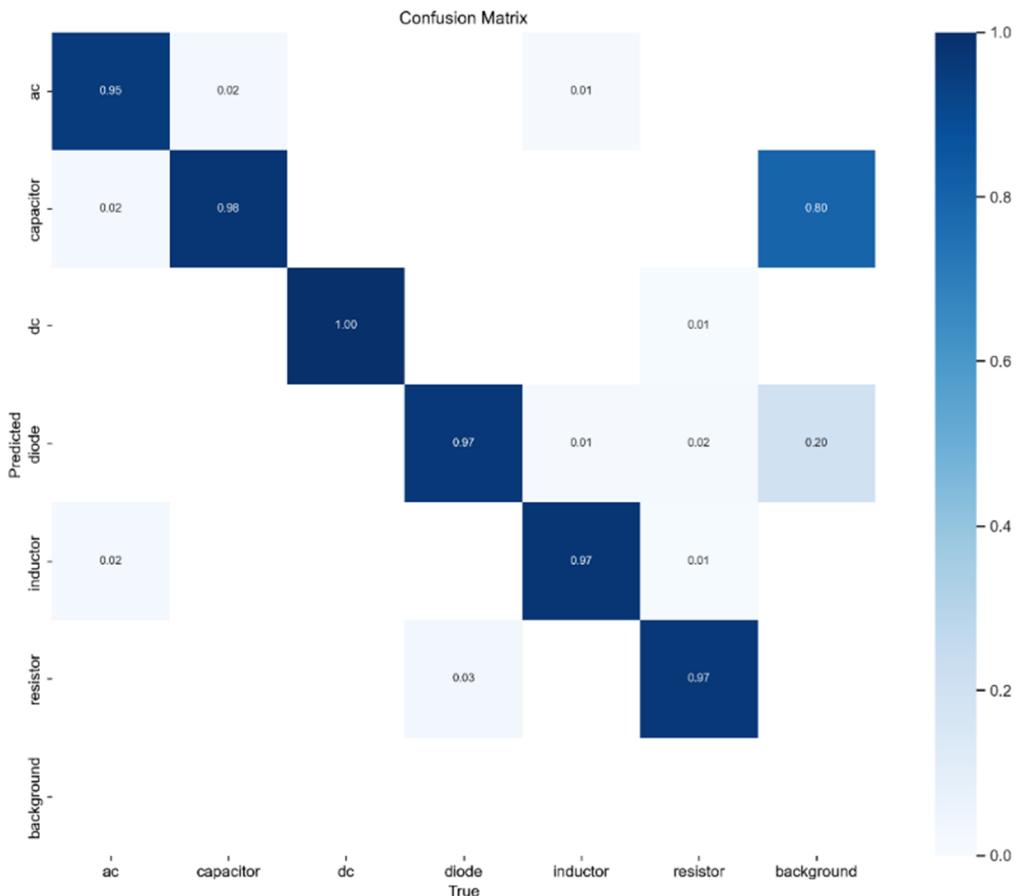


Figura 17. Matricea de confuzie pentru modelul antrenat cu *DatasetNr2*

Matricea de confuzie conținută în Figura 17., pe diagonala principală reprezintă proporția de instanțe clasificate corect pentru fiecare clasă. Se observă că modelul YOLOv5 clasifică corect majoritatea instanțelor, cu o performanță excelentă pentru clasele *dc* (sursă de tensiune continuă) - 1.00, *capacitor* (condensator) - 0.98, și *resistor* (rezistor) - 0.97. Diagonala secundară a acestei matrici de confuzie indică principalele confuzii între clase. Cele mai semnificative erori apar între clasele *diode* (diodă) și *background*, unde 20% (valoarea 0.20) din instanțele *diode* au fost clasificate greșit, și între *resistor* și *background*, cu o rată de eroare de 3%. De asemenea se observă că 2% dintre instanțele din clasa *inductor* (bobină) au fost clasificate greșit ca *resistor* (rezistor).

În ansamblu, modelul performează eficient, dar ar putea beneficia de ajustări pentru a reduce confuziile între aceste clase.

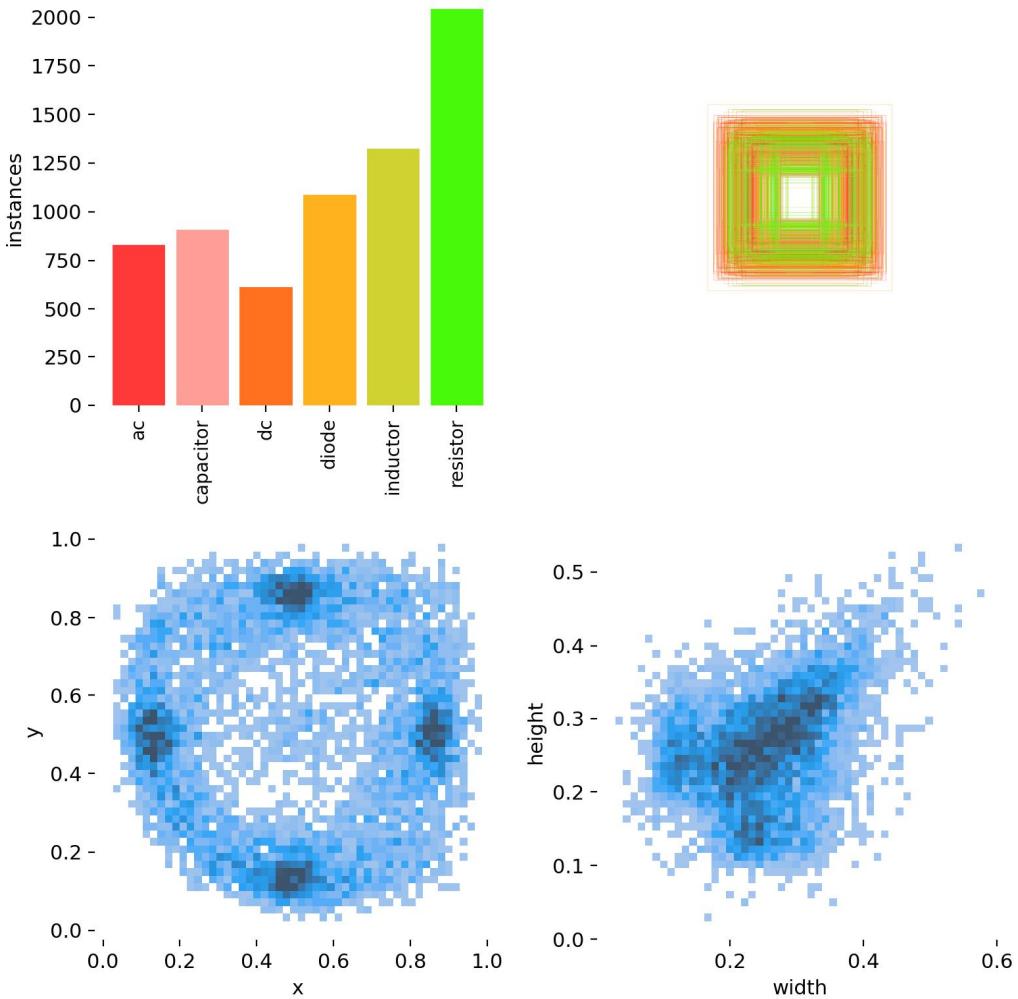


Figura 18. Distribuția etichetelor a modelului antrenat cu *DatasetNr2*

Figura 18. în partea stângă sus reprezintă distribuția instanțelor pentru fiecare clasă din setul de date, indicând că *resistor* (rezistor) are cel mai mare număr de instanțe, cu aproximativ 2000 de exemple, urmat de *inductor* (bobină) și *capacitor* (condensator), în timp ce clasele *dc* (sursă de tensiune continuă) și *ac* (sursă de tensiune alternativă) au cele mai puține instanțe.

Evident, din informațiile furnizate anterior, se concluzionează ca *DatasetNr2* are o distribuție a instanțelor de clase mai echilibrată decât în cazul *DatasetNr1*.

Graficul din dreapta sus din Figura 18. reprezintă suprapunerea bounding box-urilor în cadrul imaginii, arătând o concentrare centrală și dimensiuni similare pentru majoritatea obiectelor.

Graficul din stânga jos din Figura 18. ilustrează distribuția bounding box-urilor conform coordonatelor imaginii pe axele X și Y, care sunt uniform distribuite în cadrul acesteia, cu unele concentrații notabile în zona de stânga, sus, dreapta și jos, ceea ce prezintă un avantaj pentru aplicația dezvoltată în lucrare. În dreapta jos a Figurii 18. este reprezentată distribuția dimensiunilor bounding box-urilor, aceasta la rândul său demonstrează că majoritatea obiectelor au lățimi și înălțimi variate, dar există o tendință de concentrare în jurul unor dimensiuni medii.

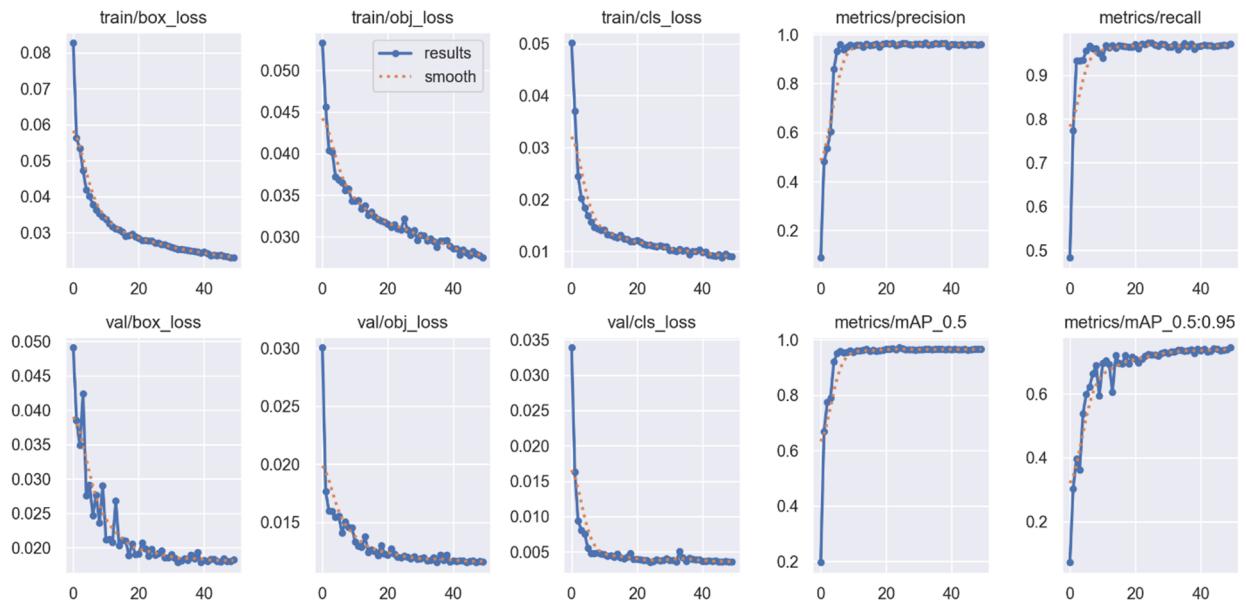


Figura 19. Curbele de pierderi și metricile antrenării și validării cu *DatasetNr2*

Curbele de pierderi și metricile de performanță din Figura 19. pentru modelul YOLO antrenat cu *DatasetNr2*, indică o convergență eficientă pe parcursul a 50 de epoci. Pierderile de antrenament și validare (*train/box_loss*, *train/obj_loss*, *train/cls_loss*, *val/box_loss*, *val/obj_loss*, *val/cls_loss*) scad constant, ajungând la valori semnificativ mici, ceea ce sugerează că modelul a învățat bine din setul de date respectiv. Metricile *precision* și *recall* prezintă valori ridicate, aproape de 1.0, ceea ce este în concordanță cu valorile tabelare afișate de *Anaconda Prompt* în Tabelul 4., unde precizia (*precision*) generală este 96.1% și un *recall* general este 97.1%.

De asemenea, valorile *mAP_0.5* și *mAP_0.5:0.95* se stabilizează rapid, reflectând performanțe excelente, cu valori de 96.5% și respectiv 74.5% la nivel global.

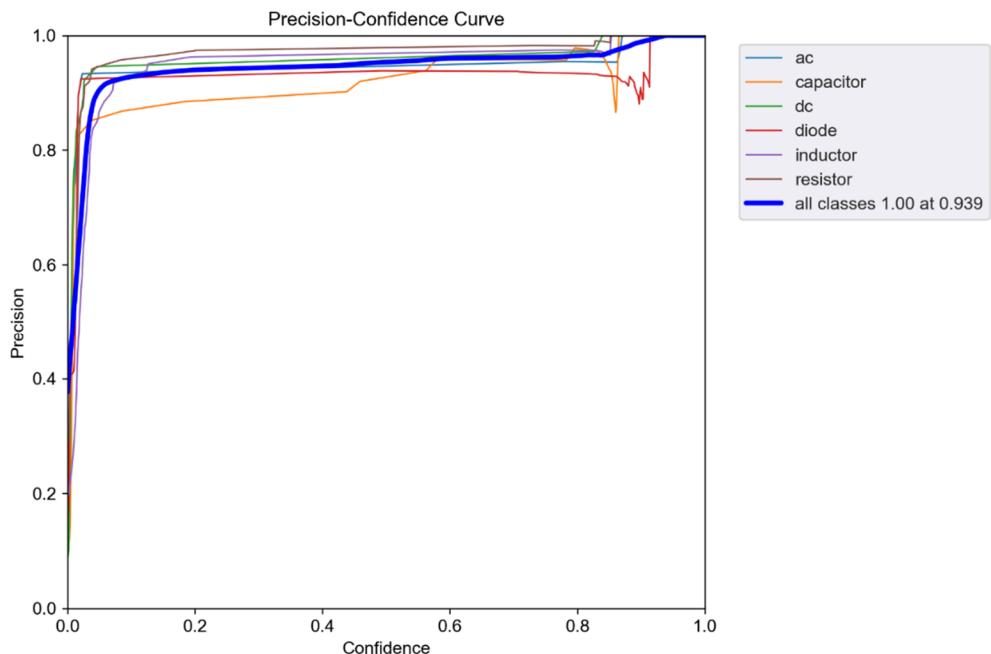


Figura 20. Curba Precision-Confidence în urma antrenării cu *DatasetNr2*

Graficul Precision-Confidence Curve din Figura 21. arată că modelul folosit în lucrare are o performanță ridicată în ceea ce privește precizia pentru toate clasele, cu o curbă ce prezintă valori foarte ridicate la un nivel de încredere înalt. Conform valorilor din Tabelul 4., clasa *resistor* are o precizie de 98.1%, reflectată în grafic printr-o curbă care atinge aproape 1.0 pe întreaga gama de încredere.

De asemenea, clasele *dc* și *capacitor* prezintă și ele rezultate solide, corespunzătoare preciziilor de 96.5% și 95.8%, în timp ce clasa *diode* are o evoluție mai scăzută decât în cazurile celorlalte clase, indicând o precizie de 93.8%.

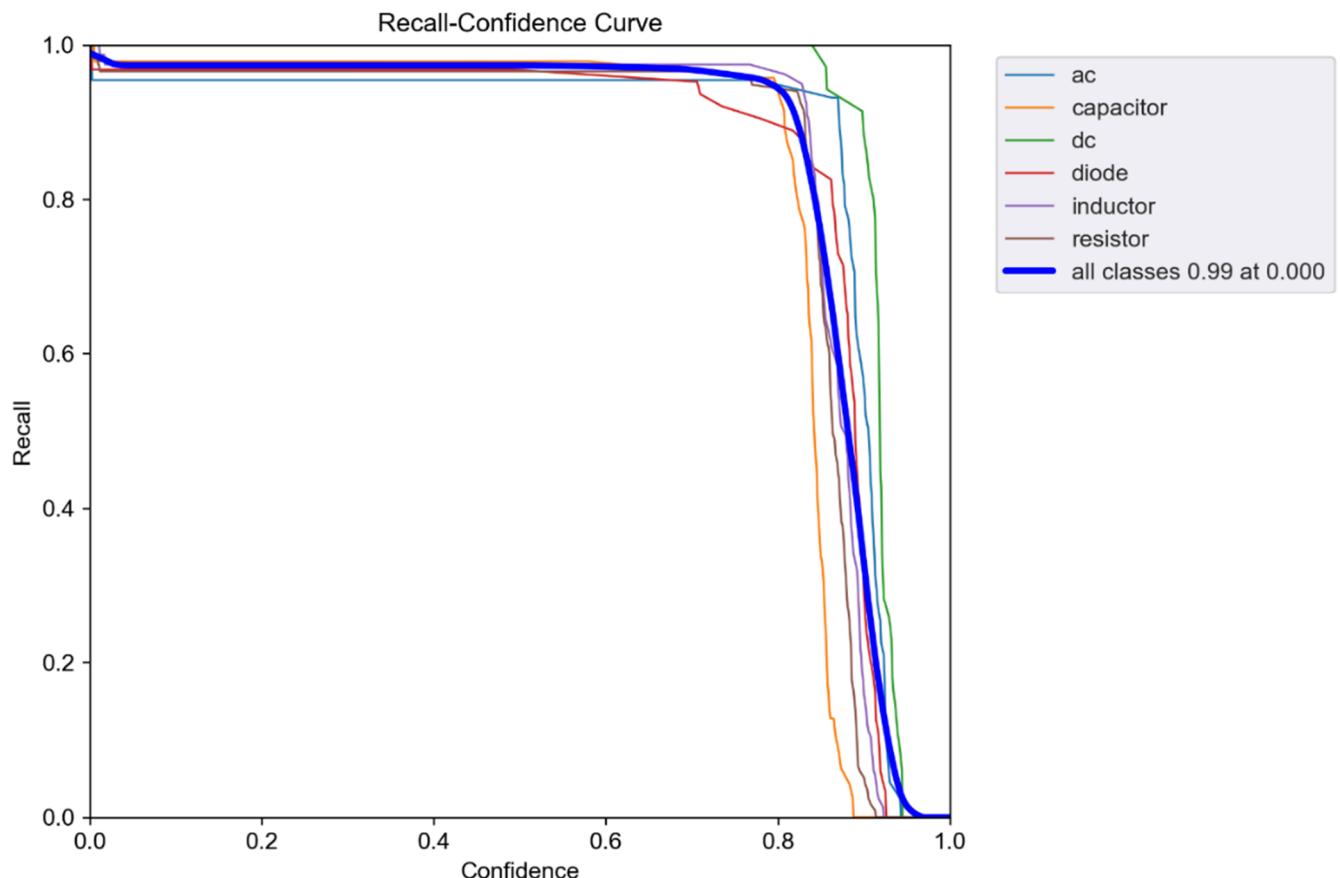


Figura 21. Curba Recall-Confidence a antrenării cu *DatasetNr2*

Graficul Recall-Confidence Curve reprezentat în Figura 21., arată că modelul YOLO menține un nivel ridicat de acuratețe pentru toate clasele pe întreaga gamă de nivele de încredere, cu o scădere abruptă doar la nivele de încredere semnificativ mari. În concordanță cu datele obținute în urma antrenării din Tabelul 4., clasa *dc* atinge un *recall* ideal de 100%, reflectat prin curba sa care se menține la valori de 1.0 foarte aproape de valoarea maximă de pe axa de încredere. Clasele *resistor* și *inductor* prezintă, de asemenea, rezultate solide, corespunzătoare unui *recall* de 96.6% și 97.5%. Clasa *diode* ca și în cazul curbei de Precision-Confidence prezintă un *recall* ușor mai scăzut, de valoare 95.8%, aceasta reflectându-se printr-o ușoară scădere a graficului începând de la un nivel de încredere mai ridicat.



Figura 22. Predicțiile modelului și clasele obiectelor pentru un batch (16 imagini) din subdirectorul de validare a setului *DatasetNr2*

6.2.1 Evaluarea rezultatelor obținute pentru *DatasetNr2*

Antrenarea modelului YOLOv5 pe setul de date *DatasetNr2* a arătat performanțe remarcabile, cu o valoare *mAP_0.5* de 96.5% pentru toate clasele, ceea ce sugerează o detecție și clasificare foarte precisă a componentelor circuitelor electronice.

Majoritatea claselor au atins valori ridicate ale preciziei și recall-ului, evidențiind astfel facilitatea integrării modelului antrenat în aplicația de generare a netlisturilor LTspice. Cu toate acestea, clasa *diode* a prezentat o performanță ușor inferioară, indicând posibile deficiențe în setul de date, mai degrabă decât în model.

Pentru îmbunătățirea generală a modelului utilizat în aplicație, este recomandat să se reechilibreze setul de date, asigurând o distribuție uniformă a instanțelor pentru fiecare clasă. De asemenea, îmbunătățirea etichetării, incluzând orientarea terminalelor componentelor, ar putea spori acuratețea detecției și eficiența. În plus, adăugarea denumirilor de modele SPICE pentru fiecare componentă pe imagini și a valorilor de tensiuni, rezistențe, curenti, capacități, inductanțe ar putea automatiza aplicația respectivă.

Augmentarea setului de date cu instanțe noi și variante ar putea aduce îmbunătățiri semnificative, permitând aplicației să fie și mai precisă în viitor.

6.3 Funcționalitatea scriptului *generate_netlist.py*

Cu scopul de a testa scriptul de generare a fișierelor netlist LTspice, a fost creat un subdirector de imagini de testare, ce conține mai multe exemple a subestului de validare *DatasetNr2*, mai jos sus reprezentate câteva exemple ce confirmă funcționalitatea robustă a aplicației respective.

Netlistul LTspice și anazila .op pentru imaginea *Test11.jpg* din subdirectorul de testare reprezentat în Figura 23. și Figura 24. :

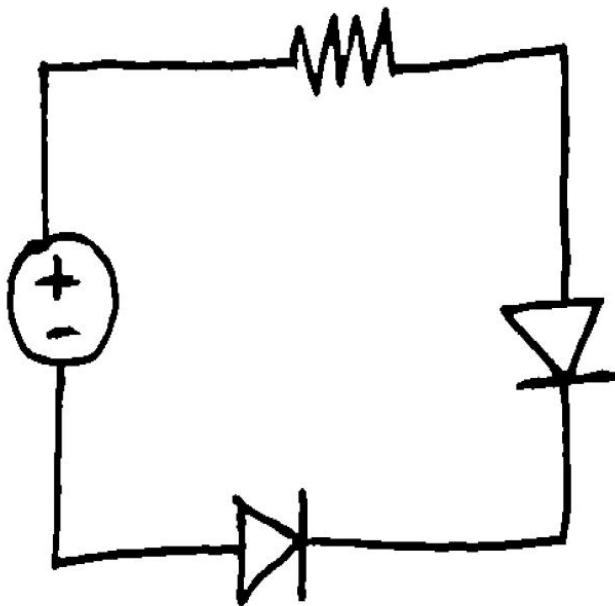


Figura 23. Imaginea *Test11.jpg*

```
* C:\Users\Username\Desktop\Validation netlist\Draft1.asc
V1 N001 0 5
R1 N002 N001 1k
D1 N002 N003 D
D2 N003 0 D
.model D D
.lib C:\Users\Username\Documents\LTspiceXVII\lib\cmp\standard.dio
.op
.backanno
.end
--- Operating Point ---
V(n001):      5          voltage
V(n002):    1.37683      voltage
V(n003):    0.688416     voltage
I(D2):      0.00362317   device_current
I(D1):      0.00362317   device_current
I(R1):     -0.00362317   device_current
I(V1):     -0.00362317   device_current
```

Figura 24. Fisierul netlist și simularea .op pentru *Test11.jpg*

Netlistul LTspice și anazila .op pentru imaginea *Test2.jpg* din subdirectorul de testare reprezentat în Figura 25. și Figura 26. :

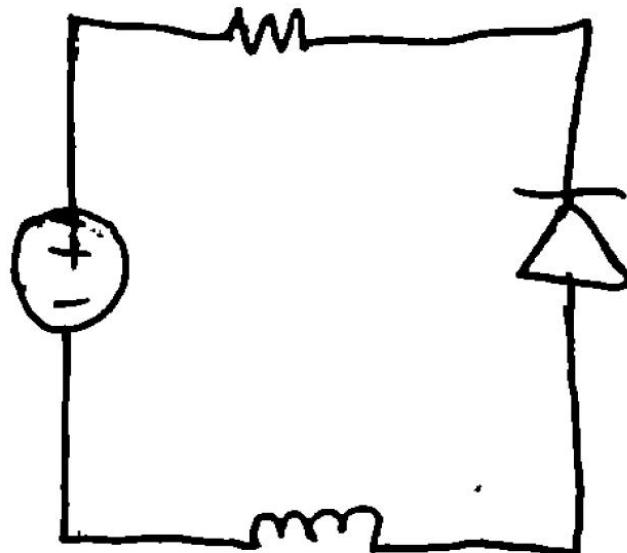


Figura 25. Imaginea *Test2.jpg*

```
* C:\Users\Username\Desktop\Validation netlist\Draft1.asc
V1 N001 0 5
R1 N002 N001 1k
D1 N002 N003 D
L1 N003 0 10m
.model D D
.lib C:\Users\Username\Documents\LTspiceXVII\lib\cmp\standard.dio
.op
.backanno
.end

--- Operating Point ---

V(n001) :      5          voltage
V(n002) :    0.692893    voltage
V(n003) :  4.30711e-006  voltage
I(D1) :     0.00430711  device_current
I(L1) :     0.00430711  device_current
I(R1) :    -0.00430711  device_current
I(V1) :    -0.00430711  device_current
```

Figura 26. Fișierul netlist și simularea .op pentru *Test2.jpg*

Netlistul LTspice și anazila .op pentru imaginea *Test9.jpg* din directorul de validare reprezentat în Figura 27. și Figura 28. :

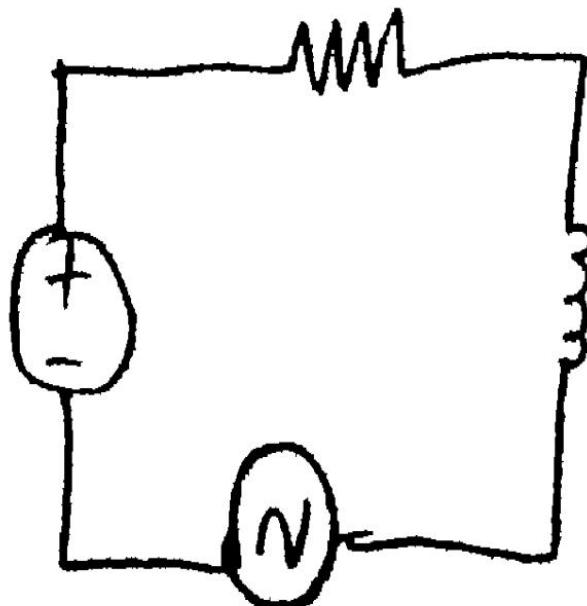


Figura 28. Imaginea *Test9.jpg*

```
* C:\Users\Username\Desktop\Validation netlist\Draft1.asc
V1 N001 0 5
R1 N002 N001 1k
L1 N002 N003 10m
V2 N003 0 SINE(0 1 1K)
.model D D
.lib C:\Users\Username\Documents\LTspiceXVII\lib\cmp\standard.dio
.op
.backanno
.end
    --- Operating Point ---
V(n001):      5          voltage
V(n002):  4.99999e-006  voltage
V(n003):      0          voltage
I(L1):        0.005      device_current
I(R1):       -0.005      device_current
I(V2):        0.005      device_current
I(V1):       -0.005      device_current
```

Figura 29. Fișierul netlist și simularea .op pentru *Test9.jpg*

Netlistul LTspice și anazila .op pentru imaginea *Test12.jpg* din directorul de validare reprezentat în Figura 30. și Figura 31. :

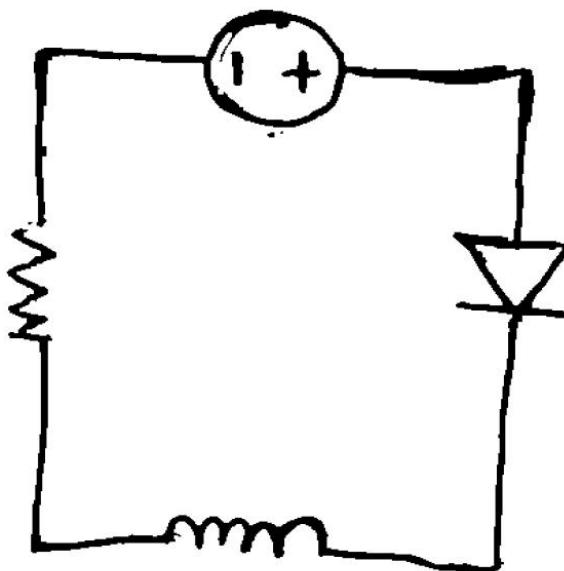


Figura 30. Imaginea *Test12.jpg*

```
* C:\Users\Username\Desktop\Validation netlist\Draft1.asc
R1 N001 0 1k
V1 N002 N001 5
D1 N002 N003 D
L1 N003 0 10m
.model D D
.lib C:\Users\Username\Documents\LTspiceXVII\lib\cmp\standard.dio
.op
.backanno
.end
--- Operating Point ---
V(n001) : -4.30711      voltage
V(n002) : 0.692893      voltage
V(n003) : 4.30711e-006  voltage
I (D1) : 0.00430711    device_current
I (L1) : 0.00430711    device_current
I (R1) : -0.00430711   device_current
I (V1) : -0.00430711   device_current
```

Figura 31. Fișierul netlist și simularea .op pentru *Test12.jpg*

Figurile respective ce conțin imaginile specifice setului de date *Datasenr2* și capturile fișierelor LTspice oferă informații despre detecția corectă a componentelor electronice de către modelul YOLOv5. Datorită acestui fapt scriptul *generate_netlist.py* mapează corect componentele electronice în structura fișierului de extensie NET, respectând și integrând regulile de netlistare

utilizate de LTspice, asigurând astfel o conversie corectă și consistentă a schițelor circuitelor electronice în netlisturi.

Datorită faptului că în script a fost integrată funcționalitatea atribuirii de valori clasice pentru componente, este posibilă vizualizarea simulărilor DC operating point, afișate automat după deschiderea fișierelor generate LTspice. O altă cauză ce a determinat succesul afișării analizei .op și absența erorilor returnate de LTspice este directiva adăugată de cod ‘.model D D’ și specificarea căii către librăria ce conține informația despre modelul diodei.

7. Concluzii

Lucrarea prezentată oferă o soluție robustă pentru automatizarea recunoașterii componentelor într-un circuit electronic, folosind tehnologia rețelelor neuronale conoluționale (CNN).

Aplicația dezvoltă un prototip funcțional, capabil să identifice corect tipurile de componente și pozițiile acestora, contribuind astfel la eficientizarea procesului de proiectare electronică. Avantajul principal al aplicației este facilitarea procesului de proiectare și simulare a circuitelor electronice prin automatizarea completă a generării fișierelor netlist direct din imagini, reducând astfel timpul și efortul necesar inginerilor pentru a transforma documentația grafică în modele de simularefuncționale.

În această lucrare, modelul ce folosește tehnologia rețelelor neuronale conoluționale (CNN) YOLOv5 a fost utilizat pentru recunoașterea componentelor electronice și identificarea poziției acestora într-un circuit. Datele din imagini au fost preprocesate înainte de a fi folosite pentru antrenarea modelului, incluzând operațiuni de normalizare, redimensionare și augmentare. Aceasta a asigurat o mai bună generalizare a modelului pe imagini noi și a permis obținerea unor performanțe mai bune în identificarea componentelor. Modelul a fost antrenat pe 2 seturi de date, iar apoi a fost realizată evaluarea performanței prin metrii precum *precision*, *recall*, și *mean Average Precision (mAP)* pentru a obține ponderile conexiunilor neurale compatibile cu aplicația dezvoltată. Limbajul de programare Python a fost folosit pentru a implementa întreaga soluție, începând de la descărcarea bibliotecilor necesare și sfârșind cu crearea scriptului de generare a netlisturilor LTspice. Pentru o implementare eficientă a aplicației de generare a netlisturilor au fost utilizate și regulile specifice aplicației LTspice.

Deși prototipul actual prezintă un potențial semnificativ, el este limitat de incapacitatea sa de a distinge direcția terminalelor componentelor, ceea ce este esențial pentru o simulare realistă. Această limitare, datorată folosirii YOLOv5 care nu suportă recunoașterea terminalelor, reduce acuratețea și aplicabilitatea modelului în scenarii complexe și necesită soluții pentru dezvoltări.

Pentru a îmbunătăți performanța aplicației și a o adapta la nevoile circuitelor mai complexe, o soluție robustă ar fi implementarea codului ce folosește maparea nodurilor și a componentelor matricială. O altă soluție pentru dezvoltarea aplicației este adăugarea adnotărilor și etichetelor pentru clasificare orientării terminalelor în setul de date. Direcția terminalelor reprezintă polaritatea surselor de tensiune, a condensatorilor și a tranzistoarelor, anodul și catodul diodei și polaritatea unor modele de condensator, astfel orientarea terminalelor joacă un rol esențial pentru a obține rezultate precise pentru simulări electrice. În plus, adăugarea informațiilor în setul de date despre modele SPICE ale componentelor ar ajuta la setarea parametrilor din librăriile LTspice, fapt ce ar spori precizia simulării pentru circuitele detectate.

8. Bibliografie

- [1] <https://arxiv.org/abs/2402.08797>
- [2] <https://en.wikipedia.org/wiki/FLOPS>
- [3] Ian Goodfellow, Yoshua Bengio, Aaron Courville. "Deep Learning", MIT Press, 2016
- [4] Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6), 386–408.
- [5] [Neural network \(machine learning\) - Wikipedia](#)
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press, 2016.
- [7] Simon Haykin, Neural Networks and Learning Machines, 3rd Edition, Pearson, 2009.
- [8] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).
- [9] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.
- [10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
- [11] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.
- [12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
- [13] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2020.
- [14] Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. "Path Aggregation Network for Instance Segmentation." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
- [15] Ultralytics YOLOv5 Documentation. "YOLOv5: An in-depth Guide." Ultralytics, 2020.
- [16] Lin, T. Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., ... & Zitnick, C. L. "Microsoft COCO: Common Objects in Context." In European Conference on Computer Vision, 2014.
- [17] <https://pytorch.org/docs/stable/index.html>
- [18] <https://numpy.org/doc/>
- [19] <https://docs.scipy.org/doc/scipy/index.html>
- [20] <https://github.com/>
- [21] <https://github.com/ultralytics/yolov5>
- [22] <https://docs.microsoft.com/en-us/powershell/>
- [23] <https://docs.ultralytics.com/>
- [24] <https://onnx.ai/>
- [25] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. Proceedings of the European Conference on Computer Vision (ECCV).
- [26] Shaoqing Ren, Kaiming He, Ross B. Girshick, și Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2015.
- [27] https://github.com/tensorflow/models/tree/master/research/object_detection
- [28] <https://github.com/facebookresearch/detectron2>
- [29] Girshick, R. (2015). Fast R-CNN. Proceedings of the IEEE International Conference on Computer Vision (ICCV)

- [30] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.
- [31] NVIDIA. (2021). CUDA Toolkit Documentation. NVIDIA
- [32] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems (NeurIPS).
- [34] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
- [35] Zhu, X., & Goldberg, A. B. (2009). Introduction to Semi-Supervised Learning. Morgan & Claypool Publishers
- [36] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.
- [37] Nielsen, M. A. (2015). Neural Networks and Deep Learning. Determination Press.
- [38] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection
- [39] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*, 323(6088), 533-536.
- [40] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2015). The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1), 98-136.
- [41] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958.
- [42] Ng, A. Y. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. *Proceedings of the twenty-first international conference on Machine learning (ICML)*, 78.
- [43] <https://www.anaconda.com/download/>
- [44] <https://www.jetbrains.com/pycharm/download/?section=windows#section=windows>
- [45] <https://developer.nvidia.com/cuda-downloads>
- [46] <https://developer.nvidia.com/cudnn>
- [47] <https://universe.roboflow.com/rp-project/circuit-recognition>

9. Anexe

9.1 *train_yolov5.py*

```
Import os
from torch.utils.tensorboard import SummaryWriter

yolov5_dir = "C:/Users/Username/PycharmProjects/Yolov5Train/yolov5"
yaml_path = "E:/DatasetNr2/data.yaml"

writer = SummaryWriter(log_dir=os.path.join(yolov5_dir, 'runs', 'train'))
os.system(f"python {yolov5_dir}/train.py --img 640 --batch 16 --epochs 50 --data {yaml_path} --weights yolov5s.pt")
writer.close()
```

9.2 *generate_netlist.py*

```
import torch
import os

model = torch.hub.load('ultralytics/yolov5', 'custom',
path='C:/Users/Username/PycharmProjects/Yolov5Train/yolov5/runs/train/exp11/weights/best.pt')

img = 'C:/Users/Username/Desktop/Validation netlist/TestImage.jpg'

# Get the detection results
results = model(img)

components = []
for *box, conf, cls in results.xyxy[0]: # For each detection
    x1, y1, x2, y2 = box
    component_class = model.names[int(cls)]
    x_center = (x1 + x2) / 2
    y_center = (y1 + y2) / 2
    components.append({'class': component_class, 'x': x_center, 'y': y_center})

counts = {'resistor': 1, 'inductor': 1, 'capacitor': 1, 'dc': 1, 'diode': 1}
counts['ac'] = counts['dc'] + 1

def convert_class_to_netlist(component_class, count):
    if component_class == 'resistor':
        return f'R{count}', '1k'
    elif component_class == 'inductor':
        return f'L{count}', '10m'
    elif component_class == 'capacitor':
```

```

    return f'C{count}', '10m'
elif component_class in ['dc', 'ac']: # Common treatment for DC and AC
    return f'V{count}', '5' if component_class == 'dc' else 'SINE(0 1 1K)'
elif component_class == 'diode':
    return f'D{count}', 'D'
return None, None

components = sorted(components, key=lambda c: c['x'])

# Identify the corner components based on sorted positions
left_component = components[0]
right_component = components[-1]
middle_components = sorted(components[1:-1], key=lambda c: c['y'])
top_component = middle_components[0]
bottom_component = middle_components[-1]

# Prepare the netlist
netlist = []
ordered_components = [left_component, top_component, right_component, bottom_component]

current_node = 'N001'
next_node = 'N002'

for comp in ordered_components:
    component_type, default_value = convert_class_to_netlist(comp['class'], counts[comp['class']])

    if comp == left_component:
        node1, node2 = current_node, '0'
        value = input(
            f'{component_type} found between nodes {node1} and {node2}. Please set the value (e.g.,\n{default_value}): ')
    elif comp == top_component:
        node1, node2 = next_node, current_node
        value = input(
            f'{component_type} found between nodes {node1} and {node2}. Please set the value (e.g.,\n{default_value}): ')
    elif comp == right_component:
        node1, node2 = next_node, 'N003'
        value = input(
            f'{component_type} found between nodes {node1} and {node2}. Please set the value (e.g.,\n{default_value}): ')
    elif comp == bottom_component:
        node1, node2 = 'N003', '0'
        value = input(
            f'{component_type} found between nodes {node1} and {node2}. Please set the value (e.g.,\n{default_value}): ')

    if comp['class'] == 'ac':
        dc_offset = input(

```

```

f"Alternating Voltage Source {component_type} found between nodes {node1} and
{node2}. Please set the DC offset (e.g., 0V): "
amplitude = input("Please set the Amplitude (e.g., 1V): ")
frequency = input("Please set the Frequency (e.g., 1kHz): ")
value = f"SINE({dc_offset} {amplitude} {frequency})"

if comp['class'] in ['ac', 'dc']:
    plus_node = input(
        f"Please set the plus terminal connection of {component_type} (must be N_higher for a
normal direction of connection): ")
    minus_node = input(
        f"Please set the minus terminal connection of {component_type} (must be N_lower for a
normal direction of connection): ")
    netlist.append(f"{component_type} {plus_node} {minus_node} {value}")

elif comp['class'] == 'diode':
    anode_node = input(
        f"Please set the anode connection of {component_type} (for the normal direction of
connection must be N_lower): ")
    cathode_node = input(
        f"Please set the cathode connection of {component_type} (for the normal direction of
connection must be N_higher): ")
    netlist.append(f"{component_type} {anode_node} {cathode_node} {value}")

else:
    netlist.append(f"{component_type} {node1} {node2} {value}")

counts[comp['class']] += 1

include_op = input("Would you like to include the .op directive in the netlist? (yes/no): ")
if include_op.lower() == 'yes':
    netlist.append(".op")

netlist.append(".model D D")
netlist.append(".lib C:\\\\Users\\\\Username\\\\Documents\\\\LTspiceXVII\\\\lib\\\\cmp\\\\standard.dio")
netlist.append(".backanno")
netlist.append(".end")

output_file = "C:/Users/Username/Desktop/Validation netlist/Generated.net"
with open(output_file, 'w') as f:
    f.write("* C:\\\\Users\\\\Username\\\\Desktop\\\\Validation netlist\\\\Draft1.asc\\n")
    for line in netlist:
        f.write(line + '\\n')

print(f"Netlist saved to {output_file}")

```