FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Recognising and Modelling Electrical Circuit Diagrams

Diogo Henrique Azevedo Moreira

Master in Electrical and Computer Engineering

Supervisor: Professor Armando Sousa

Second Supervisor: Professor Mário Alves

October 31, 2024

Abstract

The main goal of this dissertation is to extend the functionality of the U=RIsolve tool, a website for self-learning the fundamentals of electric circuit analysis. This web application supports students self-learning circuit analysis and simplification methods by providing a step-by-step resolution of a linear circuit defined by its schematics.

Typically, "solving" a circuit involves the determination of its most relevant "variables", which usually are the current in every branch and the voltage in every node. In this process, a graphical representation of the circuit is paramount, including the identification of all electrical components (loads and power sources) and their topology (branches and nodes). This circuit diagram (or schematic) defines the analytical model of the electrical circuit, which in turn allows the application of the analysis/simplification method.

Users of the U=RIsolve web platform first have to "draw" the circuit diagram in a circuit simulator to generate its mathematical model (*netlist* file), which then serves as input to the algorithms that implement each analysis method (e.g. Nodal Voltage or Mesh Current). Our objective in this dissertation is to complement this possibility by using the circuit diagram image as input and generating its model in JSON, to feed to the circuit analysis modules. Computer Vision (CV) techniques and Artificial Intelligence (AI) algorithms are used to generate the mentioned model. Classical techniques, as well as YOLOV8n-seg are used to identify and segment pixels that belong to the symbols of the electrical components present in the image. Optical Character Recognition (OCR) is also employed to recognise the alphanumeric characters that identify the components and values in the circuit, and classic computer vision techniques are used to preprocess the images and find the electrical connections.

A dataset was built that includes 200 images of circuits, each including wirings and from 3 to 12 active and/or passive components and respective labels (component alphanumerical label, value and unit). All images were hand-labelled adquately for image segmentation. The images are most interesting for segmentation of components and come from 3 computer programs likely to be used by users of the web platform: QUCS, LTspice and Microsoft Visio. Another dataset focuses solely on polarized components and includes 150 images of DC and AC voltage and current sources. To recognize the labels, OCR accuracy is most often above 90 % and consistently above 59 %.

Our software module performs well for circuit schematics created in computer-based circuit editors such as LTspice or QUCS, achieving around 80 percent accuracy in the components identification. The OCR module detects characters and numbers on computer representations and can also analyse handwriting. Our software module was additionally tested on 2 hand draw circuits of 5 components, featuring an accuracy above 60 %.

Recognition

This work was carried out with the support of the Instituto Superior de Engenharia do Porto: ISEP and supervised at the institution by Professor Mario Alves.

Professor Lino Sousa collaborated with the supervising team in carrying out this work.

United Nations Sustainable Development Goals

In 2015, the United Nations created the "2030 Agenda for Sustainable Development," outlining 17 Sustainable Development Goals and 169 Targets to address global challenges. These goals require collaboration between governments, businesses, and academic institutions to create a more sustainable and fair future for all [39]. Figure 1 represents the 17 Sustainable Development Goals.

Figure 1: United Nations Sustainable Development Goals [39].

The dissertation is directly aligned with two of these goals:

- **Goal 4 - Quality Education:** This work aims at enhancing the teaching and learning of electrical circuits analysis, by creating a self-learning and teaching tool.
- **Goal 12 - Life on Land:** This software can reduce the need for paper when analyzing electrical diagrams. By incorporating a module that can recognize and analyze electrical circuits, the U=RIsolve platform will be capable of taking an image of an electrical circuit and analyzing it. This saves paper and time for the user, who would otherwise have to perform calculations and possibly create a new circuit representation by hand.

This dissertation illustrates the indirect contribution of this work to the United Nations Sustainable Development Goals. Education is pivotal in sustainable development, particularly in technical disciplines like electrical circuit analysis. Enhancing educational resources and promoting access to technical knowledge can empower students and professionals to devise innovative and sustainable solutions to environmental issues, such as improving energy efficiency and advancing green technologies. Furthermore, progress in education and technology has the potential to foster more sustainable behaviours and the responsible utilization of natural resources.

Agradecimentos

Ao Prof. Doutor Mário Alves e ao Prof. Doutor Armando Jorge Sousa , que sempre esteveram disponíveis para me ajudar nesta fase final do meu percurso e que ultrapassaram largamente o papel de orientadores por diversas ocasiões.

Ao Professor Lino Sousa, pela sua dedicação e conhecimentos que me transmitiu para a realização deste trabalho.

A todos aqueles que de certa forma me ajudaram neste percurso.

À Andreia, pessoa que está sempre presente nas minhas aventuras e desafios, mas que ficou lesada com o desenvolvimento deste trabalho a quem devo tempo e paciência que lhe foram retirados.

Por fim, dedico esta dissertação à minha mãe, que acompanhou neste processo, com muito carinho e apoio. Obrigada por toda a paciência e amor incondicional, que foram indispensáveis para ter sucesso nesta etapa da minha vida.

Diogo Henrique Azevedo Moreira

"Give me six hours to chop down a tree and I will spend the first four sharpening the axe."

Abraham Lincoln

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AC	Alternating Current
AI	Artificial Intelligence
AUC	Area Under the Curve
CNN	Convolutional Neural Network
DC	Direct Current
FN	False Negative
FP	False Positive
FPR	False Positive Rate
IoU	Intersection over Union
JSON	JavaScript Object Notation
mAP	Mean Average Precision
OCR	Optical Character Recognition
RGB	Red, Green and Blue
RGBA	Red, Green, Blue and Alpha
ROC	Receiver Operating Characteristic
ROI	Region of Interest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
TPR	True Positive Rate
VGG	Visual Geometry Group
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Research Context and Motivation

Diagrams are a popular and widely used way of graphically representing systems in engineering. In electrical engineering, there are standard ways of graphically representing electric circuits of different types, from power distribution systems to low-power electronics. Traditionally, an electric circuit is graphically represented in the so-called "circuit diagram" or "circuit schematics" following specific rules and symbology for drawing power sources (voltage, current), passive components (resistance, inductance, capacitance), non-linear and active components (e.g. Ampop, diode, transistor), measuring instruments (e.g. Wattmeter, Ammeter, Voltmeter, Oscilloscope), circuit breaking/protection components (e.g. switch, fuse, circuit-breaker), electrical earth/ground, connections and conductors, etc.

In this context, this dissertation aims at designing a software tool capable of interpreting and converting an electrical schematic (image) into its analytical model. Ultimately, this model serves as an input to the *U=RIsolve* [40] web application, a framework for teaching and self-learning the fundamentals of DC and AC circuits. Our software module will avoid the user/student having to draw/edit the circuit schematics, allowing him/her to just use its picture (captured by photographing an exercise book or making a "print screen" in the PC for example).

The input to this software module is supposed to be an image file of a circuit diagram, created through a computer-based application (circuit editor or simulator), and the corresponding image can be captured by photograph or already pre-stored in a file with a standardised format. We are particularly interested in linear DC and AC circuits featuring passive components (R, L, C), which are usually the baseline for learning the fundamental laws and algorithms for circuit analysis and simplification, typically addressed in the first curricular year of *Electrical and Computer Engineering*-related courses.

The software tool developed in the context of this Thesis is thus expected to be integrated in the *U=RIsolve* [40] web application.

Figure 1.1 illustrates the process of analysing electrical circuits in the *U=RIsolve* [40]. It shows how users can create and submit netlists or schematic images, which then undergo file validation

and component recognition. Following this, various types of analysis are executed, and the results are presented.

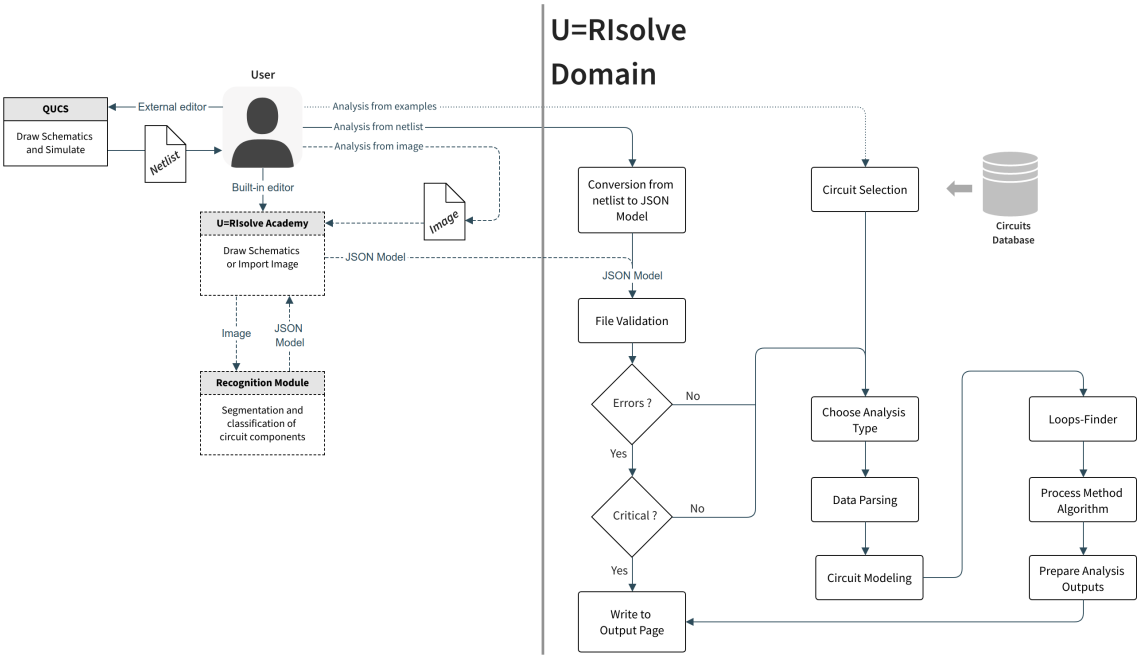


Figure 1.1: U=RIsolve flow diagram.

1.2 Overall objective and research approach

The main objective of this Thesis was to conceive a software module that can visually process the image of a circuit schematics, characterize its topology, classify its components and recognize their labels and values/units, and build an analytical model that represents the logic and topology of the (same) circuit.

With this aim, we organized our work in the following approach:

- Analysis of the state-of-the-art in electrical circuit/components classification and image segmentation techniques.
- Creation of one dataset with computer-edited electrical circuit schematics.
- Selection and implementation of the algorithms for detecting and extracting the following elements:
  - Passive components and power sources, as defined in the dataset.
  - Polarities of voltage and current sources (DC and AC)
  - Labels of the components, in particular their name/identification (e.g. R1) and numerical value (e.g., 0.47 kΩ).

  - Interconnection between the different components of the circuit, including nodes and conductor wires.

- Generation of an analytical model of the circuit, compatible to feed the *U=RIsolve* app.
- Performance evaluation and fine-tuning of the algorithms, in a way to assess and optimize their accuracy and identify limitations and directions for potential improvements.

1.3 High-level software architecture

A high-level architecture of the software is depicted in Figure 1.2, which reflects the following operation logic and sequence:

1. The user uploads an image file of the electrical circuit to be received and processed by our software application.
2. The application is responsible for segmenting the image, Block 2 in Figure 1.2. Identification of nodes, electrical connections, text and components. In addition to identifying the components, it is also necessary to classify them.
3. The segmentation results enable the identification of the text. Block 3 in Figure 1.2 involves the recognising and interpreting the text.
4. The application returns a data model with information about the circuit schematic supported by the U=RIsolve web app, Block 4 in Figure 1.2.
5. The U=RIsolve application is now ready to provide a step-by-step analysis of the circuit (one method needs to be solved)

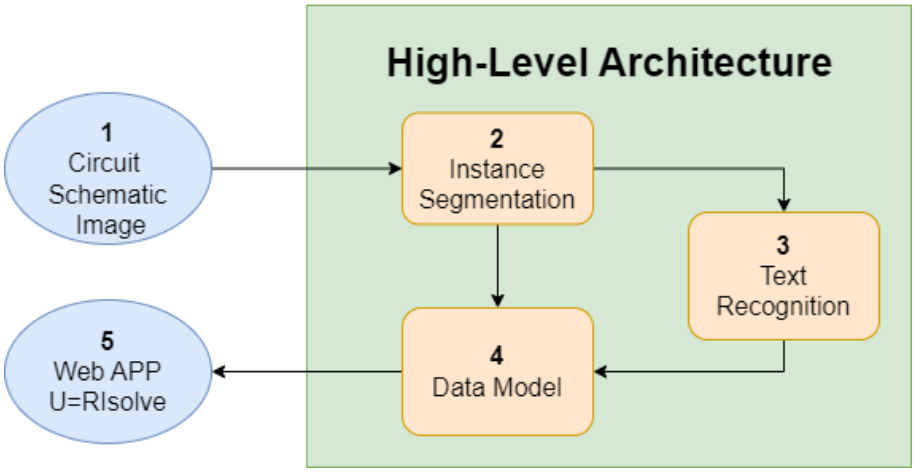


Figure 1.2: High-level system architecture

This software architecture and functionality, together with the research approach, paved the way for the study of the state-of-the-art and comparison of different technology that are summarized in Chapter 2.

1.4 Contributions

To our best knowledge, and building on the analysis of state-of-the-art and related works on electric circuit schematics recognition, namely on image segmentation, computer vision and machine learning techniques applied to image processing, we believe that this work embeds the following contributions:

- Creation of a "segmentation" dataset with images of electrical circuit diagrams, using the QUCS and LTspice simulators and taken from exercise books. The dataset does not include hand-drawn images of electrical circuits. These images were annotated pixel by pixel to obtain the mask of each element using the Robflow platform.
- Creation of a "polarity" dataset, with annotated images of power sources (voltage and current, DC and AC), to improve the identification of their electrical polarity.
- Training and testing of YOLOv8 [19] models to segment electric circuits and find the power sources polarity.
- Adaptation and implementation of a data model (in JSON [32]) compatible with the input and circuit editor of the U=RIsolve application [40] in collaboration [33].

1.5 Document Organization

The remainder of this document is structured as follows: Chapter 2 presents the literature review on image processing and image recognition and state-of-the-art methodologies applied to electrical circuit recognition; Chapter 3 elaborates on the software architecture and implementation aspects; Chapter 4 presents six case-studies created by QUCS simulator, LTspice simulator and hand-draw in order to evaluate the proposed methodology and software tool for different circuit complexity and editing platforms; Chapter 5 provides some final considerations and outlines some directions for future developments and improvements.

Chapter 2

Technological Background and Related Work

This chapter presents methods and algorithms from the field of computer vision that are fundamental to the software development for this project. It demonstrates different image processing techniques for recognising and classifying electrical circuits and describes some existing computer vision software tools.

2.1 Image Processing and "Classic" Computer Vision Techniques

Image processing involves applying computer algorithms to manipulate digital images to enhance an existing image or to sift out important information from it. This process plays a crucial role in various computer vision applications, including deep learning systems [8].

2.1.1 Digital Image

Computers see an image as a multidimensional matrix made up of picture elements, better known as pixels. The pixel is the smallest unit in the image. It has a specific position and contains values that indicate its colour. An image is represented by its dimensions, which are about the number of pixels, i.e., it has a width (number of columns in the matrix) and a height (number of rows in the matrix). So, if the dimensions of an image are 480 * 320, the image has 153600 pixels. The representation of an image for a computer follows a referential in which the origin is at the top left-hand corner, as shown in the Figure 2.1:

From a computer's point of view, images can be of the following types:

- **Binary Image** — Image in which each pixel can have only two values: 0 (represents the colour black) and 1 (represents the colour white).
- **Grayscale Image** — Or monochrome image or 8-bit image, each pixel can have a value between 0 and 255, where 0 represents black and 255 represents white, the other values are shades of grey.

- **RGB Image** — In the real world, we are used to seeing images in color, but for computers, they are represented by 16-bit matrices. An RGB pixel is made up of three values, with (0, 0, 0) representing black and (255, 255, 255) representing white. Each value is called a channel, and (255, 0, 0) represents red (R), (0, 255, 0) represents green (G), and (0, 0, 255) represents blue (B). By adjusting these channels, it is possible to create any colour we know.
- **RGBA Image** — RGBA images are RGB images with an additional channel called "alpha", which controls the image's opacity, ranging from 0 to 1.

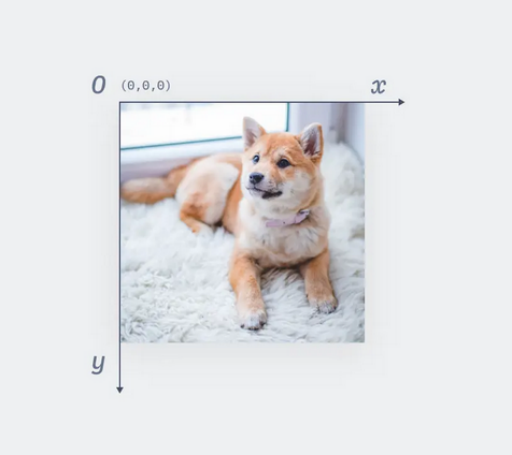


Figure 2.1: System coordinate used in an image [6].

2.1.2 Smoothing Filters

The smoothing filters, often called blurring, result in an image blurred and less sharp appearance. Although this is generally not desired in photographs, it is extremely useful in image processing tasks. Smoothing or blurring is a typical pre-processing step in computer vision and image processing.

There are various reasons for its application, the most common being the reduction of high-frequency content, such as noise and fine details, making it easier to identify objects of interest while keeping significant structural elements remain emphasised. The main smoothing methods are explored below.

- **Average Filter**

  The average filter, or normalized box filter, is a simple yet effective method for reducing noise in digital images. It operates by replacing each pixel value in the image with the average value of its neighbouring pixels, defined by a kernel. As the size of the kernel increases, the image becomes progressively more blurred. This can lead to a point where the edges of important structural objects are lost in the image. Figure 2.2 shows the application of the average filter with a 3 by 3 kernel.

Figure 2.2: The average filter: original image on the left left; output image on the right [15]

.

- **Gaussian Filter**

  The Gaussian filter is used to remove noise following a Gaussian distribution. This method results in less blur compared to the average filter, allowing for better preservation of existing edges in the image [25] because the Gaussian kernel gives much more weight to the pixels in its centre than its boundaries, as shown in equation 2.1. Figure 2.3 shows the application of gaussian filter.

$$K = \frac{1}{141}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 7 & 26 & 41 & 26 & 7 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{2.1}$$

Figure 2.3: The Gaussian filter: original image on the left left; output image on the right [15].

- **Median Filter**

  The median filter calculates the pixel value by taking the median based on a specified kernel size. This process is applied to each position of the window along the image or signal [15]. Unlike the average filter, the median filter effectively preserves sharp edges in an image, as the median is not influenced by extreme values as much as the average. Figure 2.4 illustrates the application of this filter.

Figure 2.4: The Median filter: original image on the left left; output image on the right [15].

- **Bilateral Smoothing Filter**

  The bilateral smoothing filter, also known as edge-preserving smoothing, reduces noise in an image while maintaining the edges of objects. It involves using two Gaussian distributions [10]:

  - **Spatial distribution** — This considers only spatial neighbours, meaning pixels that are close together in the image's coordinate space (x, y).
  - **Intensity distribution** — This models the intensity of the pixels in the neighbourhood, ensuring that only pixels with similar amplitudes are included in the filtering calculation.

  When pixels in the same neighbourhood have similar values, it generally indicates they belong to the same object. However, if two pixels in the same neighbourhood have different values, this can indicate the presence of an object border or boundary. The bilateral filter can preserve edges in the image while reducing noise, as shown in Figure 2.5. However, it should be noted that this method requires more processing time than other filtering methods.

Figure 2.5: The Bilateral Smoothing filter: original image on the left left; output image on the right [9].

2.1.3 Threshold Operations

The threshold operation is used to segment an image. The main aim of the threshold is to simplify the image by transforming a grey-scale image into a binary one. The more used Threshold Operations are explored below:

– **Simple Thresholding**

To demonstrate how these processes work, let's consider an input image with the pixel intensity defined as src(x, y), shown in Figure 2.6 The blue line sets the threshold value on the graph.

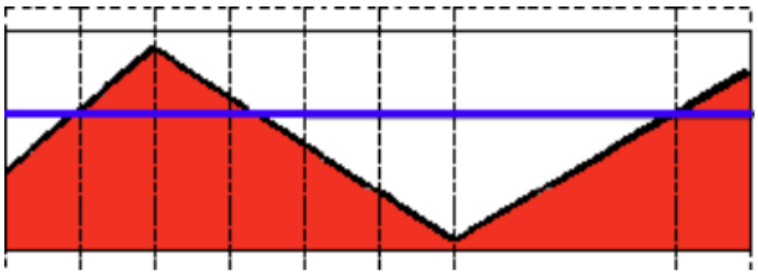


Figure 2.6: Grey-scale image pixel intensity graph for thresholding [17].

The simple threshold has the following operating modes:

∗ **Binary Threshold**

The binary threshold method is defined using Equation 2.2. In this equation, *dst(x, y)* represents the output image, and "max" is the chosen maximum value. When the amplitude of an image pixel is greater than the threshold, its value is set to "max"; otherwise, it is set to zero, as represented in Figure 2.7.

$$\text{dst}(x,y) = \begin{cases} \text{maxVal} & \text{if src}(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

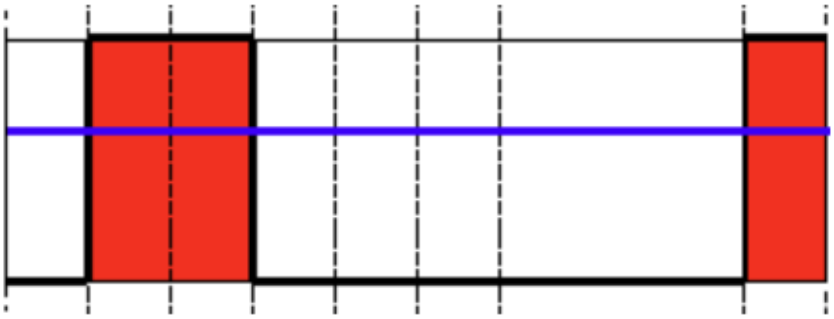


Figure 2.7: Results of the binary threshold application [17].

* **Inverted Binary Threshold**
  The inverted binary threshold corresponds exactly to the inverse process of the and is expressed by Equation 2.3. Therefore, an image pixel is set to zero if its intensity exceeds the threshold; otherwise, it is changed to the maximum value and represented in Figure 2.8.

$$\text{dst}(x,y) = \begin{cases} 0 & \text{if src}(x,y) > \text{thresh} \\ \text{maxVal} & \text{otherwise} \end{cases} \tag{2.3}$$

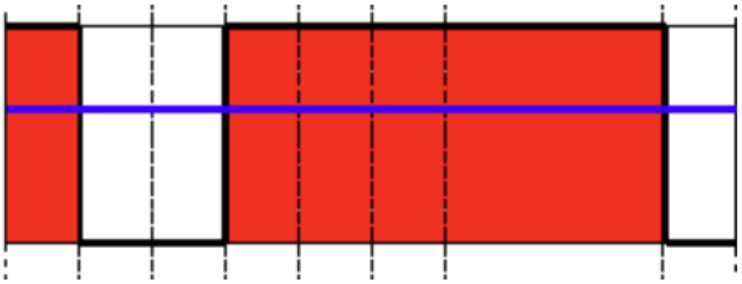


Figure 2.8: Results of the inverted binary threshold application[17].

* **truncated Threshold**
  The truncated threshold allows the limitation of pixel intensities above a certain threshold while retaining the intensities of the other pixels. This can be proven by Equation 2.4 and demonstrated by Figure 2.9.

$$\text{dst}(x,y) = \begin{cases} \text{threshold} & \text{if src}(x,y) > \text{thresh} \\ \text{src}(x,y) & \text{otherwise} \end{cases} \tag{2.4}$$

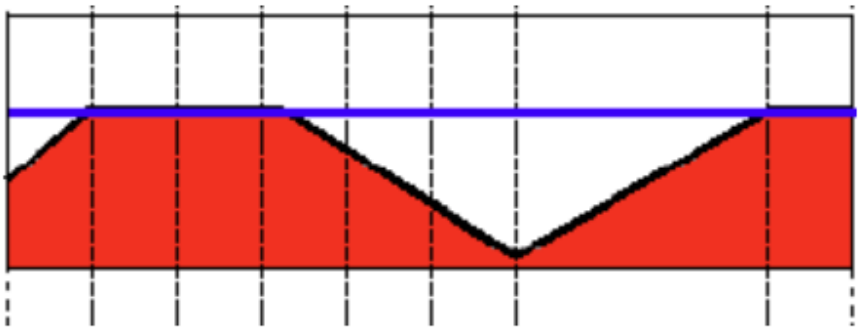


Figure 2.9: Results of the truncated threshold application [17].

* **Threshold to zero**
  The threshold to zero method implies that when the amplitude of an image pixel is less than the threshold, its value is changed to zero, without modifying the other pixels. This is stated by Equation 2.5 and shown in Figure 2.10.

$$\text{dst}(x,y) = \begin{cases} \text{src}(x,y) & \text{if } \text{src}(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

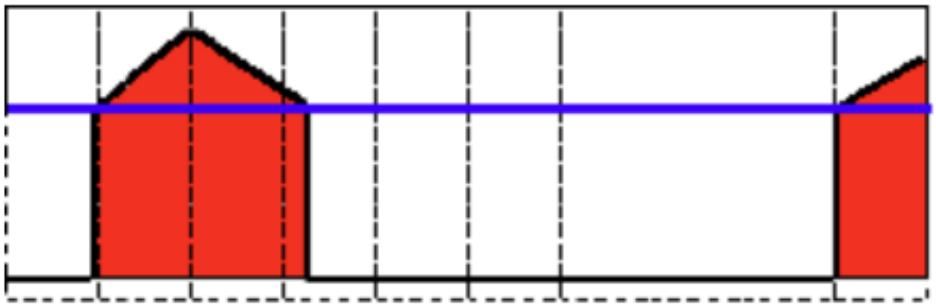


Figure 2.10: Results of the threshold to zero application [17].

* **Inverted threshold to zero**
  As expressed by Equation 2.6, the threshold for inverted zero defines a pixel as zero when its intensity is greater than the threshold. This represents the opposite behaviour of the threshold for zero, as illustrated in Figure 2.11.

$$\text{dst}(x,y) = \begin{cases} 0 & \text{if } \text{src}(x,y) > \text{thresh} \\ \text{src}(x,y) & \text{otherwise} \end{cases} \tag{2.6}$$

Figure 2.12 shows how simple thresholding is applied to a grey scale image.

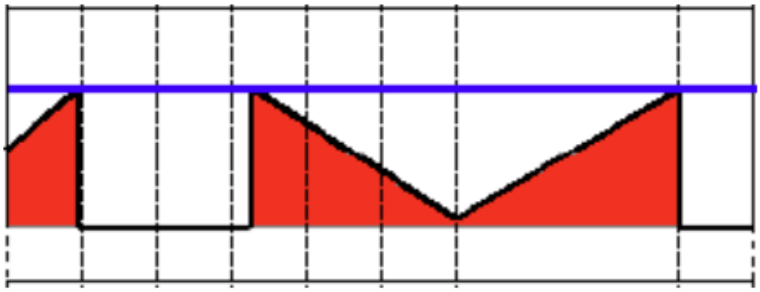


Figure 2.11: Results of the inverted threshold to zero application [17].

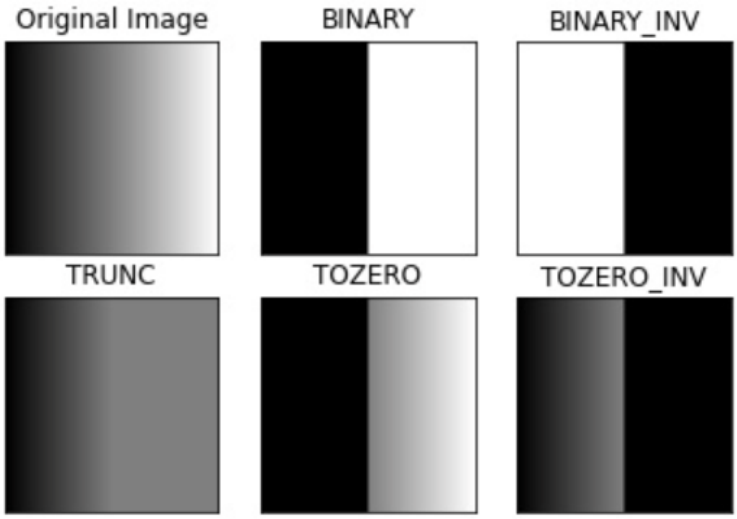


Figure 2.12: Simple thresholding [28].

– **Adaptive Thresholding**

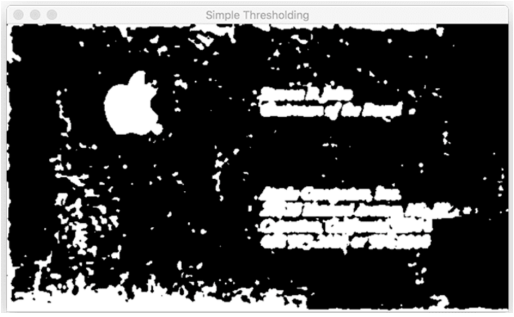
  The methods mentioned above use a single threshold value for the entire image. However, this can work well in one area of the image and fail completely in another.

  Adaptive thresholding is a technique that aims to solve problems of non-uniform illumination in images. This method divides the image into small regions, and a threshold value is calculated separately for each region. This is especially useful when the lighting varies significantly in the image, allowing each region to have its own threshold value.

  The threshold calculation is based on a statistical analysis of the different regions of the image, commonly using the arithmetic mean or the Gaussian mean. With the arithmetic mean, all the pixels in the neighbourhood contribute equally to the threshold calculation. On the other hand, with Gaussian averaging, the pixels furthest from the center of the region have less influence on the calculation, resulting in smoother and more accurate segmentation.

In Figure 2.13, the goal is to identify the symbols and text in the input image. Figure 4.24 displays the results, showing that adaptive thresholding provides the most effective solution for the issue at hand. Simple thresholding results in a very noisy image, while Otsu thresholding only allows for the identification of the symbol in the image.

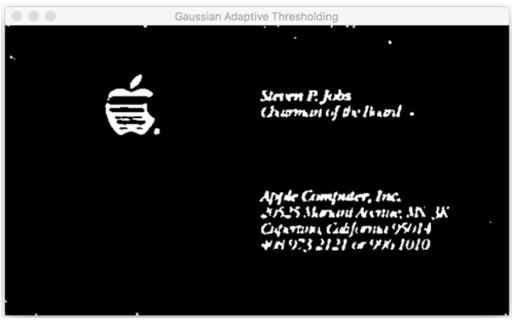
Figure 2.13: Original image for adaptive thresholding [1].



(a) Simple Thresholding

(b) Otsu Thresholding

(c) Mean Adaptive Thresholding



(d) Gaussian Adaptive Thresholding

Figure 2.14: Possible results [1].

2.1.4 Morphological Transformations

Morphological operations are essential techniques in image processing used to analyze and manipulate structures in images. They are critical in areas such as Optical Character Recognition (OCR), machine learning, and deep learning.

Erosion is a morphological operation that removes pixels from the edges of objects and is effective for eliminating noise and separating connected objects. On the other hand, dilation adds pixels to the edges of objects, fills in gaps, and connects disjointed components. Figure 2.15 demonstrates the use of these two morphological operations.

(a) Original image (b) Erosion (c) Dilation

Figure 2.15: Application of erosion and dilation [18].

Opening, which consists of an erosion followed by a dilation, is used to remove small objects and smooth out contours without significantly altering the area of the objects. Closing, which is a dilation followed by erosion, is effective for closing small holes inside objects and connecting nearby objects. Figure 2.16 represents the opening operation, and Figure 2.17 demonstrates the closing operation.

Figure 2.16: Dilation application. Original image on the left, result image on the right [18].

Figure 2.17: Closing application. Original image on the left, result image on the right [18].

2.2 Image Segmentation and AI

This section explores the details of CNNs, including their architecture and the critical processes of feature learning and classification. Accurate and reliable evaluation metrics are crucial for assessing the performance of these advanced models. Metrics such as accuracy, precision, recall, F1-score, and mean average precision (mAP) comprehensively evaluate a model's effectiveness. In addition, tools like confusion matrices, Intersection over Union (IoU), and Receiver Operating Characteristic (ROC) curves aid in analysing and visualising the model's predictive capabilities. This section also discusses the importance of inference time in real-time applications, emphasizing the need for efficient processing. By combining theoretical insights with practical evaluation techniques, this section aims to provide a comprehensive overview of CNNs and the metrics that determine their success in image recognition.

2.2.1 CNNs: *Convolutional Neural Networks*

Convolutional Neural Networks (CNNs) are a primary tool for instance segmentation and classification. They can learn features from the ROIs of the original image, enabling the model to distinguish between different components and labels. Creating a well-trained model with a strong database is essential to achieve a high success rate in classifying components.

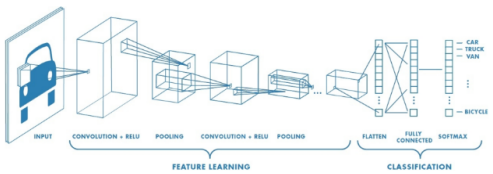


Figure 2.18: Form of a typical convolutional network [5].

As Figure 2.18 shows, CNNs can be divided into two phases: feature learning and classification:

1. **Feature learning**

- **Convolutional Layers** — Convolutional layers serve as feature detectors for edges, textures, and simple shapes.
- **Pooling Layers** — Pooling layers, such as max pooling or average pooling, are used to simplify the information from the previous layer.

2. **Classification**

- **Flattens** — The flatten operation converts this multi-dimensional array into a one-dimensional array, which is then fed into fully connected layers for further processing.
- **Fully connected** — Used in the final layers of a neural network where the network is required to make predictions or decisions based on the learned features from the earlier layers.

The activation functions bring non-linearity to the system during the feature learning and classification phases. Several activation functions exist, such as sigmoid, tanh, and softmax. However, the most commonly used one for convolutional networks is Relu [5].

2.2.2 Metrics to analyse results

Evaluation metrics play a critical role in quantifying the performance and effectiveness of image recognition models. They provide a measurable framework for comparing different models, adjusting hyperparameters, and ensuring the model aligns with the application's unique requirements.

Must consider the matrix in the Figure 2.19 to better understand the evaluation metrics and understand the following terms:

- Each element evaluated by the neural network receives a label indicating whether it is true or false and positive or negative.
- **True Positive (TP)** — The number of times the model correctly identified a positive sample as positive.
- **True Negative (TN)** — The number of times the model incorrectly identified a positive sample as negative.
- **False Positive (FP)** — The number of times the model correctly identified a negative sample as positive.
- **False Negative (FN)** — The number of times the model correctly identified a negative sample as negative.

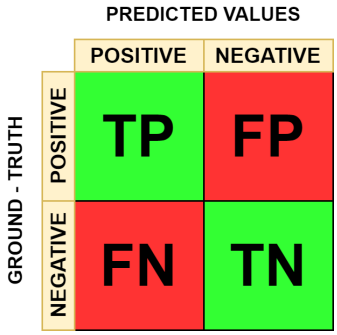


Figure 2.19: Matrix to understand the relationship between TP, TN, FP and FN [6].

2.2.3 K-folding

K-folding or K-fold Cross-Validation is a cross-validation technique widely used to evaluate the performance of machine learning models. In image processing, K-folding ensures that a computer vision model is evaluated robustly and reliably.

Figure 2.20 demonstrates the process of K-Fold Cross-Validation, which involves the following steps:

1. Divide the entire dataset into K equal subsets, which are referred to as folds in this technique. In this case, K is equal to 5.

2. The model trains the data K times and varies the training and test folders.

3. For each fold, metrics are calculated and averaged for overall performance.

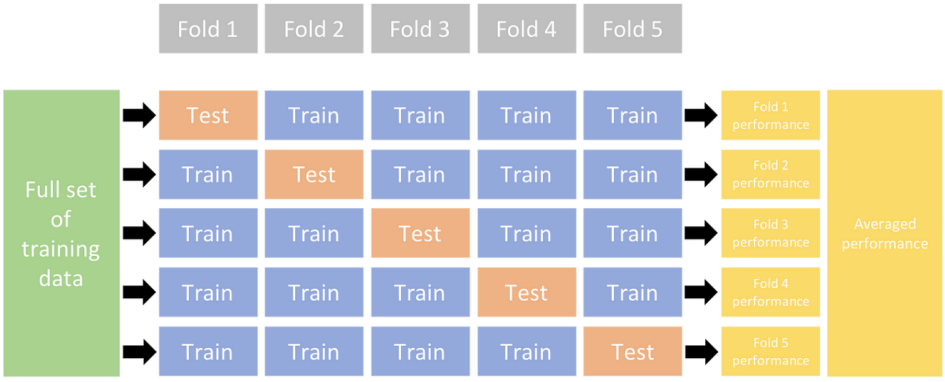


Figure 2.20: K-folding schema [19].

In summary, when a prediction is wrong, it is classified as false; if it is correct, it is classified as true. The aim is to maximise the true positives and true negatives and minimise the false positives and false negatives.

- **Accuracy**

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.7}$$

It measures how well a model correctly predicts the labels of the instances in the dataset by dividing the number of correct predictions by the total prediction number. Accuracy is intuitive and simple, but when it comes to datasets with unbalanced classes, it becomes an inaccurate metric for model evaluation.

- **Precision**

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2.8}$$

Is the ratio of true positive prediction to the total number of positive predictions. It indicates how many of the predicted positive instances are actually positive.

- **Recall**

$$\text{Recall / Sensitivity} = \frac{TP}{TP+FN} \tag{2.9}$$

Recall(Sensitivity) represents the ratio of true positive predictions to the total number of positive instances in the dataset. It measures the model's ability to predict actual positive instances correctly. The higher the recall, the more positive samples detected.

- **F1-Score**

$$\text{F1-Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.10}$$

The F1-Score is the harmonic mean of precision and recall, balancing the two. This metric is useful in unbalanced datasets where accuracy can be misleading. It is maximum when precision is equal to recall.

- **Confusion Matrix**

A confusion matrix summarises the performance of a model, showing the sum of true positives, true negatives, false positives and false negatives. It is a useful tool for understanding model errors and the distribution of predictions between different classes.

- **Intersection over Union (IoU)**

Intersection over Union is calculated by taking the area of overlap between the predicted bounding box and the ground-truth bounding box divided by the union of the two. Figure 2.21 represents possible results for the predicted bounding box and the ground-truth bounding box.

As can be seen from the Figure 2.22, the more the predicted bounding box overlaps with the original box, the closer the IoU result is to 1.

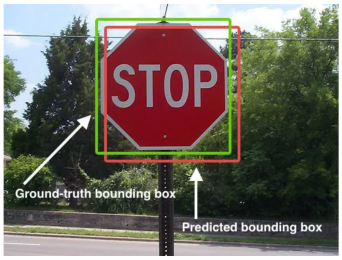


Figure 2.21: Example of a predicted box from a model to detect the stop signal [34].

Figure 2.22: Some possible results of the IoU indicator. The red bounding box represents the ground-truth, and the green box represents the model's prediction [34].

- **Mean Average Precision (mAP)**

  mAP is a metric used to measure the performance of models that are more focused on object detection. This value is dependent on the following metrics:

  - Precision
  - Recall
  - IoU
  - Confusion matrix

  In Figure 2.23, the precision-recall curve summarises the trade-off between the rate of true positives and positive predicted values. The combination of precision and recall values maximises the model's performance. When a model has a high precision value but a low recall value, it means that the model can only classify a few samples as positive. On the other hand, a high recall value and a low precision value mean that the model classifies as many negative samples as it does positive samples [3].

$$\text{Mean Average Precision} = \frac{1}{n}\sum_{k=1}^{n} AP_k \tag{2.11}$$

  The formula in equation 2.11, where n represents the number of classes and APk is the average precision of each class k, represents that for a given class k, must be calculated its average precision to obtain the model's mAP.

- **Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC)**

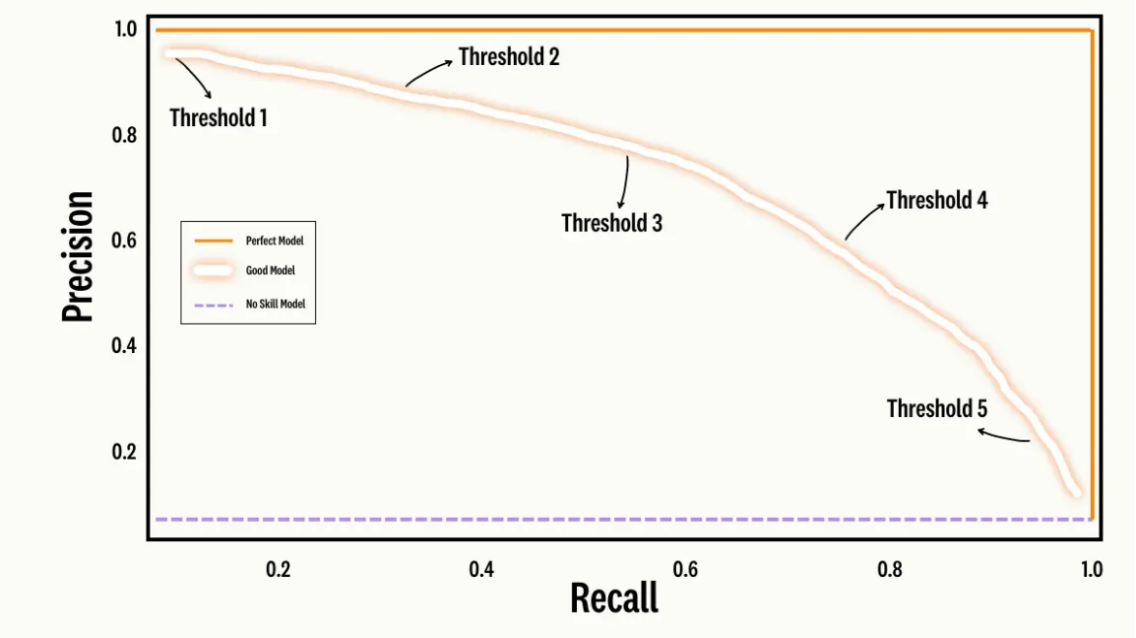


Figure 2.23: Example of a precision-recall curve representation [3].

To determine the ROC curve and AUC, first, it is necessary to take into consideration the recall/sensitivity equation 2.9 and the following equations:

$$\text{Specificity/ True Negative Rate} = \frac{TN}{TN+FP} \tag{2.12}$$

$$\text{False Negative Rate (FNR)} = \frac{FP}{TN+FP} \tag{2.13}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{TN+FP} = 1 \text{ - Specificity} \tag{2.14}$$

The ROC curve is a graph that relates Sensitivity on the y-axis and False Positive Rate (FPR) on the x-axis for different decision threshold values. Figure 2.24 illustrates an example of a ROC curve. The diagonal represented in the graph (grey line) indicates the performance of a random model, where the true positive rate is equal to the false positive rate for any given threshold.

The Area Under the Curve (AUC) is calculated as the area under the ROC curve. AUC values vary between 0 and 1; the higher the value, the better the model is at distinguishing classes.

- **Inference Time**

  Inference time refers to the time required for a Convolutional Neural Network (CNN) to process an input and produce an output. This is a crucial aspect, especially in applications

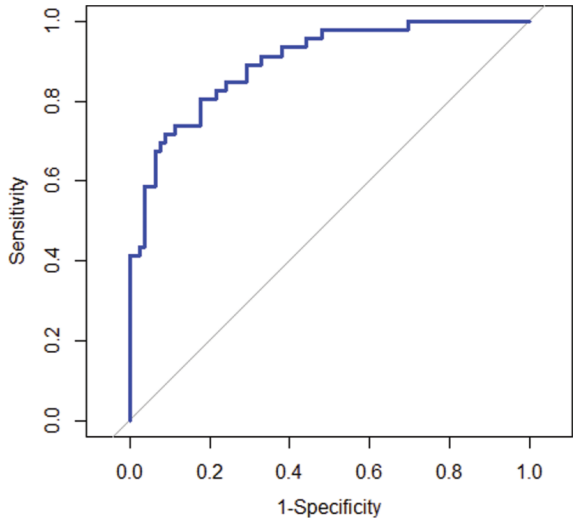


Figure 2.24: Example of a ROC curve representation [21].

that require real-time processing. This time depends on various factors such as the model used, size and number of channels in the input image, type of hardware, and software optimization.

2.3 Software tools for image processing

In the rapidly evolving field of computer vision, several software tools have become essential resources for researchers and developers. These tools enable various applications, from basic image processing to advanced machine learning and real-time object detection.

Exploring four prominent software tools: Roboflow Universe, OpenCV, YOLO, and EasyOCR, this chapter delves into the features, architectures, and applications of these tools, providing a comprehensive overview of their capabilities and contributions to the field of image processing.

2.3.1 Roboflow Universe

Roboflow Universe is a platform equipped with various tools specifically tailored for computer vision and machine learning professionals [16]. It is primarily utilized for organizing and enhancing datasets meant for training computer vision models. Furthermore, it offers various other resources relevant to the computer vision development workflow. The main functionalities are:

- **Annotation tools** — This tool helps users to efficiently annotate their own data with model-assisted labelling, making the image annotation process easier.
- **Co-operation** —Supports collaboration among team members, enabling multiple users to work at the same time on a dataset.

- **Version control** — Control of changes to the dataset can be managed over time, allowing for iterations and improvements.
- **Pre-processing and augmentation** — The platform offers image pre-processing techniques to improve data quality and data augmentation to increase the quantity of images in order to help the robustness of the applied models.

2.3.2 OpenCV: *Open Source Computer Vision*

OpenCV, short for Open Source Computer Vision Library, is an open-source library extensively utilised for image processing, computer vision, and machine learning. Originally developed by Intel in 1999, it has emerged as a key tool for real-time image processing, object recognition, motion tracking, video analysis, and various other applications [26]. It is primarily written in C++ but includes interfaces for Python, Java, and other languages. Its popularity stems from its efficiency, flexibility, and the wide range of functions and algorithms it offers. These range from basic image processing operations like filtering and geometric transformations to advanced machine-learning techniques for recognizing patterns and detecting objects.

One of OpenCV most powerful features is the ability to handle a wide variety of image and video formats, making it an ideal choice for projects involving the analysis of large sets of visual data. Furthermore, its active community, with more than 47,000 members, and an estimated number of downloads exceeding 25 million, ensures that new features and improvements are constantly being developed and made available [26]. OpenCV is structured in a modular way, with the package containing several shared or static libraries [29].

2.3.3 YOLO: *You Only Look Once*

YOLO is a CNN that stands for "You Only Look Once". It is one of the most widely used and effective algorithms for real-time object detection and instance segmentation in images and videos [30].

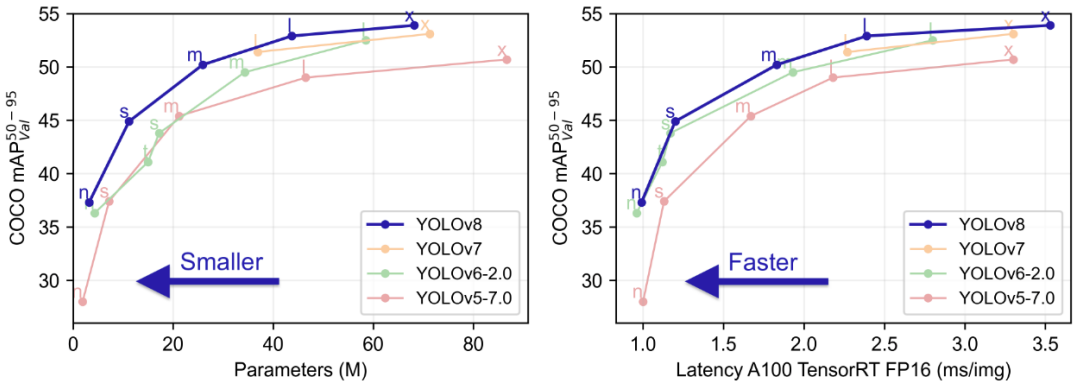


Figure 2.25: Performance comparison of several YOLO models [19].

YOLO has multiple algorithm versions. This project used version 8, which had the best documentation and performance at the time of realization of this thesis.

In Figure 2.25, it is evident that YOLOv8 outperforms the other models regarding mAP values, as shown in both graphs. YOLOv8 achieves higher accuracy with fewer parameters, indicating that it is a more efficient model when working with limited data, as seen in the left graph. The right graph demonstrates that YOLOv8 has lower latency while maintaining high accuracy, which is crucial for real-time applications where fast inference is essential.

Figure 2.25, also represents that for the same YOLO version there are different models[19]:

- **n (nano)** — Extremely light and fast, ideal for devices with limited resources.
- **s (small)** — Light and efficient, with a good balance between precision and speed.
- **m (medium)** — Intermediate model, offering good precision at an acceptable speed.
- **s (small)** — High precision, but with higher computational costs.
- **x (extra large)** — The largest model, designed for maximum precision, is suitable for use in powerful servers.

The Figure 2.26 shows the capabilities of the YOLOv8 model:

- **Classify** — Identify objects in an image and assign labels.
- **Detect** — Localize specific objects within an image with bounding boxes and confidence percentages.
- **Segment** — Precisely delineating the contours of objects within an image.
- **Track** — Tracks objects through a sequence of frames.
- **Pose** — Identify and track key points on the human body to determine postures and movements.

Figure 2.26: YOLOv8 operating modes [19].

As can be seen in Figure 2.27, the YOLOv8 architecture consists of three essential blocks: Backbone, Neck, and Head [30]:

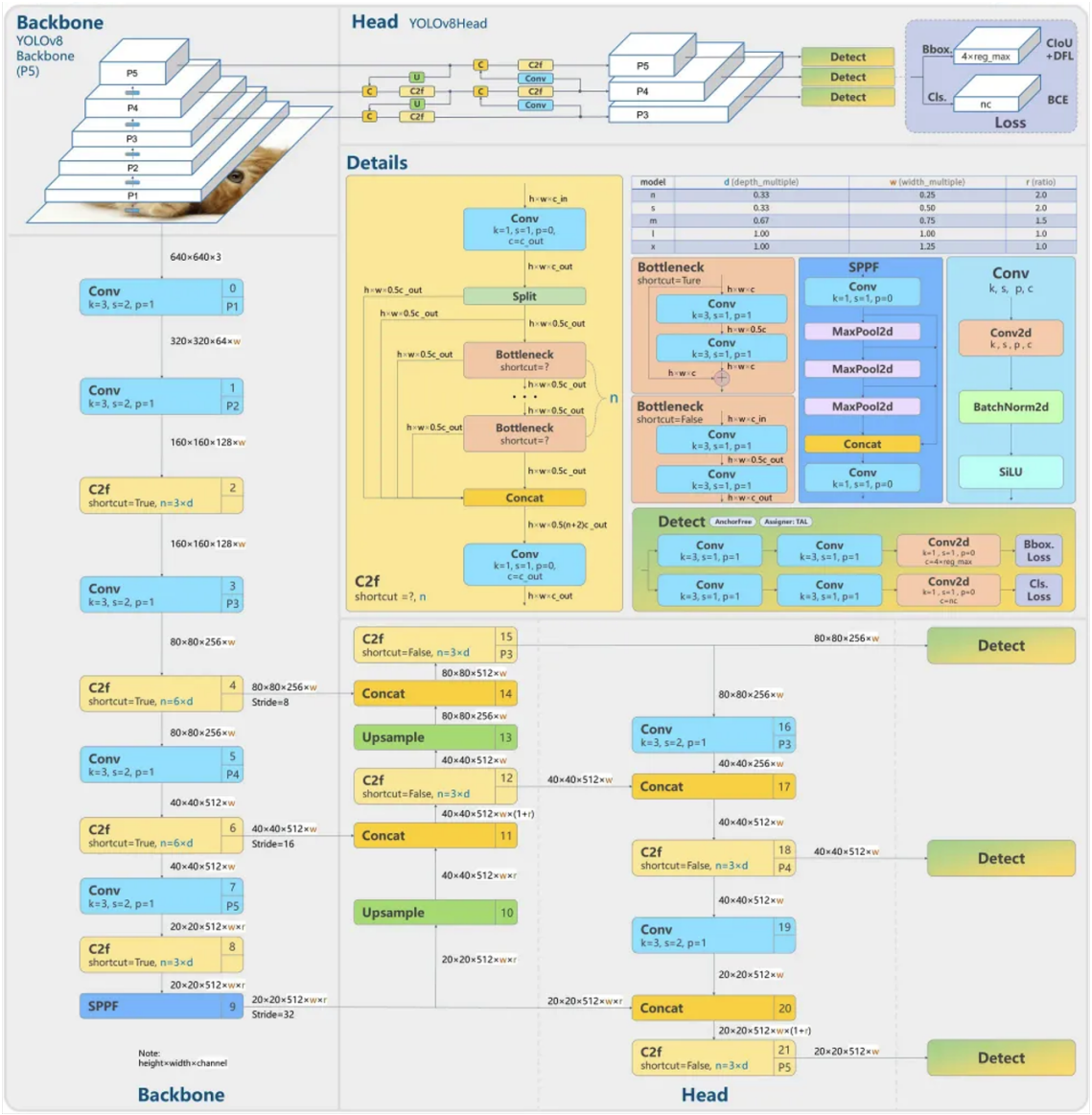


Figure 2.27: YOLOv8 Architecture [36].

- **Backbone** — This component extracts important features from the input image, capturing simple patterns and creating hierarchical representations.
- **Neck** —The Neck acts as a bridge between the Backbone and Head, merging features from different scales and integrating contextual information to improve detection accuracy.
- **Head** —This part generates the network's final outputs, such as bounding boxes and confidence scores, determining the presence and category of objects.

2.3.4 EASYOCR: *Easy Optical Character Recognition*

EasyOCR is an open-source optical character recognition (OCR) library developed by the Japanese company Jaided AI. It is designed to be simple, fast, and effective, using deep learning technology and offering support for over 80 languages [4]. EasyOCR stands out for its ease of use and accuracy in various OCR applications, making it one of the best open-source algorithms available [38].

2.4 Related Work

Recognising electrical circuits involves advanced image processing, segmentation, and component classification techniques. Electrical schematics are complex, consisting of various symbols and connections, making it crucial to develop sophisticated algorithms to interpret and identify each element accurately.

Classifying electrical components is a crucial step in the process of interpreting schematics. Machine learning algorithms such as Support Vector Machines (SVM) or neural networks are commonly used for this purpose. These algorithms assign labels to the different symbols present in the schematics. It is essential to extract features such as orientation and gradients to create robust classification models that can deal with various graphical representations of electrical components [24].

There are already several approaches to recognising and classifying electrical circuits. In this section, we'll give a short summary.

2.4.1 Eletrical Circuit Recognition and U=RIsolve

This section focuses on the essential stages of image pre-processing and segmentation in recognizing electrical circuits. It begins by converting RGB images to grayscale and applying filters to remove noise. It then explores crucial steps like threshold operations and morphological operations. The next segmentation process is introduced through an algorithmic approach, systematically identifying connections, components, nodes, and labels. The creation of Regions of Interest (ROI) for detailed analysis is discussed, laying the foundation for robust electrical circuit recognition.

2.4.1.1 Pre-processing and Segmentation

Efficient segmentation in electrical circuit recognition requires robust image pre-processing tasks, as outlined in this thesis [7]. We can divide this process into the following tasks.

- **Conversion of Image Type** — Transformation of the RGB image into a grey-scale image.
- **Image Filtering** — Application of smoothing filters to eliminate image noise.

- **Conversion to Binary Image** — To convert the image to black and white, we can use threshold operation [20].

About the type of filter applied to the image, some studies propose the use of a median filter [2] and others prefer a Gaussian filter to reduce noise [24].

The morphological operations of dilation and erosion are useful techniques for processing some images. Dilation expands the region of interest in an image, increasing the object's size. It eliminates discontinuities and irregularities, making it possible to join together any existing fragments. Erosion is a technique used to reduce the size of objects in an image, which consequently shrinks the region of interest. This technique removes fine details and small objects and the separation of connected objects [27].

To simplify and compact the elements present in an image, skeletonisation or thinning [24] is a valuable morphological operation for handwriting analysis. This technique simplifies the shape of objects, preserving essential features and connected structures, and facilitates topological analyses such as identifying endpoints, bifurcations and object lengths.

After image pre-processing, identifying electrical circuits involves an important step known as segmentation. This step extracts various circuit elements, including components, nodes, connections from the image and labels for each component. The segmentation process is commonly based on the physical shape of electronic component symbols. These symbols are categorized into three distinct groups based on their geometric structure: symbols with closed shapes, symbols with connected lines, and symbols with disconnected lines [23].

In the recent work presented in [7], the author creates an algorithm that follows a systematic and iterative approach divided into the following steps:

1. **Connection Detection** — Detection of interconnections between circuit elements which are stored in a list.

2. **Component Detection** — Detection of the components present in the circuit which are stored in a list.

3. **Updating the Connection List** — The list of previously detected connections is updated based on the new detection of components.

4. **Node Detection and New Connection List Update** — The circuit is scanned for nodes and the connection list is updated.

5. **Detection of Ports (Connection Points)** — Component ports are detected using the extracted connections.

6. **Updating the Component List** — The list of detected components is updated, discarding those without ports.

7. **Label Detection** — It detects the labels present in the circuit.

8. **Label Association** — If labels are detected, they are associated with the respective circuit elements, providing a more complete and comprehensible identification.

The algorithm is designed to deal with different cases and guarantee robustness in detecting elements.

After segmentation, the author creates the circuit components' ROI (Regions of Interest) and their respective labels. The ROI consists of a specific, delimited image area selected for more detailed analysis and processing. This well-defined area of the image is particularly useful for reducing the computational load in detection tasks, pattern recognition and the extraction of relevant features

2.4.1.2 CNN for component classification

According to the work carried out in the following thesis [13] within the framework of the *U=RIsolve* platform, the author adopts the following approach to solve the problem of classifying electrical components:

- **Hand draw component dataset** — Creation of a dataset of hand-drawn components due to a lack of data. This dataset consists of 3441 component images categorized into 22 classes.
- **CNN design and test** —The author has developed a model that takes a grayscale image of a component with a size of 64 by 64 pixels as input and outputs the corresponding image label. This model consists of a convolutional layer with ReLU activation function, and the output is transformed into a one-dimensional vector through a flattening process. The flattened output then passes through a fully connected layer, which assigns the appropriate image label based on the evaluated classes.

Using this dataset and the network created, the author achieved accuracy rates in classifying the test samples of over 90 per cent and with very short inference times.

2.4.2 Instance segmentation for electrical circuit images

The research conducted by Ohannes Bayer, Amit Kumar Roy, and Andreas Dengel from the German Research Center for Artificial Intelligence in Kaiserslautern, Germany, tackles the task of digitizing handwritten circuit diagrams. These diagrams are frequently encountered in educational or historical settings. The objective is to automatically extract their functional principles or detect errors.

The authors propose a method that uses instance segmentation and keypoint extraction to identify electrical components, including their terminals and descriptive texts, as well as their interconnections, such as junctions and wire hops. This approach simplifies the graph extraction process into a two-step procedure: model inference and straightforward geometric keypoint matching.

The paper describes the preparation of a dataset, which is crucial for training the machine learning model. The dataset, called CGHD (Circuit Graph Handwritten Diagram) [37], has been expanded to include instance segmentation ground truth through semi-automated processing steps.

The authors used the Mask R-CNN architecture for their model, implemented with Detectron 2. The model was trained to predict bounding boxes, instance masks, and keypoints. The dataset consists of 2,208 images with 185,641 bounding box annotations covering 58 object classes.

The experiment conducted with the Mask R-CNN model yielded promising results, with a minimum training loss of 0.44, a maximum training mask accuracy of 0.94, and a minimum validation loss of 1.39. However, visual inspection of the validation set revealed that while the model's recognition was reasonable, it was not entirely accurate. Figure 2.28 shows the validation result of the mask prediction, with the text and junction omitted.

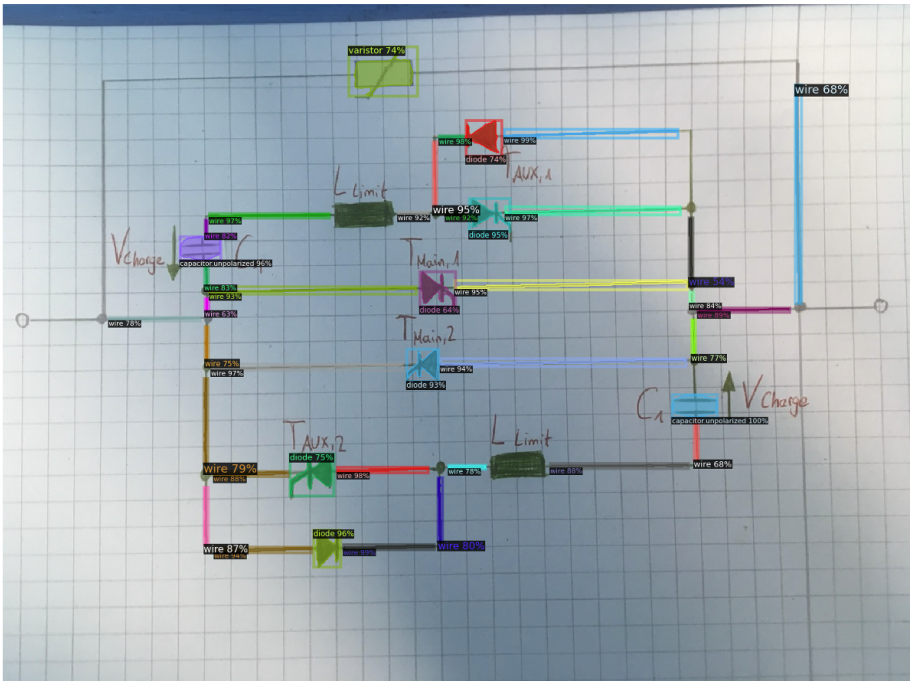


Figure 2.28: Mask prediction [37].

2.4.3 Preliminary work for Electrical Circuit Recognition

This subsection will discuss my approach to an image classification project involving neural networks. Our primary goals are to load images, create a convolutional neural network (CNN), and train it. This project was made with Python in Google Colab to develop the software. The CNN created has six convolutional layers along with max pooling and dropout.

This preliminary work was carried out with the following objectives:

- Load images from a given dataset and define a percentage of elements for testing and training.
- Create a neural network responsible for classifying these images.
- Train and test this network.

- Compare the training results with two other already trained networks.

The following libraries were needed to develop the code:

- **PyTorch** — Dynamic, open-source machine learning framework that uses tensors for data representation, with a modular neural network module and GPU support, and is widely used for building and training neural networks.
- **Matplotlib** — Open Source library for creating diverse and publication-quality visualisations.
- **OpenCV**

To solve this problem, we used the following flow diagram:

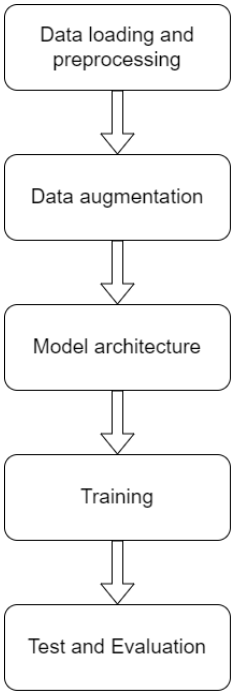


Figure 2.29: Flow diagram.

The following steps were applied to augment the data:

- Random rotations from -90 degrees to 90 degrees.
- Randomly resized and cropped to a maximum of 70% to not cut out useful information.
- Small distortion between x and y (ratio = (0.9, 1.1)).

We created a neural network with the following characteristics:

- **Convolutions Layer** — Six convolutional layers, each followed by ReLU activation. These layers are responsible for learning hierarchical features from the input images.

- **Maxpooling** — Max pooling is applied two times, after the first and second convolutional layers, to reduce spatial dimensions and computational complexity.
- **Fully Connected Layers** — After the convolutional layers, the data is flattened and passed through three fully connected layers with ReLU activation.
- **Dropout** — Is used for regularization to prevent overfitting. It randomly sets a fraction of input units to zero during training.

The loss function used to train the network was Stochastic Gradient Descent (SGD). The SGD includes a learning rate parameter that controls the size of the parameter update steps, in this case we used 0.005. This CNN model required 180 epochs of training with a batch size of 10 samples, more than 180 iterations led the model to overfit.

We used 20 % of the dataset samples to test the model, which were not used in the training to ensure the reliability of the CNN. To evaluate the model, we determine the accuracy using the test samples, the F1-score, the confusion matrix and the AUC per class.

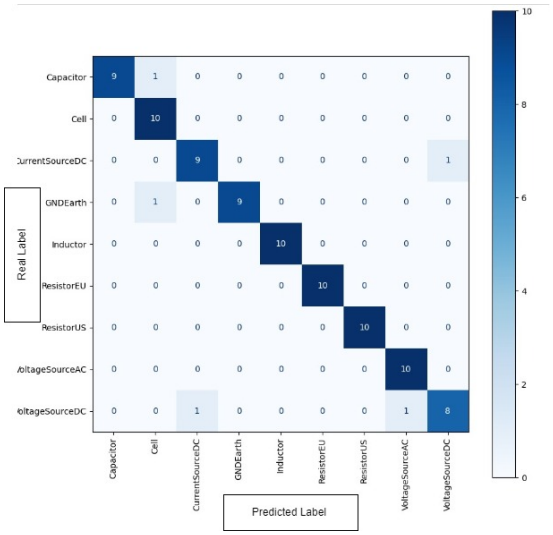


Figure 2.30: Confusion matrix.

Table 2.1: AUC per class.

Class	AUC
Capacitor	99.4%
Cell	100 %
DC Current Source	99.4%
Ground	100 %
Inductor	100 %
EU Resistor	100 %
US Resistor	95 %
AC Voltage Source	95 %
DC Voltage Source	100 %

With this CNN and this test model, an accuracy of 94 % was obtained. Figure 2.30 presents the confusion matrix and Table 2.1 the AUC per class values.

In order to assess the behaviour of the network created, dataset was tested with other pre-trained networks, taking into account the same percentage of samples from the training dataset (20%). The first network used for comparison was VGG16 [14] , which has the following characteristics:

- **Convolutions Layers** — Sixteen convolutional layers, each followed by ReLU activation, expect the last output layer where the softmax[41] function is applied. These layers are responsible for learning hierarchical features from the input images.
- **Maxpooling** — Max pooling is applied two times.

- **Dropout** — Is used in two convolutional layers.

The value obtained for the accuracy of this network was 98%. Below, we present the confusion matrix and the AUC per class values.

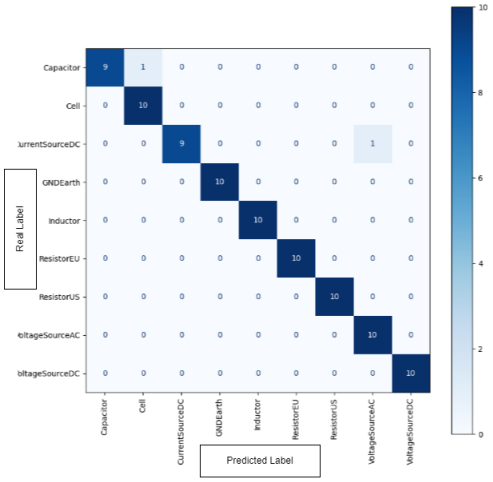


Figure 2.31: Confusion matrix of VGG.

Table 2.2: AUC per class of VGG.

Class	AUC
Capacitor	95%
Cell	99.4 %
DC Current Source	95%
Ground	100 %
Inductor	100 %
EU Resistor	100 %
US Resistor	100 %
AC Voltage Source	99.4%
DC Voltage Source	100 %

The dataset was tested with another CNN model, ResNet 18.

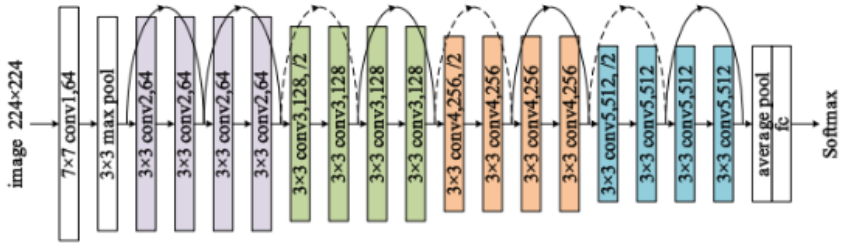


Figure 2.32: ResNet 18 network architecture [42].

As can be seen in Figure 2.32, ResNet 18 has 18 layers, including 16 convolutional layers and 2 fully connected layers, uses max pooling layers, the activation function used throughout the network is ReLU and this network does not use dropout but residual blocks that allow the input of the layer to be added directly to the output. With this configuration, an accuracy value of 98% was achieved. Below, I present the confusion matrix and the AUC per class values.

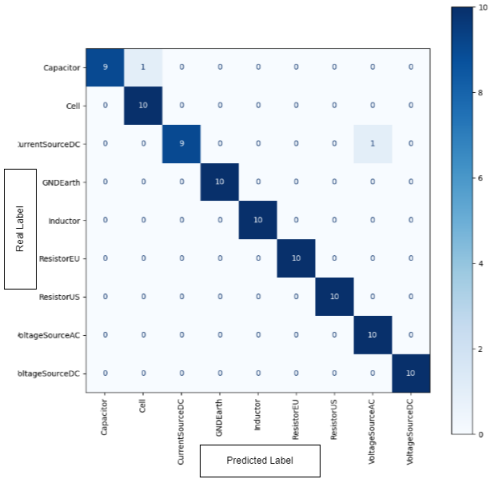


Figure 2.33: Confusion matrix of ResNet 18.

Table 2.3: AUC per class of ResNet 18.

Class	**AUC**
Capacitor	99.4%
Cell	100 %
DC Current Source	99.4%
Ground	100 %
Inductor	100 %
EU Resistor	100 %
US Resistor	95 %
AC Voltage Source	95%
DC Voltage Source	100 %

Chapter 3

Software Architecture and Implementation

In this chapter, a comprehensive overview of the software architecture designed for image processing of electrical circuit schematics is presented, along with the generation of JSON files compatible with the U=RIsolve application. The content includes the specifications of the development workstation and a detailed exposition of the system's block diagram, offering an in-depth understanding of each constituent element, ranging from image segmentation to identifying electrical connections and character recognition. Insights into the datasets utilized for training the machine learning models and a discussion of the obtained results are also provided. Additionally, the data model's structure and the source code's organisation are delineated and information about the software's functioning.

3.1 Software Architecture

The architecture of the software is illustrated in the block diagram in Figure 3.1. The system is designed to take an image of an electric circuit design on a computer as input and produce a JSON file as output that is compatible with the U=RIsolve web platform. In order to achieve the main objective of this work, the following blocks were required:

- **Components, nodes and labels segmentation** — This block is responsible for receiving the circuit image and segmenting it as a way to identify all components, nodes, junctions, and text.
- **Image processing to define electrical connections** — In this part, the software cuts out all the content found in the segmentation from the original image and applies filters to the image created from the segmentation cuts so that only the circuit connections remain.
- **Electrical connection detection** — Through the use of computer vision algorithms and mathematical calculation functions, this module provides the necessary information about electric circuit connections.

- **Prepocessing text** — Using filters in the text's regions of interest (ROI) to simplify its classification. By enhancing the clarity and contrast of text areas, it prepares the text for accurate recognition and extraction.
- **Optical character recognition (OCR)** — This module extracts text from the preprocessed image regions using OCR technology. It identifies component names, values, and units, converting the textual information in the image into a machine-readable format.
- **Polarity detection** — This section is responsible for determining the polarity of components, ensuring correct identification of directional properties for accurate circuit analysis and simulation.
- **Circuit map** — This block compiles all the extracted information into a structured JSON file. The JSON file includes details about components, nodes, and electrical connections, providing a comprehensive digital representation of the circuit that can be integrated into the U=RIsolve platform.

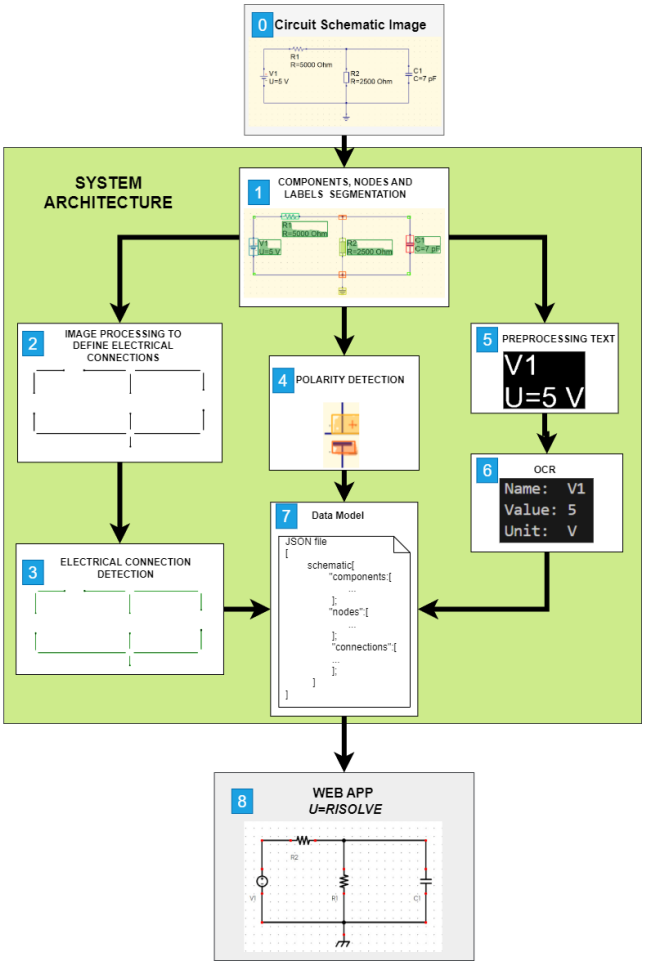


Figure 3.1: Block diagram of the circuit modelling software.

3.2 Datasets

In the realisation of this project, it was crucial to create two datasets. The segmentation dataset has data on the components, nodes (junction points of 3 or more electrical connections), junctions of two electrical connections and electrical connections called "wire"; the polarity dataset has images of components featuring polarity, voltage DC source, voltage AC source, and current source.

The segmentation dataset comprises 200 images created from electrical circuit simulators (such as QUCS and LTspice) and various circuit workbooks. The images within this dataset were designed to depict potential electrical configurations for the purpose of evaluation by prospective users of U=RIsolve.

This dataset was annotated to identify the masks of all electrical circuit elements in the images, as shown in Figure 3.2. It comprises a total of 12 classes and 6715 annotations distributed according to Table 3.1.

Table 3.1: Class balance of segmentation dataset.

Classe Name	Number of Samples
wire	2691
text	1391
junction	792
cross	471
resistorus	311
capacitor	194
voltagedc	192
ground	183
coil	161
current	140
resistoreu	111
voltageac	78

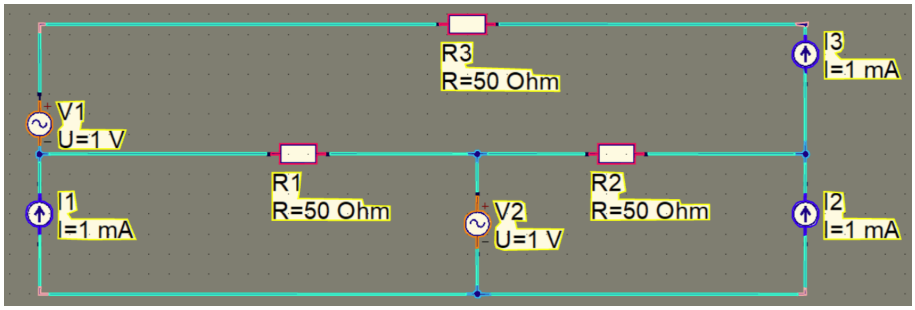
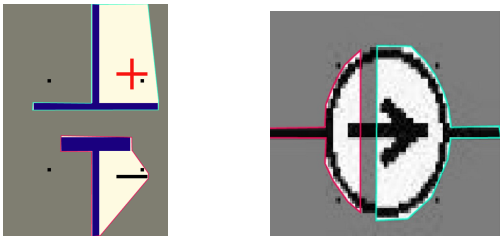


Figure 3.2: Example of a segmentation image dataset.

The polarity dataset is annotated for segmentation and comprises 150 images with 4 classes distributed across the Table 3.2. Figure 3.3 illustrates an example of an annotation for this dataset.



(a) DC voltage source. (b) DC current source.

Figure 3.3: Examples of polarity images dataset.

Table 3.2: Class balance of polarity dataset.

Class Name	Number of Samples
voltage negative	93
voltage positive	93
current negative	57
current positive	57

3.3 Machine learning models

3.3.1 Segmentation model without "wire" class

In order to segment all the components, nodes, and text in the circuit diagram, machine-learning approaches were employed, as represented by block 1 in Figure 3.1.

The segmentation dataset was divided into training, validation, and test sets in a ratio of 70:20:10. This split ensures that the model has sufficient data to learn from while also being evaluated on unseen diagrams to validate its performance.

Using the Roboflow platform, it was possible to preprocess an augmentation of the dataset. The images were resized to 640 by 640 pixels as a way to standardize the size of all the images in the dataset. Then, the images were converted to greyscale to simplify the training process. In this particular application, the focus is only on the shapes of the elements in the images; color was unnecessary for this particular application. The augmentation techniques used were:

- **Flip** — Horizontal and vertical.
- **90º Rotate** — Clockwise, counter-clockwise and Upside down.
- **Crop** — Random crop to a maximum of 95
- **Rotation** — Random rotation between -15 degrees to 15 degrees.
- **Shear** — Random shear -5 degrees to 5 degrees horizontal and vertical.
- **Blur** — Up to 0.5 pixels.
- **Noise** — Up to 1.01% of pixels.

Using the Robflow platform and the augmentation techniques defined above, it was possible to train the model with 1050 images. Appendix A presents several images that illustrate various augmentation techniques.

In the development of this model, a deliberate decision was made to refrain from utilizing the "wire" class. This choice was underpinned by the notably lower confidence in accurately identifying electrical connections when using our machine learning model in comparison to the identification of other elements within the image. The outcomes of a model trained with the "wire" class are presented in section 3.3.2.

The YOLOv8n-seg model was trained using the following parameters:

- **Batch size** — 26
- **Number of epoch** — 2000
- **Save period** — 10
- **Learning rate** — 0.01
- **Optimizer** — SGD (Stochastic Gradient Descent) [12]
- **Input image size** — 640 by 640 pixels.
- **Data augmentation** — Random augmentation from YOLOv8 [19]

The best training results calculated with validation data can be found in Table 3.3.

Table 3.3: Results from segmentation model.

Class	Instances	Box				Mask			
		Precision	Recall	mAP50	mAP50-90	Precision	Recall	mAP50	mAP50-90
capacitor	49	0.924	0.942	0.943	0.74	0.879	0.893	0.883	0.630
coil	28	0.825	1	0.950	0.777	0.758	0.893	0.871	0.524
cross	82	0.872	0.827	0.879	0.386	0.818	0.766	0.739	0.347
current	20	0.930	1	0.976	0.869	0.886	0.950	0.947	0.851
ground	33	0.986	0.970	0.994	0.808	0.989	0.970	0.994	0.677
junction	141	0.784	0.671	0.729	0.280	0.396	0.312	0.251	0.092
resistorus	49	0.990	1	0.995	0.881	0.992	1	0.995	0.672
resistoreu	17	0.959	1	0.995	0.849	0.965	1	0.995	0.685
text	254	0.954	1	0.972	0.859	0.955	0.996	0.970	0.673
voltagedc	40	0.969	1	0.995	0.869	0.975	1	0.995	0.858
voltadeac	15	0.967	1	0.995	0.945	0.971	1	0.995	0.854
all	**728**	**0.924**	**0.942**	**0.943**	**0.740**	**0.879**	**0.893**	**0.883**	**0.63**

In Figure 3.4, the graphs depict the results of training this model. The loss graphs demonstrate a decreasing trend, signifying effective learning of the model during both training and validation without overfitting. The precision and recall metrics also exhibit an increasing trend, indicating that the model is enhancing its predictions over time.

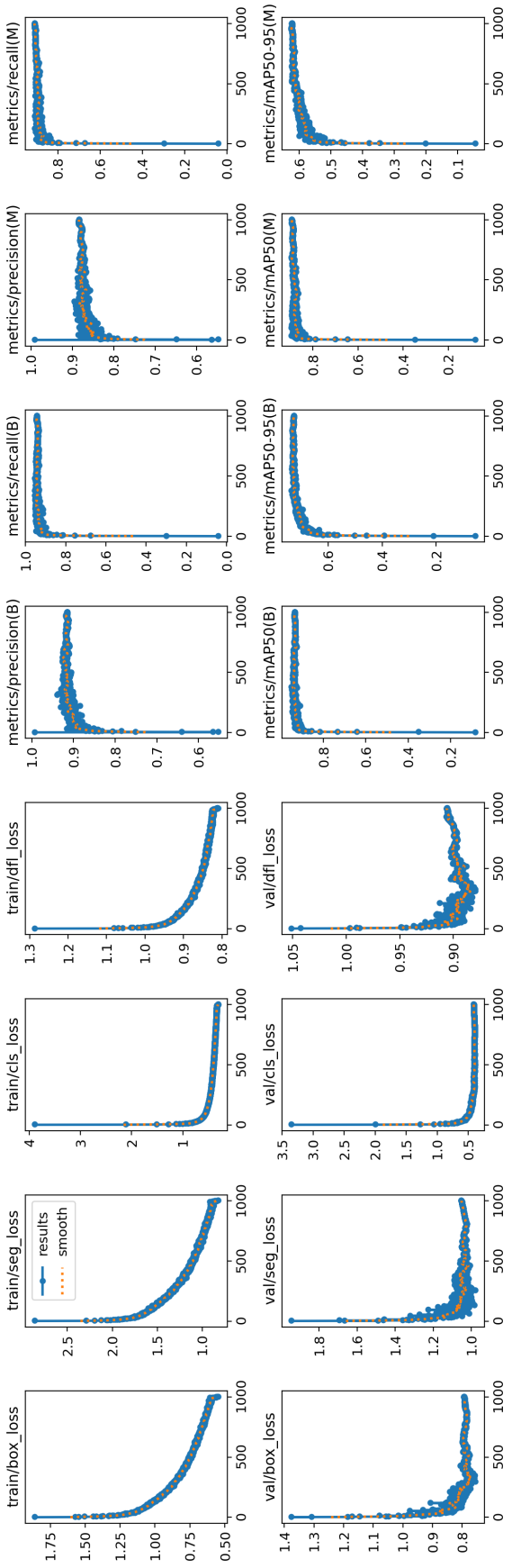


Figure 3.4: Graphs resulting from segmentation training: the top graphs pertain to the training samples and the bottom graphs to the validation samples.

The confusion matrix in Figures 3.5 and 3.6 indicates that the model performs robustly for the various classes.

This was the segmentation model utilized in the software and can be located at *models/segmentation* in the project's software directory. This folder contains more metrics on the training, validation and testing of this model.

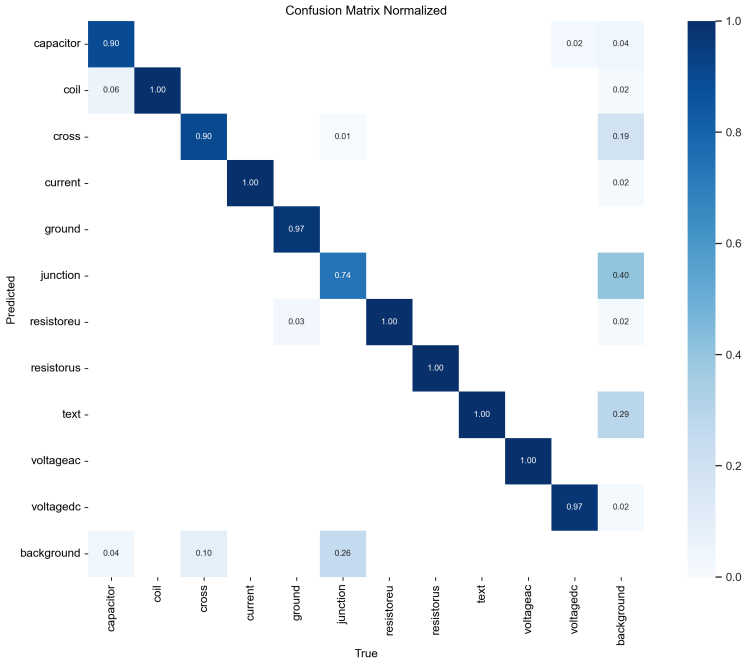


Figure 3.5: Confusion matrix created with validation data.

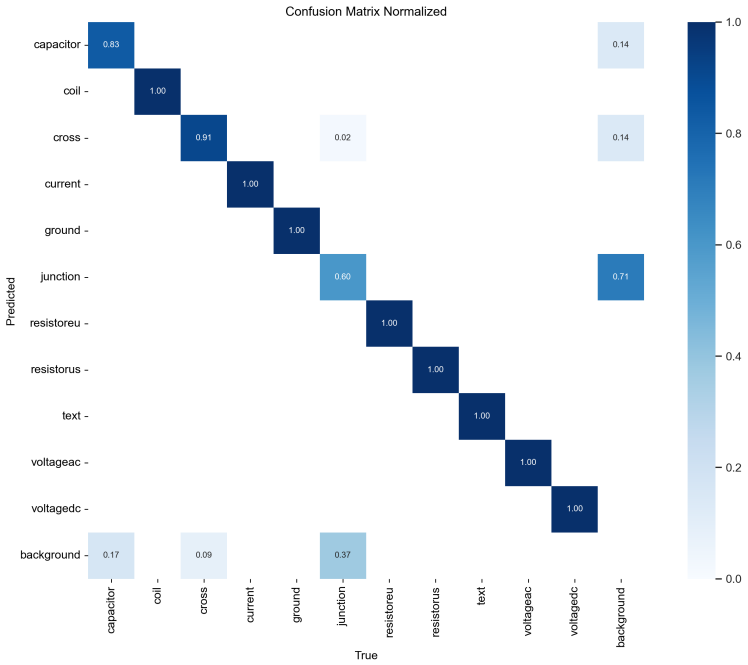


Figure 3.6: Confusion matrix created with test data.

3.3.2 Segmentation model with "wire" class

It was not possible to identify electrical connections in the circuit images for this dataset. To confirm that, a model with the class "wires" was trained under the same conditions as those defined in section 3.3.

In Figure 3.7, the validation losses start to increase or stabilize while the training losses continue to decrease. This is a clear sign of overfitting. However, since I save the model's results every 10 epochs, analyzing the graphs at epoch 200 shows that overfitting isn't happening.

Table 3.4: Results from segmentation model at epoch 200.

Class	**Instances**	**Box**				**Mask**			
		Precision	**Recall**	**mAP50**	**mAP50-90**	**Precision**	**Recall**	**mAP50**	**mAP50-90**
capacitor	49	0.872	0.898	0.904	0.587	0.941	0.929	0.948	0.591
coil	28	0.855	1	0.945	0.779	0.768	0.893	0.876	0.529
cross	82	0.857	0.902	0.885	0.340	0.737	0.756	0.703	0.291
current	20	0.952	0.997	0.988	0.911	0.914	0.950	0.955	0.857
ground	33	0.920	0.970	0.970	0.843	0.972	0.970	0.970	0.749
junction	141	0.735	0.729	0.687	0.266	0.359	0.319	0.213	0.074
resistorus	49	0.984	1	0.995	0.887	0.987	1	0.995	0.709
resistoreu	17	0.855	1	0.992	0.855	0.862	1	0.992	0.673
text	254	0.951	1	0.974	0.874	0.950	0.996	0.973	0.693
voltagedc	40	0.975	0.981	0.995	0.866	0.978	0.975	0.995	0.842
voltadeac	15	0.984	1	0.995	0.962	0.956	1	0.995	0.841
wire	504	0.457	0.351	0.604	0.253	0.406	0.133	0.261	0.195
all	**1232**	**0.713**	**0.691**	**0.783**	**0.502**	**0.642**	**0.543**	**0.575**	**0.568**

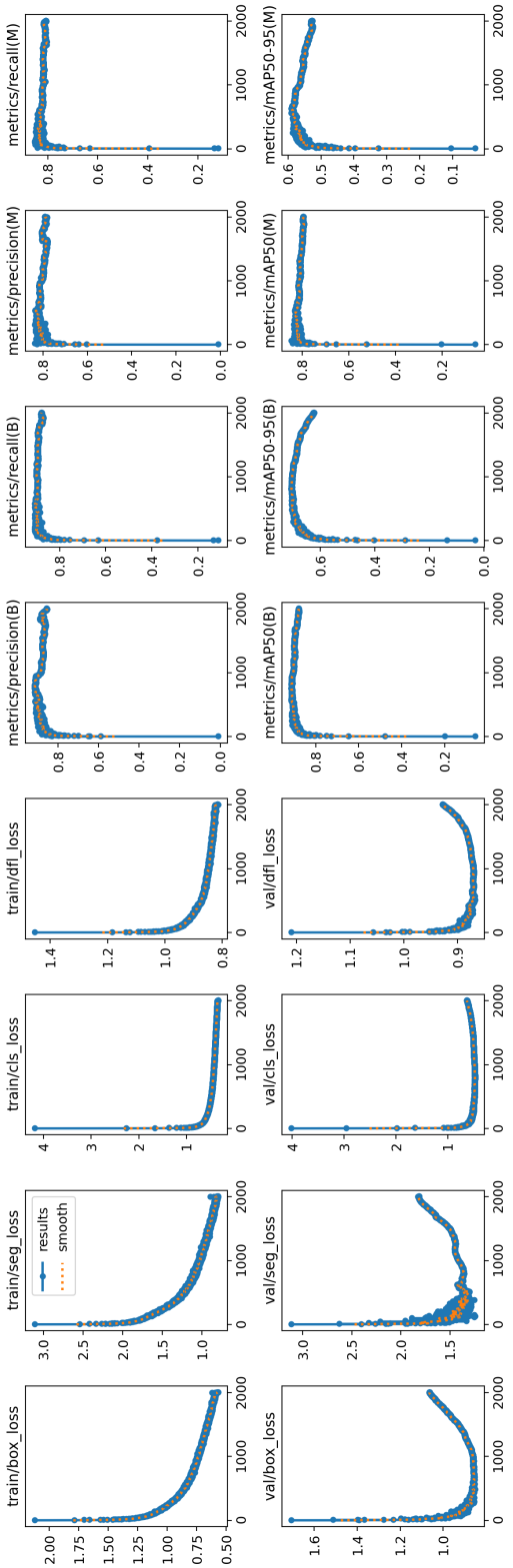


Figure 3.7: Graphs resulting from segmentation training with "wires" class: the top graphs pertain to the training samples and the bottom graphs to the validation samples.

As we can see from the confusion matrix in Figures 3.8 and 3.9, the model behaves robustly across classes except for the "wire" class.

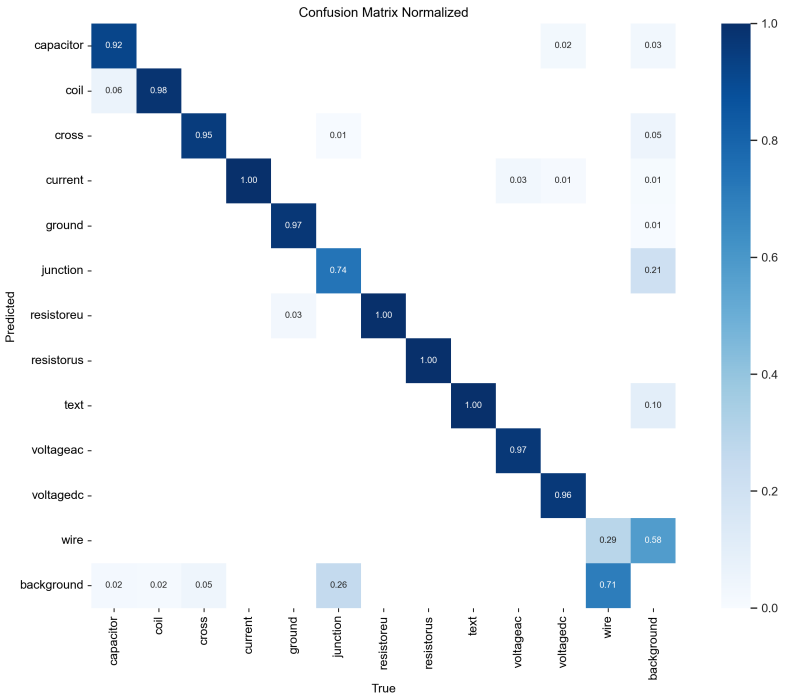


Figure 3.8: Confusion matrix created with validation data.

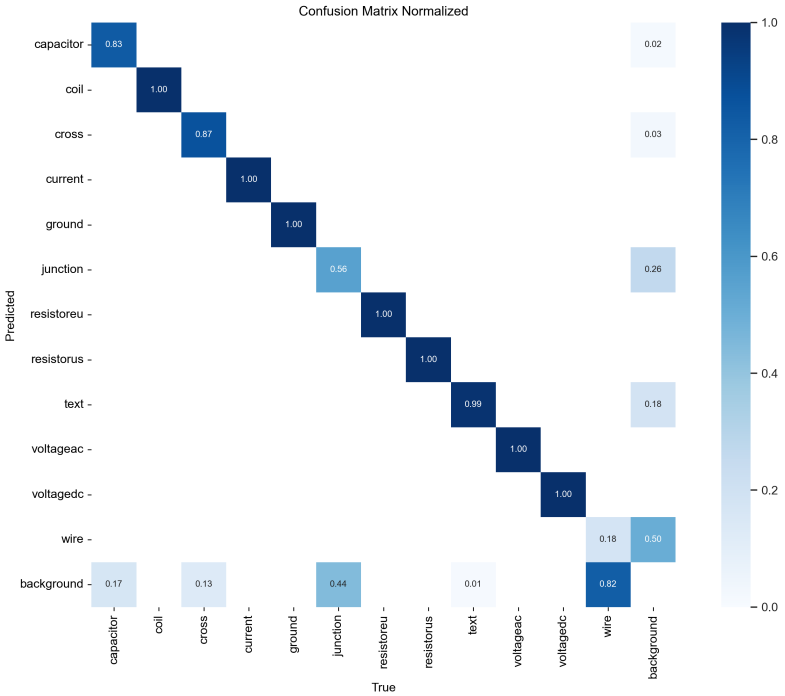


Figure 3.9: Confusion matrix created with test data.

3.3.3 Polarity model

YOLOv8n-seg was also applied to find the polarity in the voltage and current sources, but on the polarity dataset, as depicted by block 4 of Figure 3.1.

The dataset was divided into training, validation, and testing sets in a 70:20:10 ratio. The images were resized to 96x96 pixels, which is the minimum size accepted by YOLOv8n-seg. Furthermore, the images were converted to grayscale. The augmentation techniques used were the same as the segmentation dataset.

The YOLOv8n-seg model was trained using the following parameters:

- **Batch size** — 8
- **Number of epoch** — 500
- **Save period** — 25
- **Learning rate** — 0.001
- **Optimizer** — AdamW [11]
- **Input image size** — 96*96 pixels.
- **Data augmentation** — Random augmentation from YOLOv8 [19]

Based on the graphs in Figure 3.10, the validation metrics stabilize around epoch 300. The model generated at this epoch was used to determine the polarity of the sources. Table 3.5 displays the results of this model for the validation data.

Table 3.5: Results from polarity model at epoch 300.

Class	Instances	Box				Mask			
		Prescision	Recall	mAP50	mAP50-90	Precision	Recall	mAP50	mAP50-90
current negative	14	0.918	0.799	0.847	0.531	0.905	0.786	0.787	0.595
current positive	14	0.872	0.786	0.879	0.653	0.873	0.786	0.854	0.656
voltage negative	15	1	0.976	0.995	0.722	0.931	0.904	0.908	0.608
voltage positive	15	0.926	1	0.995	0.773	0.929	1	0.995	0.601
all	**29**	**0.929**	**0.890**	**0.929**	**0.670**	**0.910**	**0.869**	**0.886**	**0.595**

As can be seen from the confusion matrix in Figures 3.11 and 3.12, the model behaves robustly.

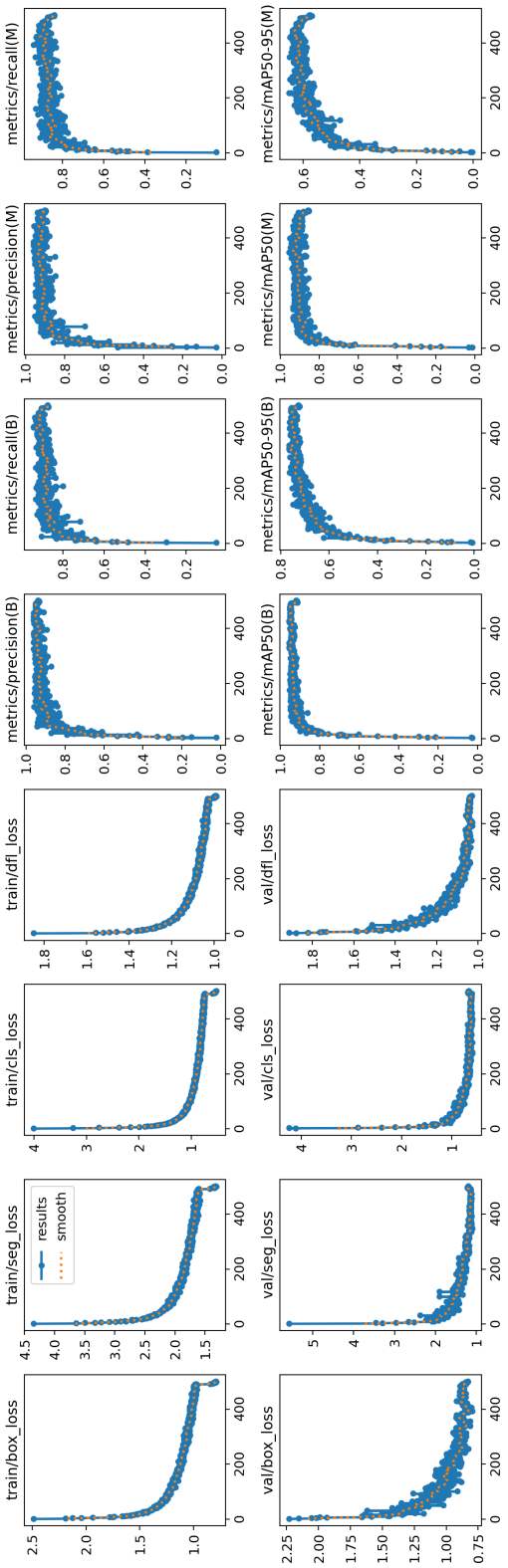


Figure 3.10: Graphs resulting from polarity training: the top graphs pertain to the training samples and the bottom graphs to the validation samples.

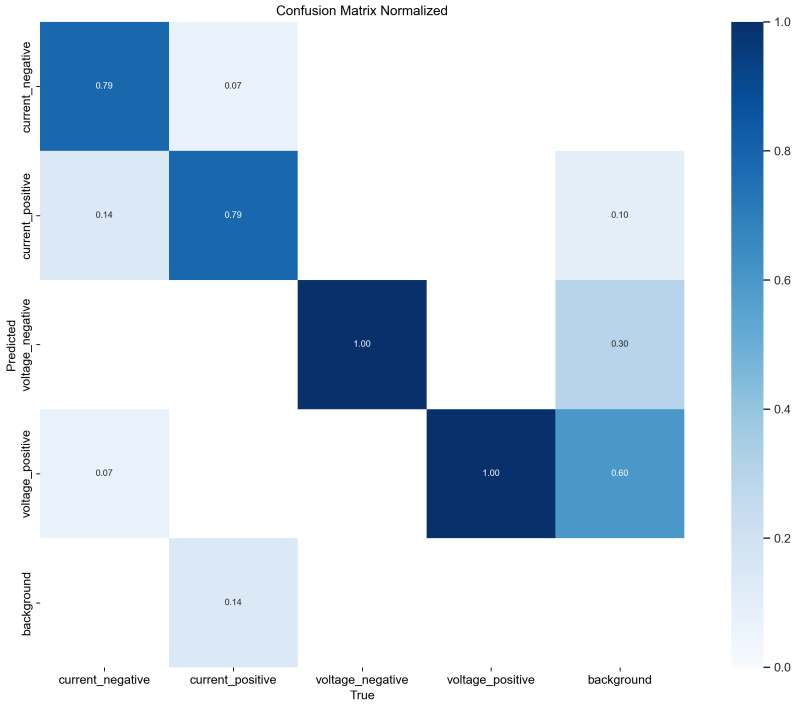


Figure 3.11: Confusion matrix created with validation data.

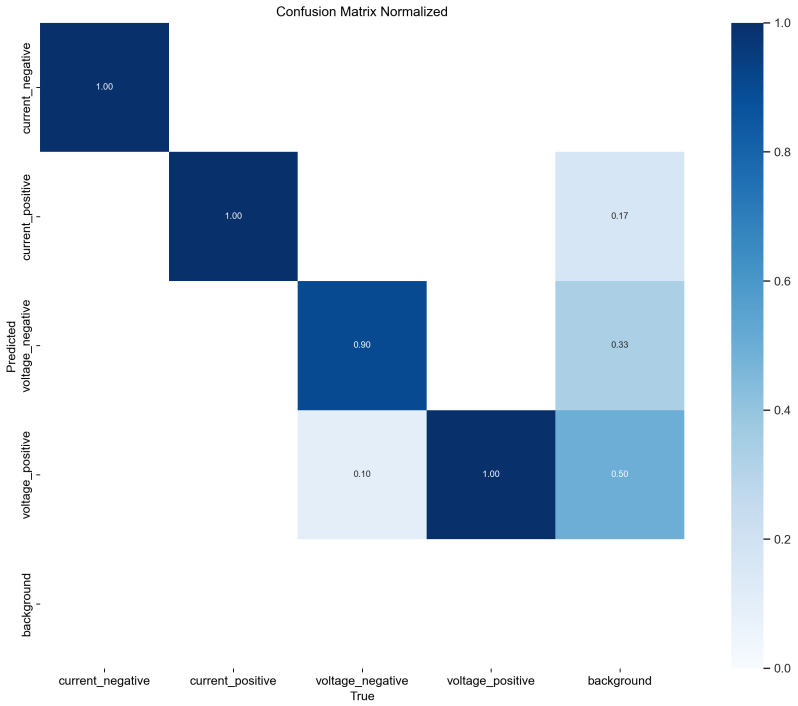


Figure 3.12: Confusion matrix created with test data.

The polarity classes were not annotated in the segmentation dataset because the representation of DC voltage sources is similar to that of capacitors. As seen in Figure 3.13, the polarity model

could identify polarity information on capacitors. In this application, the definition of capacitor polarity is not anticipated.

Figure 3.13: Capacitor evaluated in the polarity model. The orange colour represents the negative voltage class.

3.4 Data Model

This section briefly presents the organisation of the data model that must be created from the input image of the circuit schematic. The model must be compatible with the U=RIsolve [40] application in order to be imported and used. The data model for this work is executed in JSON format file. The JSON data format is superior to a netlist due to its enhanced capability for the efficient addition and removal of data.

3.4.1 First data model

The first model developed during this project has the following structure:

- **component** — The component class consists of all the electrical components in the circuit and possesses the attributes defined in Table 3.6.
- **node** — This class comprises all the nodes shown in the circuit schematic evaluated by the software. Table 3.7 presents all of its parameters.
- **connection** — Table 3.8 displays this category's parameters, which consists of all the electrical connections found in the electrical circuit.
- **schematic** — The schematic class represents the comprehensive diagram of the electrical circuit, formed by a combination of components, nodes, and connections

Table 3.6: Component class

Level 0	Level 1	Description
type		Type (e.g. resistance, voltage dc, capacitor, coil)
id		Id unique
seg_confidence_in_prediction		Confidence in the prediction of the segmentation model
name		Name of the component resulted from OCR
value		Value of the component resulted from OCR
unit		Unit of the component resulted from OCR
OCR_confidence_in_prediction		Confidence in the prediction of the OCR model
position	x	Component Position
	y	
	z	
port (array)	id	Id unique
	polarity	Classification of the polarity model, if the component has polarity
	probability_polarity	Confidence in the prediction of the polarity model, if the component has polarity
	x	Port position
	y	

Table 3.7: Node class

Level 0	Level 1	Description
type		"cross"
id		Id unique
probability		Confidence in the prediction of the segmentation model
position	x	Node position
	y	
	z	
connection (array)		Vector with the ids of the connection linked to the nodes

Table 3.8: Connection class

Level 0	Level 1	Description
type		Type (e.g. straight, curve)
id		Id unique
name		Name of connected components (e.g. 'resistor to ground')
source	id	Id of the connection start component
destination	id	Id of the connection destination component
vertex (array)	x	Vertex position of connections
	y	

3.4.2 Data model for U=RIsolve Simulator

The data model used by the simulator has the general structure shown in Table 3.9.

The structure of the components is defined in Table 3.10.

The nodes of the model defined for the simulator have the exact definition as the JSON format described in the section above, and their class is represented in Table 3.11.

There is a key distinction in defining a connection, as defined by Table 3.12. In my model, a connection is a single electrical wire. However, in the simulator's JSON model, a connection is described as any sequence of connected wire with the same electric potential.

Table 3.9: General structure of the data model to simulator.

Level 0	Level 1	Level 2	Description
components (array)			List of components present in the circuit
connections (array)			List of connections present in the circuit
nodes (array)			List of nodes present in the circuit
properties	grid	x	Pixels per grid unit (horizontal)
		y	Pixels per grid unit (vertical)
		active	Grid visibility
	view	x1	Left boundary of the editor window
		y1	Top boundary of the editor window
		x2	Right boundary of the editor window
		y2	Bottom boundary of the editor window
		xPos	Horizontal position of the editor window
		yPos	Vertical position of the editor window
		scale	Global scale of the editor

Table 3.10: Component class for simulator.

Level 0	Level 1	Level 2	Description
id			Unique identifier of the component
type			Type of the component
name	value		Name of the component
	position	x	Name position
		y	
	visible		Visibility of the name
position	x		Component position
	y		
	z		Component layer in relation to other components
	angle		Angle of the component [0, 1, 2, 3] for 0º, 90º, 180º, 270º
frequency	unit		Frequency unit of the component
	value		Frequency value of the component
	visible		Visibility of the frequency
properties	impedance	unit	Impedance unit of the component
		value	Value of the component
	temperature	unit	Simulation temperature unit
		value	Simulation temperature value
port (array)	net		Identifier of the electrical node the component port is connected to
	position	x	Port position
		y	
	connections (array)	wire	Identifier of the wire the port is connected to
		point	Point of the wire the port is connected to {'begin', 'end'}
visible			Visibility of the component

Table 3.12: Connection class for simulator.

Level 0	Level 1	Level 2	Level 3	Description
id				
ports (array)	component			Identifier of the connected component
	port			Port number of the connected component
	position	x		
		y		
wires (array)	id			Wire identifier
	label	text		Wire label text
		position	x	
			y	
	begin	connectedPorts (array)	component	Identifier of the connected component
			port	Port number of the component
		connectedWires (array)	wire	Identifier of the connected wire
			point	Point of the wire to which it is connected ['begin', 'end']
		x		
		y		
	end	connectedPorts (array)	component	Identifier of the connected component
			port	Port number of the component
		connectedWires (array)	wire	Identifier of the connected wire
			point	Point of the wire to which it is connected ['begin', 'end']
		x		
		y		

Table 3.11: Node class for simulator.

Level 0	Level 1	Level 2	Description
id			Node identifier
name	value		Node name
	position	x	Name position
		y	
connection (array)	wire		Connected wire identifier
	point		Point of the wire to which it is connected ['begin', 'end']
position	x		Node position
	y		

3.4.3 Data model Improvements

With a view to improving the data model for future use and implementation on the U=RIsolve platform, the following points could be implemented:

- Incorporation of parameters in the component class with the model's confidence level about the type of component, the text found and the orientation of the component, as represented in the data model referred in Table 3.10. In the future, if the confidence levels do not reach a predefined threshold value, the program interacts with the user to check and correct, if necessary, the information classified.
- Ensure that the data type for each field in the model is clearly defined and documented. This includes specifying all possible values interpreted by the software to eliminate any ambiguity. For instance, differentiating between component type, 'Vdc', and 'voltage_dc' is crucial to ensure accurate interpretation by the software.

3.5 Software Organization

The code is located in the *U_RI* folder, which is divided into various sub-folders and files to organize all the necessary data and operations for the project's development, following this structure:

- ***data*** — This folder contains the datasets used for training two models and the images for software validation.
- ***models*** — It contains the trained models that are used in this software and two files with the code needed to do the training and validation.
- ***results*** — Folder where the software results are stored, including the *JSON* files and various images produced throughout the circuit recognition and modeling process.
- ***src*** — Contains the associated files for implementing the software architecture depicted in Figure 1.1 of Chapter 1.
- ***main*** — Contains the source code for this project

3.6 Software Operation

The software follows the scheme shown in Figure 3.1 and the time scale of the numbers represented in the blue boxes. In order to comprehend the entire pipeline, I will outline and explain the following system architecture components in the subsequent sections.

3.6.1 Component, Nodes and Labels Detection - block 1

After receiving the image, it is assessed using the segmentation model described in section 3.3.1. The goal is to identify all components, nodes, labels, and junctions created by the intersection of two wires at a 90º angle. Using this model, we can find out the following information about each class and that all the blocks of the system's architecture are interdependent.

- Id to identify the class type.
- Center of the bounding box, x and y coordinates in pixels.
- Height and width of the bounding box in pixels.
- Confidence values by class ranging from 0 to 1.

3.6.2 Image processing to define electrical connections - blocks 2 and 3

This block receives the original image and the segmentation results. Figure 3.14 demonstrates the possible results of segmentation. In order to find only the electrical connections in the image, block 2 creates a copy of the original image and blanks out all the regions of interest found in the segmentation model in the copied image, as represented in Figure 3.15.

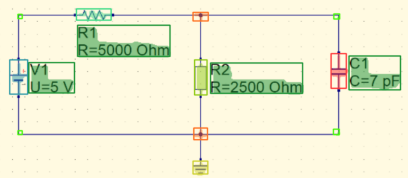


Figure 3.14: Original image with segmentation results.

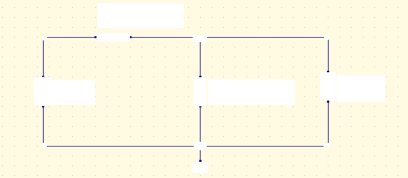


Figure 3.15: Copy of the original image without the segmentation results.

To reduce the noise in the image that shows only the wires (Figure 3.15) and make it easier to detect them, the image was converted to grayscale. Then, a Gaussian filter with a kernel size of 9 was applied, and an Otsu thresholding operation was performed. The result of these transformations can be seen in Figure 3.16 and Figure 3.17.

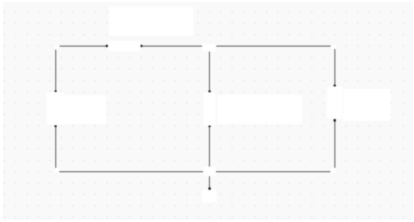


Figure 3.16: Gaussian filter with a kernel size of 9.

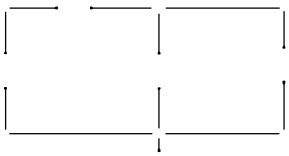


Figure 3.17: Otsu thresholding application.

Block 3 in Figure 3.1 applies the contour finder functions [35] in the image with the inverted binary scale after the threshold operation to detect electrical connections. The contours found by this function are shown in green in Figure 3.18. The block calculates the ends of the wires and identifies the component closest to each end. It then returns a list of vectors, with each vector representing a wire. Each vector includes the coordinates of the center of the component, junction, or node associated with each end found in the segmentation model.

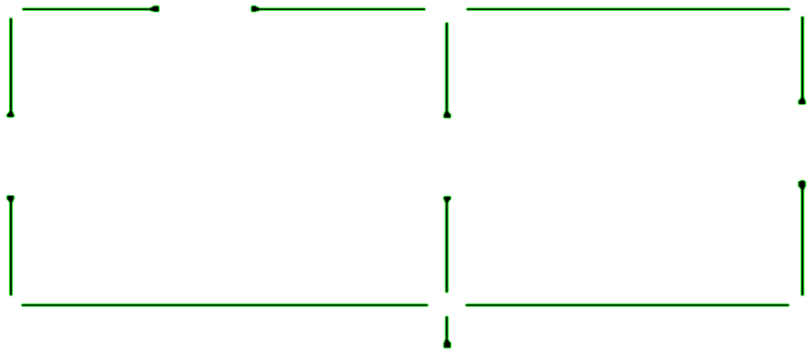


Figure 3.18: Results of the detection of electronic connection in green.

3.6.3 Polarity detection - block 4

This section of the software verifies our segmentation results, checking for components with polarity. If any are found, they are cropped from the original image based on the component's bounding box and then evaluated using a pre-trained polarity model. The software then returns, for each component, the negative and positive polarity with a correction of the center of each polarity in relation to the reference of the original image. This means that it adjusts the coordinates of the center of the component in the original image with the coordinates of each polarity in an image where its size is limited to the size of the bounding box. Additionally, it provides confidence values for each polarity detected.

3.6.4 Processing text and OCR - blocks 5 and 6

Block 5 of Figure 3.1 receives the segmentation results and the original image. It selects the text from the segmentation results and then uses EasyOCR to extract labels and component values from the image. To enhance the accuracy of the OCR, the image undergoes pre-processing, where it is converted to a reverse binary version to highlight the text by inverting the colours. Using an inverse binary image makes EasyOCR's results more reliable than just a binary image, as EasyOCR often confuses the number 0 with the letter "o" in binary images. Switching to the inverse scale makes it easier to detect the text in the image. To eliminate the loose points in the text image resulting from the simulator grid from which the image was taken, an opening filter with a kernel of size 3 is used. Figure 3.19 shows all the processing on the text images.

(a) Sample image text. (b) Inverse image text. (c) Application of opening filter.

Figure 3.19: Text processing.

The text extracted by EasyOCR is analyzed to determine if it represents valid labels. For example, it must start with a letter and be followed by digits, or contain only one letter. After that, the values and units of the texts are identified and extracted, starting from specific characters that indicate the beginning of a value. In theory, after an '=' found by the OCR, the value and unit of the component should follow if the picture shows it. However, Easy OCR sometimes confuses the '=' with the '-' and 'z', which is normal given the small scale of text images. To solve this problem, a function was needed to iterate through all these possibilities.

The location of the texts in the image is then compared with the coordinates of the identified components. The text closest to each component is associated with it, ensuring that the labels and values are correctly assigned to the corresponding components.

The resulting OCR data can contain multiple readings for the same coordinates. Therefore, the results are combined, consolidating multiple readings into a single entry for each set of coordinates. This ensures that each component has a unique and consistent label, value, and unit.

3.6.5 Circuit map

To create the data model referred to in Chapter 3.3.1, it was necessary to combine all the information from blocks 3, 4 and 6 of Figure 3.1: electrical connection detection, polarity detection and OCR, respectively.

In order to visualize the results of the entire project pipeline, it was necessary to adapt the data model, as explained in chapter 3.3.2. I want to highlight the algorithm for determining connections, which in the simulator's data model represents all electrical connections with the same electrical potential.

The algorithm starts by iterating over each wire in the electrical diagram from the first JSON model. The origin and destination are identified for each wire, and their coordinates are determined. If the origin or destination is a node or junction, the coordinates are taken from the respective lists of nodes or junctions in the first JSON model. Otherwise, the coordinates are taken from the list of components in the first JSON model. The source and destination coordinates are then rounded to the nearest 5, due to the limitations of the simulator using a grid. To determine whether the connection is horizontal or vertical, the algorithm calculates the angle of connection between the source and destination coordinates. The source and destination coordinates are adjusted to ensure that the link is horizontal or vertical, i.e. for horizontal links the value of y must be the same at the source and destination, and for vertical links, the value of x must be the same at the ends of the link.

After making the adjustments, the information about the central component, including names, IDs, adjusted coordinates, angle and wire ID, is stored in a list. The function performs a final check to ensure that the coordinates are consistent, i.e. for the same source or destination ID, the coordinates must be the same for any link. In addition, the x and y values are the same for vertical and horizontal links, respectively. For junctions, the function checks and adjusts their coordinates if necessary, as this element is an intersection point of two links that form a 90-degree angle due to the simulator being standardized to a grid.

At this point, it is possible to obtain detailed information about the centres of the ends of the electrical connections, including their names, IDs, adjusted coordinates, connection angle and IDs of the connected wires. What the algorithm does next is assigned for each connection; if the end is a component, the centre value for the component port is associated with the respective connection.

Chapter 4

Performance Evaluation

This chapter showcases the performance of our software tool considering different circuit complexity levels (namely in terms of number of components, branches and nodes) and circuit schematic "sources" (drawn in the QUCS and LTSpice circuit simulators and also by hand). Note that, although our software tool was not targeted for hand-drawn schematics, we decided to consider it to asses how far it could perform.

The "simpler" circuit is a mix series-parallel with two resistors (labeled as R1 and R2) and one capacitor (labeled as C1) and one DC source (labelled V1). The more complex circuit includes a total of 7 passive components (R, L, C) components and 3 different power sources (DC voltage, DC current and AC voltage) in a 9 branches and 5 nodes topology. Besides each component label, the physical quantity (U, R, C) and respective value and unit (voltage of the DC source, resistance of the resistors and capacitance of the capacitor) are also represented in the schematics.

From our experiences, we believe this is a representative set which reflects some complexity concerning the mix between "graphical" and "textual" elements in each image. Mind that what is presented in this chapter is just a selection of examples that we believe showcase, assess and validate our tool in a simple organized and effective way. We did many other tests, including with other "sources", e.g. the one in appendix C where the image has been captured by photographing a circuit schematics drawn in VISIO [22] in an exercise book [31].

4.1 Case studies schematics draw using *QUCS*

The first case study represents a simple computer-designed electrical circuit using the QUCS electrical circuit simulator. The image has a width of 1275 pixels and a height of 677 pixels and is displayed in Figure 4.1.

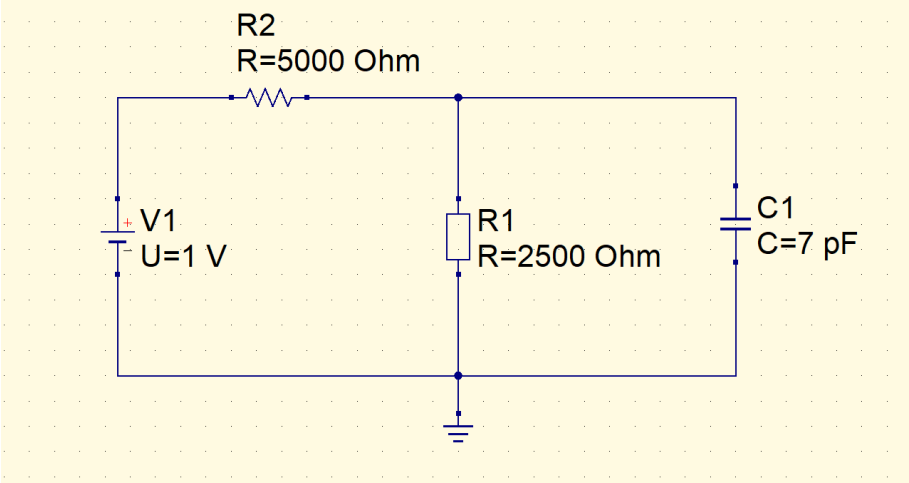


Figure 4.1: Segmentation result for case study 1 - QUCS.

The results of the case study 1 processing can be found at *results/test1_1275_677* in the project's software directory.

Figure Segmentation result for case study 1 displays the segmentation result of the first case study. All the classes classified by this model are visible in the image. Table 4.1 displays the accuracy of each element as well as the average accuracy across all elements. The segmentation model inference time was 106.3ms.

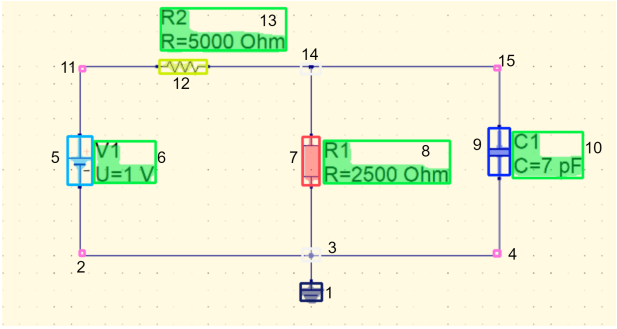


Figure 4.2: Segmentation result for case study 1.

Table 4.1: Accuracy of each element detected for the case of study 1.

Element	Classe Name	Accuracy
1	ground	90.6%
2	junction	66.7 %
3	cross	79.9%
4	junction	68.7 %
5	voltagedc	90.7 %
6	text	93.6 %
7	resistoreu	91.6 %
8	text	93.4%
9	capacitor	86.4 %
10	text	92.2 %
11	junction	69.1 %
12	resistorus	86.4 %
13	text	92.2 %
14	cross	80.7 %
15	junction	67.1 %
Average		82.5 %

In this case study, there is a voltage source in which the polarity needs to be evaluated. Figure 4.3 shows the result of the evaluation of this component, which obtained an average accuracy

percentage of 54.5%.

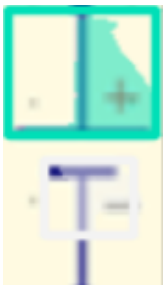
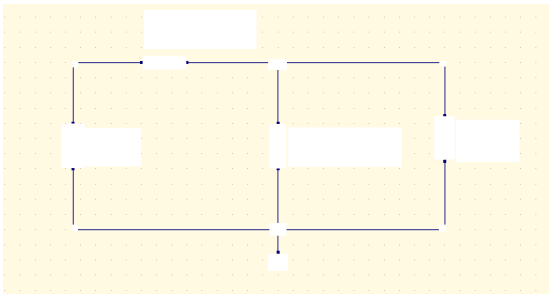
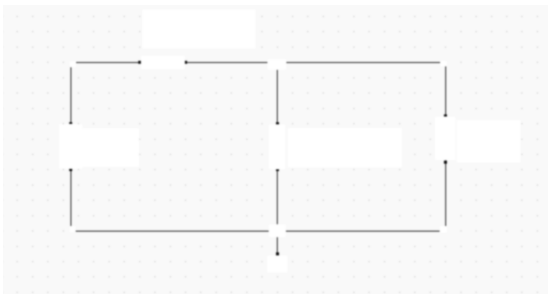


Figure 4.3: Polarity result for case study 1. Blue represents the positive port, and white is associated with the negative port.

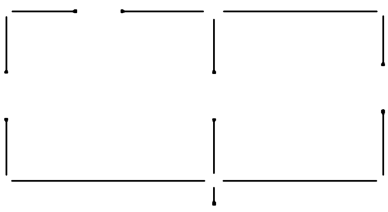
To detect electrical connections, the image was first processed with a Gaussian blur using a kernel of 9, Figure 4.4b. Then, the Otsu threshold was applied to eliminate the noise in the image, Figure 4.4c. With this process it was possible to detect the 12 electrical connections shown in Figure 4.4d.



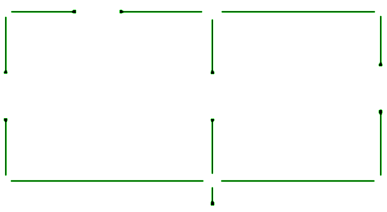
(a) Copy of the original image without the segmentation results.



(b) Gaussian filter with a kernel size of 9.



(c) Otsu thresholding application.



(d) Result of the contours.

Figure 4.4: Results of the detection of electronic connection in green

Table 4.2 shows the results of OCR and the software's interpretation to separate the data into labels, values and units.

Table 4.2: Text results from case study 1.

Text Image	Predicted text	Accuraracy	Label	Value	Unit
6	'V1'	99,7%	V1	-	-
	'U=1 V'	76,0%	-	1	V
8	'R1'	99,9%	R1	-	-
	'R-2500 Ohm'	83,9%	-	2500	Ohm
10	'C1'	99,9%	C1	-	-
	'C-7pF'	94,9%	-	7	pF
13	'R2'	98,4%	R2	-	-
	'R=5000 Ohm'	59,9%	-	5000	Ohm

The data model for this case study was created correctly, and it was possible to integrate it with the U=RIsolve platform, as shown in Figure 4.5. Appendix B represents the JSON model of this case study compatible with U=RIsolve.

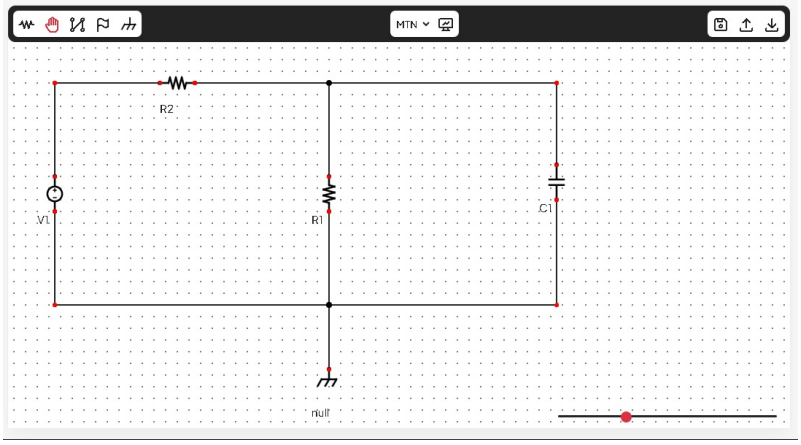


Figure 4.5: Integration result with U=RIsolve for case study 1.

The second case study represents a computer-designed electrical circuit using the QUCS electrical circuit simulator. It is more complex than the first one because this schematic has more components. The image has a width of 1132 pixels and a height of 916 pixels and is displayed in Figure 4.6.

The case study 2 processing results can be found at *results/complexwithbackground* in the project's software directory.

Figure 4.7 displays the segmentation result of the second case study. All the classes classified by this model are visible in the image. Table 4.3 displays the accuracy of each element and the average accuracy across all elements. The segmentation model inference time was 117.3 ms.

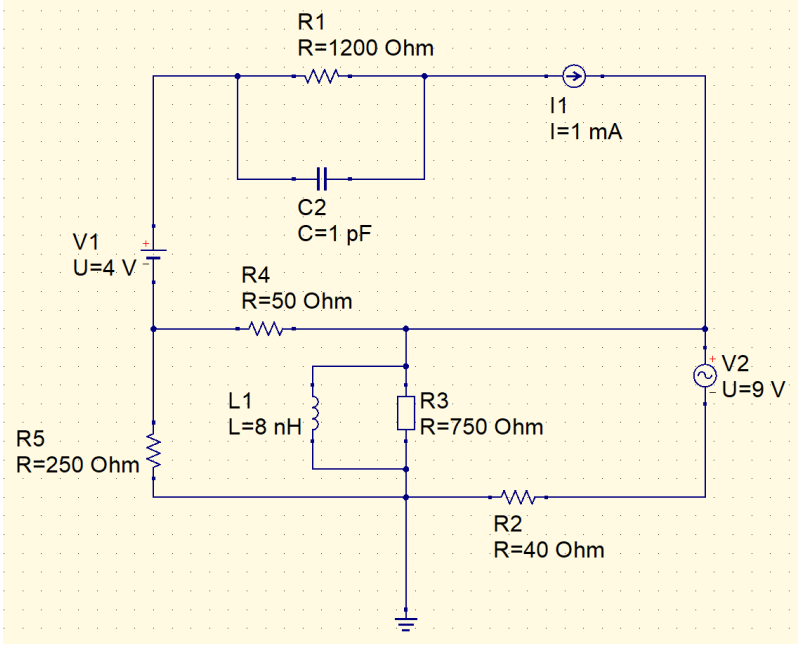


Figure 4.6: Electrical circuit schema for case study 2 - QUCS.

Table 4.3: Accuracy of each element detected for the case of study 2.

Element	Classe Name	Accuracy	Element	Classe Name	Accuracy
1	ground	88.3%	20	resistorus	86.6 %
2	text	92.1 %	21	cross	77.1 %
3	junction	69.9%	22	cross	83.0 %
4	cross	77.3 %	23	text	91.8 %
5	resistorus	87.9 %	24	voltagedc	89.5 %
6	junction	69.9 %	25	text	93.4 %
7	text	92.2 %	26	text	93.1 %
8	resistorus	86.7%	27	junction	29.9 %
9	junction	73.3 %	28	capacitor	89.1 %
10	cross	81.6 %	29	junction	44.6 %
11	text	92.5 %	30	junction	60.2%
12	coil	79.0 %	31	cross	78.5 %
13	resistoreu	90.1 %	32	resistorus	85.1 %
14	text	91.9 %	33	cross	81.9 %
15	junction	49.0 %	34	text	90.6 %
16	cross	83.7 %	35	current	90.8%
17	voltageac	88.9 %	36	junction	70.0 %
18	text	92.0 %	37	text	92.2 %
19	cross	83.4 %	Average		81.1 %

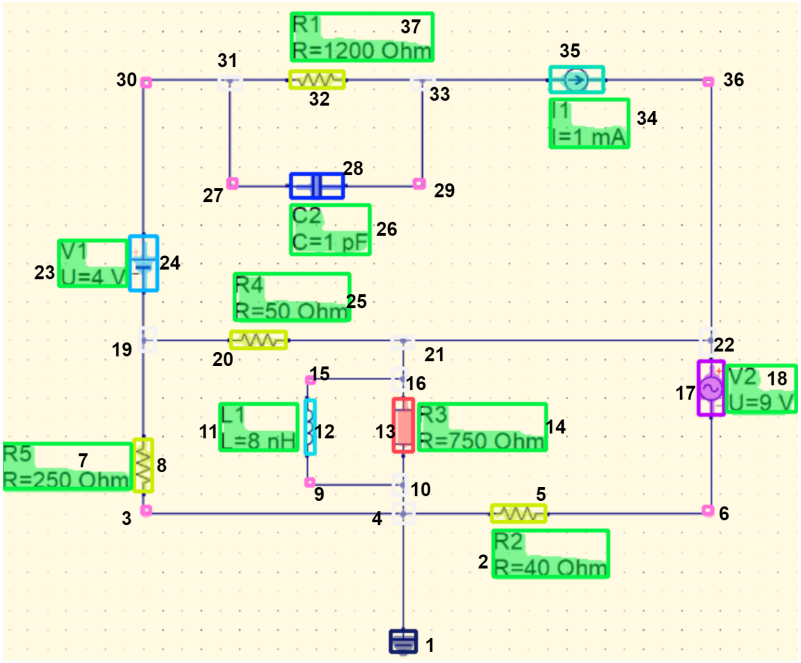


Figure 4.7: Segmentation result for case study 2.

Figure 4.8 illustrates the evaluation of the source polarity model for the second case study. In the voltage sources, blue indicates the positive part and white indicates the negative part. In the current sources, light blue indicates the positive part and dark blue indicates the negative part. The average accuracy for 4.8a is 35.4%, for 4.8b is 74.8%, and for 4.8c is 73.9%.

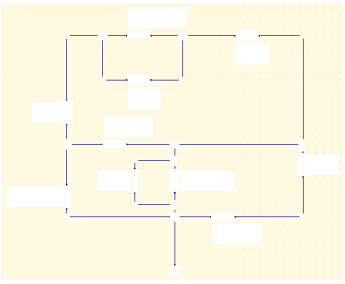
(a) Element 24

(b) Element 35

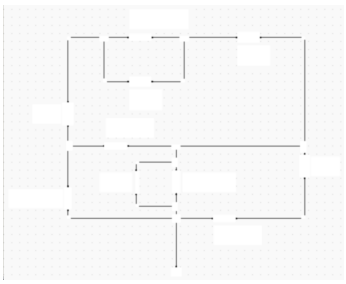
(c) Element 17

Figure 4.8: Text images from segmentation results.

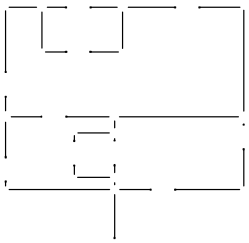
To detect electrical connections, the image was first processed with a Gaussian blur using a kernel of 9, Figure 4.9b. Then, the Otsu threshold was applied to eliminate the noise in the image, Figure 4.9c. Finally, the cv2.findContours() function is applied to the image with the inverted binary scale after the threshold operation, where 31 electrical connections were detected, as demonstrated in Figure 4.9d.



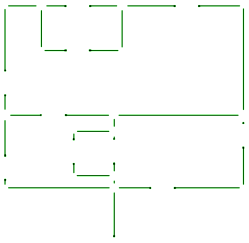
(a) Copy of the original image without the segmentation results



(b) Gaussian filter with a kernel size of 9.



(c) Otsu thresholding application.



(d) Result of the contours.

Figure 4.9: Results of the detection of electronic connection in green.

Table 4.4 shows the results of OCR and the software's interpretation to separate the data into labels, values and units.

The data model for this case study was created correctly, and it was possible to integrate it with the U=RIsolve platform, as shown in Figure 4.10.

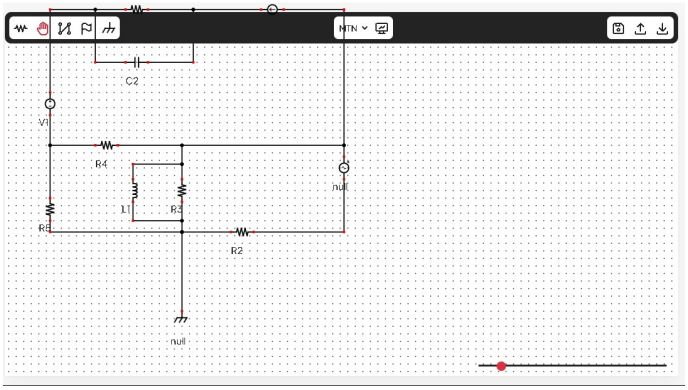


Figure 4.10: Integration result with U=RIsolve for case study 2.

4.2 Case studies schematics draw using *LTspice*

The third case study represents a simple computer-designed electrical circuit using the LTspice electrical circuit simulator. The image has a width of 1233 pixels and a height of 811 pixels and is

Table 4.4: Text results from case study 2.

Text Image	Predicted text	Accuracy	Label	Value	Unit
37	'R1'	99,9%	R1	-	-
	'R=1200 Ohm'	96,2%	-	1200	Ohm
2	'R2'	98,6%	R2	-	-
	'R=40 Ohm'	78,0%	-	40	Ohm
14	'R3'	99.9%	R3	-	-
	'R=750 Ohm'	75,3%	-	750	Ohm
25	'R4'	94,9%	R4	-	-
	'R=50 Ohm'	37,5%	-	50	Ohm
7	'R5'	68,4%	R5		
	'r-250 Ohm'	87,9%	-	250	Ohm
35	'11'	97,8%	-	-	-
	'I=1 mA'	56,1%	-	1	mA
23	'V1'	98,8%	V1	-	-
	'U=4 V'	97,5%	-	4	V
18	'V2'	97,8 %	V2	-	-
	'U=9 V'	54,9%	-	9	V
26	'C2'	97,9%	C2	-	-
	'C= 1 pF'	69,9%	-	1	pF
11	'L1'	99,6%	L1	-	-
	'L- 8 nH'	60,2%	-	8	nH

displayed in Figure 4.11.

The case study 3 processing results can be found at *results/simple_ltspice* in the project's software directory.

Figure 4.12 displays the segmentation result of the third case study. All the classes classified by this model are visible in the image. Table 4.5 displays the accuracy of each element as well as the average accuracy across all elements. The segmentation model inference time was 153.0 ms.

In this case study, there is a voltage source in which the polarity needs to be evaluated. Figure 4.13 shows the result of the evaluation of this component, which obtained an average accuracy percentage of 62.9%.

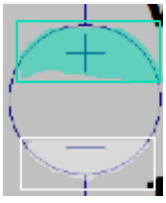


Figure 4.13: Polarity result for case study 3. Blue represents the positive port, and white is associated with the negative port.

In this case study, the image processing for detecting the electronic connections differs from the previous cases. After removing the elements found in the segmentation, the resulting image,

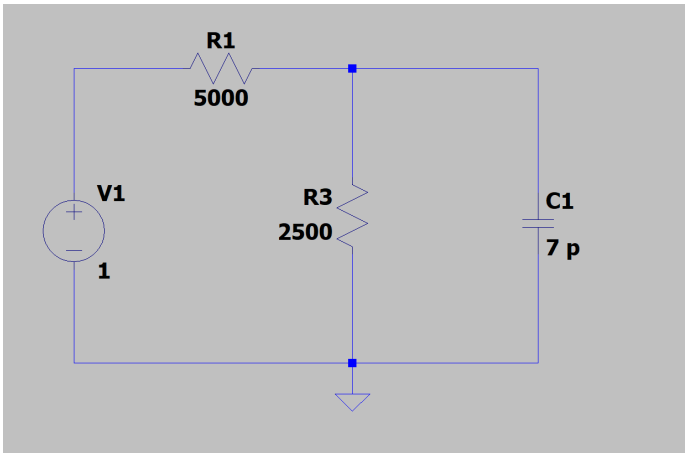


Figure 4.11: Electrical circuit schema for case study 3 - LTspice.

Figure 4.14a, does not have noise. In this case, applying a binary threshold filter with a threshold value of 150 to the grayscale image, Figure 4.14b, is sufficient. Finally, the cv2.findContours() function is applied to the inverted binary image, and it successfully detected 12 electrical connections, as shown in Figure 4.14d.

Table 4.6 shows the results of OCR and the software's interpretation to separate the data into labels, values and units.

Table 4.6: Text results from case study 3.

Text Image	Predicted text	Accuracy	Label	Value	Unit
5	'1'	99.9%	-	1	-
7	'V1'	70.2%	V1	-	-
8	'2500'	99.9%	-	2500	-
9	'R3'	99.9%	R3	-	-
12	'C1'	95.4%	C1	-	-
13	'7 p'	99.9%	-	7	p
15	'5000'	98.8%	-	5000	-
19	'R1'	59.9%	R1	-	-

The data model for this case study was created correctly, and it was possible to integrate it with the U=RIsolve platform, as shown in Figure 4.15.

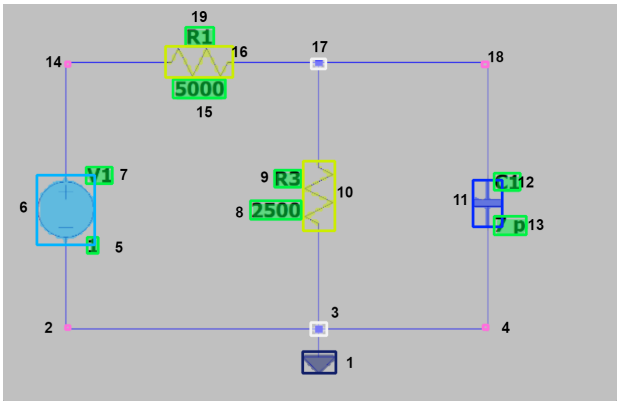


Figure 4.12: Segmentation result for case study 3.

Table 4.5: Accuracy of each element detected for the case of study 3.

Element	Classe Name	Accuracy
1	ground	90.0%
2	junction	64.6 %
3	cross	81.5%
4	junction	63.2 %
5	text	90.2 %
6	voltagedc	94.7 %
7	text	89.0 %
7	resistoreu	91.6 %
8	text	89.7%
9	text	87.8 %
10	resistorus	88.9 %
11	capacitor	83.2 %
12	text	88.1 %
13	text	84.1 %
14	junction	58.5 %
15	text	86.7 %
16	resistorus	87.3&
17	cross	81.&
18	junction	58.2&
19	text	86.7&
Averege		82.2 %

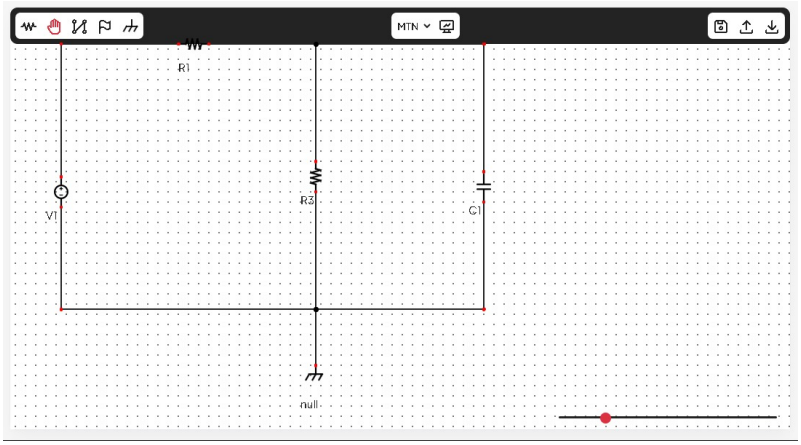


Figure 4.15: Integration result with U=RIsolve for case study 3.

The fourth case study represents a computer-designed electrical circuit using the LTspice electrical circuit simulator. It is the same circuit that was used in case study one but represented in LTspice. The image has a width of 868 pixels and a height of 622 pixels, slightly smaller than case study 2, and is shown in Figure 4.16.

(a) Copy of the original image without the segmentation results

(b) Gray image

(c) Binary thresholding application.

(d) Result of the contours.

Figure 4.14: Electrical connection detection for case study 3.

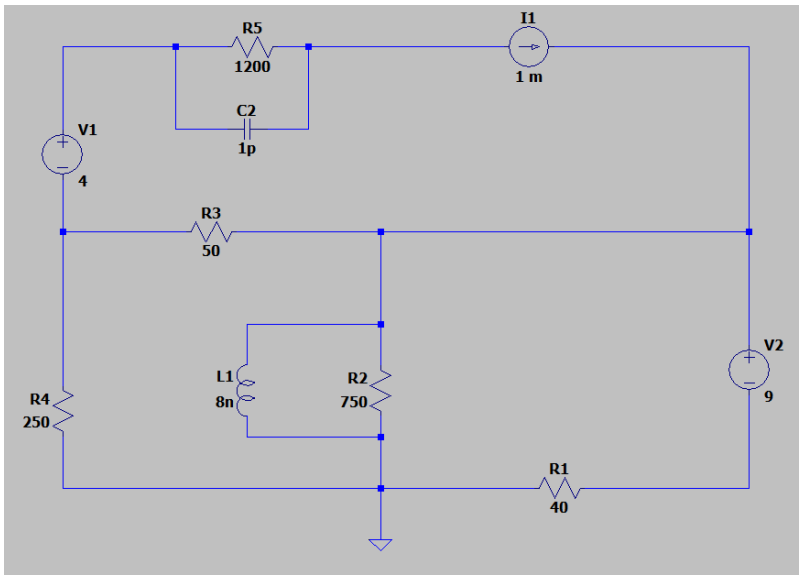


Figure 4.16: Electrical circuit schema for case study 4 - LTspice.

Figure 4.17 displays the segmentation result of the fourth case study. All the classes classified by this model are visible in the image. Table 4.7 displays the accuracy of each element and the average accuracy across all elements. The segmentation model inference time was 103.0 ms.

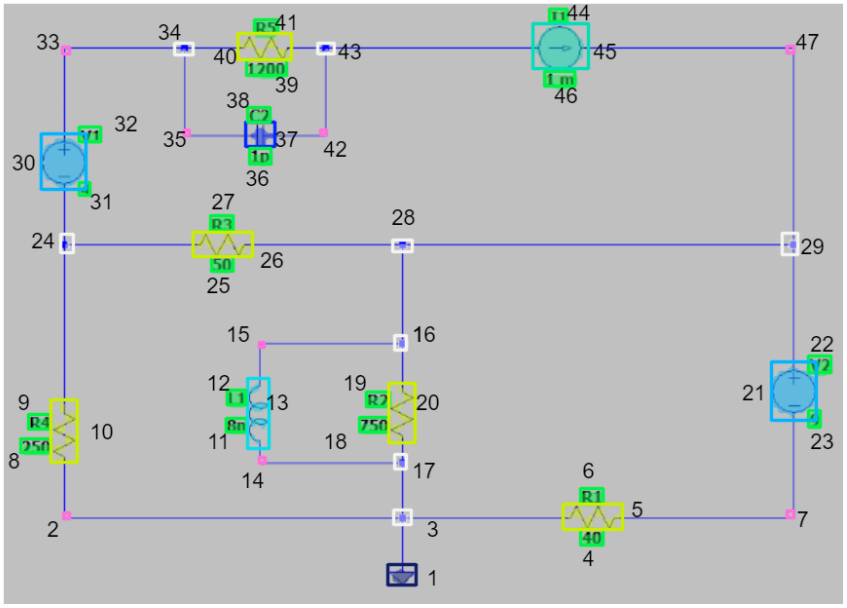
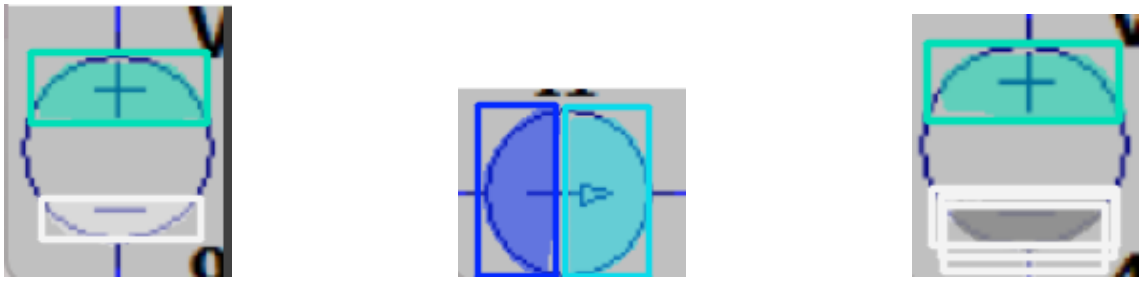


Figure 4.17: Segmentation result for case study 4.

Table 4.7: Accuracy of each element detected for the case of study 4.

Element	Classe Name	Accuracy	Element	Classe Name	Accuracy
1	ground	87.0%	25	text	83.2 %
2	junction	30.4 %	26	resistorus	84.6 %
3	cross	76.4%	27	text	83.9 %
4	text	79.3 %	28	cross	77.7 %
5	resistorus	85.1 %	29	cross	70.4 %
6	text	84.3 %	30	voltagedc	90.0 %
7	junction	27.7 %	31	text	68.0 %
8	text	81.6%	32	text	85.3 %
9	text	83.6 %	33	junction	20.7 %
10	resistorus	86.2 %	34	cross	79.8%
11	text	82.0 %	35	junction	62.6 %
12	text	86.6 %	36	text	86.5 %
13	coil	82.3 %	37	capacitor	76.9 %
14	junction	43.8 %	38	text	86.9 %
15	junction	44.7 %	39	text	80.9%
16	cross	77.6%	40	resistorus	87.7 %
17	cross	78.4 %	41	text	86.0 %
18	text	83.5 %	42	junction	61.3 %
19	text	84.1 %	43	cross	75.3 %
20	resistorus	86.5 %	44	text	74.4 %
21	voltagedc	91.9 %	45	current	91.0 %
22	text	83.6 %	46	text	83.6 %
23	text	77.8 %	47	cross	43.5 %
24	cross	75.4 %	Average		75.2%

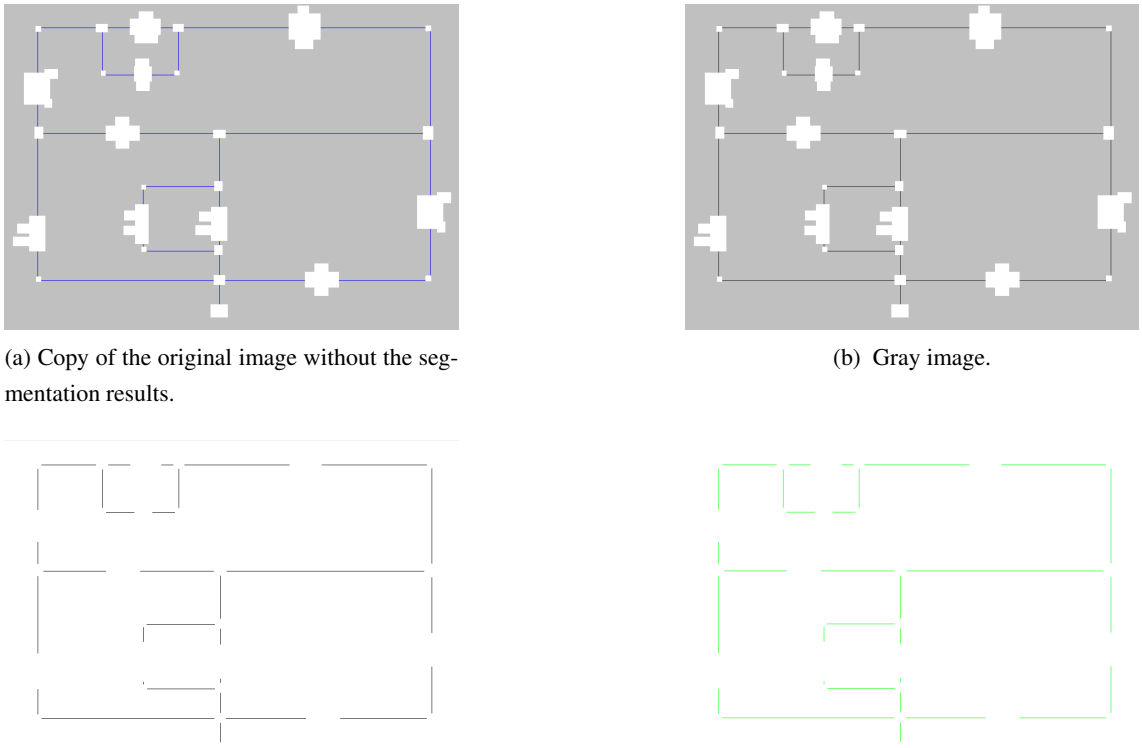
Figure 4.18 illustrates the evaluation of the source polarity model for the second case study. In the voltage sources, blue indicates the positive part, and white indicates the negative part. In the current sources, light blue indicates the positive part, and dark blue indicates the negative part. The average accuracy for (a) is 69.7%, for (b) is 85.3%, and for (c) is 70.5%.



(a) Element 30 (b) Element 45 (c) Element 21

Figure 4.18: Source images from polarity results.

The method used to find the electrical connections was the same as in the previous case study, as shown in Figure 4.19. A total of 31 electrical connections were identified.



(a) Copy of the original image without the segmentation results.

(b) Gray image.

(c) Binary thresholding application.

(d) Result of the contours.

Figure 4.19: Electrical connection detection .

Table 4.8 shows the results of OCR and the software's interpretation to separate the data into labels, values and units.

Table 4.8: Text results from case study 4.

Text Image	Predicted text	Accuracy	Label	Value	Unit
4	’40’	99.9%	-	40	-
6	’R1’	84.9%	R1	-	-
8	’250’	99.9%	-	250	-
9	’R4’	98.3%	R4	-	-
11	’8n’	99.9%	-	8	n
12	’L1’	82.7%	L1	-	-
18	’750’	99.9%	-	750	-
19	’R2’	96.2%	R2	-	-
22	’V2’	97.9%	V2	-	-
23	’9’	100%	-	9	-
25	’50’	99.9%	-	50	-
31	”	0%	-	-	-
32	’V1’	90.2%	V1	-	-
36	’1p’	37.5%		1	p
38	’C2’	96.0%	C2	-	-
39	’1200’	99.9%	-	1200	-
41	’RS’	69.5 %	-	-	-
44	’M1’	92.1 %	M1	-	-
46	’1m’	97.6 %	-	1	m

The data model for this case study was created correctly, and it was possible to integrate it with the U=RIsolve platform, as shown in Figure 4.20.

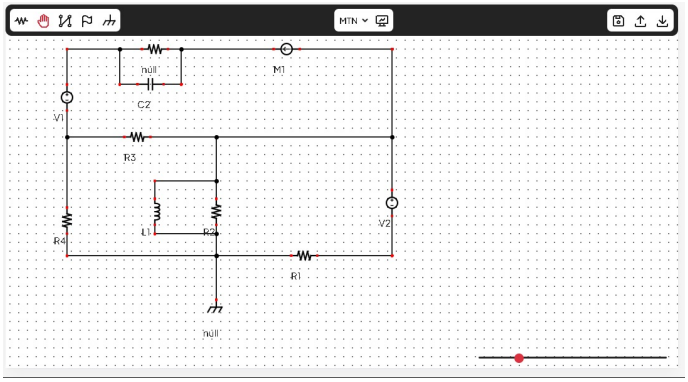


Figure 4.20: Integration result with U=RIsolve for case study 4.

4.3 Hand-written case studies

In addition to exploring the performance limits of the software, the fifth case study was created by hand and is shown in Figure 4.21. The image has a height of 700 pixels and a width of 1218 pixels.

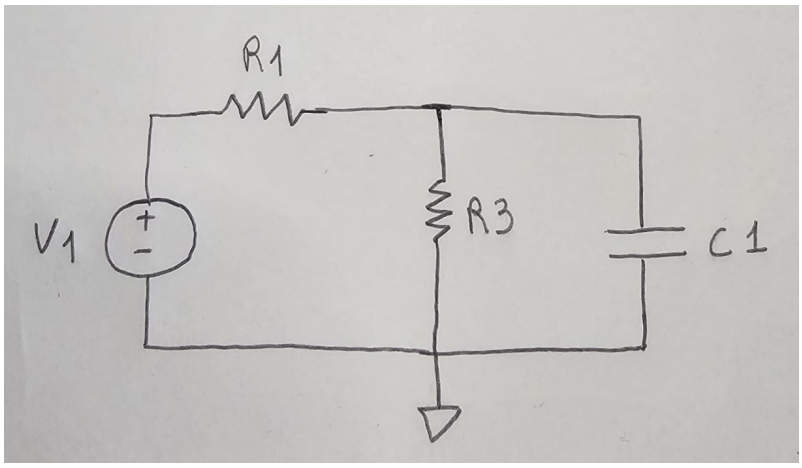


Figure 4.21: Segmentation result for case study 5 - hand drawn.

The case study 5 processing results can be found at *results/manual_teste* in the project's software directory.

Figure 4.22 displays the segmentation result of the third case study. All the classes classified by this model are visible in the image. Table 4.9 displays the accuracy of each element and the average accuracy across all elements. The segmentation model inference time was 114.8 ms.

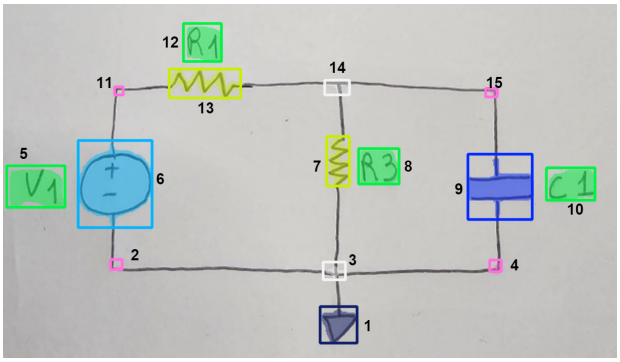


Figure 4.22: Segmentation result for case study 5.

Table 4.9: Accuracy of each element detected for the case of study 5.

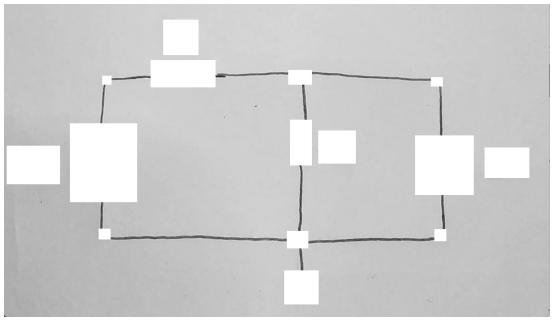
Element	Classe Name	Accuracy
1	ground	85.8%
2	junction	38.2 %
3	cross	62.9%
4	junction	45.3 %
5	text	77.1 %
6	voltagedc	91.8 %
7	resistorus	84.5 %
8	text	88.1%
9	capacitor	83.7 %
10	text	86.5 %
11	junction	44.8 %
12	text	86.1 %
13	resistorus	87.9%
14	cross	47.0 %
15	junction	65.3 %
Average		71.7 %

Figure 4.23 represents the polarity the result model for the DC voltage represented in this study case. The average accuracy of this prediction is 86.6%.

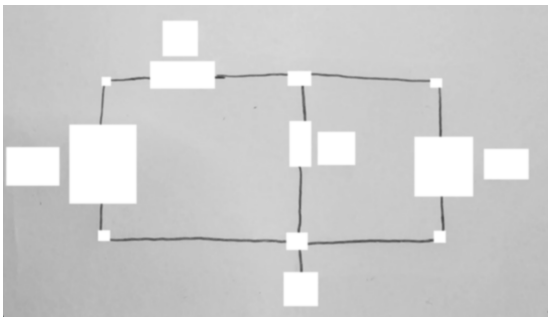
The image was first processed with a Gaussian blur to detect electrical connections using a kernel size of 7 (Figure 4.24b). Next, the mean adaptive threshold with a threshold value of 140

Figure 4.23: Polarity result for case study 5. Blue represents the positive port, and white is associated with the negative port.

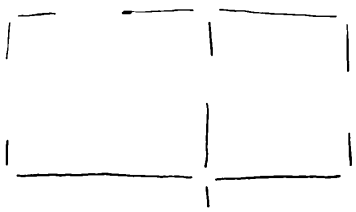
was applied to remove noise in the image (Figure 4.24c). Finally, the cv2.findContours() function was used on the image with the inverted binary scale after the threshold operation, resulting in the detection of 12 electrical connections (as shown in Figure 4.24d).



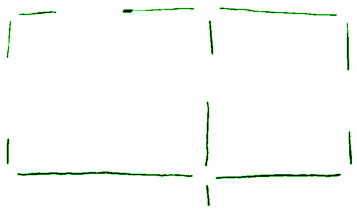
(a) Copy of the original image without the segmentation results



(b) Gaussian filter with a kernel size of 7.



(c) Mean adaptive threshold application.



(d) Result of the contours.

Figure 4.24: Electrical connection detection for study case 5.

Table 4.10 shows the results of OCR and the software's interpretation to separate the data into labels, values and units.

Table 4.10: Text results from case study 5.

Text Image	Predicted text	Accuracy	Label	Value	Unit
5	'VA'	44.9%	-	-	-
8	'R3'	95.7%	R3	-	-
10	'C1'	31.8%	C1	-	-
12	'Ri'	99.9%	-	-	-

The data model for this case study was created correctly but with an error in one electrical connection, and it was possible to integrate it with the U=RIsolve platform, as shown in Figure 4.25.

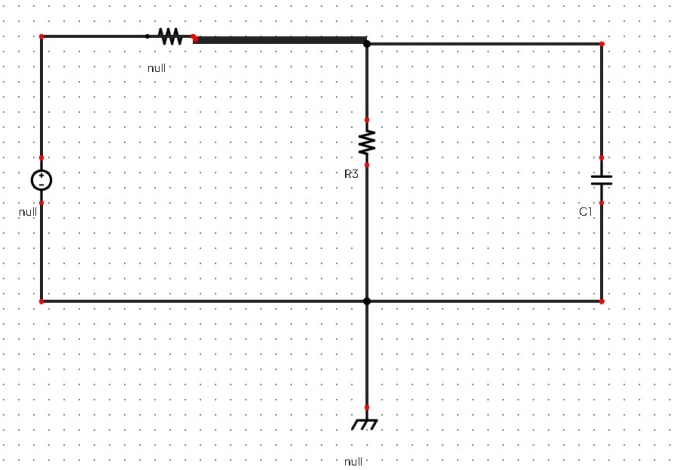


Figure 4.25: Integration result with U=RIsolve for case study 5.

The sixth case involves the same circuit as in cases 2 and 4, but hand-drawn. The image has a width of 800 pixels and a height of 765 pixels and is displayed in Figure 4.26.

The case study 6 processing results can be found at *results/complex_manual* in the project's soft- ware directory

Figure 4.27 displays the segmentation result of the sixth case study. All the classes classified by this model are visible in the image. Table 4.11 displays the accuracy of each element and the average accuracy across all elements. The segmentation model inference time was 94.8 ms.

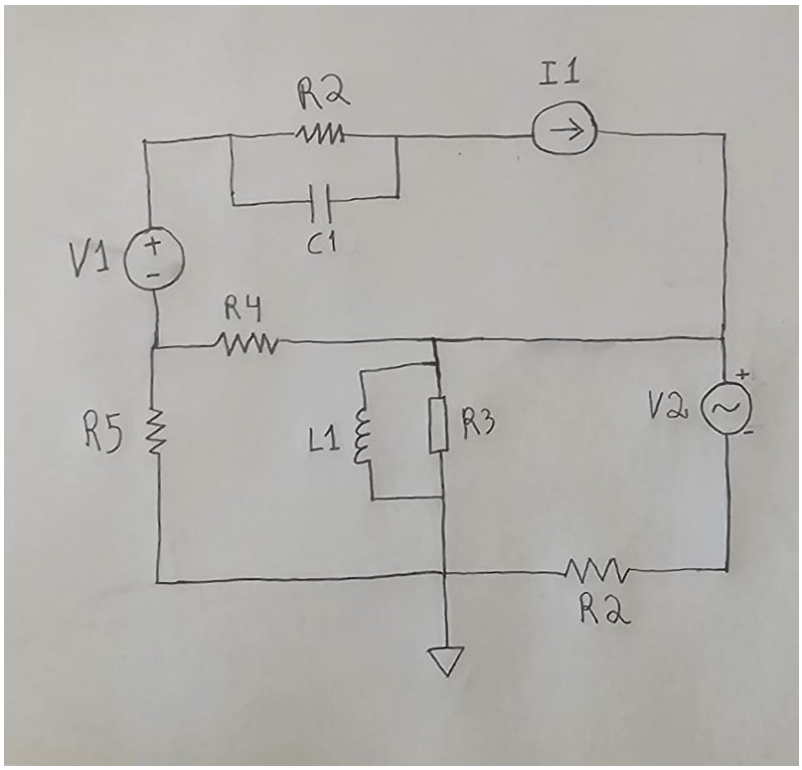


Figure 4.26: Electrical circuit schema for case study 6 - hand drawn.

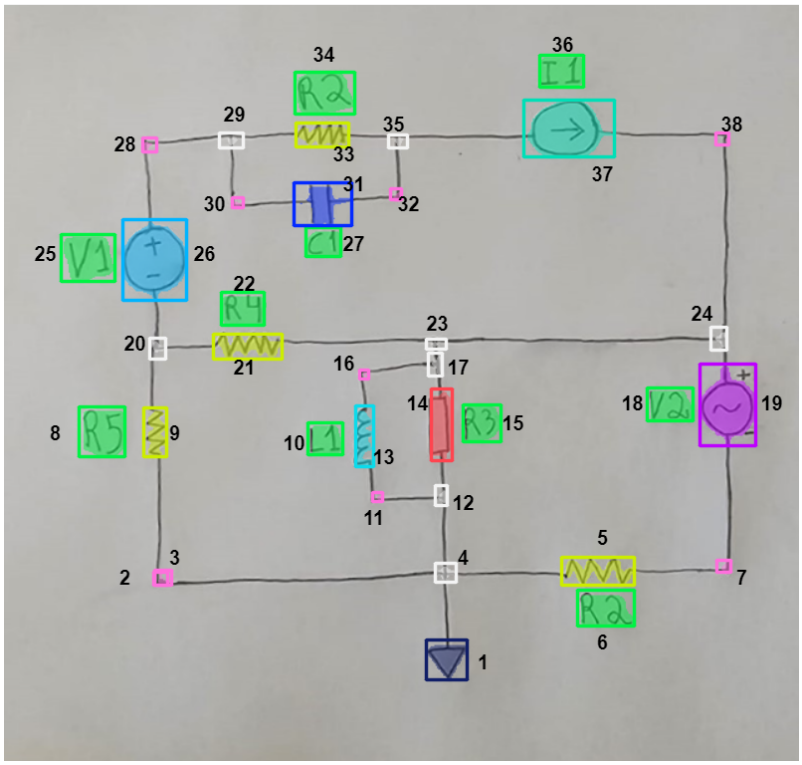
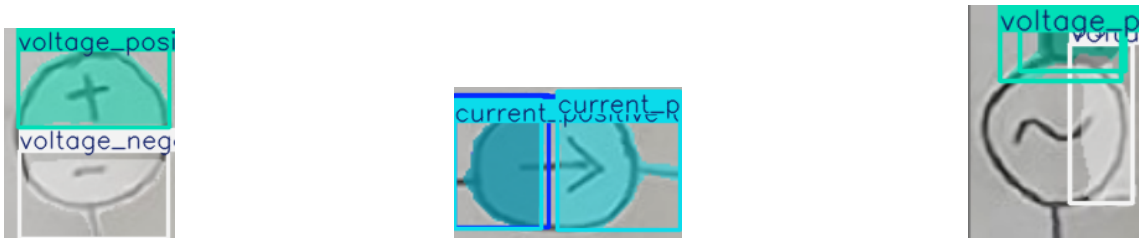


Figure 4.27: Segmentation result for case study 6.

Table 4.11: Accuracy of each element detected for the case of study 6.

Element	Classe Name	Accuracy	Element	Classe Name	Accuracy
1	ground	82.6%	20	cross	52.0 %
2	junction	30.4 %	21	resistorus	83.8 %
3	junction	39.3%	22	text	80.9 %
4	cross	67.5 %	23	cross	38.0 %
5	resistorus	85.4 %	24	cross	49.5 %
6	text	84.3 %	25	text	76.9 %
7	junction	50.0 %	26	voltagedc	90.6 %
8	text	86.0%	27	text	74.9 %
9	resistorus	86.6 %	28	junction	40.9 %
10	text	80.7 %	29	cross	54.8%
11	junction	35.8 %	30	junction	46.6 %
12	cross	45.5 %	31	capacitor	82.2 %
13	coil	80.0 %	32	junction	47.6 %
14	resistoreu	74.4 %	33	resistorus	82.8 %
15	text	78.4 %	34	text	86.6%
16	junction	27.7%	35	junction	42.1%
17	cross	58.8%	36	text	81.6 %
18	text	82.7 %	37	current	87.0 %
19	voltagedc	88.6 %	38	junction	31.7 %
Average		65.7 %			

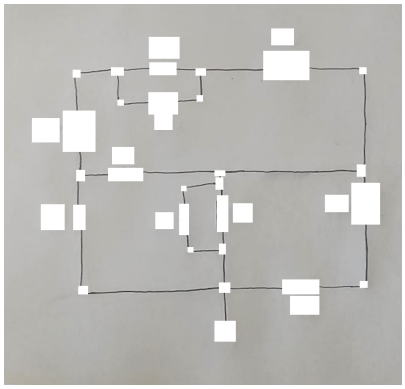
Figure 4.28 illustrates the evaluation of the source polarity model for the second case study. In the voltage sources, blue indicates the positive part and white indicates the negative part. In the current sources, light blue indicates the positive part and dark blue indicates the negative part. The average accuracy for 4.28a is 75.5%, for 4.28b is 64.5%, and for 4.28c is 40.7%.



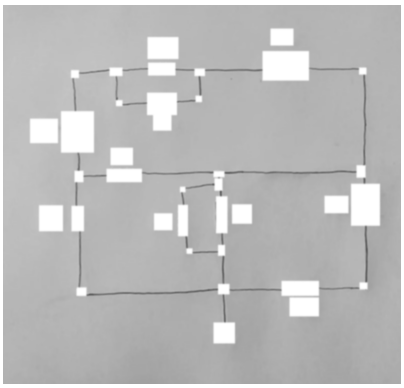
(a) Element 26 (b) Element 37 (c) Element 19

Figure 4.28: Sources images from polarity results.

In this case study, the same techniques as in the fifth case study were used to determine the electrical connection, as seen in Figure C.4, and 29 electrical connections were detected.



(a) Copy of the original image without the segmentation results



(b) Gaussian filter with a kernel size 7

(c) Mean adaptive thresholding application.

(d) Result of the contours.

Figure 4.29: Electrical connection detection .

Table 4.12 shows the results of OCR and the software's interpretation to separate the data into labels, values and units.

Table 4.12: Text results from case study 6.

Text Image	Predicted text	Accuracy	Label	Value	Unit
6	'R5'	41.8%	R5	-	-
8	'85'	40.7%	R3	-	-
10	'L1'	98.4%	L1	-	-
15	'R3'	83.1%	R3	-	-
18	'V3'	60.0%	V3	-	-
22	'R4'	95.1%	R4	-	-
25	'V1'	88.3%	V1	-	-
27	'C1'	80.0%	C1	-	-
34	'R?'	21.1%	-	-	-
36	'Il'	21.7%	-	-	-

The data model for this case study was created correctly, but the data model has some errors defining the electrical connection, and it was possible to integrate it with the U=RIsolve platform, as shown in Figure 4.30.

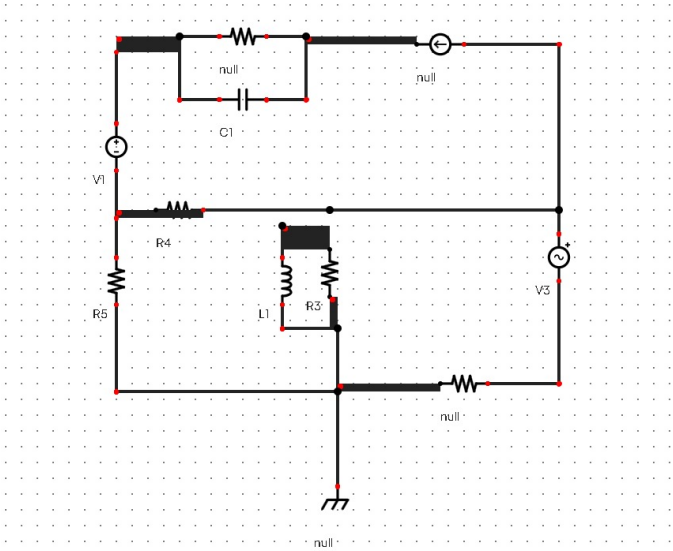


Figure 4.30: Integration result with U=RIsolve for case study 6.

4.4 Overall appreciation

In the context of the segmentation model, on computer-designed electrical circuits cases of study, consistently achieves accurate detection of all components and text, with an average accuracy exceeding 80%. However, it exhibited lower accuracy in identifying the 'junction' and 'cross' classes and may sometimes fail to detect them based on the applied confidence level. In the realm of hand-drawn illustrations, the model demonstrates an impressive performance due to the absence of hand-drawn electrical schematic images in the training dataset. The automated detection of electrical connections is unattainable due to the necessity of employing diverse image processing techniques tailored to the specific case in order to isolate the electrical connections selectively.

The OCR tool is capable of identifying text, including handwritten text. The algorithm designed to categorize text into label, value, and unit has demonstrated successful performance in these specific case studies. However, this tool was unable to identify the ohm symbol (Ω).

The polarity model occasionally fails to yield the intended outcome of distinguishing a positive and a negative zone of the source. As illustrated in Figure 4.18c, the category 'voltage negative' presents multiple solutions, necessitating code implementation to ascertain the optimal result. In instances where the polarity model fails to delineate the positive and negative zones accurately, errors akin to those depicted in Figure 4.30 may arise, wherein the current source is oriented in the opposite direction.

In case studies 5 and 6, the integration with the U=RIsolve platform is failing because the algorithm for approximating the coordinates of the elements is not working correctly. This results in thicker connections, as shown in Figures 4.25 and 4.30. In case study 6, the segmentation module identifies nodes 23 and 17 (Figure 4.27) as being too close together, leading to an incorrect definition of the electrical connection.

The software necessitates an algorithm to adjust the size of the electrical connection to ensure that it fits within the grid of the U=RIsolve simulator, as depicted in Figure 4.10.

Chapter 5

Conclusions

The primary motivation for this project was to enhance the teaching and learning of electrical circuit analysis, specifically for students in the early stages of the Electrical Engineering course. This dissertation focused on developing a software module capable of interpreting and modelling electrical diagrams from an image.

Initially, this journey started with gaining knowledge about computer vision and machine learning techniques in a computer vision course. The first experiment for this project involved developing an algorithm to classify electrical components based on hand-drawn sketches. Self-designed and realised work on this course. Although the classification approach was not used in this project, it proved to be a valuable learning experience for applying existing machine learning methods.

One of the goals of this project was to create a dataset of computer-generated electrical circuit diagrams. Two datasets were created. One of them was needed for segmenting the electrical circuit, while the other was created to accomplish one of the project's objectives, which was to determine the polarity direction of the voltage and current sources. Collecting these images and, in particular, annotating them pixel by pixel (to use image segmentation) required a significant amount of time and perseverance. Hopefully, using image segmentation techniques allows for better determination of components and associated labels, even if some components have overlapping boxes.

The initial segmentation approach involved using the "wires" class from the segmentation dataset to identify electrical connections in the images. However, this approach resulted in unsatisfactory results for this project. The electrical connections were not consistently detected; when they were, their confidence value was lower than other classes in the images. In order to address this issue, a model was trained without this class. This allowed for the removal of all other elements defined in the segmentation dataset from the image. By detecting these elements and applying computer vision techniques, all the components, nodes and electrical connections in an image of an electronic circuit were successfully located and identified. Additionally, text regions of the circuit were also detected. An OCR engine was used to interpret the text regions of interest in the images. As a way to solve the problem of identifying the name, value and unit of each

component, it was necessary to create functions to interpret the OCR output.

While integrating the JSON data into the U=RIsolve software, some difficulties and limitations were found. Identified limitations included placing of components in a grid of 10 pixels, component size also in multiple of 10 pixels and inexistence of polarized components. As such, improvements to the JSON format were defined and exported to the rest of the U=RIsolve web application. Additionally, assigning electrical meaning to the electrical connections and ensuring that all connections with the same electrical potential are joined posed a challenge. This work also included the creation of a graph search in order to adapt connections to the standard format.

Overall, this project has achieved its objectives: uploading the image of a circuit, our software module sends the JSON circuit model to the U=RIsolve web application. The images are supposed to be generated by computer-based circuit editors. Limited tests with handwritten circuits were also performed to assess how far the software could perform. A combination of machine learning and "classic" computer vision techniques were used throughout this work.

5.1 Future Work

The developed systems leave room for future work, as they could be further improved to achieve better performance.

Naturally, adding more images to the datasets would benefit segmentation and polarity definition blocks. Using a neural network other than YOLO, such as Mask R-CNN, and compare its behaviour with YOLO could be an alternative. We could see some performance gains from different models for this data.

In order to enable the automated recognition of electrical connections, incorporating an algorithm facilitating automatic image processing became imperative, independently of the electrical schematic under consideration. Should this approach prove ineffective, an alternative solution could involve training a model tailored specifically for assessing electrical connections.

The OCR approach can be enhanced by incorporating machine learning methods to recognize specific unit symbols in the images, like the ohm symbol (Ω). Before running the OCR, a model trained on these symbols can predict their presence and include this information in the OCR results, leading to a more precise interpretation of the text.

References

[1] A. Rosebrock. Adaptive Thresholding with OpenCV (cv2.adaptiveThreshold). Available at https://pyimagesearch.com/2021/05/12/adaptive-thresholding-with-opencv-cv2-adaptivethreshold/. Last accessed on 2024/6/25.

[2] MS Abhinav, PN Ganesh, PV Nishanth, K Bhargavi, et al. Electric circuit diagram detection recognition and simulation. *International Research Journal of Engineering and Technology(IRJET)*, 7(09), 2020.

[3] Nisha Arya Ahmed. Mean average precision (map): A complete guide, Dec. 2023. Available at https://kili-technology.com/data-labeling/machine-learning/mean-average-precision-map-a-complete-guide#mean-average-precision-(map)--definition, Last accessed on 2024/5/29.

[4] Jaided AI. Ultralytics YOLO, May 2023.

[5] Ethem Alpaydin. *Introduction to machine learning*. MIT press, Fourth edition, 2020.

[6] Bhandari Aniruddha. Understanding & interpreting confusion matrix in machine learning (updated 2024), Dec. 2023. Available at https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/, Last accessed on 2024/5/29.

[7] Hugo Miguel Ribeiro Barbosa. Interpretação e modelação de esquemas elétricos com recurso a algoritmos de visão computacional. Master's thesis, Instituto Superior de Engenharia do Porto (ISEP), 2022.

[8] Hritam Basak, Rohit Kundu, and Ram Sarkar. Mfsnet: A multi focus segmentation network for skin lesion segmentation. *Pattern Recognition*, 128:108673, 2022. https://www.sciencedirect.com/science/article/pii/S0031320322001546.

[9] Rein van den Boomgaard. Syllabus 2016 - 2017, 2016. Available at https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20162017/LectureNotes/IP/LocalOperators/bilateralfilter.html#bilateral-filtering.

[10] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[11] PyTorch Contributors. Adamw, 2023. Available at https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html, Last accessed on 2024/06/13.

[12] PyTorch Contributors. Sgd, 2023. Available at https://pytorch.org/docs/stable/generated/torch.optim.SGD.html, Last accessed on 2024/06/13.

[13] Emanuel Jorge Pereira da Cunha. Identificação de componentes de esquemas elétricos através de técnicas de classificação de imagem. Master's thesis, Instituto Superior de Engenharia do Porto (ISEP), 2024.

[14] datagen. tech. Understanding vgg16: Concepts, architecture, and performance, Apr. 2017. Available at https://datagen.tech/guides/computer-vision/vgg16/#, Last accessed on 2023/12/28.

[15] Shashika Dilhani. Digital image processing filters, Aug 2021. Available at https://medium.com/@shashikadilhani97/digital-image-processing-filters-832ec6d18a73.

[16] B. Dwyer, J. Nelson, T. Hansen, et al. Roboflow (version 1.0), 2024. Available from https://roboflow.comLast accessed on 2024/6/11.

[17] A. Huamán. Basic thresholding operations., Dec 2021. Available at https://docs.opencv.org/4.5.5/db/d8e/tutorial_threshold.htmlLast accessed on 2024/6/3.

[18] A. Huamán. Morphological transformations, May 2021. Available at https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.htmlLast accessed on 2024/6/8.

[19] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, January 2023.

[20] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. " O'Reilly Media, Inc.", 2016.

[21] Francisco Melo. *Area under the ROC Curve*, pages 38–39. Springer New York, New York, NY, 2013.

[22] Microsoft Corporation. Microsoft Visio, 2018. Available at https://products.office.com/en/visio/flowchart-software. Last accessed on 2024/6/25.

[23] Momina Moetesum, Syed Waqar Younus, Muhammad Ali Warsi, and Imran Siddiqi. Segmentation and recognition of electronic components in hand-drawn circuit diagrams. *EAI Endorsed Transactions on Scalable Information Systems*, 5(16):e12–e12, 2018.

[24] Akshatha Mohan, Athulya Mohan, B Indushree, M Malavikaa, and C P Narendra. Generation of netlist from a hand drawn circuit through image processing and machine learning. In *2022 2nd International Conference on Artificial Intelligence and Signal Processing (AISP)*, pages 1–4, 2022.

[25] Mark Nixon and Alberto Aguado. *Feature extraction and image processing for computer vision*. Academic press, 2019.

[26] OpenCV. About. Available at https://opencv.org/about/, Last accessed on 2024/5/31.

[27] OpenCV. Eroding and dilating, May 2024. Available at https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html, Last accessed on 2024/6/.

[28] OpenCV. Image thresholding, Jul 2024. Available at https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.htmlLast accessed on 2024/6/3.

[29] OpenCV. Introduction, May. 2024. Available at https://docs.opencv.org/4.x/d1/dfb/intro.html, Last accessed on 2024/5/31.

[30] João Pedro. Detailed explanation of yolov8 architecture — part 1, Dec. 2023. Available at https://medium.com/@juanpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e, Last accessed on 2024/6/1.

[31] Francisco Pereira, António Castro Vide, Maria Judite Ferreira, Susana Amado, and Ana Viana. Exercícios propostos: Condensadores em corrente contínua, March 2019. Licenciatura em Engenharia Eletrotécnica e de Computadores, Instituto Superior de Engenharia do Porto.

[32] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273. International World Wide Web Conferences Steering Committee, 2016.

[33] Ângelo Pinheiro. Unificação e expansão da plataforma u=risolve academy: Projeto e implementação de um editor de circuitos. Master's thesis, Instituto Superior de Engenharia do Porto (ISEP), 2024. work in progress.

[34] Adrian Rosebrock. Intersection over union (iou) for object detection, Nov. 2016. Available at https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, Last accessed on 2024/5/29.

[35] Raqueeb Shaikh. Opencv (findcontours) detailed guide, Jun 2022. Available at https://medium.com/analytics-vidhya/opencv-findcontours-detailed-guide-692ee19eeb18Last accessed on 2024/7/11.

[36] Jacob Solawetz and Francesco. Top models for instance segmentation reviewed, Jan. 2023. Available at https://blog.roboflow.com/whats-new-in-yolov8/, Last accessed on 2024/6/1.

[37] Felix Thoma, Johannes Bayer, Yakun Lie, and Andreas Dengel. A public ground-truth dataset for handwritten circuit diagram images, Nov. 2022. Available at https://doi.org/10.5281/zenodo.7355865, Last accessed on 2024/06/12.

[38] Leo Ueno. Best ocr models for text recognition in images, Mar 2024. Available at https://blog.roboflow.com/best-ocr-models-text-recognition/, Last accessed on 2024/6/1.

[39] United Nations. THE 17 GOALS. Available at https://sdgs.un.org/goals. Last accessed on 2024/7/24.

[40] U=RIsolve. "u=risolve simulator.", Aug. 2022. Available at https://urisolve.pt/app/. Last accessed on 2024/5/14,.

[41] Thomas Wood. Softmax function, dec. 2016. Available at https://deepai.org/machine-learning-glossary-and-terms/softmax-layer, Last accessed on 2023/12/28.

[42] Thomas Wood. Softmax function, jan. 2023. Available at https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html, Last accessed on 2023/12/28.

Appendix A

Images used to train the segmentation model

This appendix presents an annotated image from the segmentation dataset, alongside several examples of the same image that have been subjected to various augmentation techniques.

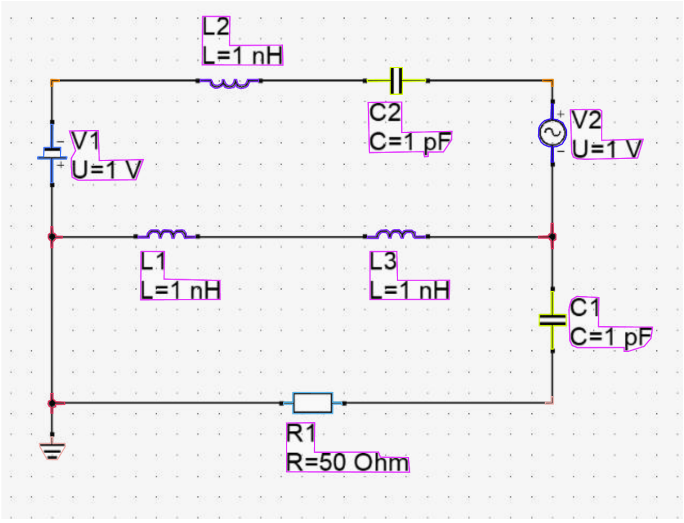


Figure A.1: Original image.

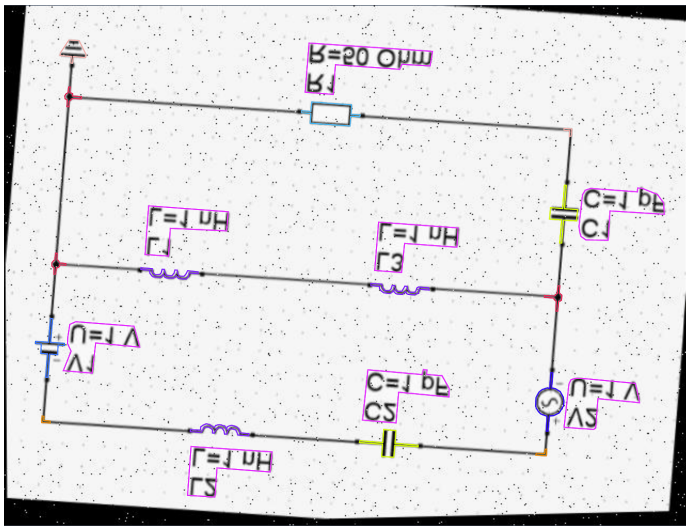


Figure A.2: Augmentation example 1.

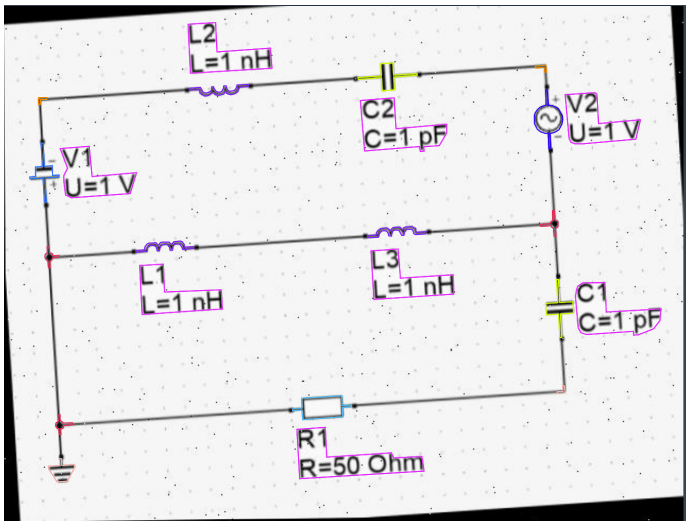


Figure A.3: Augmentation example 2.

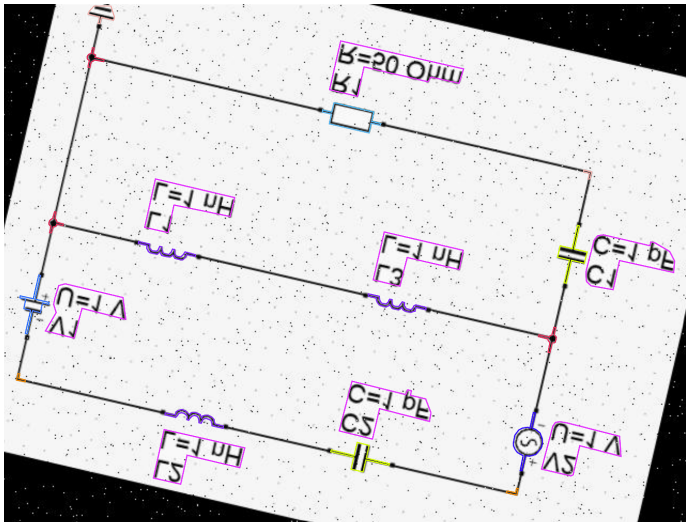


Figure A.4: Augmentation example 3.

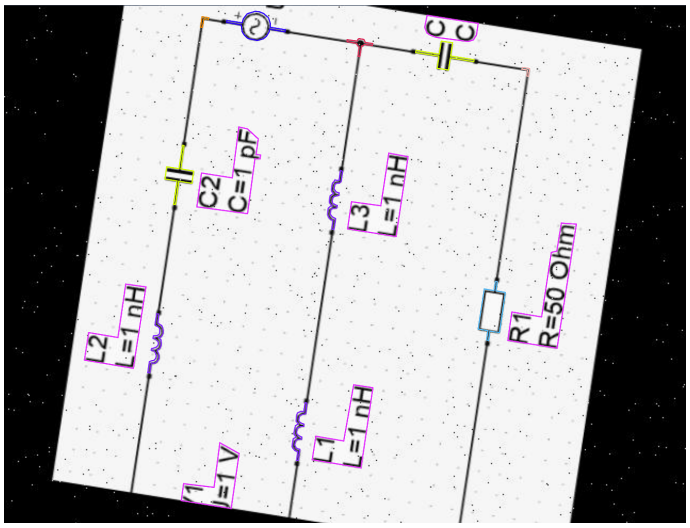


Figure A.5: Augmentation example 4.

Appendix B

JSON code for case study 1

This appendix outlines the JSON model generated from the analysis of Case Study 1 represented in section 4.1.


```
{
    {
    "components": [
        {
            "id": "r6bb73bf",
            "type": "R",
            "active": 1,
            "name": {
                "value": "R1",
                "position": {
                    "x": 10,
                    "y": 20
                }
            },
            "value": {
                "value": "2500",
                "unit": "Ohm",
                "visible": 1
            },
            "position": {
                "x": 64,
                "y": 33,
                "angle": 1,
                "mirrorx": null
            },
            "symbol": null,
            "frequency": {
                "value": null,
                "unit": null,
                "visible": null
            },
```

```
            "phase": null,
            "properties": {
                "impedance": {
                    "value": null,
                    "unit": null,
                    "visible": null
                },
                "temperature": {
                    "value": "26.85",
                    "visible": "0"
                },
                "tc1": {
                    "value": "0",
                    "visible": "0"
                },
                "tc2": {
                    "value": "0",
                    "visible": "0"
                },
                "tnom": {
                    "value": "26.85",
                    "visible": "0"
                },
                "initialValue": null,
                "damping": null
            },
            "port": [
                {
                    "position": {
                        "x": 64,
                        "y": 30
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w6",
                            "point": "end"
                        }
                    ]
                },
                {
                    "position": {
                        "x": 64,
                        "y": 36
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w3",
```

```
                            "point": "begin"
                        }
                    ]
                }
            ]
        },
        {
            "id": "re7a7776",
            "type": "Vdc",
            "active": 1,
            "name": {
                "value": "V1",
                "position": {
                    "x": 10,
                    "y": 20
                }
            },
            "value": {
                "value": "1",
                "unit": "V",
                "visible": 1
            },
            "position": {
                "x": 17,
                "y": 33,
                "angle": 1,
                "mirrorx": null
            },
            "symbol": null,
            "frequency": {
                "value": null,
                "unit": null,
                "visible": null
            },
            "phase": null,
            "properties": {
                "impedance": {
                    "value": null,
                    "unit": null,
                    "visible": null
                },
                "temperature": {
                    "value": "26.85",
                    "visible": "0"
                },
                "tc1": {
                    "value": "0",
                    "visible": "0"
                },
```

```
                "tc2": {
                    "value": "0",
                    "visible": "0"
                },
                "tnom": {
                    "value": "26.85",
                    "visible": "0"
                },
                "initialValue": null,
                "damping": null
            },
            "port": [
                {
                    "position": {
                        "x": 17,
                        "y": 30
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w7",
                            "point": "end"
                        }
                    ]
                },
                {
                    "position": {
                        "x": 17,
                        "y": 36
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w4",
                            "point": "begin"
                        }
                    ]
                }
            ]
        },
        {
            "id": "r830fe75",
            "type": "GND",
            "active": 1,
            "name": {
                "value": null,
                "position": {
                    "x": 10,
                    "y": 20
```

```
                }
            },
            "value": {
                "value": null,
                "unit": null,
                "visible": 1
            },
            "position": {
                "x": 64,
                "y": 63,
                "angle": 0,
                "mirrorx": null
            },
            "symbol": null,
            "frequency": {
                "value": null,
                "unit": null,
                "visible": null
            },
            "phase": null,
            "properties": {
                "impedance": {
                    "value": null,
                    "unit": null,
                    "visible": null
                },
                "temperature": {
                    "value": null,
                    "visible": null
                },
                "tc1": {
                    "value": null,
                    "visible": null
                },
                "tc2": {
                    "value": null,
                    "visible": null
                },
                "tnom": {
                    "value": null,
                    "visible": null
                },
                "initialValue": null,
                "damping": null
            },
            "port": [
                {
                    "position": {
                        "x": 64,
```

```
                        "y": 63
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w0",
                            "point": "end"
                        }
                    ]
                }
            ]
        },
        {
            "id": "r3138d02",
            "type": "R",
            "active": 1,
            "name": {
                "value": "R2",
                "position": {
                    "x": 10,
                    "y": 20
                }
            },
            "value": {
                "value": "5000",
                "unit": "Ohm",
                "visible": 1
            },
            "position": {
                "x": 38,
                "y": 14,
                "angle": 0,
                "mirrorx": null
            },
            "symbol": null,
            "frequency": {
                "value": null,
                "unit": null,
                "visible": null
            },
            "phase": null,
            "properties": {
                "impedance": {
                    "value": null,
                    "unit": null,
                    "visible": null
                },
                "temperature": {
                    "value": "26.85",
```

```
                    "visible": "0"
                },
                "tc1": {
                    "value": "0",
                    "visible": "0"
                },
                "tc2": {
                    "value": "0",
                    "visible": "0"
                },
                "tnom": {
                    "value": "26.85",
                    "visible": "0"
                },
                "initialValue": null,
                "damping": null
            },
            "port": [
                {
                    "position": {
                        "x": 35,
                        "y": 14
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w11",
                            "point": "end"
                        }
                    ]
                },
                {
                    "position": {
                        "x": 41,
                        "y": 14
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w10",
                            "point": "begin"
                        }
                    ]
                }
            ]
        },
        {
            "id": "r2ad2808",
            "type": "C",
```

```
            "active": 1,
            "name": {
                "value": "C1",
                "position": {
                    "x": 10,
                    "y": 20
                }
            },
            "value": {
                "value": "7",
                "unit": "CpF",
                "visible": 1
            },
            "position": {
                "x": 103,
                "y": 31,
                "angle": 1,
                "mirrorx": null
            },
            "symbol": null,
            "frequency": {
                "value": null,
                "unit": null,
                "visible": null
            },
            "phase": null,
            "properties": {
                "impedance": {
                    "value": null,
                    "unit": null,
                    "visible": null
                },
                "temperature": {
                    "value": "26.85",
                    "visible": "0"
                },
                "tc1": {
                    "value": "0",
                    "visible": "0"
                },
                "tc2": {
                    "value": "0",
                    "visible": "0"
                },
                "tnom": {
                    "value": "26.85",
                    "visible": "0"
                },
                "initialValue": null,
```

```
                "damping": null
            },
            "port": [
                {
                    "position": {
                        "x": 103,
                        "y": 28
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w8",
                            "point": "end"
                        }
                    ]
                },
                {
                    "position": {
                        "x": 103,
                        "y": 34
                    },
                    "net": null,
                    "connections": [
                        {
                            "wire": "w5",
                            "point": "begin"
                        }
                    ]
                }
            ]
        }
    ],
    "connections": [
        {
            "id": "ra3221b3",
            "wires": [
                {
                    "id": "w0",
                    "begin": {
                        "x": 64,
                        "y": 52,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 64,
                        "y": 63,
                        "connectedWires": [],
                        "connectedPorts": []
```

```
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w0"
                },
                {
                    "id": "w1",
                    "begin": {
                        "x": 64,
                        "y": 52,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 103,
                        "y": 52,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w1"
                },
                {
                    "id": "w2",
                    "begin": {
                        "x": 17,
                        "y": 52,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 64,
                        "y": 52,
                        "connectedWires": [],
```

```
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w2"
                },
                {
                    "id": "w3",
                    "begin": {
                        "x": 64,
                        "y": 36,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 64,
                        "y": 52,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w3"
                },
                {
                    "id": "w4",
                    "begin": {
                        "x": 17,
                        "y": 36,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 17,
                        "y": 52,
```

```
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w4"
                },
                {
                    "id": "w5",
                    "begin": {
                        "x": 103,
                        "y": 34,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 103,
                        "y": 52,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w5"
                },
                {
                    "id": "w6",
                    "begin": {
                        "x": 64,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 64,
```

```
                        "y": 30,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w6"
                },
                {
                    "id": "w7",
                    "begin": {
                        "x": 17,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 17,
                        "y": 30,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w7"
                },
                {
                    "id": "w8",
                    "begin": {
                        "x": 103,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
```

```
                        "x": 103,
                        "y": 28,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w8"
                },
                {
                    "id": "w9",
                    "begin": {
                        "x": 64,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 103,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w9"
                },
                {
                    "id": "w10",
                    "begin": {
                        "x": 41,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
```

```
                    "end": {
                        "x": 64,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w10"
                },
                {
                    "id": "w11",
                    "begin": {
                        "x": 17,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "end": {
                        "x": 35,
                        "y": 14,
                        "connectedWires": [],
                        "connectedPorts": []
                    },
                    "label": {
                        "text": null,
                        "position": {
                            "x": 0,
                            "y": 0
                        },
                        "distance": null
                    },
                    "node_set": null,
                    "wire": "w11"
                }
            ],
            "ports": [
                {
                    "position": {
                        "x": 35,
                        "y": 14
                    },
```

```
                    "component": "r3138d02",
                    "port": 1,
                    "connections": []
                },
                {
                    "position": {
                        "x": 17,
                        "y": 30
                    },
                    "component": "re7a7776",
                    "port": 1,
                    "connections": []
                }
            ]
        }
    ],
    "nodes": [
        {
            "position": {
                "x": 64,
                "y": 52
            },
            "id": "rb28108b",
            "name": "C",
            "connections": [
                {
                    "wire": "w0",
                    "point": "begin"
                },
                {
                    "wire": "w1",
                    "point": "begin"
                },
                {
                    "wire": "w2",
                    "point": "end"
                },
                {
                    "wire": "w3",
                    "point": "end"
                }
            ]
        },
        {
            "position": {
                "x": 64,
                "y": 14
            },
            "id": "raebe6fa",
```

```
            "name": "C",
            "connections": [
                {
                    "wire": "w6",
                    "point": "begin"
                },
                {
                    "wire": "w9",
                    "point": "begin"
                },
                {
                    "wire": "w10",
                    "point": "end"
                }
            ]
        }
    ],
    "properties": {
        "grid": {
            "x": 10,
            "y": 10,
            "active": 1
        },
        "view": {
            "x1": 0,
            "y1": 0,
            "x2": 128,
            "y2": 68,
            "xPos": 0,
            "yPos": 0,
            "scale": 1
        }
    }
}
```

Appendix C

Case study with a VISIO circuit schematic

This case study represents a simple computer-designed electrical circuit using VISIO electrical circuit simulator. The image has a width of 959 pixels and a height of 348 pixels and is displayed in Figure C.1.

Theprocessing results can be found at *results/exer_anexo* in the project's software directory.

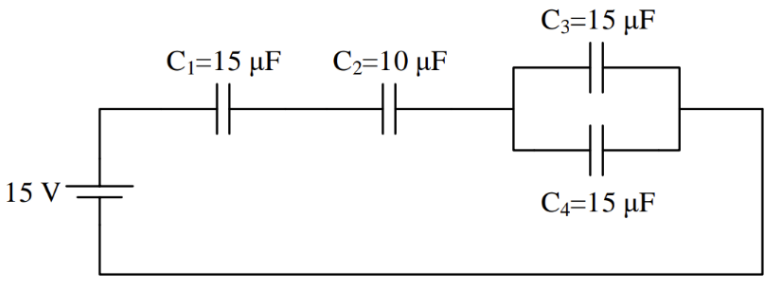


Figure C.1: Original image - Visio circuit schematic.

Figure C.2 displays the segmentation result of the third case study. All the classes classified by this model are visible in the image. Table C.1 displays the accuracy of each element and the average accuracy across all elements. The segmentation model inference time was 114.8 ms.

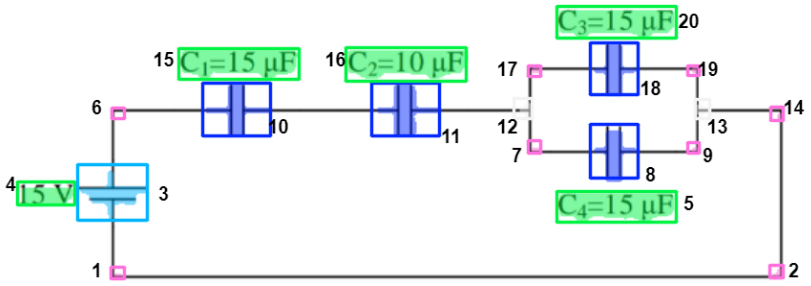


Figure C.2: Segmentation result for case study 1.

Table C.1: Accuracy of each element detected.

Element	Classe Name	Accuracy	Element	Classe Name	Accuracy
1	junction	56.8%	11	capacitor	81.8 %
2	junction	59.1 %	12	cross	78.0 %
3	voltagedc	89.0%	13	cross	73.8%
4	text	77.9 %	14	junction	58.5 %
5	text	89.3 %	15	text	88.0 %
6	junction	51.8 %	16	text	87.6 %
7	junction	80.7 %	17	junction	73.5 %
8	capacitor	83.9%	18	capacitor	82.6 %
9	junction	47.0 %	19	junction	71.7 %
10	capacitor	80.7 %	20	text	88.9%
Average		75.0%			

Figure C.2 represents the polarity the result model for the DC voltage represented in this study case. The average accuracy of this prediction is 83.9

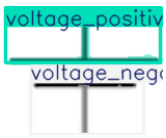


Figure C.3: Polarity result. Blue represents the positive port, and white is associated with the negative port.

Table C.2 shows the results of OCR and the software's interpretation to separate the data into labels, values and units.

In this case study, it was not necessary to use filters on the image after removing the segmentation results because the image is noiseless. The only operation required was to change the image to a binary scale in order to apply cv2.findcountours() functions, as shown in the figure.

Table C.2: OCR results.

Text Image	Predicted text	Accuracy	Label	Value	Unit
4	’15 V’	93,3%	-	15	v
15	’C1-15 pF ’	49.9%	C1	15	pF
16	’C2-10 uF’	65.7%	C2	10	uF
20	’C3-15pF’	39.4%	C3	15	pF
15	’C4-15 pF’	52.1%	C4	15	pF

(a) Copy of the original image without the segmentation results

(b) Result of the contours.

Figure C.4: Electrical connection detection .

The data model for this case study was created correctly, but the data model has some errors defining the electrical connection, and it was possible to integrate it with the U=RIsolve platform, as shown in Figure C.5.

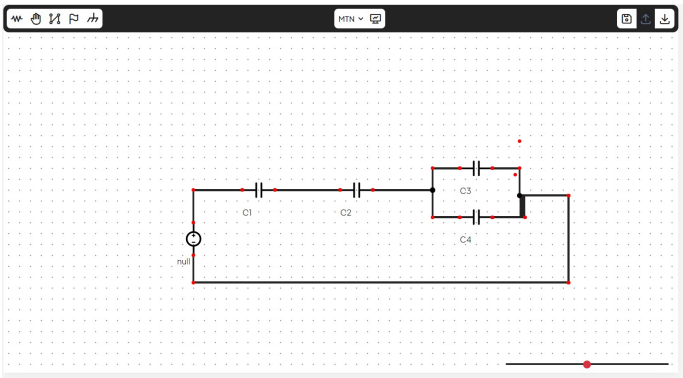


Figure C.5: Integration result with U=RIsolve.