Eberhard Karls Universität Tübingen
Department of Computer Science
Embedded Systems

Master's Thesis Machine Learning

Multimodal Deep Learning for Automated Schematic Analysis

Submitted by: Vojtěch Sýkora

25.7.2025

First Supervisor

Prof. Dr. Oliver Bringmann

Faculty of Science
Department of Computer Science
Embedded Systems
University of Tübingen

Second Supervisor

Prof. Dr.-Ing. Cristóbal Curio

Faculty of Computer Science
Department of Computer Science
Vice Dean of Research
Reutlingen University

Graduate Advisor

M.Sc. Tobias Hald

FZI Forschungszentrum Informatik

Sýkora, Vojtěch:
Multimodal Deep Learning for Automated Schematic Analysis
Master's Thesis Machine Learning
University of Tübingen
Processing period: 26.1.2025 - 25.7.2025

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst habe, dass ich keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, dass die Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist, dass ich die Arbeit weder vollständig noch in wesentlichen Teilen bereits veröffentlicht habe, dass die Inhalte der für die Plagiatsprüfung eingereichten Datei mit den Inhalten des eingereichten gedruckten Berichts übereinstimmen, dass ich keine Inhalte aus Berichten anderer Studierender übernommen habe und ich mir bewusst bin, dass eine Plagiatsprüfung durchgeführt wird. Falls ich KI-basierte Textgeneratoren (z.B. ChatGPT oder GPT-3) für die Erstellung von Textteilen genutzt habe, habe ich sichergestellt, dass diese keine Plagiate enthalten. Mir ist bewusst, dass ich allein für die Qualität der eingereichten Arbeit verantwortlich bin.

Tübingen, 28.7.2025 .....

Ort, Datum Unterschrift

Abstract

Object detection for automating the extraction of electric schematics remains affected by two main issues: severe class imbalance and a lack of good training data, forcing a substantial domain shift between messy handwritten and clean computer-generated schematics. This thesis tackles both. First, a new Raspberry Pi (RPi) dataset of clean high-resolution CAD schematics is created with 22 images of 1675 annotated symbols, aimed at working with a domain shift from the CGHD hand-drawn dataset of 3137 images and 246k annotations. Second, the Faster R-CNN object detector from the Modular Graph Extraction pipeline is enhanced using losses, architectural changes, hyperparameter tuning, and image transformations. The combination of Focal and GIoU losses, followed by tuning, shows a 7.3% mAP increase on CGHD and 3.7% on the RPi dataset, both compared to a strong baseline. Next, a pretrained VLM, Molmo-7B-D, falls short of 32.0% accuracy of the Faster R-CNN with 17.5% accuracy on the same object detection task; however, it shows basic understanding of the task. This work confirms that a carefully modified specialist model beats a pretrained VLM for symbol detection, and releases a new RPi dataset to help with cross-domain adaptation.

Kurzfassung

Die Automatisierte Objekterkennung von elektrischen Schaltplänen ist von zwei Problemen betroffen: zum Einen die Verteilung der Elemente auf verschiedene Klassen weist eine starke Ungleichheit auf, zum Anderen besteht ein Mangel an guten Trainingsdaten, was zu einem ungleichgewicht zwischen unordentlichen handschriftlichen und computergenerierten Schaltplänen führt. Diese Arbeit befasst sich mit beiden Problemen. Zunächst wird ein neuer Raspberry Pi (RPI)-Datensatz mit computergenerierten, hochauflösenden CAD-Schaltplänen erstellt, welcher 22 Bilder mit 1675 annotierten Symbolen umfasst. Dabei tritt eine Verschiebung der Domäne zum handgezeichneten CGHD-Datensatz mit 3137 Bildern und 246.000 Annotationen auf. Zudem wird der Faster R-CNN-Objektdetektor aus der Modular Graph Extraction-Pipeline durch die Wahl von Loss-Functions, architektonische Änderungen, Hyperparameter-Optimierung und Bildtransformationen verbessert. Die Kombination aus Focal- und GIoU-Loss-Function, gefolgt von einer Optimierung, zeigt eine Steigerung der mAP um 7,3% bei CGHD und um 3,7% beim RPi-Datensatz, im Vergleich zur Baseline. Zudem bleibt das vortrainiertes VLM, Molmo-7B-D, mit einer Genauigkeit von 17,5% bei derselben Aufgabe hinter der Genauigkeit von 32,0% des Faster R-CNN zurück. Das VLM zeigt jedoch ein grundlegendes Verständnis der Aufgabe. Diese Arbeit bestätigt, dass ein sorgfältig modifiziertes Spezialmodell ein vortrainiertes VLM bei der Symbolerkennung übertrifft.

Acknowledgements

To begin, I would like to thank my advisor Tobias Hald for his guidance. The knowledge he has about this field combined with his insightful and quick responses were invaluable over the course of this thesis. He has pointed me in the right direction, refined my ideas, and his constructive feedback has significantly improved the quality of this thesis.

Next, I would like to thank Professor Bringmann and Professor Curio for supervising me on this topic. I find this topic very interesting and have learned a lot about the field of computer vision.

I am grateful to Johannes Bayer for the foundational code on which this thesis builds, for clarifying its inner workings, and for kindly providing his pretrained baselines for comparison.

I would like to thank Nam Nguyen The for his insightful advice and technical guidance.

Finally, I would like to thank my family, friends, and partner for their support and encouragement throughout my academic journey.

Table of Contents

Chapter 1
Introduction

Circuit schematics remain the main language of electrical engineering, yet most exist only on paper or in PDFs, limiting the automated analysis and search within their elements. Previous work *Modular Graph Extraction* (Section 2.2) showed that a sequential multi-modal vision pipeline trained on the *Circuit Graph Hand-Drawn Dataset (CGHD)* can recover the graph-based structure from handwritten photographed diagrams with a lot of artefacts. However, it suffers from class imbalance and poor cross-domain generalisation to clean computer-generated schematics.

CGHD offers 3137 images, 246k object detection annotations for 59 different symbol classes. It has a large class imbalance combined with numerous real-world artefacts and mixed symbol standards (IEEE/ANSI, IEC/DIN) described in Section 2.1. To evaluate domain shift, we created a new dataset from official Raspberry Pi datasheets (Section 4.1), which includes 22 high-resolution images with 1675 labelled objects mapped to the original 59 classes. We used the whole dataset for evaluation.

In this work, we target the object detection stage in the pipeline using the Faster R-CNN model (Section 2.3). We switch to Focal loss as classification loss and Generalized Intersection over Union (GIoU) loss as box regression loss (Section 4.2), addressing the mentioned symbol class imbalance and bounding box location sensitivity. Following, we explore architectural changes in the form of Efficient Channel Attention (ECA) and dilated convolutions (Section 4.2). Finally, we add nine image augmentations and execute hyperparameter tuning (Section 4.2).

During training, the combined Focal and GIoU addition improved the accuracy by 10% over the baseline, beating other enhancements and their combinations (Subsection 5.2.1). During inference, it increases on unseen CGHD drafters by 7.3% and on the Raspberry Pi dataset by 3.7%, confirming knowledge was transferred (Subsection 5.2.3).

Finally, we examine whether a state-of-the-art Vision Language Model (Molmo-7B-D, Subsection 2.7.1) can perform the same object detection task in zero- or few-shot modes. Two metrics were used, which focused on detected symbol classes and assigned locations to found symbols. Even though the model shows some understanding of the smaller diagrams, it struggles to grasp the task for more complex diagrams. The results fall behind specialised object detection even after rigorous prompt engineering (Section 5.3).

Chapter 2

Fundamentals

Section 2.1

Handwritten Circuit Diagram Dataset (CGHD)

The CGHD is a large-scale and openly licensed dataset containing images of hand-drawn electrical circuit diagrams and accompanying annotation and segmentation ground-truth files. It is intended to train models for extracting electrical graphs from raster graphics. Originally introduced in [1], but the number of images and the annotation quality have been continuously expanded. For this work, we used version 14 of the dataset[1].

Table 2.1 provides a comprehensive overview of the dataset's content and statistics for version 14 of the CGHD dataset. Moreover, the maximum number of bboxes found in a single image was 542, showing how complex some diagrams truly are.

Furthermore, in Figure 2.1.1, we can see a major flaw of this dataset: the uneven class distribution. The plot shows the number of bounding box (bbox) labels of each class in the whole dataset. The minimum is 16 and the maximum is 90,391; hence, the plot had to employ the logarithmic scale. We will look at a possible solution to this flaw in Subsection 2.4.1.

[1]CGHD Version 14, released 8 November 2024, DOI: `https://zenodo.org/records/14042961`

Table 2.1.: CGHD Dataset (version 14) Statistics and Content Overview

Category	Count
Annotated Raw Images	3,173
Bounding Box Annotations	245,962
Object Classes	59
Rotation Annotations	39,955
Mirror Annotations	1,339
Text String Annotations	84,431
Text String Completeness	93.41%
Text Characters	286,467
Character Types	98
Binary Segmentation Maps	284
Polygon Annotations	21,186

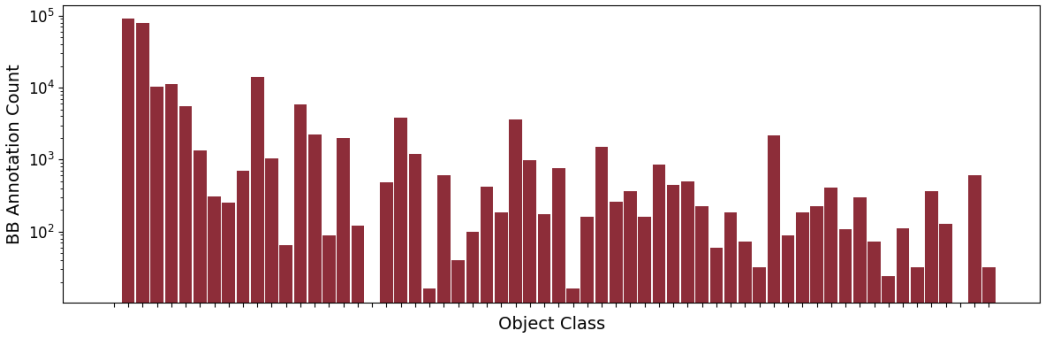


Figure 2.1.1.: Distribution of bbox annotations across object classes in the CGHD dataset

Subsection 2.1.1

Imperfections and Variations in Images

The CGHD dataset contains a vast variety of imperfections and differences, which make it very challenging for machine learning models. These imperfections try to make the models robust against any real-world scenarios that could be encountered when capturing hand-drawn diagrams. Let us explore 2 example figures from the dataset while explaining the possible artefacts in this data.

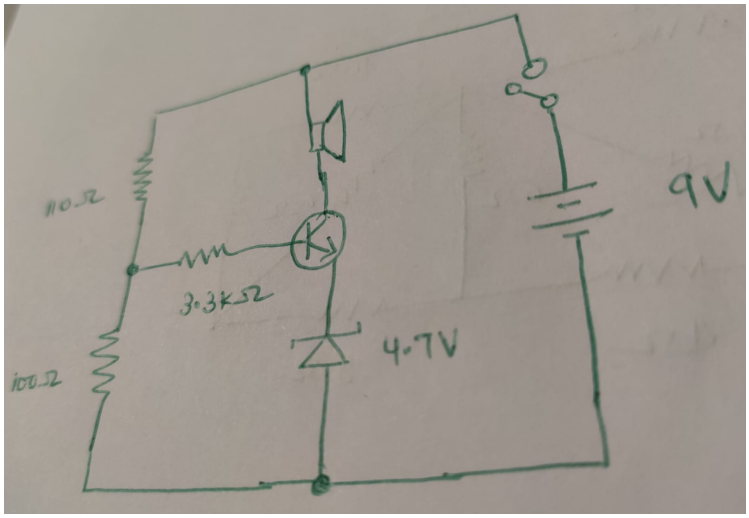


Figure 2.1.2.: Sample C32_D2_P3 from drafter 3.

As seen in Figure 2.1.2, the image includes a marker with a non-standard green colour. We can also see a blur on the left side of the image, as well as a tilt of the view. The image is on a white paper; however, there is a slightly seethrough diagram from the other side. The overall view of the circuit diagram is very close-up, and the symbols are relatively large because of this. The diagram includes only simple symbols.

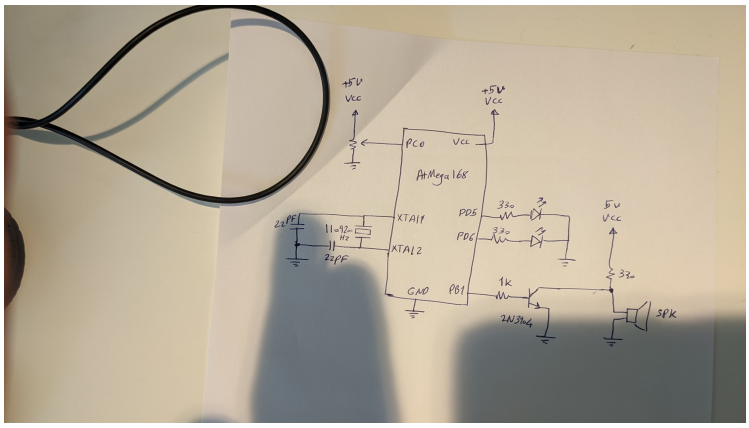


Figure 2.1.3.: Sample C184_D2_P2 from drafter 16.

As seen in Figure 2.1.3, there can be shadows, bends, or even objects over the paper. The circuit diagram is photographed from further away, making all symbols smaller. There can be more complicated symbols, such as an integrated circuit.

These were only two examples from the whole dataset, which includes quite a large number of possible variations and imperfections, which are listed in the original

paper [1]. They include variations in drawing instruments, drawing surfaces, visual artefacts, physical distortions, capturing conditions, and background context.

Subsection 2.1.2

Challenging Symbol Standardization

Circuit diagrams are symbolic, graph-based descriptions of electric and electronic structures. There are various standards for symbol representation, and this CGHD dataset includes both IEEE/ANSI [2] and IEC/DIN [3] standards. The images were allowed to contain both variants, a mix of them, and interestingly, non-standard-conforming symbols. This lack of firm standardisation is a usual occurrence even outside of this dataset. It is one of the reasons why symbol recognition in this field is quite tricky. For transfer learning, this might also be an issue, since the pre-trained model might have been trained on a different standard of the same symbol. Therefore, we will need to check that our training set and testing sets have the same standardisation.

Section 2.2

Modular Graph Extraction

The modular graph extraction approach represents a multi-modal pipeline for extracting structured semantic graph representations of electric circuit diagrams from images of pen-on-paper sketches. This system combines computer vision techniques with semantic reasoning to achieve comprehensive circuit understanding with acceptable accuracy [4]. This system was trained on the CGHD dataset explored in Section 2.1.

In the original work [4], the author focuses on the educational domain; however, in this thesis, we build upon his work by enhancing the pipeline's performance and adapting it to a domain regarding computer-generated schematics of Raspberry Pi computers, which are substantially more commonly used in the modern world.

Subsection 2.2.1

Pipeline Architecture

The extraction pipeline, based on [4], follows a modular design consisting of three primary components: **Dataloader**, **Processor**, and **Model**. These components

are configured through JSON configuration files that define the specific algorithms, parameters, and processing strategies for different extraction tasks.

The core pipeline includes the following models in the respective order (a visualisation of the pipeline execution can be seen in Figure 2.2.1):

Step 1: Object Detection

The first step uses a Faster R-CNN model [5] with a ResNet-152 backbone, yielding region of interest bbox positions and classes for the symbols and text. The system is trained to detect 59 different classes, such as diodes, capacitors, or resistors, but also larger specialised components such as integrated circuits. The configuration files also include helper classes such as `__background__` and `unknown`. The object detection processor handles pre-processing of input images into tensor format and post-processing of predictions into bbox annotations with confidence scores.

Step 2: Orientation Recognition

The second step uses a CNN-MLP model for orientation recognition, where the input includes not only the symbol snippet but also the respective symbol template. The output of this model is the encoded smallest possible angle between the template and the snippet. Some symbols are indeed mirror-symmetric, such as resistors, which are in this approach solved by reducing the angle's range down to a maximum of 180 degrees from the original 1 to 360 degrees. The current version includes 32 supported classes with known templates, with 8 of them being symmetric. This step is crucial for components like diodes or transistors, where orientation affects their function in the circuit.

Step 3: Text Recognition

The third step employs a CNN-LSTM hybrid architecture for transcribing text found in the images. The model combines a Convolutional Neural Network (CNN) for feature extraction with a bidirectional Long Short-Term Memory (LSTM) [6] for text prediction. The model can predict variable text lengths, which is a crucial advantage when working with both short texts, such as the voltage "9V", or a long text explaining an unusual component in the diagram. The text recognition model is capable of handling special characters common in circuit diagrams, such as the ohm (Ω) or micro (μ) symbols for measurements.

Step 4: Shape Recognition

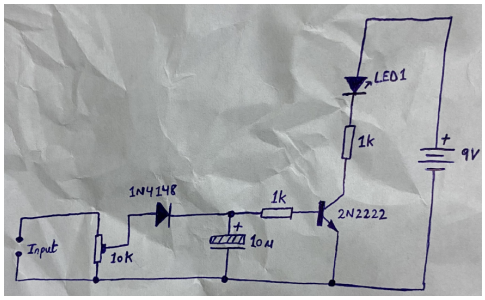
The fourth step uses instance segmentation with U-Net [7] to refine the boundaries of symbols. The instance segmentation processor extracts 128 by 128 pixel patches around each component, using both the original polygon (from the raw image) and the segmentation map as input (both included in the CGHD dataset). In the end, transforming the local coordinates to global ones for saving the refined polygons of symbol boundaries.

Step 5: Connection Recognition

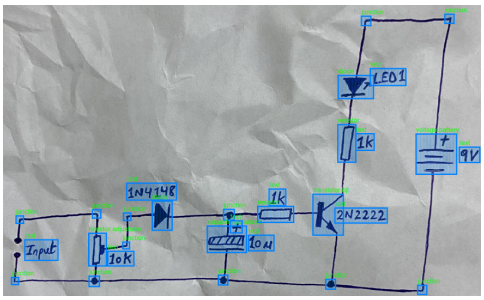
The fifth step performs semantic segmentation to identify the connections (wires) between components. It begins by creating a binary segmentation map again using U-Net [7], followed by a removal of object regions. Since these object regions also include structural elements such as line corners and crossings, the remaining image is expected to contain only line segments. These line segments are then detected as blobs and an edge is created for each blob that touches two objects.

Final Step: Graph Construction

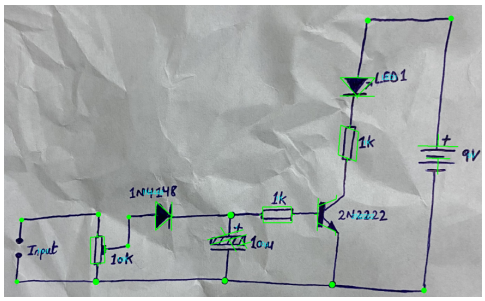
The extracted components and connections are integrated into a unified graph structure, which extends NetworkX's [8] directed graph implementation. This graph representation maintains spatial relationships, component properties, and connectivity information essential for circuit analysis. Afterwards, the post-processing takes care of wire hops, rectification of uneven edge lines, and ports are corrected in position with their respective names. This graph structure can be saved in various file types and evaluated on test samples.



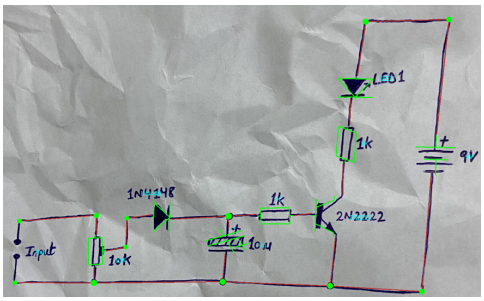
(a) Raw input image



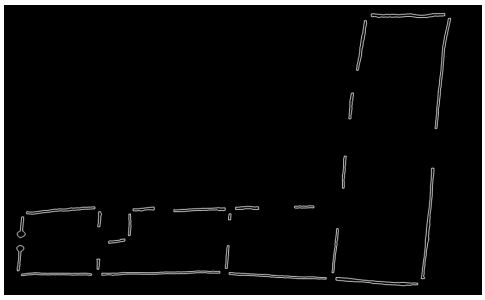
(b) Object detection



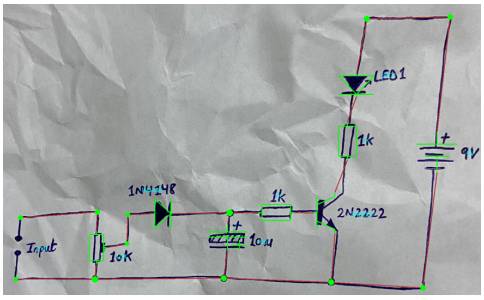
(c) Orientation and text recognition



(d) Edge extraction



(e) Edge line segments



(f) Final graph rectification

Figure 2.2.1.: Modular graph extraction pipeline visualisation showing the progression from raw input image through various processing steps to final graph rectification. *All images taken from original paper* [4].

Section 2.3

Faster R-CNN

Faster R-CNN [5] represents a well-respected baseline in object detection, serving as part of the modular extraction pipeline described in section 2.2. This section will provide an overview of this model and its significance for our task.

Faster R-CNN is built on a Region-based Convolutional Neural Network (R-CNN) [9], which evolved from R-CNN to Fast R-CNN [10] to Faster R-CNN, improving

computational efficiency along the way by introducing Region of Interest (RoI) pooling layer and further by adding a Region Proposal Network (RPN), making it the first end-to-end trainable object detection framework. This inclusion helps with the precise localisation of symbols in our electric circuit diagrams.

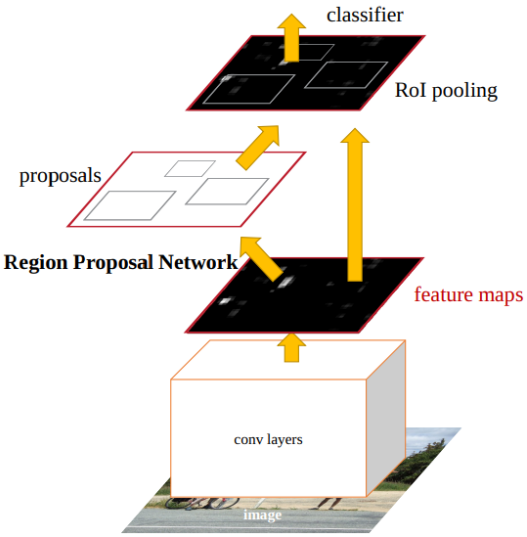


Figure 2.3.1.: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the "attention" of this unified network [5].

The Faster R-CNN architecture seen in Figure 2.3.1 is a combination of the RPN, which proposes RoIs, and a Fast R-CNN detector that uses the proposed regions. Both of these parts share convolutional features coming from one backbone, reducing computational overhead while keeping accuracy. In our case, it is a ResNet backbone [11] with a Feature Pyramid Network (FPN) [12], specifically pretrained `resnet152` from torchvision [13]. The convolutional layers were pretrained on the ImageNet-1K classification challenge [14] with over 1.2 million images with 1000 different classes. The FPN layers are initialised from scratch and trained downstream on our detector. Nonetheless, not including electric schematics. This large backbone enables obtaining a wide range of features from narrow wires to large symbols.

The RPN works by sliding a window over the backbone feature maps and generates region proposals through anchor boxes based on multiple scales or aspect ratios, as seen in Figure 2.3.2. Such an approach is crucial for this domain since there are cases, such as the grounding symbol being tiny compared to the integrated circuit symbol.

Moreover, the detection head combines the predicted regions and features from the backbone to provide us with classification and bbox locations. The RoI pooling is possible in batches since we extract same sized tensors after max pooling from

the predicted anchors which enables us to process hundreds of RoIs through the detection head.

The Faster R-CNN loss includes a classification loss and a bbox regression loss for both the RPN and the detection head. We will focus on working with both losses in later chapters.

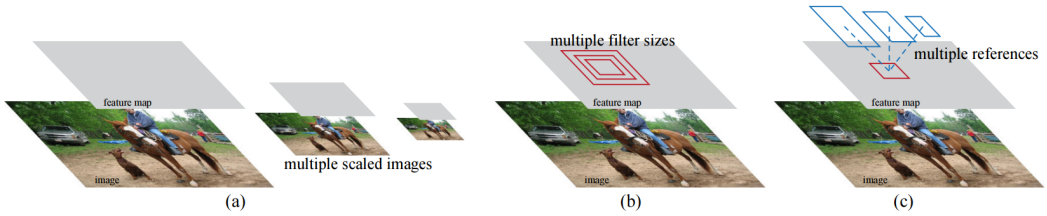


Figure 2.3.2.: Different schemes for addressing multiple scales and sizes. (a) Pyramids of images and feature maps are built, and the classifier is run at all scales. (b) Pyramids of filters with multiple scales/sizes are run on the feature map. (c) We use pyramids of reference boxes in the regression functions [5].

Section 2.4

Loss Functions

Subsection 2.4.1

Focal Loss

Class imbalance commonly occurs in image datasets because of an imbalance between background and foreground objects. Focal Loss [15] is a classification loss that tries to mitigate this issue.

The focal loss reduces the loss contribution of the well-classified classes while increasing the loss contribution of the often misclassified classes. In practice we use this α-balanced variant of the focal loss [15]:

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{2.1}$$

where p_t represents the estimated probability for the true class, α_t is a weighting factor for class t, and γ is the focusing parameter that controls the rate at which the easy examples are down-weighted.

The main point of focal loss is in $(1-p_t)^{\gamma}$. Where if an example is misclassified, p_t is small, and the modulating factor is almost 1, then the loss is unaffected. As p_t

approaches 1, showing a well-classified example, the modulating factor approaches 0, and the loss is down-weighted.

Subsection 2.4.2

Generalised Intersection over Union Loss

The default Faster R-CNN uses the Smooth L1 loss [16] for bounding box regression, which treats all location errors equally. For object detection, a variant of Intersection over Union (IoU) [17] is more suited. IoU takes the area of the intersection of the two compared bboxes and divides it by their combined area. Generalised Intersection over Union (GIoU) [17] takes this one step further by considering the smallest box that encloses both bboxes.

GIoU is defined as:

$$\text{GIoU} = \text{IoU} - \frac{|E \setminus (P \cup G)|}{|E|} \tag{2.2}$$

where P and G are the predicted and ground truth bboxes respectively, E is the smallest enclosing box that contains both P and G, and $|E \setminus (P \cup G)|$ represents the area of E that is not covered by the union of P and G.

The GIoU loss is then computed as:

$$\text{L}_{\text{GIoU}} = 1 - \text{GIoU} \tag{2.3}$$

Therefore, by using GIoU, we directly optimise the IoU metric while being scale-invariant and giving non-zero gradients even when bboxes do not overlap. Since GIoU $\in [-1, 1]$, the GIoU loss $\in [0, 2]$, where 0 is a perfect overlap and 2 shows maximum dissimilarity.

Section 2.5

Architectural Components

Subsection 2.5.1

Efficient Channel Attention (ECA)

ECA [18] is a lightweight method for achieving higher performance while not increasing model complexity. It proposes a local cross-channel interaction strategy

without dimensionality reduction, which can be efficiently implemented using global average pooling followed by a 1D convolution:

$$\mathbf{z} = \mathrm{GAP}(\mathbf{X}) \in \mathbb{R}^C, \tag{2.4}$$

$$\mathbf{s} = \sigma\big(\mathrm{Conv1D}(\mathbf{z};\, k)\big) \in \mathbb{R}^C, \tag{2.5}$$

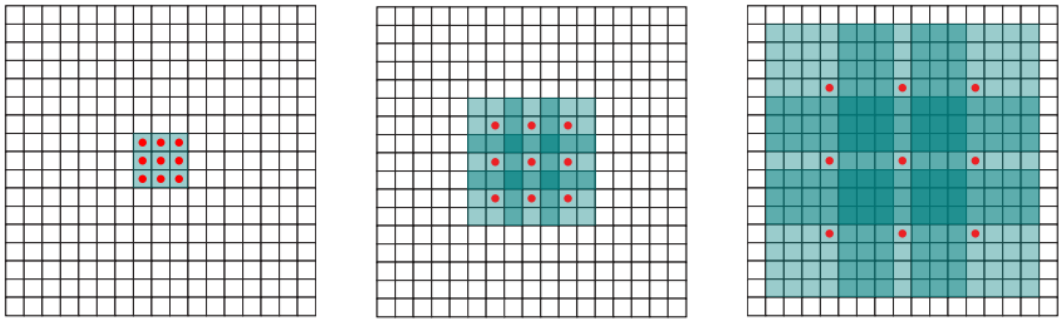
where $\mathbf{X} \in \mathbb{R}^{C\times H\times W}$ is the input feature map, GAP denotes global average pooling, $\mathrm{Conv1D}(\cdot; k)$ is a 1D convolution with kernel size k (an odd integer determined adaptively, e.g. $k = |C/\gamma|_{\mathrm{odd}} + b$), and σ is the sigmoid activation. The output is then channel-wise multiplied:

$$\hat{\mathbf{X}}_c = \mathbf{s}_c\, \mathbf{X}_c, \quad c = 1, \ldots, C. \tag{2.6}$$

Subsection 2.5.2

Dilated Convolutions

Dilated convolutions [19] help expand the area of the input image covered. They introduce a wider field of view without additional computational cost by spreading the pixels from the convolution instead of having them in a solid square.



(a) **(b)** **(c)**

Figure 2.5.1.: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly. [19]

Given an input feature map $\mathbf{X} \in \mathbb{R}^{C \times H \times W}$ and a kernel $\mathbf{W} \in \mathbb{R}^{C' \times C \times K \times K}$, the dilated convolution at output position (i, j) is:

$$y_{c'}(i,j) \;=\; \sum_{c=1}^{C} \sum_{u=1}^{K} \sum_{v=1}^{K} W_{c',c,u,v}\; X_c(i + r\,(u - \lceil K/2 \rceil),\; j + r\,(v - \lceil K/2 \rceil))\,, \qquad \textbf{2.7}$$

where r is the dilation rate, K is the kernel size, and $\lceil \cdot \rceil$ is the ceiling operation.

Section 2.6

Image Transformations

A staple in making object detection models more robust and generalised are image transformations. These represent geometric or photometric changes to the image and have repeatedly been shown to work [20]. Our nine image transformations shown in Figure 2.6.1 are the following:

Grayscale. Removes the colour information from the image while preserving luminance, leaving it only in shades of grey. This is useful for reducing the model's dependency on colour and increasing its focus on shapes.

Rotation. Rotates an image by its centre point by an angle, filling in the new pixels with black colour. This helps the model when the data can come with different orientations.

Scaling. Performs a geometric change of the scale of the image while preserving the ratios of distances and orientations of lines. This is useful for tasks where the same object is stretched to different sizes in different images.

Perspective. Simulates 3D perspective effects. Therefore, it can simulate images that were taken from different viewpoints, as real-life data usually is.

Colour Jitter. Randomly changes the values of contrast, brightness, saturation and hue. This helps with model robustness towards variations in these settings.

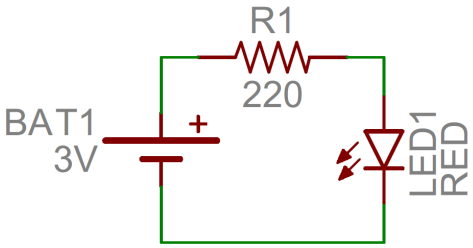
Auto Contrast. Focuses only on adjusting contrast. It stretches the histogram to use the full range of the available intensities. This is used for robustness against different lighting and temperature.

Gaussian Blur. Smooths the image using a Gaussian kernel which reduces details and noise while keeping the structural features. This helps the model focus on understanding the general structure rather than guiding decision-making based on a few details.

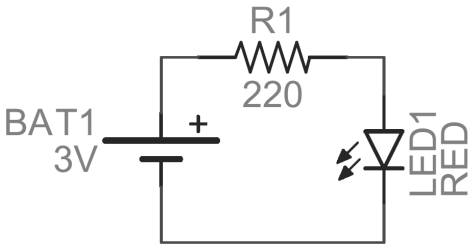
Noise. Introduces random pixel-level changes which can represent sensor noise in real life or perhaps compression artefacts. This again helps with robustness against real-world images.

Erasing. Randomly removes a small part of the image, forcing the model to understand all parts of the objects instead of relying on one standout feature.

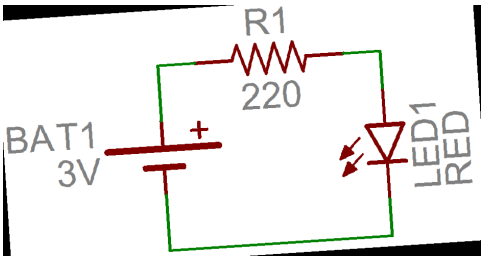
These transformations help increase generalisation and performance of computer vision models.



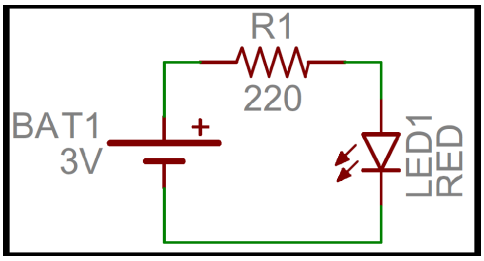
(a) Original Image



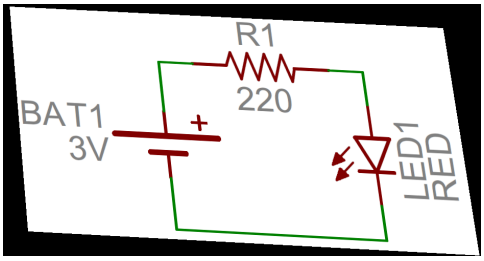
(b) Grayscale



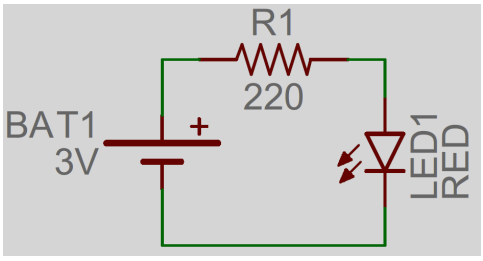
(c) Rotation



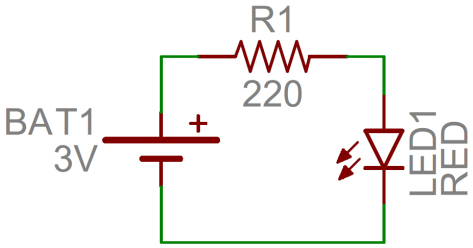
(d) Scaling



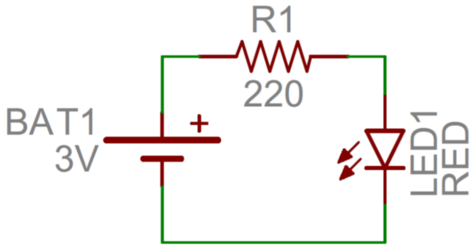
(e) Perspective



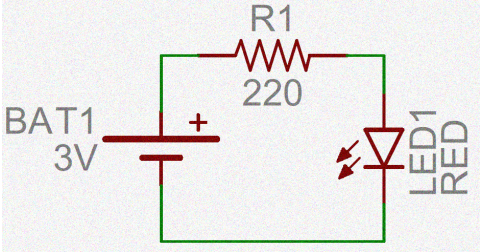
(f) Colour Jitter



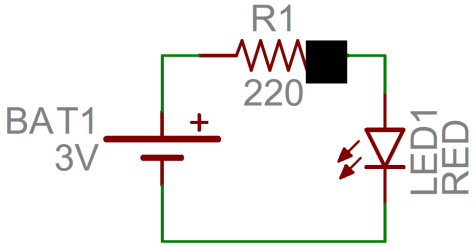
(g) Auto Contrast



(h) Gaussian Blur



(i) Noise Addition



(j) Random Erasing

Figure 2.6.1.: Visualisation of various image transformations applied to a circuit diagram for data augmentation during training.

Section 2.7

Vision Language Models (VLMs)

We are in the year 2025, and a lot of new information in the field of Artificial Intelligence (AI) is focused on Large Language Models (LLMs) and Multimodal Large Language Models (MLLMs), which are possible because of the Transformer architecture [21]. These have the advantage of size and general understanding of the whole input without needing to split it into different tasks such as is with the Modular Graph Extraction pipeline introduced in Section 2.2. These systems also have good out of the box capabilities which is why we wanted to experiment with using one out-of-the-box on the same object detection task as the Faster R-CNN model and compare them. For this to work, we need a subset of MLLMs with visual capabilities, specifically called a Vision Language Model (VLM).

VLMs integrate an image encoder with an LLM to allow joint reasoning over image pixels and text. The image encoder is typically a convolutional or vision transformer backbone. They project the tokens from the image into the LLM's embedding space. Even though these models might be good at general question answering or captioning tasks, they usually struggle with complex images, which is what we evaluate. The diagrams range from quite simple and small to massive assemblies of wires and symbols.

Subsection 2.7.1

Molmo-7B-D

Molmo-7B-D [22] comes from the Molmo family of open VLMs released by the Allen Institute for AI. It is a multimodal AI model that has vision and language abilities. In academic benchmarks, Molmo-7B-D positions itself on a very similar level to state-of-the-art models such as GPT-4o [23], Gemini 1.5 Pro [24], or Claude 3.5 Sonnet [25], even outperforming the latter. Therefore, it is a great open-source candidate for our experiments.

It combines a frozen vision backbone in the form of OpenAI's CLIP ViT-L/14 [26], which is followed by a decoder-only LLM built on Gwen2-7B architecture [27]. The model was trained on roughly 1 million image-text pairs from the PixMo dataset [22]. The PixMo dataset consists of 6 specialised subsets, one of which is used for pre-training, while the other 5 are for supervised fine-tuning. One subset used for fine-tuning is called PixMo-Docs, which contains diagrams, charts, tables, and other technical images. Even though these give Molmo a general understanding of

schematics, it does not specifically focus on electric circuit diagrams; therefore, its knowledge is nowhere near a fine-tuned model for such data.

Subsection 2.7.2

Few-shot Learning

Few-shot learning, sometimes called in-context learning, uses the ability of large transformers to learn by providing example prompts and answers directly in the prompt rather than through gradient descent updates. Papers such as [28] show that adding just a few of these examples during inference time can be equivalent to finetuning hundreds to thousands of annotated samples while massively reducing the time and computational cost.

For LLMs, this might be as simple as adding both the text of the example prompt and the ideal answer in the system prompt of the LLM. For VLMs, few-shot prompting requires the addition of example images in the prompt. For few-shot prompting in VLMs, the image needs to be converted into a text format, such as base64 and then included it in the prompt. However, such text might be too long for the context window. We solve this by building up the conversation history in a format which the model understands, and we pass all images as their path. This does not take up the token limit, and it ensures that the VLM has access to all needed images.

Chapter 3
Related Work

There are various datasets of electric circuit diagrams publicly available; however, most include images in the tens or hundreds, which is not enough for a robust generalised system. That is why for this work we decided to work with the CGHD dataset [1] introduced in Section 2.1. Even though it is using handwritten circuit diagrams, while our test set focuses on computer-generated circuit diagrams, the benefits of the dataset size and features might outweigh the negatives for such a domain shift.

Nonetheless, there are some promising datasets in the works, such as AMSNet: Netlist dataset for AMS Circuits [29], which focuses on a dataset for automated conversion of analogue/mixed-signal integrated circuit schematics to SPICE format netlists. The dataset is, as of writing, not yet publicly available.

This paper [30] focused on segmenting and recognising all two-terminal components and wires in images. Unfortunately, their evaluation dataset consisted only of 15 real-world schematics, and their pipeline was limited to 30 distinct symbols, while the modular graph extraction worked with 59. Moreover, their pipeline does not understand the directionality of symbols and does not understand the text labels of components. Overall, this work did tremendous work in segmentation and symbol recognition; however, it lacks in the other steps of the needed pipeline.

What all of these mentioned works have in common is the focus on one domain, one standard, one type of electric circuit diagrams, while our work aims to contribute to a larger goal of a generalised system for circuit diagram extraction that can understand all needed aspects of such diagrams, such as symbol orientation.

Furthermore, in the realm of VLMs, there are some works worth mentioning. Chain-of-Reason paper [31] uses rule-based detection using OpenCV, followed by semantic aggregation, whose output is then given to a VLM for reasoning. This does not use any deep learning techniques for object detection, and therefore it just focuses on recognising general shapes, which are then input into the VLM.

The author, however, uses it for scientific diagram question answering, which is a recurring theme with most other papers on this topic. In our thesis, we focus on extracting the symbols and graph structure from the diagram instead of purely answering a high school physics exam question. For such question answering tasks, modern VLMs have average performance as tested in ElectroVizQA [32].

Nonetheless, there are papers focusing more on our task. Wan et al. [33] propose a two-stage pipeline that first encodes electric circuit images with a CLIP vision encoder [26] and then decodes the fused visual and text embeddings using an LLaMA-based language decoder [34] to produce both symbol annotations and full-sentence captions. However, they use a proprietary dataset and fine-tune their transformers, which is also the approach of another paper, which finetunes GPT-2 XL [35] variants to output SPICE-style netlists [36]. Our approach does not use finetuning.

Overall, our topic has been touched on by related work; however, none were focused on the same goal.

Chapter 4

New Dataset and Approach

As previously mentioned, this work builds on the code of Modular Graph Extraction for Handwritten Circuit Diagram Images [4] and the CGHD dataset [1]. We enhanced the performance of the pipeline while adapting it to a quite different domain of Raspberry Pi schematics, for which we created our dataset.

The codebase from Bayer et al. [4] provided a valuable foundational framework for our research. However, during the implementation phase, we identified several technical challenges that required resolution. The training pipeline for object detection and segmentation models required substantial debugging and changes. Additionally, we added the ability to load a pretrained model. Furthermore, we addressed the memory inefficiency of the object detection model during training by adding garbage collection, mixed precision training, and explicit memory cleanup. Moreover, the inference process provided us merely with a predicted graph. Our first addition was implementing mean Average Precision (mAP) and IoU metrics for understanding the model's performance. Our second addition was automated inference on the whole dataset or a subset of test drafters, with comprehensive statistics about the mAP of the model. These changes, along with our additional work described in this section, were crucial for adapting the framework to the unique characteristics of the Raspberry Pi schematics domain.

Section 4.1

Raspberry Pi Schematics Dataset (RPi)

This research project was done to help automate the extraction of semantics out of Raspberry Pi datasheets [37]. The Raspberry Pi (RPi) datasheets are official technical documentation containing detailed circuit schematics, pinout diagrams, and electrical specifications for various Raspberry Pi models. These computer-generated documents provide comprehensive information about component layouts,

signal routing, and hardware interfaces essential for understanding the boards' electronic design. These datasheets include schematics of varying complexity and a wide range of symbols, which were ideal for testing the vision model pipeline.

Since the datasheets included a lot of text and images not needed for our goal, we started with preprocessing.

Subsection 4.1.1

Dataset Creation

To begin the automated preprocessing, the program scrapes the name and URL of all datasheets on the official Raspberry Pi website [37]. There is an optional filter phrase which only scrapes datasheets whose name contains a filter phrase, such as in our case 'pico'. Following that, we move to downloading of all PDF files of the scraped datasheets.

During the next step, we extract only the images that contain valid electric schematics. These are picked out based on a manually predefined list, excluding all other diagrams and photographs. This step includes specific modifications, which are all included in a modifications JSON file. Some images need to be rotated. Other images include a multitude of schematics instead of only one. Each of the subschematics is extracted into a separate image file using pre-labelled bounding polygons, and the background rectangle is filled with the background colour of the schematics. This colour varies between different datasheets, which our script automatically detects and handles through colour specifications defined in the modifications file. For all subschematics to have good quality for our object detection model, we opted for scaling the images by a factor of 6 during the extraction using `Docling` [38].

Following this process, we leveraged the existing drafter system from CGHD to easily integrate the new Raspberry Pi (RPi) dataset to be compatible with the graph extraction codebase while still allowing domain-specific adaptations.

Moreover, we labelled our whole RPi dataset as a test set using the Pascal VOC XML format. During labelling, we only used the classes seen by the Faster R-CNN model during training on the CGHD dataset for a more precise comparison of the performance and domain-adaptation success. Even though Raspberry Pi datasheets generally adhere to both IEEE/ANSI and IEC/DIN standards for schematic symbols, which are the same standards used by the CGHD dataset, there were still discrepancies in how the symbols were portrayed and on top, we discovered possible

incorrect labels in the CGHD dataset. Overall, the differences in visualising the same symbols between the CGHD and RPi datasets were quite visible.

The annotation statistics for the RPi dataset are summarised in Table 4.1, and the per-image bbox counts are shown in Figure 4.1.1.

Table 4.1.: Annotation statistics for the Raspberry Pi Pico sample dataset

Metric	Value
Total files	22
Total bounding boxes	1,675
Average bounding boxes	76.1
Median bounding boxes	41.5
Minimum bounding boxes	16
Maximum bounding boxes	436

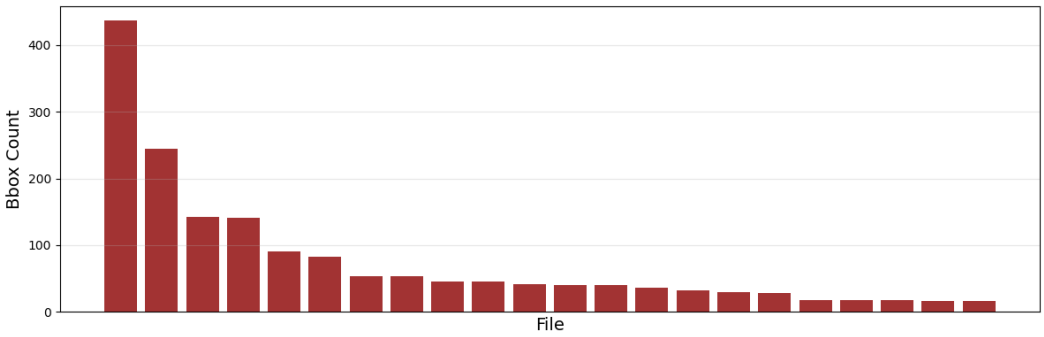


Figure 4.1.1.: Object detection bbox counts in the Raspberry Pi dataset

Due to copyright considerations, the RPi dataset images cannot be made publicly available, as they are derived from proprietary Raspberry Pi documentation [37].

Subsection 4.1.2

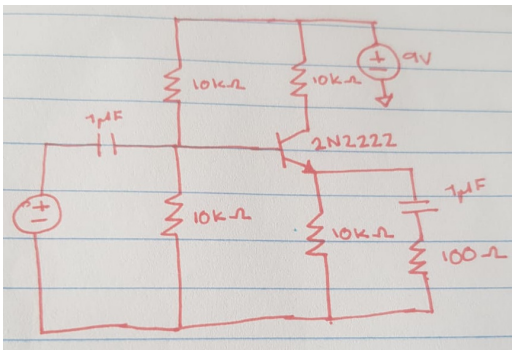
Differences between CGHD and RPi images

The training set from the CGHD dataset uses images of hand-drawn diagrams, which include many artefacts as described in Subsection 2.1.1.

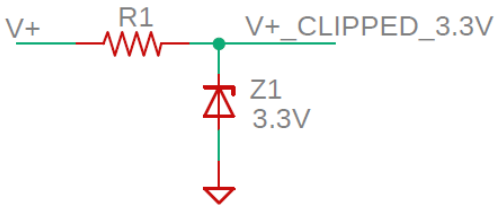
The test set from the RPi dataset was purely computer-generated schematics. The background was always perfectly white. The text was black, blue, green, and red. The symbols and wires were sometimes distinguished from each other using different colours, and sometimes there were sections of the schematics completely

changed to red signalling optional/external circuits to not mistake them for on-board parts. The complexities and sizes varied; nevertheless, the shapes and lines were always perfect as a computer-generated schematic should be. The RPi dataset included vastly more text than the CGHD dataset.

Because of these prominent differences, the training pipeline has to be generalised and robust to properly transfer the knowledge.



(a) CGHD example (drafter_1/C2_D2_P1)



(b) Computer-generated schematic similar to RPi images[1]

Figure 4.1.2.: Comparison of hand-drawn CGHD dataset images versus computer-generated RPi-style schematics

Section 4.2

Faster R-CNN Enhancements

To increase the performance of our object detection model, Faster R-CNN with ResNet backbone, we introduced focused changes to the loss, architecture, and dataloader.

Focal Loss

As was shown in Figure 2.1.1 in Section 2.1, the class imbalance in the dataset is quite strong. Even after removing the background helper class and the text class, a pronounced class imbalance still persists. This can lead to the model becoming biased during training for predicting the majority classes. Therefore, our first change to the default Faster R-CNN was focused on mitigating this imbalance. We introduced Focal Loss [15] instead of the default cross-entropy classification loss [39], which should help mitigate this common issue in object detection tasks. In our implementation we use $\alpha = 1.0$ and $\gamma = 2.0$ with Equation 2.1.

[1] Source: `https://electronics.stackexchange.com/questions/414563/reduce-noise-added-with-zener-clipper`

Generalised Intersection over Union (GIoU) Loss

Moreover, we switched to the GIoU bbox regression loss instead of the default Smooth L1 loss. This is beneficial in case of circuit diagram schematics since it considers non-overlapping bboxes which are very close, resulting in faster convergence (details in Subsection 2.4.2).

Efficient Channel Attention (ECA)

Furthermore, we modified the architecture of the Faster R-CNN model with its ResNet Backbone to enhance the performance on our domain. We implemented ECA following Eq. 2.7 from Subsection 2.5.1.

This is beneficial in our domain of circuit diagrams because it emphasises channels that respond to smaller symbols or narrow wires. It helps the detector focus on minute differences between similar types of symbols, such as `capacitor.polarized` vs `capacitor.unpolarized`. Moreover, this leads to better distinguishing of the same symbols that are placed right next to each other. Furthermore, ECA's efficiency proved beneficial by adding minimal computational cost compared to what other attention mechanisms require.

Dilated Convolutions

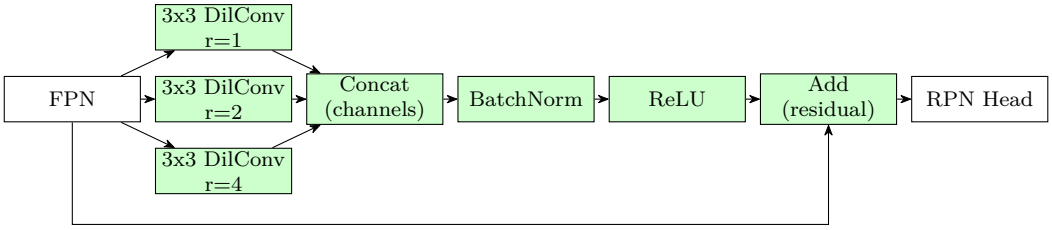


Figure 4.2.1.: All green blocks were added in our implementation of dilated convolutions addition to Faster R-CNN.

Another architectural change was attaching dilated convolution blocks to the FPN layers. In our implementation, we process the image through multiple dilated convolutions, specifically, we use dilation rates $r \in \{1, 2, 4\}$ to give us more comprehensive features across these scales. We do the processing in parallel, and then we concatenate them along the channel dimensions, we apply batch normalisation, and finish with ReLU to return it to the original format.

We allow the model to capture much larger areas and with that, more complex symbols, such as the different types of integrated circuits, which can span across

a third of the whole image. This method preserves the details, unlike other methods such as pooling or strided convolutions. Therefore, dilated convolutions should help the Faster R-CNN model increase its performance without too much model complexity.

Image Transformations

To enhance the model's robustness against real-world artefacts, we implemented a set of image transformations introduced in Section 2.6. Each combination of transformations was applied to half of the training data, ensuring the other half remained unmodified. Transformations were not applied to the validation or test sets. The parameters were chosen based on our knowledge of the topic and some light testing.

Grayscale is implemented using torchvision's `RandomGrayscale` [13].

Rotation is added using torchvision's `RandomRotation` with a maximum rotation angle of 5 degrees in either direction, mimicking the light rotation found in the CGHD dataset.

Scaling is achieved using torchvision's `RandomAffine` with scaling factors ranging from 0.9 to 1.1. These moderate scaling bounds were selected to improve model generalisation while avoiding excessive distortion that could obscure symbol features.

Perspective is created using torchvision's `RandomPerspective` with a distortion scale of 0.05.

Colour Jitter is implemented using torchvision's `ColorJitter` and jitter coefficients brightness 0.2, contrast 0.2, saturation 0.01, and hue 0.05.

Auto Contrast is done using torchvision's `RandomAutocontrast`.

Gaussian Blur is achieved using torchvision's `GaussianBlur` with gaussian kernel size of 3.

Noise is implemented using a lambda function that adds Gaussian noise with 2% intensity to the input tensor.

Erasing is implemented using torchvision's `RandomErasing` with the random erased rectangle taking up 0.5% to 2% of the image size. The rectangles have a shape ratio in the range 0.3 to 3. We kept this smaller to not remove whole symbols using one erasing. This way, it removes a part of the symbol for the model to learn the features of the rest.

Section 4.3

VLM Implementation

We ran `allenai/Molmo-7B-D-0924` [40] model locally for our experiments using Python and HuggingFace's Transformers library [41]. The moderate size of this model allowed us to run it comfortably on one consumer GPU (More info in Appendix C).

Subsection 4.3.1

Accuracy Metrics

To assess Molmo's ability, we decided to evaluate it on two metrics in object detection.

Metric 1: Predicted Classes

The first metric was focused on finding the unique set of classes that appear in the image. We gave it the set of symbol classes used by the Faster R-CNN model without the helper classes (background, unknown, junction, crossover) and without the text class. The metric was computed as the Intersection over Union mentioned in 2.4.2. The formula being:

$$\text{IoU} = \frac{|\text{Predicted Classes} \cap \text{Ground Truth Classes}|}{|\text{Predicted Classes} \cup \text{Ground Truth Classes}|} \times 100$$

Where the predicted and ground truth classes were sets of unique classes.

Metric 2: Predicted Locations

The second metric focused on counting how many predicted locations of a symbol were inside the correct bbox. The VLM is not able to predict a bbox, but a predicted centre point. We ran one prompt for each singular class to let the model fully focus on one task.

The metric was computed as:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \times 100$$

Where a prediction is considered correct if the predicted point (x, y) satisfies:

$$x_{\min} \leq x \leq x_{\max} \text{ and } y_{\min} \leq y \leq y_{\max}$$

for the ground truth bbox $[x_{\min}, x_{\max}, y_{\min}, y_{\max}]$.

Subsection 4.3.2

Prompt Engineering

We developed two distinct prompts for our VLM evaluation, each tailored to a specific task and designed to ensure clear, structured outputs.

Prompt 1: Class Identification

For the first metric of identifying symbol classes present in circuit diagrams, we used the following prompt:

Class Identification Prompt:

```
You are a specialised circuit-diagram interpreter. Given the image of
an electrical circuit diagram, identify which symbol classes from the
list below appear in the image. IMPORTANT RULES: - Only list symbols
you can clearly see and identify in the image - Do not guess or assume
symbols are present - If you're uncertain about a symbol, do not
include it - Look for distinctive visual features of each symbol type -
A typical simple circuit might only contain 2-5 different symbol types.
List only the classes you find, in a python list, and nothing else.
Symbol classes: ["terminal", "gnd", "vss", "voltage.dc", "voltage.ac",
"voltage.battery", "resistor", "resistor.adjustable", "resistor.photo",
"capacitor.unpolarized", "capacitor.polarized", "capacitor.adjustable",
"inductor", "inductor.ferrite", "inductor.coupled", "transformer",
"diode", "diode.light_emitting", "diode.thyrector", "diode.zener",
"diac", "triac", "thyristor", "varistor", "transistor.bjt",
"transistor.fet", "transistor.photo", "operational_amplifier",
"operational_amplifier.schmitt_trigger", "optocoupler",
"integrated_circuit", "integrated_circuit.ne555",
"integrated_circuit.voltage_regulator", "xor", "and", "or", "not",
"nand", "nor", "probe", "probe.current", "probe.voltage", "switch",
"relay", "socket", "fuse", "speaker", "motor", "lamp", "microphone",
"antenna", "crystal", "mechanical", "magnetic", "optical", "block"].
Do it for this specific image:
```

Prompt 2: Location Detection

For the second metric of predicting symbol locations, we used the following prompt template:

Location Detection Prompt:

```
You are a specialised circuit-diagram interpreter performing zero-shot
inference.  Given the image of an electrical circuit diagram, identify
all symbols of class "CLS_NAME".  For each symbol from that class,
output one <points> tag and nothing else, in the exact format:  <points
x1="..." y1="..." x2="..." y2="..." x3="..." y3="..."
alt="CLS_NAME">CLS_NAME</points> Use the coordinates of the center of
the bounding box of each symbol.  Do not include any additional text or
tags.  Example for 3 different locations:  <points x1="20.8" y1="42.4"
x2="21.0" y2="53.6" x3="22.3" y3="54.8"
alt="CLS_NAME">CLS_NAME</points>
```

In this template, "CLS_NAME" is replaced with the specific symbol class name (e.g. "resistor", "capacitor.unpolarized") for each query. This allows us to generate targeted prompts for detecting specific symbol types in circuit diagrams.

These prompts were carefully designed to constrain the model's output format and improve parsing reliability. The first prompt explicitly instructs the model to return results as a Python list, while the second prompt enforces XML formatting with specific coordinate attributes for precise location tracking. For more information about the process which we took to get to these final prompts, see Appendix D.1.

In short, we focused on giving the model its role, the context of the domain, its task, rules limiting excessive and incorrect output, instructions for the format of the output, and a list of classes to choose from. Interestingly, adding Chain-of-thought-like reasoning [42], shown to enhance performance in zero-shot settings, did not help in our domain.

Subsection 4.3.3

Few-shot Examples

For few-shot learning, we annotated two new images. For the first metric of predicting the classes appearing in the image, we did two-shot learning with both images. These included six different symbol classes, with only one overlapping both images. For the second metric of predicting locations of classes in the image, we did one-shot

learning with the location of resistors in our first image. More information on these prompts can be found in Appendix D.2.

We converted the images into RGB and built up the conversation history, inserting user messages, images, and VLM answers for each shot. At the end, we appended the current user prompt with the current image. We did not include a special system prompt since that was included in each shot and also in the final user prompt to ensure the model's compliance. We then processed this message history through the Molmo processor and generated a result with a maximum new token length of 512. We experimented with adding more; however, that did not make a difference with our metrics.

Chapter 5
Results and Discussion

This chapter will first focus on how we achieved up to a 7.3% higher test mAP for object detection by modifying the losses, architecture, and image processor of the Faster R-CNN model during training. And secondly, we analysed a Visual Language Model's ability to detect symbols and understand their location. Taking the results and comparing them to the Faster R-CNN model to see how a specialised system copes against an out-of-the-box general system with a greater range of understanding of the principles that work in electric schematics.

Section 5.1
Maximum Bounding Boxes

Before experimenting with additions to the training pipeline, we tested our baseline model trained on a configuration file that the author of the original repository provided us with. This model is not yet publicly available.

As noted in Chapter 2, the maximum number of bboxes in the CGHD dataset is 542, while in the RPi dataset it is only 436. Our first experiment was to test limiting the maximum number of bboxes our Faster R-CNN model can predict:

- limit to 50
- limit to 100
- set to maximum in each dataset
- set to maximum in each dataset + 50

This was done due to two reasons. Firstly, to observe how effective the model is at detecting large numbers of bboxes above 100. And secondly, whether allowing predictions over the actual count degrades the model performance or introduces

hallucinations. It is important to note that the threshold for accepting a prediction was set to 0.8.

Table 5.1.: Impact of maximum predicted bboxes on model performance (mAP)

Max BBox	CGHD val
50	33.9
100	42.5
max	**44.3**
max+50	**44.3**

"CGHD val" are only our validation drafters 27 and 28.

As can be seen in Table 5.1, there is a considerable advantage to allowing the model to predict more than 50 and 100 bboxes when looking at the CGHD dataset. Looking at the numbers, the model correctly predicts a significant amount of correct bboxes above the 50 mark and only slightly more above the 100 mark. This reduction could be due to most diagrams in these drafters having ground truth of around 50-100 bboxes.

We did not observe any benefits to increasing the limit above the maximum. It is important to note that the minimum and maximum accuracy over the validation set were very similar in the latter three experiments.

From observing the results visually, there were instances of hallucinations and misclassifications in the cases of using the maximum number of bboxes.

Misclassifications occur when the model predicts a correct bbox, nevertheless, assigns it the wrong class. In our case, misclassifications were very rare, only occurring with very similar symbols, such as different diodes or integrated circuits. Rare misclassifications were most probably achieved through a high acceptance threshold.

Hallucinations occurred when the model predicted objects in places where there was not an object present or when it predicted many smaller objects instead of the true whole object being in one bbox. The Faster R-CNN very often predicted multiple smaller bboxes around a larger part of the text instead of one bbox around the whole text. For example, instead of detecting "Audio Power" as one text object, the model predicted "Audio" and "Power" as separate objects. To mitigate this risk, one would have to include semantic understanding in the object detection model.

The model also often predicted junctions at the corners of rectangular symbols. This is an unfortunate artefact of the implementation of helper classes in the Modular Graph Extraction paper [4]. The implementation added helper object classes for

turns in wires to be able to reconstruct the graph by connecting straight wires to objects rather than taking complex wires with many turns as one object.

The predicted bboxes from the model were well placed around the symbols, even in the most complex schematics. This shows the power of the two-stage detection process, where the Region Proposal Network first generates Region of Interest proposals, which are then refined by the classification and regression heads to produce the final bboxes and class predictions. The main drop in accuracy in more complex schematics was due to missed symbols. Even when we increased the resolution of the images to give the model more detail, the more complex diagrams always resulted in scaled-down symbols, making it more difficult for the model to spot them.

We also tried the models on a few images of the RPi dataset. A pleasant surprise when transferring the domain from CGHD to RPi datasets was that the model was able to accurately predict many symbols that were of a different standard. Even within the RPi dataset, there were two types of grounding symbols represented by the same `gnd` class. Sometimes it was shown as a triangle, meaning signal ground, and sometimes as four parallel lines, each shorter than the other, meaning earth ground. This comes back to the most pressing issue in the field of electric schematics. There are various similar symbols and various standards, and one cannot be certain that documentation from one company will contain only one symbol standard.

Section 5.2

Faster R-CNN Architecture Adjustments

In this section, we present results from our Faster R-CNN enhancements in the form of losses, architectural changes, and image transformations, as well as hyperparameter tuning. These additions combined resulted in a substantial increase in accuracy. All our changes were compared to the original baseline from the Modular Graph Extraction paper [4], which we retrained using the same parameters (except for drafters) to have an accurate comparison. Therefore, in this section, "Our Baseline" is this retrained baseline with our set of drafters.

All experiments in this section were trained from scratch on the same sets of drafters of the CGHD dataset:

```
drafters_set_train: [-1,23,24,25,26],
drafters_set_val: [27,28],
drafters_set_test: [29,30]
```

Subsection 5.2.1

Losses & Architecture changes

Table 5.2.: CGHD validation mAP performance for different model configurations

Configuration	CGHD val mAP (%)
Baseline	44.3
Focal loss	54.2
GIoU loss	54.2
Focal + GIoU	**54.3**
ECA	53.3
Dilated convolutions	52.2
Focal + GIoU + ECA	53.4
Focal + GIoU + Dilated	53.1

We experimented with more combinations; however, those did not yield any further improvements. The best configuration seen in Table 5.2 was a combination of focal and GIoU losses which resulted in an increase of 10% validation mAP over the baseline. Quite interestingly, both losses, separate or combined, beat the score of both architectural changes. ECA and dilated convolutions increased the performance; however, they did not surpass the losses.

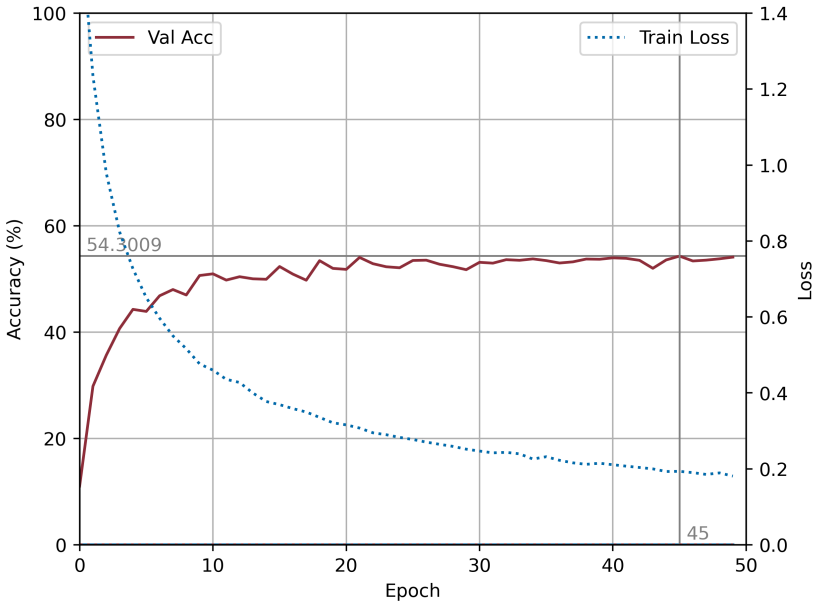


Figure 5.2.1.: Learning curve showing training progress of *Focal + GIoU* run with accuracy represented by mAP

The training of the best model was quite stable, as seen in Figure 5.2.1, relatively quickly rising close to the maximum mAP. To achieve such results, we employed hyperparameter tuning with the resulting model having this set of hyperparameters different to the baseline:

Table 5.3.: Hyperparameter changes from baseline configuration

Parameter	New Value	Old Value
learning rate	0.0035	0.01
momentum	0.9	0
gamma	0.98	0.99
batch size	1	3
box detections	542	500

Where `box detections` is the maximum number of predicted bboxes which was tested in Section 5.1.

The whole backbone remained trainable, and no other changes were beneficial. More information on the full set of hyperparameters can be found in the accompanying GitHub repository found in Appendix A.

Subsection 5.2.2

Image Transformations

As described in Section 4.2, we added image transformations to enhance the robustness of the model. We tried nine different image augmentation techniques as well as adjusting their coefficients.

We took the best model obtained in the previous experiments (Focal + GIoU) and applied each singular image transformation to the training of this model. Again, using the same settings as described in this Section.

Table 5.4.: CGHD validation mAP performance for data augmentation techniques added to Focal + GIoU

Configuration	CGHD val mAP (%)
Baseline	44.3
Focal + GIoU	54.3
ColorJitter	**55.3**
RandomErasing	54.7
RandomGrayscale	54.6
RandomAutocontrast	54.5
GaussianBlur	54.2
GaussianNoise	53.6
RandomPerspective	47.2
RandomRotation	45.3
RandomAffine	43.2

Table 5.4 shows our best results for each transformation, after testing different parameters. Four resulted in better validation mAP, with the ColorJitter transformation increasing mAP by one whole percentage, bringing the increase from the baseline to 11%.

The dataset itself was already built on a vast variety of imperfections which is what is often the job of image transformations. In this case, it seems like adjusting the images did, in some cases, have a positive effect on its learning.

Subsection 5.2.3

Evaluating on both datasets

Given our discoveries in this Section, we took our best-performing models and ran a comparison with the baseline on our newly created Raspberry Pi dataset and the CGHD test set. Our first model was the best-performing model among all loss or architecture changes (Focal + GIoU), and the second model was the best-performing model from our image transformations experiments (Focal + GIoU + ColorJitter).

As the results in Table 5.5 show, there was a significant increase of 3.7% mAP on the RPi dataset and 7.3% mAP on the CGHD test set. This transferred well from the validation mAP increase of 10%. Interestingly, the better validation accuracy of the ColorJitter model from the training did not translate into better performance on the unseen test sets for both datasets. A possible explanation for this result is that each drafter was hand-drawn and photographed by a different person in a different setting, giving the validation set slightly different artefacts from the test set.

Therefore, our best model remains the Focal + GIoU version.

Table 5.5.: Comparison of baseline and enhanced model performance on test datasets

Model	CGHD test mAP (%)	RPi mAP (%)
Baseline	41.3	28.3
Focal + GIoU	**48.6**	**32.0**
Focal + GIoU + ColorJitter	48.0	28.6
Improvement (Focal + GIoU)	+7.3	+3.7

These transformations include coefficients which substantially influence the model's accuracy. There could be a better combination than the ones we found, and it could be beneficial to combine a few transformations; however, what we showed with these results is that there is a positive impact of using these transformations.

Section 5.3

VLM testing

In the second part, we focused on testing an out-of-the-box visual language model, Molmo-7B-D, on the same object detection task as Faster R-CNN. We divided the RPi diagrams into three size categories (small, mid, big) to understand how the

model understands each level of complexity. From Figure 4.1.1, we took all images with a number of bboxes less than 29 as the small category. Larger than small, but below 100 are mid, and above 100 are big. We handpicked two good candidates from each category for our testing in this chapter. The ones picked have number of bboxes $244, 143, 53, 30, 18, 16$, giving a good representation of the whole dataset. The prompts used were shown in Subsection 4.3.2 with the process shown in Appendix D.

Table 5.6.: VLM performance comparison across prompting strategies and diagram sizes on the RPi dataset

	Metric 1: Symbols found (%)		Metric 2: Locations Correct (%)	
	Normal	Few-shot	Normal	Few-shot
Total avg	16.7	**29.6**	**9.1**	5.4
Avg small	11.3	33.8	2.4	10.0
Avg mid	15.1	23.2	20.2	6.3
Avg big	23.7	31.7	4.6	0.0

The metrics used were described in Subsection 4.3.1. Since the VLM sometimes gave an answer too long, it did not finish within the allowed token limit, we decided to limit Metric 2 to use a maximum of 20 locations. Any extra locations always resulted in a worsening of performance.

Surprisingly, the experiments did show positive results for Molmo. As seen in Table 5.6, it performed relatively well, achieving at best an accuracy of 33.8% for symbol classes found enhanced by few-shot learning.

Nevertheless, during the experiments, we observed the following unfortunate situations.

For Metric 1, Molmo sometimes repeated one class until it ran out of tokens and sometimes did not result in a unique set of classes. Both these issues were addressed in the prompt; however, they were not fixed entirely.

For Metric 2, Molmo sometimes repeated the same coordinates multiple times for the same symbol class. In our inspection of the results, it only ever got a location correct for these classes: capacitor, gnd, resistor, and diode. Our findings showed that using few-shot learning for this task worsened the results. Few-shot learning is a very effective strategy for VLMs in cases where the shots have a similar topic in mind; however, they have to be phrased very differently with a different example. In our case Molmo often predicted the exact location from the shot for more symbols than the resistor that it belonged to in the shot. This was an issue even after

modifying the shot prompt to be different from the user prompt, to avoid such behaviour. It is not enough to have a different image for the model to avoid such an issue. The model takes too much inspiration from the structure of the correct answer and interchanges the class name and outputs it as its prediction. This happened more than half of the time, making it a large skew of results. For future work, it would be beneficial to experiment with different types of diagrams and schematics as shots to see if it mitigates this issue while enhancing performance compared to no-shot learning.

Molmo's prediction strategy of locations was different to predicting a bbox. Molmo predicted points instead of boxes. If it thought that a symbol was large, it predicted multiple points in a grid pattern around the predicted location. On top of that, it sometimes predicted different symbols extremely close to each other, creating a cluster of predicted classes in a place where only one class truly existed.

What was nevertheless a success, was the models correct style of formatting of the outputs, exactly as requested. On top of that, the model never hallucinated new symbols. Moreover, few-shot prompting helped reduce the output length during testing of Metric 2, which helped avoid long lists of hallucinated locations.

Section 5.4

Final thoughts

During this thesis, we enhanced the performance of a Faster R-CNN model on object detection and compared it to a Visual Language model, Molmo-7B-D, on the same tasks. Averaging the results from both metrics for Molmo gives us these results:

Table 5.7.: Performance comparison of final models on RPi dataset

Model	RPi accuracy (%)
Faster R-CNN Baseline	28.3
Faster R-CNN (Focal + GIoU)	32.0
Molmo-7B-D	12.9
Molmo-7B-D (Few-shot)	17.5

We achieved a significant increase of 3.7% accuracy of the Faster R-CNN model by adding Focal and GIoU losses. On the other hand, Molmo performed poorly. However, poor performance was expected with no finetuning on our domain and a smaller size of 7 billion parameters. Nevertheless, during its experimentation,

we uncovered many interesting observations, some of which pointed to this model having some understanding of the task. Having achieved up to 17.5% accuracy on a domain it quite probably never saw is quite an achievement.

Chapter 6
Conclusion and Future Work

This thesis aimed to improve an existing multi-modal computer vision pipeline and harden it on two main problems: severe class imbalance and a poor ability to transfer knowledge from handwritten, messy diagrams to clean computer-generated schematics. By replacing the default Faster R-CNN classification and box regression losses with Focal and GIoU losses, respectively, combined with hyperparameter tuning, we were able to increase evaluation accuracy on CGHD by 7.3% and on the RPi dataset by 3.7% compared to the baseline. It showed increases of up to 11% during training on the CGHD validation set. These results show that a targeted loss focused on the task at hand is, in this case, more effective than complex architectural changes in the form of ECA or dilated convolutions. Following up with image transformations showed some increase on the validation set; however, it did not translate the full gains into the evaluation set.

The newly created Raspberry Pi dataset exposed the sharp visual and stylistic gap between noisy hand-drawn sketches and clean CAD exports. It showed uniform white backgrounds, multi-colour symbol encodings and more text density. Despite being trained only on CGHD, the improved model generalised to the new data, albeit at a lower accuracy. This shows both the future possibility of this system as well as the prevailing domain gap, especially with symbol variants, which remain a core obstacle for reliable automated schematic analysis.

Experiments with Molmo-7B-D provided us with an outlook on the current state of smaller open VLMs. Zero- and few-shot learning could lead the model to recognise a set of symbol classes in a diagram, reaching an accuracy of up to 33.8% on small diagrams. Unfortunately, the accuracy of locating symbols was poor and brittle, including coordinate repetition, copying of the few-shot template, and only successfully recognising very common classes, such as a resistor. These results reinforce the opinion that pretrained VLMs are not a drop-in replacement for specialised detectors.

Overall, we have shown that targeted loss choice, architectural changes, and careful hyperparameter choices yield significant gains in electric schematic object detection and, importantly, that it transfers to a measurably different domain. Even though the performance remains below what would be needed for a fully automated extraction pipeline, the results have shown to be promising and therefore, research in this field should continue.

Future work should focus on four key directions. Firstly, progress in this field may depend more on the variety and quality of labelled data used for training. That includes increasing the size of the RPi dataset and potentially modifying it with real-world camera effects [43] to combine it with the CGHD dataset for training. Secondly, the approach of using VLMs should be explored more with larger models and agentic workflows that can focus on separate parts of the diagrams, splitting the work into more manageable pieces. Image embeddings search could help with this scenario. Another approach that should be explored is a hybrid between a classic object detection model and a VLM, combining focused precision and more general context understanding. Lastly, the issue of symbol standardisation should be given more focus, with potential mapping tables or adding sub-labels.

References

[1] F. Thoma, J. Bayer, and Y. Li, *A public ground-truth dataset for handwritten circuit diagram images*, 2021. arXiv: 2107.10373 [cs.CV].

[2] *Graphic symbols for electrical and electronics diagrams (including reference designation letters)*, New York, NY, USA: Institute of Electrical and Electronics Engineers, 1975.

[3] *Graphical symbols for diagrams – part 1: General information and indexes*, Also published as DIN EN 60617-1:2012 by Deutsches Institut für Normung, Geneva, Switzerland: International Electrotechnical Commission, 2012.

[4] J. Bayer, L. van Waveren, and A. Dengel, *Modular graph extraction for handwritten circuit diagram images*, 2024. arXiv: 2402.11093 [cs.CV].

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, ser. Lecture Notes in Computer Science, vol. 9351, Springer, 2015, pp. 234–241.

[8] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx", in *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, G. Varoquaux, T. Vaught, and J. Millman, Eds., 2008, pp. 11–15.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[10] R. Girshick, "Fast r-cnn", in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[12] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 936–944.

[13] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 8024–8035.

[14] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet large scale visual recognition challenge", *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015, Accessed: 2 July 2025.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal loss for dense object detection*, 2018. arXiv: 1708.02002 [cs.CV].

[16] N. Khanzhina, A. Lapenok, and A. Filchenkov, *Towards robust object detection: Bayesian retinanet for homoscedastic aleatoric uncertainty modeling*, arXiv:2108.00784v2, 7 Sep 2021, 2021. arXiv: 2108.00784 [cs.CV].

[17] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, *Generalized intersection over union: A metric and a loss for bounding box regression*, https://arxiv.org/abs/1902.09630, Last revised: 15 April 2019; Accessed: 2 July 2025, Feb. 2019. arXiv: 1902.09630 [cs.CV].

[18] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "Eca-net: Efficient channel attention for deep convolutional neural networks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Accessed: 2 July 2025, 2020, pp. 11 531–11 539.

[19] F. Yu and V. Koltun, *Multi-scale context aggregation by dilated convolutions*, https://arxiv.org/abs/1511.07122, Last revised: 30 April 2016; Accessed: 2 July 2025, Nov. 2015. arXiv: 1511.07122 [cs.CV].

[20] T. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning", *Journal of Big Data*, vol. 6, p. 60, 2019.

[21] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, https://arxiv.org/abs/1706.03762, Accessed: 2 July 2025, Jun. 2017. arXiv: 1706.03762 [cs.CL].

[22] M. Deitke, C. Clark, S. Lee, *et al.*, "Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models", *arXiv preprint arXiv:2409.17146*, 2024, Submitted: 25 September 2024; Revised: 5 December 2024; Accessed: 2 July 2025.

[23] OpenAI, *GPT-4o system card*, https://openai.com/index/hello-gpt-4o/, Accessed: 2 July 2025, May 2024.

[24] S. Pichai, D. Hassabis, and the Google Gemini Team, *Gemini 1.5 pro: Our next-generation multimodal model*, https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/, Accessed: 2 July 2025, Feb. 2024.

[25] Anthropic, *Model card addendum: Claude 3.5 sonnet*, https://www-cdn.anthropic.com/fed9cc193a14b84131812372d8d5857f8f304c52/Model_Card_Claude_3_Addendum.pdf, Accessed: 2 July 2025, Jun. 2024.

[26] A. Radford, J. W. Kim, C. Hallacy, *et al.*, "Learning transferable visual models from natural language supervision", in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 8748–8763.

[27] Y. et al., *Qwen2 technical report*, https://arxiv.org/abs/2407.10671, Submitted: 15 July 2024; Revised: 10 September 2024; Accessed: 2 July 2025, 2024.

[28] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[29] L. Tao, W. Chen, and A. Kumar, "AMSNet: A public corpus of analogue/mixed-signal schematics with spice netlists", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 2, pp. 567–579, 2025.

[30] C. Kelly and J. Cole, "Digitizing images of electrical-circuit schematics", *APL Machine Learning*, vol. 2, Mar. 2024.

[31] X. Li, Y. Sun, W. Cheng, Y. Zhu, and H. Chen, "Chain-of-region: Visual language models need details for diagram analysis", in *International Conference on Learning Representations (ICLR)*, Poster at ICLR 2025, Apr. 2025.

[32] S. Meshram, K. Shah, G. Shete, A. Raj, A. Mandal, and R. R. Shah, "ElectroVizQA: A benchmark for evaluating multimodal large language models on undergraduate-level electronics questions", *arXiv preprint arXiv:2412.00102*, 2024.

[33] T. Wan, H. Liu, and L. Geng, "Application of multi-modal large models in electronic circuit image automatic annotation and caption generation", in *Proceedings of the 2024 International Symposium on Integrated Circuit Design and Integrated Systems (ICDIS)*, 2024.

[34] H. Touvron, T. Lavril, G. Izacard, *et al.*, *Llama: Open and efficient foundation language models*, 2023. arXiv: 2302.13971 [cs.CL].

[35] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, *Language models are unsupervised multitask learners*, OpenAI Blog, https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf, 2019.

[36] L. W. Nagel and D. O. Pederson, "SPICE2: A computer program to simulate semiconductor circuits", University of California, Berkeley, Electronics Research Laboratory, Berkeley, CA, Tech. Rep. ERL-M382, 1973.

[37] Raspberry Pi Foundation, *Raspberry pi datasheets*, https://datasheets.raspberrypi.com/, Accessed: 2025-06-01, 2024.

[38] N. Livathinos, C. Auer, M. Lysak, *et al.*, *Docling: An efficient open-source toolkit for ai-driven document conversion*, 2025. arXiv: 2501.17887 [cs.CL].

[39] A. Mao, M. Mohri, and Y. Zhong, *Cross-entropy loss functions: Theoretical analysis and applications*, 2023. arXiv: 2304.07288 [cs.LG].

[40] Allen Institute for AI, *Molmo-7b-d-0924 model card*, https://huggingface.co/allenai/Molmo-7B-D-0924, Accessed: 2 July 2025, 2024.

[41] T. Wolf, L. Debut, V. Sanh, *et al.*, "Huggingface's transformers: State-of-the-art natural language processing", *CoRR*, vol. abs/1910.03771, 2019. arXiv: 1910.03771.

[42] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, *Large language models are zero-shot reasoners*, 2023. arXiv: 2205.11916 [cs.CL].

[43] A. Carlson, K. A. Skinner, R. Vasudevan, and M. Johnson-Roberson, "Modeling camera effects to improve visual learning from synthetic data", in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 505–520.

[44] A. Shevchenko, M. Lytvyn, and D. Lider, *Grammarly: Free ai writing assistance*, https://www.grammarly.com/, Initial release: 1 July 2009; accessed 18 July 2025, 2009.

[45] GitHub and OpenAI, *Github copilot: Ai pair programmer for visual studio code*, https://github.com/features/copilot/, Version 1.7.4421; first announced 29 June 2021; accessed 18 July 2025, 2021.

[46] Anthropic, *Claude sonnet 4*, https://www.anthropic.com/news/claude-4, Version released May 22, 2025, 2025.

[47] DeepL GmbH, *Deepl translator for windows*, version 25.7.1.16851, 2025.

List of Abbreviations

CGHD	Circuit Graph Hand-Drawn Dataset
RPi	Raspberry Pi
bbox	bounding box
ECA	Efficient Channel Attention
FPN	Feature Pyramid Network
GIoU	Generalized Intersection over Union
IoU	Intersection over Union
mAP	mean Average Precision
R-CNN	Region-based Convolutional Neural Network
RoI	Region of Interest
RPN	Region Proposal Network
AI	Artificial Intelligence
CPU	Central Processing Unit
GPU	Graphics Processing Unit
LLM	Large Language Model
MLLM	Multimodal Large Language Model
VLM	Vision Language Model

List of Figures

List of Tables

Appendix A
GitHub Repository

Code accompanying this thesis can be found here `https://github.com/sykoravojtech/od-symbol/`. Code for locally run Molmo-7B-D can be found here `https://github.com/sykoravojtech/Molmo-7B-D/`.

The foundational code on which this thesis builds was obtained from this repository `https://codeberg.org/Circuitgraph`. There is a version on GitLab, however, that is older.

Appendix B
Usage of GenAI

This thesis employs Generative AI tools for grammar correction and wording refinement to enhance cohesion and clarity. All original content was authored by the student, and the use of AI was limited to linguistic improvements without altering or significantly expanding the original ideas. Additionally, an AI coding assistant was used for code completion tasks, strictly avoiding the generation of substantial program components. The specific software versions used are listed in Table B.1.

Purpose	**Software**	**Latest used version**
Grammar correction and wording refinement	Grammarly Web [44]	14.1244.0
Minor Code completion	VSCode Copilot [45] (Claude Sonnet 4 [46])	1.7.4421
Translation of abstract to German	DeepL for Windows [47]	25.7.1.16851

Table B.1.: Generative AI software usage and versions

Appendix C
Hardware

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

We ran a batch job with these requested values, and the choice of exact Graphics Processing Unit (GPU) and Central Processing Unit (CPU) were on the job scheduling system. Here are the values:

- Inference: 1 GPU, 1 CPU, 16GB memory.
- Object detection training with all CGHD drafters: 1 GPU, 1 CPU, 165GB of memory.
- Object detection training with our set of CGHD drafters: 1 GPU, 1 CPU, 80GB of memory.

Appendix D

VLM Prompt Engineering

Section D.1

General Prompt Improvements

To achieve the final prompt shown in Subsection 4.3.2, we followed general guidelines on prompt engineering with addition of our ideas. This section includes the main steps we tried in prompt engineering; however, for each of the prompts we tried many subtle variations of it. Logs of conversations can be found in the accompanying GitHub repository.

We started by giving the VLM context with what task it needs to do and what information it has access to. And lastly, we only want it to predict symbol classes from a list, so instead of LIST_OF_CLASSES, we added the whole list of possible classes (Seen in Subsection 4.3.2). Here is how the initial version looked.

```
You are a specialised circuit-diagram interpreter.  Given the image of
an electrical circuit diagram, identify which symbol classes from the
list below appear in the image.  Symbol classes:  LIST_OF_CLASSES
```

Following this, we focused on limiting hallucinations and making sure the answer is in our preferred format. Adding this sentence to the prompt.

```
List only the classes you find, in a python list, and nothing else.
```

And we tried being more specific with its confidence level; however, that did not result in better performance on Metric 1.

```
Only include classes you are highly confident about (>80% certainty).
Be conservative - it's better to miss a symbol than to incorrectly
identify one.
```

So instead we tried giving it a set of rules, which sometimes gave us better results, which, after more testing, was most probably because of the emphasis on "might only contain 2-5 different symbol types". This was beneficial because the biggest issue overall with this VLM was that it always predicted too many classes, and this part of the prompt reduced the false positives.

```
IMPORTANT RULES: - Only list symbols you can clearly see and identify
in the image - Do not guess or assume symbols are present - If you're
uncertain about a symbol, do not include it - Look for distinctive
visual features of each symbol type - A typical simple circuit might
only contain 2-5 different symbol types.
```

We also wanted to explore some chain-of-thought reasoning, trying both "Think step by step" and the following more detailed addition. Even just adding the four words was shown to have a positive effect [42]. Both of these resulted in the model giving us a longer predicted list. This was, unfortunately, worse; therefore, we removed it.

```
For each symbol you identify: 1. First, locate the symbol visually in
the image 2. Then, verify it matches the standard representation for
that symbol type 3. Only include it if you can clearly see its
distinctive features. Be precise and conservative.
```

Since the main issue was overpredicting the amount of classes, we tried to solve that issue again unsuccessfully with this addition.

```
Do NOT include classes that: - You cannot clearly see in the image -
Might be confused with other symbols - Are partially obscured or
unclear - You are guessing about based on context. Only include
symbols with clear, unambiguous visual representation.
```

We also tried changing the beginning of the main prompt with the following; however, that resulted in more false positives.

```
You are an expert at object recognition of symbols in a circuit diagram
```

And enforcing only a smaller number of items made the model respond with more hallucinated classes.

```
so OUTPUT **NO MORE THAN 6 ITEMS**
```

When we did it this way, it reduced hallucinations with "8" while increasing hallucinations when we tried another number, such as "10". This was very confusing.

```
Give me a list with max 8 elements without repeating them.
```

Overall, this prompt engineering did not work as expected, and some of the modifications to the prompt did not work as literature or other experiments said.

Section D.2

Few-shot prompts

The following prompts were used for few-shot learning experiments with the VLM model. These examples were provided to the model before the actual test image to show it the expected format and behaviour.

Shot 1: Example for Metric 1

User Prompt (Shot 1):

```
Analyze this first image of a circuit diagram and identify which
electronic symbol classes are present. Rules: Only list symbols
clearly visible, don't guess, exclude uncertain ones. Return as python
list. Available classes: LIST_OF_CLASSES
```

Assistant Response (Shot 1):

```
["voltage.battery", "resistor", "diode.light_emitting"]
```

Shot 2: Example for Metric 1

User Prompt (Shot 2):

```
Analyze this second image of a circuit diagram and identify which
electronic symbol classes are present. Rules: Only list symbols
```

```
clearly visible, don't guess, exclude uncertain ones. Return as python
list. Available classes: LIST_OF_CLASSES
```

Assistant Response (Shot 2):

```
["resistor", "diode.zener", "gnd", "terminal"]
```

Shot 3: Example for Metric 2

This training example was used specifically for metric 2 (location detection) to teach the model how to output coordinate information for specific symbol classes.

User Prompt (Shot 3):

```
You are a specialised circuit-diagram interpreter performing zero-shot
inference. Given the image of an electrical circuit diagram, identify
all symbols of class "resistor". For each symbol from that class,
output one <points> tag and nothing else, in the exact format: <points
x1="..." y1="..." x2="..." y2="..." x3="..." y3="..."
alt="resistor">resistor</points> Use the coordinates of the center of
the bounding box of each symbol. Do not include any additional text or
tags. Example for 3 different locations: <points x1="20.8" y1="42.4"
x2="21.0" y2="53.6" x3="22.3" y3="54.8"
alt="resistor">resistor</points>
```

Assistant Response (Shot 3):

```
<points x1="56.6" y1="24.4" alt="resistor">resistor</points>
```