Detection of Hand Drawn Electrical Circuit Diagrams and their Components using Deep Learning Methods and Conversion into LTspice Format

Master's Thesis in Computer Science

submitted
by

Dmitrij Vinokour

born 19.12.1993 in St. Petersburg

Written at

Lehrstuhl für Mustererkennung (Informatik 5)
Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg.

Advisor: Florian Thamm M. Sc., Felix Denzinger M. Sc., Prof. Dr. Andreas Maier

Started: 01.01.2021

Finished: 31.07.2021

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien des Lehrstuhls für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Erlangen, den 11. August 2021

Übersicht LTspice ist ein Simulationsprogramm für elektrische Schaltungen, in dem Schaltungen modelliert und simuliert werden können, um benötigte Parameter zu erhalten, wie beispielsweise den Strom der durch ein Bauteil fließt. Während das Simulieren schnell ist, kann das Modellieren einer Schaltung durchaus Zeit beanspruchen und ist oft unnötig, da Schaltungen bereits in handgezeichneter Form existieren. Eine automatisierte Umwandlung von handgezeichneten elektrischen Schaltungen in das LTspice Schematik Datenformat, könnte daher die Nutzerfreundlichkeit des Programms erhöhen. Daher beschäftigt sich die vorliegende Arbeit mit der Umwandlung von handgeschriebenen elektrischen Schaltungen in das LTspice Schematik Datenformat. Die Pipeline beinhaltet das Erkennen von elektrischen Bauteilen und deren Annotationen, sowie das Segmentieren der gesamten Schaltung. Hierbei kommt das Objekt Erkennungs-Netzwerk You Only Look Once (YOLO), sowie das Segmentierungs-Netzwerk MobileV2-UNet zum Einsatz. Um die bestmöglichen Ergebnisse zu erzielen, werden die optimalen Hyperparameter beider Netzwerke über eine Reihe von Experimenten ermittelt. Hierbei, wird zunächst eine initialie Lernrate bestimmt, dann werden verschiedene Augmentierungen auf ihre Parameter getestet und abschließend with eine Gittersuche über mehrere Loss Funktionen, Lernraten und Batch Größen durchgeführt. Die weiteren Schritte der vorgestellten Pipeline beinhalten zudem, das Erzeugen der Topologie einer Schaltung, das Verbinden von Annotationen mit den zugehörigen elektrischen Schaltungs-Bauteilen, sowie das Erkennen des Texts in den Annotationen. Um die Umwandlung quantifizieren zu können wird außerdem ein Evaluations Algorithmus vorgestellt, der mit dem Graph Edit Distance Algorithmus [Che18] verwandt ist.

Abstract LTspice is a simulation program for electrical circuits, in which circuits can be modeled and simulated to obtain required parameters, such as the current flowing through an electrical circuit component. While simulating is fast, modeling a circuit can take time and is often unnecessary, primarily when circuits already exist in hand-drawn form. An automated conversion of hand-drawn electrical circuits into the LTspice schematic data format could increase the programs's usability. Therefore, this thesis deals with the conversion of hand-drawn electrical circuits into the LTspice schematic file format. The conversion pipeline includes the detection of electrical components and their annotations and the segmentation of the entire circuit. The object detection network You Only Look Once (YOLO) and the segmentation network MobileV2-UNet are used. In order to achieve the best possible results, the hyperparameters of both networks are improved over a series of experiments. Here, first an initial learning rate search is conducted, followed by the testing of different augmentations and their parameters. Finally, a grid search is performed to find the optimal configuration of loss function, learning rate, and batch size. The further steps of the presented pipeline also include the generation of the topology of the circuit, matching of annotations with their corresponding electrical circuit component and the recognition of the text in the annotations. In order to quantify the results, an evaluation algorithm related to the Graph Edit Distance algorithm [Che18] is proposed.

Contents

Chapter 1

Introduction

1.1 Motivation

Electrical Circuit Diagrams (ECDs) are used in electrical engineering to represent the functionality of an electrical circuit visually. In early design phases, especially hand-drawn sketches of ECDs are used to get an initial concept of the underlying problem. Engineers have more time to concentrate on the problem than on the design in Computer-Aided Design (CAD) programs. In general hand-drawn methods are considered to be 90% more time-efficient than CAD methods [Ref08]. When studying electrical engineering or related fields, students often have to draw sketches of ECDs and calculate the physical properties of the Electrical Circuit Components (ECCs) in the circuit. The complexity of the calculations grows with the circuit's size and also depends on the used ECCs. For example, when sources with auxiliary properties are used, calculations become dependent on the frequency of the sources, which adds additional complexity. In the real world, such calculations are normally done in a circuit simulation software such as LTspice. Here, the circuit is designed with its ECCs and their physical properties. Parameters of interest can then be obtained by simulating the circuit in that program. While the simulation of a circuit is fast and the parameters of interest can be obtained fast, designing a circuit in such a program is generally slower than drawing it by hand.

To increase the efficiency of engineers and students in designing and verifying their calculations respectively, a method to convert an image of a hand-drawn ECD could ease the use of CAD software and circuit simulation software.

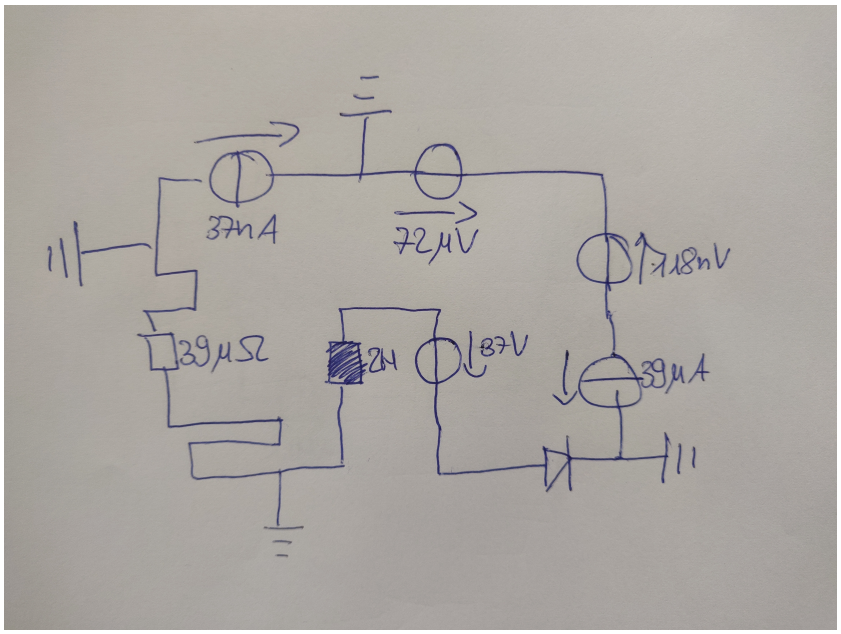


Figure 1.1: A drawing of an ECD on a grid paper background with its ECCs and annotations.

1.2 Problem Statement

To understand how a conversion of a hand-drawn ECD into a digital format can be achieved first, the various components of an ECD have to be presented. Typically, an ECD consists of ECCs, where for each ECC, a unique symbol is defined in the international standard [IEC]. ECCs are connected with lines, which correspond to wires in the real world. Wires and ECCs together form the topology of the underlying circuit.

Additionally, ECCs can be further be specified by an annotation next to their symbol, which consists of a digit followed by a unit. This annotation determines the physical properties of the underlying ECC. Voltage sources and current sources are annotated with an arrow next to their symbol, which indicates the direction of the potential difference or, in the latter case, the direction of the current flow.

Furthermore, it is common that ECDs are drawn on gridded paper since it provides a simple way to align the different components. An example of an ECD drawn on grid paper with its ECCs and annotations can be seen in figure 1.1.

So a conversion pipeline has to be able to capture all of the above properties of an ECD. To achieve this the necessary subtasks have been identified as follows:

1. Detect the class and position of an ECC and its annotations
2. Match annotations to their respective ECC
3. Interpret the text in the textual annotations
4. Detect the topology of the ECD, regardless of the used paper background
5. Embed the information into the target format

1.3 Related Work

To provide some context for the task at hand, in the following related approaches to the goal of this thesis are presented. The gathered works can be structured into three categories:

1. Classification of ECCs and related components (section 1.3.1).
2. Extraction of ECCs from an image of an ECD with following classification (section 1.3.2).
3. Extraction of all related information from ECDs or related structured diagrams and classification of the components (section 1.3.3).

1.3.1 Classification

Günay et al. [Gü20] compared in their work the accuracy of various common Convolutional Neural Network (CNN) architectures trained on a dataset of hand-drawn ECCs. The dataset consisted of four classes (resistor, capacitor, inductor, voltage source) and had overall a size of 863 images. With the best performing CNN architecture, they were able to obtain an accuracy of 83%.

Rabbani et al. [Rab16] tested the classification performance of letters, numbers, and ECCs in their work. For each class, a total of 20 samples was used. The classification was done using a Multilayer Perceptron (MLP), where different geometric and moment features were fed into the network. Overall they reported an f1-score of 83.5%.

Roy et al. [Roy20] used in their dataset 20 different ECCs with around 150 samples per class. As a classifier, Sequential Minimal Optimization (SMO) was used, which was fed with

different features based on texture and shape. Further, they compared the performance of the SMO classifier to the Random Forest classifier, an MLP, a K-Nearest-Neighbor (KNN) classifier, and the Naive Bayes classifier. An accuracy of 91.88% was reported for the SMO classifier.

1.3.2 Extraction and Classification

Dewangan and Dhole [AD18] proposed a pipeline that can extract ECCs from an ECD and classify it. The initial step was formed by a preprocessing pipeline, which binarized the image, denoised it, and then skeletonized it. Afterwards, segmentation was used to segment wires and components individually. The segmentation has not been described in full depth, thus no detailed insights can be given at this point. Finally, based on the segmented ECCs, numerous features were obtained like area, major axis length, minor axis length, centroid, orientation, eccentricity, convex area, filled area, equiv diameter, and extent. Those features were used to train a KNN classifier, where an accuracy of 90% was reported.

Moetesum et al. [Moe18] proposed a similar pipeline. A dataset of 100 images of ECDs, which contained a total of ten different classes, with 35 samples per class, was used. With a total amount of 350 samples, 200 were used for training and 150 for testing. Preprocessing was done by first converting the image to grayscale, denoising with a smoothing filter, followed by binarization of the image. Afterwards, the ECCs were segmented using different strategies based on the shapes of the ECCs. For example, diodes were segmented using the assumption that the shape is closed, and capacitors were segmented using its parallel line property. Afterwards, Histogram of Oriented Gradients (HOG) features were obtained from the segmented components, which were then fed into a Support Vector Machine (SVM). An accuracy of 87.71% was reported on the whole dataset, where the evaluation was done using transient errors, i.e., an error produced in the segmentation will automatically produce an error in the classification since the component could not be segmented and hence also not classified.

1.3.3 Extraction, Classification and Topology Creation

Edwards and Chandrun [Edw00] proposed a pipeline to create an ECD from a hand-drawn image into a digital form. The dataset comprised 449 ECCs and 107 nodes, where a node is defined as a connection or set of connections. In the preprocessing step, first a grayscale conversion was performed, followed by noise reduction and image binarization. To mitigate common errors such as small gaps in lines, morphological closing was performed, and the number of pixels per region was normalized through a thinning algorithm. In the next step, the ECCs and connections were

segmented, based on the pixel density in a region. Here the assumption was that plain lines (wires) will have a lower pixel density inside a small circular region than ECCs or connections and with an appropriate threshold, they could be segmented. Classification of the segmented components was performed with a KNN classifier, which classifies different geometric and moment features of the segmented regions. To recreate the topology, the connections between the ECCs had to be identified. This was done using the Breadth-First Search algorithm, which started at a certain component and tried to find a path to another one. When a path was found, it indicated a connection between those two components. Finally, a node recognition accuracy of 92% and a classification accuracy of 86% were reported.

Refaat et al. [Ref08] proposed a context-independent approach for structured diagram recognition utilizing SVMs. Their dataset was comprised of graphics, which were basic line primitives such as rectangles, circles, triangles, ellipse, and text annotations, as well as lines connecting the graphics. The preprocessing consisted of an adaptive thresholding approach to separate the foreground from the background. Segmentation was performed based on size filters to segment the text annotations, which were relatively small compared to the graphics. To segment the graphics and lines, a model was used, which imitates the human perception of shapes based on the smoothness and continuity property. The used algorithm can be found in [Cao]. After segmentation, the shapes were classified using an SVM classifier that was trained with 120 images of each shape. For testing, 20 images per shape were used, and a classification accuracy of 90% was reported.

Dhanushika and Ranathunga [Dha19] proposed an approach that can provide a boolean expression from an image of a logical gate diagram. Their dataset consisted of 300 diagrams, where 240 were used for training, and 60 were used for testing. To detect the logical gate components, two different object detection networks were presented. They trained a You Only Look Once (YOLO) network and a network from the open-source deep learning framework TensorFlow [Aba15] (the exact network was not mentioned). Detection of connections between components was done using a rule-based approach. First, the lines were extracted using the Hough Transform [She15] and vertical and horizontal lines were split up in different chunks. Afterwards, intersections were calculated between the lines of the chunks, and different patterns were identified based on the proposed rules. For example, a pattern for an edge was identified or for a “T” connection, which indicates that three components are connected at this particular connection. By starting from a logical gates component and tracing along the identified pattern, it could be identified whether two (or more) components were connected. From the gathered results, a boolean expression could then be generated. Three different metrics were reported based on the

different detection types. For the YOLO network, an accuracy of 40% was reported, while for the unknown TensorFlow object detection algorithm, an accuracy of 90% was reported. Further, for the line and connection detection, an accuracy of 83.11% and 89.49% were reported, respectively.

1.4 Goals of this Thesis

1.4.1 Task Description

In section 1.2, the subtasks at hand were presented, which are required to achieve a conversion pipeline. In this section, each subtask is presented with a technical solution, which will briefly be described in the following. Identifying the class and position of the ECCs and annotations is essentially an object detection problem. Therefore, the object detection network YOLO will be used to identify those. Detecting the topology will be done by combining various computer vision methods in a specifically tailored algorithm. Here also, the emphasis should be put on the different paper backgrounds, which can be handled by a neural network dedicated to the segmentation task, more precisely the MobileNetV2-UNet (MUnet). Further, a simple algorithm will be introduced, which allows the matching of annotations to their respective ECC. The interpretation of the textual annotations is the only step that is not handled, but could be solved by an Optical Character Recognition (OCR) engine such as Tesseract [Smi09]. The last step will be the embedding of the information into an output format, which will be demonstrated based on the LTspice schematic file format.

1.4.2 Contribution

The contributions of this work can be summarized as follows:

- Creation of a dataset of ECDs, with ECCs in German notation (section 3.1).
- A pipeline that receives an image of an ECD, either on white or grid paper, classifies the components and returns a schematic file for LTspice as output (section 3.2).
- A systematic ablation study-like parametrization of the two branches of the proposed pipeline:
  - ECC Detection: You Only Look Once Version 4 (YOLOv4)-Tiny (section 4.1)
  - ECD Segmentation: MUnet (section 4.2)

- The proposal of an evaluation algorithm measuring the similarity between two detected topologies (section 5.1).
- An evaluation of the proposed pipeline with the proposed evaluation algorithm (section 5.2).

Chapter 2

Theory

2.1 Electrical Circuit Diagrams

An ECD consists of ECCs where for each ECC, a unique symbol is defined in the international standard [IEC]. ECCs are connected with lines, which correspond to wires in the real world. Additionally, ECCs are further specified by an annotation next to their symbol, which consists of a digit followed by a unit. This annotation determines the physical properties of the underlying ECC. Voltage sources and current sources additionally are annotated with an arrow next to their symbol, which indicates the direction of the potential difference or, in the latter case, the direction of the current flow.

In Figure 2.1, the ECCs, which are used in this thesis, are shown with their horizontal orientation. Note that each ECC can be rotated three times by 90° and would still result in a correct notation. Furthermore, resistors, inductors, capacitors, and grounds are rotation invariant in regards to their physical properties, while sources and diodes change their physical behavior inside the ECD when rotated.

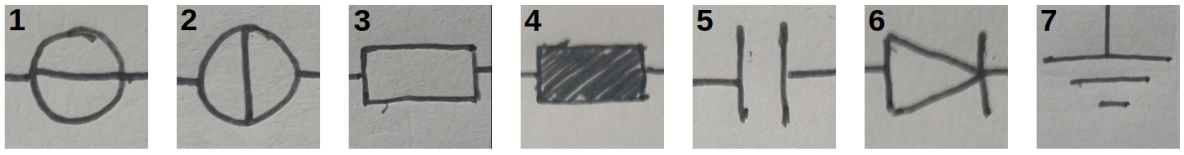


Figure 2.1: All used ECCs in this thesis in German notation: 1. Voltage Source, 2. Current Source, 3. Resistor, 4. Inductor, 5. Capacitor, 6. Diode, 7. Ground

2.2 LTspice

LTspice is a circuit simulation software where circuits can be modeled, parameterized, and simulated. A modeled circuit is stored in an LTspice schematic file with the file extension .asc. Since the last step in the pipeline of this thesis is the conversion of the hand-drawn ECDs into an LTspice schematic file, the file structure and syntax were analyzed and are presented in this section. The understanding of the syntax is necessary to understand which parameters have to be obtained throughout the pipeline to produce a valid LTspice schematic file.

LTspice files are written in plain text and are human-readable. The file structure can be interpreted line by line, meaning that one command is written on one line. Inside a line, a command is separated by a space. In most cases, the first word of a line is a keyword indicating the used command, which then is followed by parameters for the command.

LTspice itself provides a grid where components are aligned to. This grid has a size of 32×32 units, where a unit is an abstract measure inside of LTspice.

Header

Each schematic file starts with a header, which defines the used version of the syntax. Throughout this thesis, the fourth version of the syntax is used. Table 2.1 shows the syntax for the version definition command.

	Keyword	**Param1**
Syntax	VERSION	version number
Info		in this thesis 4

Table 2.1: LTspice header syntax

Symbols

In LTspice, ECCs are called symbols. The syntax to define a symbol is presented in table 2.3. The command is declared by using the keyword *SYMBOL* followed by a symbol name, where the symbol name is a mapping to an ECC. All symbol names which are used in this thesis are shown in table 2.2. The symbol name is followed by two integers defining the x and y coordinate of the symbol. The coordinates are representing the upper left corner of the image of the symbol used to represent the ECC inside LTspice. Additionally, a rotation has to be provided with Rr, where r defines the rotation in degree. The rotation r is constrained to be either $0°$, $90°$, or $270°$. So an

example for a resistor declaration would be *SYMBOL res 32 32 R90*, which means that a resistor is defined at $x = 32$, $y = 32$ with a rotation of $90°$.

Electrical Circuit Component	**LTspice keyword**
Resistor	res
Capacitor	cap
Inductor	ind
Diode	diode
Voltage Source	voltage
Current Source	current

Table 2.2: LTspice symbol names

	Keyword	**Param1**	**Param2**	**Param3**	**Param4**
Syntax	SYMBOL	symbol name	X-Coordinate	Y-Coordinate	Rotation
Info		see table 2.2	multiple of 32	multiple of 32	R0, R90, R270

Table 2.3: LTspice symbol syntax

Symbol Attributes

A symbol can be further specified through the usage of a symbol attribute. The symbol attribute is always applied to the first occurrence of a previously defined symbol relative to this command. The syntax for this command is presented in table 2.4. This command is declared using the keyword *SYMATTR* followed by the targeted attribute and the corresponding value to be set for the targeted attribute. Two attributes can be used as a target for this command. The *InstName* attribute allows declaring a name for the symbol, and the value therefore is a string. The *Value* attribute allows declaring a component value for the symbol always defined in the corresponding base unit of the component (e.g., Ω for resistors).

	Keyword	**Param1**	**Param2**
Syntax	SYMATTR	attribute	value
Info		Value, InstName	Integer, String

Table 2.4: LTspice symbol attribute syntax

Ground

Grounds are defined by using the keyword *FLAG* followed by the coordinates of the ground. An additional *0* has to be placed at the end of the line, which indicates that the used flag is indeed a ground node. Note that it was not further analyzed which effect the last parameter has on the flag definition, but it can be said that when the *0* is not present, the ground is not defined correctly. The syntax for grounds can be seen in table 2.5.

	Keyword	**Param1**	**Param2**	**Param3**
Syntax	FLAG	X-coordinate	Y-coordinate	flag indicator
Info		multiple of 32	multiple of 32	0 for ground

Table 2.5: LTspice ground syntax

Wire

After the symbols have been defined, they are connected through wires. Wires in LTspice are defined as lines. The syntax is presented in table 2.6. The command begins with the keyword *WIRE* followed by two coordinate pairs. The first pair is the beginning of the line, and the second one, the end of the line.

	Keyword	**Param1**	**Param2**	**Param3**	**Param4**
Syntax	WIRE	X1-coordinate	Y1-coordinate	X2-coordinate	Y2-coordinate
Info		multiple of 32	multiple of 32	multiple of 32	multiple of 32

Table 2.6: LTspice wire syntax

2.3 Artificial Neural Networks

2.3.1 General Concepts

The basic building blocks of an Artificial Neural Network (ANN) are Artificial Neurons, which are inspired by their biological counterparts [McC43]. The first artificial neuron, the perceptron, was proposed by Rosenblatt [Ros58]. The decision rule for the perceptron is described by eq. 2.1. It states that the output $\hat{y} \in \{-1, 1\}$ is defined by the sign of the dot product of a weight vector $\mathbf{w} \in \mathbb{R}^n$ with an input vector $\mathbf{x} \in \mathbb{R}^n$. The decision boundary of this function is a linear function,

which means non-linear functions like the XOR function ("exclusive or") cannot be solved with the perceptron.

$$\hat{y} = \text{sign}(\mathbf{w}^T\mathbf{x}) \tag{2.1}$$

Multilayer Perceptron

Since the perceptron is not able to solve non-linear problems. the MLP was introduced [Gro73] which makes that possible. Generally, an MLP consists of three layer types: one input layer, one or more hidden layers, and one output layer. The input layer is the identity function (eq. 2.2), which forwards the input $\mathbf{x} \in \mathbb{R}^n$ without change to the following hidden layer.

$$\mathbf{I}(\mathbf{x}) = \mathbf{x} \tag{2.2}$$

The hidden layer, which is also called a Fully-Connected (FC) layer [Mai20] is built out of one to $m \in \mathbb{N}$ perceptrons. The output vector $\hat{\mathbf{y}} \in \mathbb{R}^m$ is calculated by multiplying $\mathbf{x}$ with each weight vector $\mathbf{w} \in \mathbb{R}^n$ and adding a constant bias vector $\mathbf{b} \in \mathbb{R}^m$ to the calculation [Goo16].

$$\hat{\mathbf{y}} = (\mathbf{w_1}^T\mathbf{x}, \cdots, \mathbf{w_m}^T\mathbf{x})^T + \mathbf{b} \tag{2.3}$$

The calculation can further be simplified by using a weight matrix, which combines all weights $\mathbf{w}$ and the bias $\mathbf{b}$. The output $\hat{\mathbf{y}}$ is then the matrix multiplication of the weight matrix $W \in \mathbb{R}^{(n+1)\times m}$ with the input vector $\mathbf{x}$. The dimensionality of $\mathbf{x}$ has to be extended with a *1* at the end hence results in $\mathbf{x} \in \mathbb{R}^{(n+1)}$ [Mai20]. The current output vector $\hat{\mathbf{y}}$ is just a linear transformation of the input vector $\mathbf{x}$, but biological neurons are also able to process a received signal non-linearly [Goo16], therefore activation functions $f_a : \mathbb{R}^n \to \mathbb{R}^n$ are used to mimic this behavior. Some commonly used activation functions and specifically the ones used in this thesis, are presented in section 2.3.2. Combining the activation function and the matrix multiplication results in the following formula:

$$\hat{\mathbf{y}} = f_a(\mathbf{W}^T\mathbf{x}) \tag{2.4}$$

The last missing layer type is the output layer, which in a classification setting outputs a vector of conditional class probabilities $\hat{\mathbf{y}} \in \mathbb{R}^c$, where c is the number of classes. Commonly the softmax activation function is used to produce such an output [Bri90]. The softmax function takes as input the output of a previous layer and outputs a vector of pseudo probabilities. The equation 2.5 defines the i-th element of the output vector, where C is the length of the input $\mathbf{x}$, which is

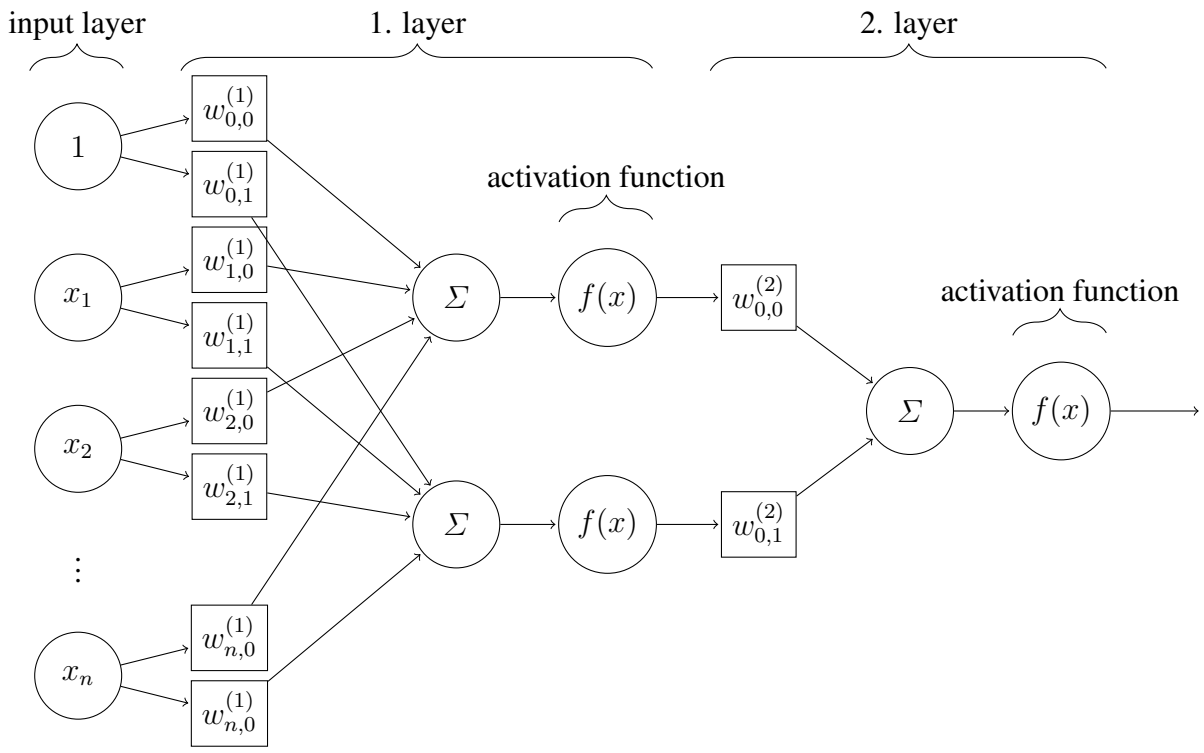


Figure 2.2: A MLP with two hidden layers (two neurons in the first and one in the last), each input gets multiplied with each weight of each neuron, summed up and fed into an activation function to produce the input for the next layer, where this process is repeated.

also the number of classes to predict.

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \tag{2.5}$$

Since the output of one layer is the input for the following layer a MLP can be mathematically described in a chain like function structure as $f^{(O)}(f^{(h)}(\cdots f^{(1)}(f^{(I)}(x))))$, where $f^{(I)}$ is the input layer, $f^{(O)}$ the output layer and $f^{(1)} \cdots f^{(h)}$ are the amount of h hidden layers, respectively. Note that in practive several different functions can be defined as a layer.

Learning Procedure

In the first step, the input data is fed to the network producing an output $\hat{\mathbf{y}}$. This step is called forward propagation and is just the above-described method of calculating the output of a layer and using it as the input for the next one.

The output $\hat{\mathbf{y}} \in \mathbb{R}^c$ is compared to the desired output $\mathbf{y} \in \mathbb{R}^c$, using a loss function $L(\mathbf{y}, \hat{\mathbf{y}}) : (\mathbb{R}^c, \mathbb{R}^c) \to \mathbb{R}$. The labels $\mathbf{y}$ normally are one-hot encoded, which means that a one is put at the

index of the respective class and the other elements (classes) are set to zero in the label vector [Rod18].

A common loss function is the Mean Squared Error (MSE) [Red15], which calculates the mean sum of the squared differences between the inputs $\mathbf{y}$ and $\hat{\mathbf{y}}$ (equation 2.6).

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{C} \sum_{i=1}^{C} (y_i - \hat{y}_i)^2 \tag{2.6}$$

Another common example of a loss function would be the Cross Entropy (CE) loss (equation 2.7), which measures the difference between two probability distributions for a given random variable or a set of events [Jad20].

$$\text{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{C} y_i \cdot log(\hat{y_i}) \tag{2.7}$$

The resulting loss value is used to calculate the gradients with respect to the weights in the last layer. Due to the above described chained function structure of an MLP the chain rule can be used to propagate the error back through the network and calculate the gradients for all remaining layers.

After all gradients have been calculated the weights of each layer are optimized. A simple optimizer is the stochastic gradient descent, whose update rule is defined as:

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} - \eta \nabla L(\mathbf{W}^{(k)}) \tag{2.8}$$

Here, k denotes the current time step, $\mathbf{W}^{(k+1)}$ the updated weights, and $\mathbf{W}^{(k)}$ the current state of the weights. The variable η is the learning rate of the network, which scales the amount of gradient that is used to update the weights. Typically an $0 < \eta < 1$ is chosen since big values have shown to destabilize the training process, resulting in divergence of the loss. Further, $\nabla L(\mathbf{W}^{(k)})$ denotes here the gradients with respect to the loss function at the respective layer.

To accelerate the training the stochastic gradient descent can be extended by a momentum term [Ram21]. The momentum term and the resulting update rule are defined as follows:

$$\mathbf{V}^{(k)} = \mu \mathbf{V}^{(k-1)} - \eta \nabla L(\mathbf{W}^{(k)}) \tag{2.9}$$

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} + \mathbf{V}^{(k)} \tag{2.10}$$

The variable μ denotes here the momentum value, which is typically set to 0.9, 0.99 respec-

tively [Kin17]. The idea is that one can incorporate the weighted value of the previous update to accelerate in directions with persistent gradient [Mai20].

2.3.2 Activation Functions

Non-linear activation functions play a crucial role in the performance of an ANN, since this enables universal function approximation [Dig19]. Where universal function approximation means that any arbitrary function can be approximated. In the perceptron the *sign* function was used, but due to its non-differentiable property, it is not suited for the use in ANNs, since backpropagation requires differentiability of the activation function. Therefore, various differentiable activation functions have been introduced.

Sigmoid

The sigmoid activation function (eq. 2.11) is a smooth differentiable activation function, which maps its input to a $\{0, 1\}$-space. It is commonly used in output layers since the output of the sigmoid can be interpreted as a probability. A major drawback of the sigmoid lies in the saturating property for $\mathbf{x} \to \pm\infty$, when it's used as an activation function in trainable layers of ANNs. Due to this the training process suffers from the so called vanishing gradient problem, where the derivative of the sigmoid tends to go towards zero and hence does not provide an update to the weights of the ANN.

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \tag{2.11}$$

Rectified Linear Unit (ReLU)

The Rectified Linear Unit (ReLU) activation function solves the gradient vanishing problem by introducing a linear term for input values $\mathbf{x} > 0$, while maintaining the non-linearity property by setting all negative input values to 0. The ReLU activation function is defined as follows:

$$\text{ReLU}(\mathbf{x}) = \begin{cases} \mathbf{x}, & \text{if } \mathbf{x} > 0 \\ 0, & \text{else} \end{cases} \tag{2.12}$$

A variation of the ReLU activation function is the Leaky ReLU (LReLU) activation function, where additionally negative values are scaled linearly. LReLU was introduced to tackle the dying ReLU problem, where the network only predicts negative values and hence all gradients become zero [Mai20]. It is defined as follows:

$$\text{LReLU}(\mathbf{x}) = \begin{cases} \mathbf{x}, & \text{if } \mathbf{x} > 0 \\ \alpha\mathbf{x}, & \text{else} \end{cases} \tag{2.13}$$

ReLU6

ReLU6 is a modification of the classical ReLU, where the output activation is limited to 6. It is often used in resource constrained environments, especially in cases of low precision computation, since it has shown to perform robust in such cases [How17].

$$\text{ReLU6}(\mathbf{x}) = \begin{cases} 6, & \text{if } 6 <= \mathbf{x} \\ \mathbf{x}, & \text{if } 0 < \mathbf{x} < 6 \\ 0, & \text{else} \end{cases} \tag{2.14}$$

2.3.3 Convolutional Neural Networks

While MLPs perform well on vectorial data, multidimensional data like images for example, can only be fed to a network when it was previously flattened into a vector. One problem that arises is that when flattening for example an image of size 100×100, this would already require the input size of the network to have $10,000$ weights per neuron in the next layer. Additionally nearby pixels in images are often highly correlated and classical unstructured ANN fails to capture such spatial dependencies [LeC99].

The proposed alternative therefor are Convolutional Neural Networks (CNNs), which have shown to perform well over the last decade in several image related benchmarks [Sze14] [He15] [Hua16]. The classical CNN architecture is comprised of three different layer types: convolutional layers, pooling layers and fully-connected layers.

Convolutional Layer

Convolutional layers form the major component in a CNN. As the name suggest the underlying mathematical foundation of those layers is the convolution operation. The equation for a convolution in continuous space (equation 2.15) states that a function f convolved with another function g, is the multiplication of those two functions, while g is shifted over f. The final result is then obtained by taking the integral over the whole domain [Mai20].

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau \tag{2.15}$$

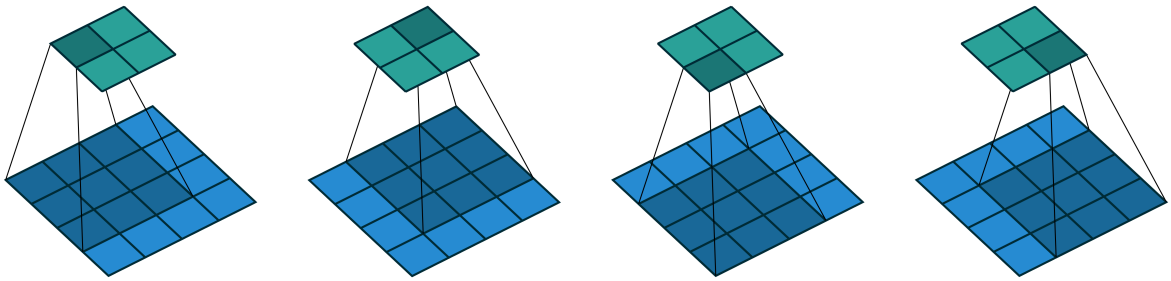


Figure 2.3: Example convolution of a 4x4 input (blue) with a 3x3 kernel (dark blue) and a stride of 1, resulting in a 2x2 output (cyan) [Dum16]

In deep learning the concept of convolutions is applied to an image $I \in \mathbb{R}^{W \times H \times C}$, where W and H are the width and height of the image and C being the number of channels. Further, there is a kernel $K \in \mathbb{R}^{k_w \times k_h \times C}$. The image and the kernel are now convolved by moving the kernel over the image and at each position an element-wise multiplication of the overlapping area of the image and the kernel is computed. The result is summed up and used as an output element in the convolution result. Finally, the kernel is shifted further until the whole image has been processed. How much pixel a kernel is shifted at a time depends on the used stride. The higher the stride the less local information is preserved. Typically a stride of 1 or 2 is used. An example of a convolution of a 4×4 input with a 3×3 kernel and a stride of 1 is given in figure 2.3.

Pooling Layer

Pooling layers in CNNs are used to further reduce the dimensionality of the output. During the pooling operation information across spatial locations is fused by sliding a window (typically of size 2×2 or 3×3) over the input and performing a function on the values inside the window. In the case of max pooling the used function is the *maximum* function, therefore only the maximum value inside the window is considered and used in the pooling output. This decreases the number of parameters and hence reduces the computational cost [Mai20].

Fully-Connected Layer

The FC layer as described in 2.3.1 is the final layer inside a CNN. The high-level features, which were previously build through the convolutional and max pooling layers, are flattened into a fixed size vector and passed for classification to the FC layer.

2.3.4 Batch Normalization

A common problem while training with the ReLU activation function is that the outputs are non-zero centered, since ReLU is non-zero centered. So with each layer the output distribution gets shifted from the input distribution. This observed phenomenon is called the *internal co-variate shift* and forces the network to adapt to the shifting output distribution. The adaption produces an overhead, which is expressed in a longer training time of the network. Therefore, Ioffe and Szegedy [Iof15] introduced Batch Normalization (BatchNorm), a trainable layer for ANNs. The BatchNorm layer normalizes the feature maps along the batch axis to have zero-mean and a standard deviation of one, which not only decreases the training time, but also allows for higher learning rates and less careful initialization of the network weights.

2.4 Data Augmentation

CNNs require a large amount of data to learn the desired objective and to prevent the problem of overfitting, where a network perfectly learns the underlying data and is not able to generalize over unseen data [Sho19]. Augmentation is a way to artificially increase the amount of data in terms of size and diversity. In the image domain for example, images are normally transformed in various ways, like changing color parameters, rotating the image, or cropping parts of the image. Other successful augmentation techniques like the copy-paste augmentation [Ghi20] exists, which takes a segmentation mask of an object and projects it on another background. The copy-paste augmentation is highly used in this thesis, foremost to increase the number of ECDs with a checkered background. In this thesis images were augmented using the albumentations library [Bus20], since it includes augmentations for plane images, as well as bounding boxes and segmentation masks. Due to its convenient interface, which can easily be incorporated into different deep learning frameworks like PyTorch [Pas19] or TensorFlow [Aba15], it is highly recommended. The different augmentations used from albumentations in this thesis are listed in table 2.7.

Normally augmentations are only applied to the training set - before or during training - but they can also benefit the predictions at test time [Mos20][Sha20]. Such augmentations are then referred to as Test-Time Augmentations (TTAs). Common augmentations used for TTA are flip, rotate, color jitters and crop [Sha20], images augmented with those augmentations are next to the original image given to the network as a separate input. The augmented predictions are "deaugmented", e.g. by again flipping the prediction in the same direction or rotating it in the opposite direction the same amount, in which the image was previously rotated. Afterwards, an

	Object Detection	Segmentation
ColorJitter	✓	✓
RandomCrop		✓
RandomBBoxSafeCrop	✓	
Rotation	✓	✓
Random Scale	✓	✓

Table 2.7: A listing of the different augmentations used from albumentations [Bus20] in this thesis and the target domain where they were applied to.

average, or weighted average for example is taken over the predictions to form the final prediction.

2.5 Object Detection

Object detection is one of the subtasks in the image domain. It is an extension of the classification task, where additionally to the predicted class, the location of the objects in the image should be predicted. The location is normally given as a bounding box, where throughout this thesis the bounding box format by Redmon et al. [Red15] is used to label object instances. In this format a bounding box is defined as:

$$bbox = (x_{rel}, y_{rel}, w_{rel}, h_{rel}) \tag{2.16}$$

Where the bounding box is presented as a tuple of four values. The first two values indicate the center of the bounding box relative to the image size, while the latter two values indicate the width and the height of the bounding box also relative to the image size. All values can be calculated by dividing the absolute value by the corresponding image value. For example x_{rel} would be calculated by dividing the absolute x-coordinate x_{abs} by the width of the image. Same applies for y_{rel}, except here the value is divided by the image height. w_{rel} and h_{rel} are calculated following the same principles. The advantages of this format are, that the definition of the bounding box becomes invariant to the image size. The image can be resized without having to recalculate the bounding box, as it is the case with other formats which use an absolute definition for bounding boxes.

2.5.1 History of Object Detection

Sliding Window

The simplest algorithm to detect objects in an image is the sliding window approach. A classifier is trained on image patches which contain the object to detect. To now detect the objects in an unseen image, the image is divided into patches of various scale and fed to the classifier. The prediction score of the patches is thresholded with a predefined value. High confidence patches are likely to contain an object and are kept [Pri20].

Regions with CNN Features (R-CNN)

While the sliding window approach is effective, it is also highly inefficient, since every generated patch has to be processed by the classifier in order to find all possible objects in an image. Regions with CNN features (R-CNN) by Girshick et al. [Gir13] improves on that by using a region proposal algorithm to obtain probable regions of an object. In their work the Selective Search algorithm [vdS11] was used to generate region proposals. How the algorithm performs and what kind of bounding boxes are produced, can be seen in fig. 2.4. 2000 of those proposed bounding boxes are taken from different scales and warped into the input shape of the following CNN, disregarding the size or aspect ratio of the proposed bounding box. Each of the bounding boxes is separately passed through the CNN and yields a feature vector which is classified by $N + 1$ binary SVMs (N classes $+1$ general background class) to produce the class prediction. Further, the bounding box is improved through N separately trained bounding box regressor SVMs [Fel10].

Fast R-CNN

While R-CNN was an improvement to the previous methods, the training and inference process was still very expensive, since it involved multiple stages. With Fast R-CNN Girshick [Ger15] improved on the training and inference time by a large margin in contrast to R-CNN.

The main difference to R-CNN is that instead of generating region proposals and passing them through the CNN, region proposals are now projected onto the feature maps of the CNN and pooled into a fixed size grid through a Region of Interest (RoI) pooling layer. Meaning that the CNN computations are shared between each bounding box proposal resulting in a drastic inference speed improvement. After pooling, the RoI it is processed by a fully-connected layer, producing a so called RoI feature vector. This feature vector is fed into a siblings output layer. The first branch is a classification layer with a softmax output, producing class probabilities and

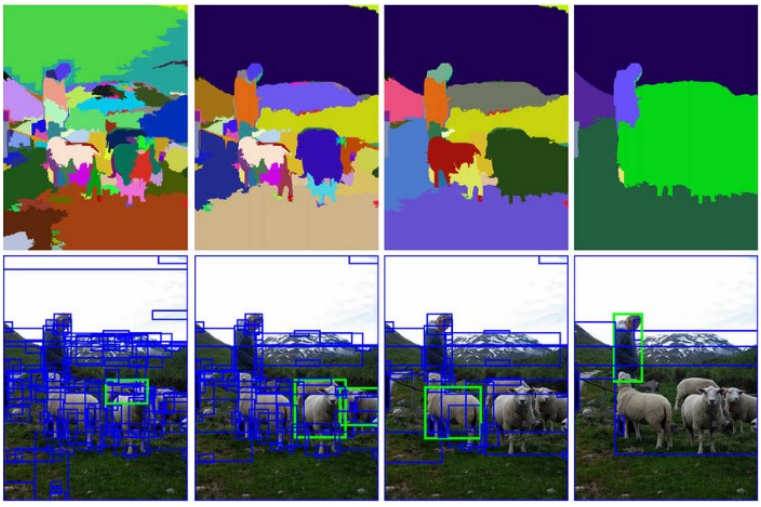


Figure 2.4: Example of results obtained through the Selective Search algorithm (top) with increasing region scale from left to right and bounding boxes drawn around those regions (bottom). Selective Search produces sub-segmentations of objects in an image, considering size, color, texture and shape based features for the grouping of the regions [vdS11].

replaces the previous SVMs. The second branch is comprised of a bounding box regression layer, which outputs bounding box regression offsets as in R-CNN [Gir13].

Anchor Box Based Single Shot Detectors

Various object detection networks such as, Faster R-CNN [Ren15], YOLO [Red15], Single Shot Multibox Detector (SSD) [Liu15] and RetinaNet [Lin17b] use anchor boxes to predict the objects in an image. Further, all the named methods predict the class as well as the bounding boxes for the objects in a single network pass, therefore the name single shot. Anchor boxes are predefined boxes of various size aspect ratio, which are used as a base for the prediction bounding box prediction of the above networks. Instead of making the network predict the bounding box directly, the network predicts bounding box regression offsets based on a certain anchor box [Red18]. The selection of good anchor box sizes is a hyperparameter in the training of an object detector and can improve the prediction quality [Ren15]. The scale and aspect ratio of the anchor boxes is often selected by using the k-means clustering algorithm on the labeled bounding boxes of the dataset [Red16].

Anchor Box Free Single Shot Detectors

Another class of single shot detectors are the anchor box free detectors such as You Only Look Once Version 1 (YOLOv1) [Red15], CornerNet [Law18], CenterNet [Kai19] and the Fully Convolutional One Stage Detector (FCOS) [Tia20]. The advantages of anchor box free detection methods are that complicated computation related to anchor boxes, e.g. calculating the Intersection over Union (IoU) at training time, can be omitted [Tia20], as well as the tuning of anchor box sizes for the specific task [Kai19]. As of now, all the above methods perform worse than their current state-of-the-art anchor box based counterparts [Boc20].

2.5.2 Intersection Over Union (IoU)

The IoU, which is also known as the Jaccard index, is a measure for how much two arbitrary shapes (volumes) are overlapping [Rez19]. In object detection IoU is often used to compare two bounding boxes A and B and also to construct various loss functions as well as metrics.

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2.17}$$

2.5.3 IoU Based Loss Functions

The MSE has shown to perform not well for the task of bounding box regression, because it assumes that the regressed variables (x, y, w, h) are independent of each other and can be optimized separately [Yu16].

To take the correlation of those variables into account, Yu et al. [Yu16] proposed the IoU loss (equation 2.18). While this was a major improvement to previously known methods the IoU loss still suffers from slow convergence and from the gradient vanishing problem, which occurs when the two bounding boxes A and B have no intersection.

$$\text{L}_{\text{IoU}}(A, B) = 1 - \text{IoU}(A, B) \tag{2.18}$$

Further, to solve these drawbacks Rezatofighi et al. [Rez19] proposed the Generalized IoU (GIoU) loss (equation 2.19). Their loss introduces an additional penalty term added to the IoU loss. Here, C is the smallest convex box enclosing A and B. Hence, when the boxes have no overlap there is still a gradient pushing them closer to each other. While the GIoU loss is a major improvement in terms of vanishing gradient, it suffers from slow convergence when A and B have overlap and at the same time A contains B (or vice versa), because the penalty term then becomes

0, as a consequence the GIoU loss becomes the IoU loss. Furthermore, it has been observed that when A and B have no overlap, instead of decreasing the spatial distance between A and B, the GIoU loss tends to increase the size of the bounding box area to reduce the loss [Zha21].

$$\mathrm{L}_{\mathrm{GIoU}}(A, B) = 1 - \mathrm{IoU}(A, B) + \frac{|C - (A \cup B)|}{|C|} \tag{2.19}$$

The next improvement in the IoU based loss function space was proposed by Zheng et al. [Zhe19], with their Distance IoU (DIoU) and Complete IoU (CIoU) loss functions. In contrast to GIoU, DIoU (equation 2.20) solves the gradient vanishing problem by considering the normalized distance of the central points of the two bounding boxes. The squared euclidean distance is normalized by the squared diagonal length of the smallest box containing A and B.

$$\mathrm{L}_{\mathrm{DIoU}}(A, B) = 1 - \mathrm{IoU}(A, B) + \frac{\|(A_{center} - B_{center})\|^2}{\|C_{diag}\|^2} \tag{2.20}$$

To further improve on that, the authors additionally considered the aspect ratio of the bounding box to be another important geometric factor for bounding box regression. Hence, the DIoU loss is further extended by a penalty term considering the aspect ratio and resulting in the improved CIoU loss (equations 2.21, 2.22, 2.23).

The penalty in CIoU is split into α and ν. The term α is a trade-off parameter which gives higher priority to the overlapping factor, especially in the case of non-overlap. Further, ν is the parameter penalizing the difference in the aspect ratios of A and B. Still, it can be noticed that ν is 0 when the aspect ratios are the same, regardless of the underlying relations between A_w, B_w and A_h, B_h. E.g. the aspect ratio is the same for all boxes with the following property $\{(A_w = kB_w,\ A_h = kB_h) \mid k \in \mathbb{R}^+\}$ [Zha21].

$$\mathrm{L}_{\mathrm{CIoU}}(A, B) = \mathrm{DIoU}(A, B) + \alpha(A, B) \cdot \nu(A, B) \tag{2.21}$$

$$\nu(A, B) = \frac{4}{\pi^2}\left[arctan\left(\frac{A_w}{A_h}\right) - arctan\left(\frac{B_w}{B_h}\right)\right]^2 \tag{2.22}$$

$$\alpha(A, B) = \frac{\nu}{1 - IoU(A, B) + \nu'} \tag{2.23}$$

Therefore, Zhang et al. [Zha21] proposed the Efficient IoU (EIoU) loss to remove this error. The aspect ratio penalty is here replaced through two separate penalties, which consider the normalized width and height of the two bounding boxes.

$$\mathrm{L}_{\mathrm{EIoU}}(A,B) = 1 - IoU(A,B) + \frac{\|(A_{center} - B_{center})\|^2}{\|C_{diag}\|^2} + \frac{\|(A_w - B_w)\|^2}{\|C_w\|^2} + \frac{\|(A_h - B_h)\|^2}{\|C_h\|^2} \tag{2.24}$$

2.6 You Only Look Once (YOLO)

In this thesis the single shot detector YOLO [Red15] is used. Single shot means that YOLO produces class predictions as well as bounding box predictions in a single network pass. In contrast to the other methods mentioned in 2.5.1, this yields a huge improvement in performance and resource efficiency. The YOLO network, more precisely YOLOv4 [Boc20] was selected, because it is currently the state-of-the-art in the commonly known object detection benchmarks. Furthermore, there exists a version of YOLOv4 (YOLOv4-tiny [Wan21]), which can be used in resource constrained environments such as mobile devices.

General

YOLOv4 unifies class prediction and bounding box regression in a single neural network. It is an anchor-based detection network and hence outputs regression offsets based on a predefined anchor box. The architecture can be structured into three main parts: a backbone, a neck and a head. The backbone receives an image as input and is responsible to extract features out of it. Further, the neck receives the output of the backbone and produces output feature maps, which are further processed by the head to produce the final prediction. The final prediction is then a vector with bounding box regression offsets for the anchors, an objectness score, which class-agnostically indicates whether an object is present, as well as a vector of independent class probabilities.

Backbone

The backbone of the YOLOv4 network is mostly formed out of YOLOConv layers (table 2.8), which is sequentially build out of a convolutional layer, a batch normalization layer and a leaky ReLU activation function.

The initial two layers (c0, c1) use an increased stride and a valid padding to first reduce the input size of the image. The following layers always use three 3×3 convolutions, followed by one 1×1 convolution and a max pooling layer to half the feature map size. The number of

Sequence	Parameter
Convolution	see table 2.9; l2 kernel regularization $5x10^{-3}$
BatchNorm	
LeakyReLU	$\alpha = 0.1$

Table 2.8: Convolutional basic building block in YOLOv4 (YOLOConv). A convolutional layer, followed by a batch normalization layer, followed by a LeakyReLU activation function.

filters for the four convolutions is always $B \cdot (K, \frac{K}{2}, \frac{K}{2}, K)$, where B denotes the block number and K the number of filters (kernels) beginning with 64. So as can be seen in table 2.9, where the whole architecture of the backbone is presented, in the first layer the kernel sizes are set to $(64, 32, 32, 64)$. Further, the backbone has three outputs, seen on the right of the table ($\mathbf{Skip_S}$, $\mathbf{Skip_M}$, $\mathbf{Skip_L}$), which are used as skip connections to the following neck network. S, M, L indicates here the size of the detected object (small, medium, large).

Neck

The neck of the YOLOv4 network is a scaled version of the Path Aggregation Network (PANet) [Liu18] and is further referred to as PANet-tiny [Wan21]. The PANet-tiny receives the outputs of the backbone as input and produces three output tensors, where each output tensor is a prediction for a particular scale of object. Each prediction output has twice the feature map size of the previous output, except for the most lowest output ($\mathbf{Pred_L}$), which has the size of the last backbone output block. Therefore, $\mathbf{Pred_L}$ has a size of $S \times S$, $\mathbf{Pred_M}$ has a size of $2S \times 2S$ and $\mathbf{Pred_S}$ has a size of $4S \times 4S$. The grid size $S \times S$ can be calculated by dividing the spatial input dimensions by 32, i.e. $S \times S = \frac{w}{32} \times \frac{h}{32}$, which also enforces input sizes to be divisible by 32. Generally, the PANet-tiny is built by using a separate output branch to produce a prediction, followed by an upsampling layer, where the feature map is bilinearly upsampled and convolved together with the skip connection from the backbone to serve again as input for the next output branch. Each output branch is a 1×1 convolution and hence acts as a fully-connected layer. The output size of each output layer is $3 \cdot (5 + C)$, where 3 is the number of anchor boxes in that particular scale, 5 is the number of bounding box parameters (4 regression offsets, 1 objectness score) and C is the number of classes. So the final output tensor of the network is $(2^{O-1} * S) \times (2^{O-1} * S) \times (3 * (5 + C))$.

Layer	Input	Type	Filters	Size	Stride	Padding	Output
Reduction Block							
c0	img	YOLOConv	32	3x3	2	valid	
c1	c0	YOLOConv	64	3x3	2	valid	
Block 1							
c2	c1	YOLOConv	64	3x3	1	same	
c3	c2	YOLOConv	32	3x3	1	same	
c4	c3	YOLOConv	32	3x3	1	same	
c5	c3 & c4	YOLOConv	64	1x1	1	same	
m0	c2 & c5	MaxPool		2x2	2	same	
Block 2							
c6	m0	YOLOConv	128	3x3	1	same	
c7	c6	YOLOConv	64	3x3	1	same	
c8	c7	YOLOConv	64	3x3	1	same	
c9	c7 & c8	YOLOConv	128	1x1	1	same	$\mathbf{Skip_S}$
m1	c6 & c9	MaxPool		2x2	2	same	
Block 3							
c10	m1	YOLOConv	256	3x3	1	same	
c11	c10	YOLOConv	128	3x3	1	same	
c12	c11	YOLOConv	128	3x3	1	same	
c13	c11 & c12	YOLOConv	256	1x1	1	same	$\mathbf{Skip_M}$
m2	c10 & c13	MaxPool		2x2	2	same	
Block 4							
c14	m2	YOLOConv	512	3x3	1	same	$\mathbf{Skip_L}$

Table 2.9: The CSPDarknet53Tiny Architecture is a scaled version of the original CSPDarknet53 architecture [Boc20]. The definition for YOLOConv can be found in table 2.8. MaxPool indicates here the Max Pooling operation as described in sec. 2.3.3. The network is build out of five blocks, the first block reducing the image size and the following blocks building up features. Further, features from three different scales are taken and used as skip connections to the following network.

Layer	Input	Type	Filters	Size	Stride	Padding	Output
c15	$\mathbf{Skip_L}$	YOLOConv	256	1x1	1	same	
Out 1							
c16	c15	YOLOConv	512	3x3	1	same	
c17	c16	YOLOConv	3 * (5 + C)	1x1	1	same	$\mathbf{Pred_L}$
Up 1							
c18	c15	YOLOConv	128	1x1	1	same	
u18	c18	UpBillinear		2x2			
Out 2							
c19	$\mathbf{Skip_M}$ & u18	YOLOConv	256	3x3	1	same	
c20	c19	YOLOConv	3 * (5 + C)	1x1	1	same	$\mathbf{Pred_M}$
Up 2							
c21	c19	YOLOConv	256	3x3	1	same	
u21	c21	UpBillinear		2x2			
Out 3							
c22	$\mathbf{Skip_S}$ & u21	YOLOConv	128	3x3	1	same	
c32	c22	YOLOConv	3 * (5 + C)	1x1	1	same	$\mathbf{Pred_S}$

Table 2.10: The PANetTiny architecture which is a scaled version of PANet [Liu18], which is the decoder in the YOLO network. It takes as inputs the skip connections from the CSPDarknet53Tiny. The network is build by alternating an output branch and an upsampling branch. First, the lowest skip connection $\mathbf{Skip_L}$ from the backbone is convolved and the output fed into the first output layer. Each output layer has again a 3×3 convolution followed by a 1×1 convolution, which form a raw bounding box prediction, fed to the YOLO head. After an output has been processed the previous convolution is again convolved and upsampled to be fed into the next output block. This is done three times in the whole PANetTiny network.

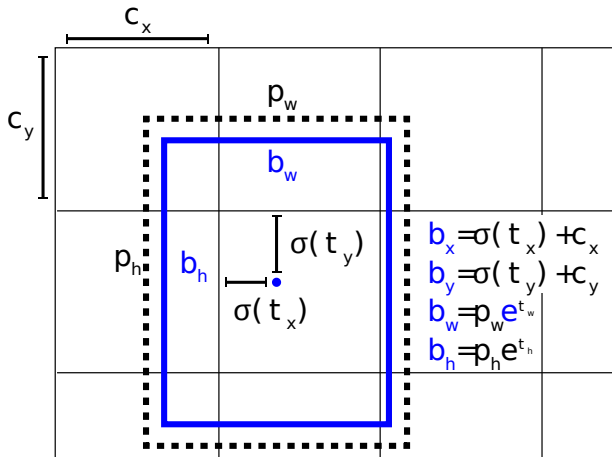


Figure 2.5: Bounding box calculation in the YOLOv4 network based on a prior anchor box [Red16]. The b_* indicate the final prediction, p_* indicate parameters of the prior bounding box and c_* indicate the prior bounding box spatial offset based on the current cell. The final center coordinate offset is predicted non-linearly with a sigmoid function, while the final width and height are predicted linearly, as it is done in [Ger15] and [Ren15].

Head

The last component in the YOLOv4 network architecture is the head, which is reused from YOLOv3 [Red18]. Each input scale from the neck ($\mathbf{Pred_L}$, $\mathbf{Pred_M}$, $\mathbf{Pred_S}$) is processed in the same manner. The input vector looks as follows:

$$\mathbf{Pred_X} = \{t_x, t_y, t_w, t_h, q_{conf}, q_{C1}, ..., q_{CC}\} \tag{2.25}$$

The final bounding box calculation as well as some visual information on how the final bounding box parameters are derived can be seen in fig. 2.5. The final spatial coordinate b_*, where $*$ denotes x and y respectively, is calculated by applying the sigmoid activation function on t_* and adding the grid cell offset c_* to the results. Further, the final size b_+, where $+$ denotes w and h respectively, is calculated by multiplying the prior assigned anchor box sizes p_+ with the exponential function applied to the predicted spatial offset t_+. Finally, the objectness score as well as the class probabilities are also calculated through the sigmoid activation function.

Loss

YOLO is trained end-to-end and requires to predict the class as well as the bounding boxes a multi-task loss. The loss can be separated into three main parts:

1. class loss
2. objectness loss
3. bounding box regression loss

In the following 1_{sb}^{obj} denotes that the anchor box b in grid cell s is responsible for detecting the object. Before training, responsibility is assigned by making that anchor box responsible which has the highest IoU with the ground truth bounding box. Furthermore, S^2 denotes the number of grid cells in a particular scale and B the number of anchor boxes present in a grid cell (three in YOLOv4).

The class loss (equation 2.26) is calculated by taking the sum of all possible anchor boxes and the sum of all independent class losses, which is calculated with the CE loss. An anchor box only contributes to the loss, when an object is labeled to be present in it.

$$\mathrm{L}_{\text{class}} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} \sum_{c \in classes} 1_{sb}^{obj} \cdot \mathrm{CE}(P_{sbc}, \hat{P}_{sbc}) \tag{2.26}$$

The bounding box regression loss (eq. 2.27) is calculated again as the sum of all possible anchor boxes, taken over an IoU based loss function denoted as XIoU (e.g. IoU, GIoU, DIoU, CIoU, EIoU). Additionally, the XIoU is multiplied with a scaling weight which enforces a multiplier based on the size of the ground truth bounding box, i.e. if the bounding box is small more weight is applied on the loss of that bounding box.

$$\mathrm{L}_{\text{IoU}} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} 1_{sb}^{obj} \cdot w_{scale} \cdot \mathrm{XIoU}(\mathbf{bbox}, \hat{\mathbf{b}}\mathbf{box}) \tag{2.27}$$

$$w_{scale} = 2 - w \cdot h \tag{2.28}$$

The last part of the multi-task loss is the objectness loss, where objectness is defined as the confidence of the network that an object is present in a grid cell or not. The objectness loss is split into two sub equations. The former (equation 2.29) defines the loss which occurs when an object is present and the latter (equation 2.30) when no object is present. Again, both are taken over the sum of all possible anchor boxes. L_{obj} takes the sum of all CE loss outputs, applied to 1_{sb}^{obj} and the predicted objectness score. Almost the same is done in $\mathrm{L}_{\text{noobj}}$, but instead here the inverse prediction score is used. Additionally, a constraint is introduced that makes the loss only contribute when the maximum IoU of the predicted bounding box $\hat{b}box_{sb}$ and any ground truth bounding box $bbox$ is below a certain threshold t_{ignore}. This does not penalize predictions which

have a high IoU, but should normally not be present, i.e. another anchor box is assigned as a predictor. For example this can happen when during prediction responsibility assignment two anchor boxes had a similar IoU with the ground truth bounding box, the network hence tries to predict both of them.

$$\mathrm{L}_{\text{obj}} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} \mathrm{CE}(1_{sb}^{obj}, \hat{P}_{sb}) \tag{2.29}$$

$$\mathrm{L}_{\text{noobj}} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} \mathrm{CE}(1_{sb}^{noobj}, 1 - \hat{P}_{sb}) \cdot \{\max(\mathrm{IoU}(\forall \mathbf{bbox}, \hat{\mathbf{bbox}}_{\mathbf{sb}})) < t_{ignore}\} \tag{2.30}$$

Finally, each of the above losses is multiplied with a tunable hyperparameter and summed up to form the final YOLO loss.

$$\mathrm{L}_{\text{YOLO}} = \lambda_{class} * L_{class} + \lambda_{IoU} * L_{IoU} + \lambda_{objectness} * (L_{obj} + L_{noobj}) \tag{2.31}$$

Non-Maximum Suppression (NMS)

During prediction time often multiple bounding boxes are predicted for one object. To suppress unnecessary bounding boxes, a Non-Maximum Suppression (NMS) algorithm is applied. More precisely DIoU-NMS [Zhe19] is used during training time, which has shown to perform better than classical NMS [Bod17], especially in cases of occlusion.

In the following the algorithm for DIoU-NMS is presented. The input is a set of bounding boxes $\mathbf{B}$, also called candidates, as well as the corresponding prediction scores $\mathbf{S}$ and a suppression threshold ϵ. The algorithm first selects the bounding box $\mathbf{b_m}$ with the highest score s_m and compares it to every other bounding box $\mathbf{b_i}$ in the candidate set $\mathbf{B}$. If the DIoU measure between $\mathbf{b_m}$ and $\mathbf{b_i}$ is above the threshold ϵ, the bounding box $\mathbf{b_i}$ gets suppressed by removing it from the set $\mathbf{B}$. This process is repeated until no bounding boxes are present in $\mathbf{B}$.

Algorithm 2.1: DIoU-NMS

Input: B = b_1,...,b_N (list of bounding box proposals)
S = s_1,...,s_N (list of bounding box scores)
ϵ (NMS threshold)
Output: $B_S = \{b_1, ..., b_M\}$ (list of bounding boxes after NMS)
$S_S = \{s_1, ..., s_M\}$ (list of bounding box scores after NMS)

```
B_S ← {};
S_S ← {};
while B ≠ empty do
    m ← argmax(S)
    B_S ← B_S ∪ B_m; S_S ← S_S ∪ S_m;
    for b_i, s_i in B, S do
        if DIoU(B_m, b_i) ≥ ε then
            B ← B - b_i; S ← S - s_i;
        end
    end
    return B_S, S_S
end
```

DIoU-NMS is used as the default baseline. Later it will be shown that some predicted classes suffer from multiple bounding boxes, which are not suppressed by DIoU-NMS, but a combination of that bounding boxes would lead to a superior prediction. Therefore, experiments are also evaluated using the Weighted Bounding Box Fusion (WBF) [Sol19] algorithm, which could help increase the overall prediction quality. The algorithm is described in the following.:

1. Each predicted bounding box is added to a single list $\mathbf{B}$ if the confidence score of that bounding box is above a threshold $\boldsymbol{\sigma}$. Afterwards, the list is sorted in decreasing order of the confidence scores $\mathbf{C}$.

2. Declare empty list $\mathbf{L}$, which will hold clusters of bounding boxes from $\mathbf{B}$. Populate it by comparing each bounding box in $\mathbf{B}$, with each other bounding box in $\mathbf{B}$ and add it to $\mathbf{L}$ if the compared boxes have the same class label and the IoU of both boxes is above a threshold $\boldsymbol{\epsilon}$. If no match can be found for a bounding box add it to $\mathbf{L}$ as a cluster of size 1.

3. Declare empty list $\mathbf{F}$, which will contain bounding boxes, which are fused from a cluster in $\mathbf{L}$. Populate $\mathbf{F}$ by recalculating the bounding box parameters of each cluster in $\mathbf{L}$ with:

$$C = \frac{\sum_{i=1}^{T} C_i}{T} \tag{2.32}$$

$$X_{1,2} = \frac{\sum_{i=1}^{T} C_i \cdot X_{1,2}}{\sum_{i=1}^{T} C_i} \tag{2.33}$$

$$Y_{1,2} = \frac{\sum_{i=1}^{T} C_i \cdot Y_{1,2}}{\sum_{i=1}^{T} C_i} \tag{2.34}$$

4. **Optional.** When bounding boxes from more than one model are used, which is the case when TTA is used, or an ensemble of multiple predictors, then the confidence score of the fused bounding box should be adapted to incorporate the uncertainty of multiple models. For example when one model predicts a bounding box, but another model does not. Therefore, the confidence is rescaled with:

$$C = C \cdot \frac{\min(T, N_P)}{N_P} \tag{2.35}$$

or

$$C = C \cdot \frac{T}{N_P} \tag{2.36}$$

Here, T is the number of bounding boxes in a cluster and N_P is the number of predictors, which have predicted a bounding box in that particular cluster. The experiments in this thesis utilize equation 2.35, because there the confidence can not be greater than one. In equation 2.36 the confidence can exceed one, when the number of predicted bounding boxes is higher than the number of predictors, which is often the case.

2.7 Segmentation

Segmentation is another subtask in the image domain. The target here is to obtain a mask of an object or objects in an image. It is related to the classical classification problem, but instead of predicting the class for a whole image, the class is here predicted for each pixel individually. Segmentation can be further divided into semantic segmentation [Net21] and instance segmentation [He17]. In semantic segmentation the type of an object in general is predicted for example cat or dog. In instance segmentation further the instance of an object is

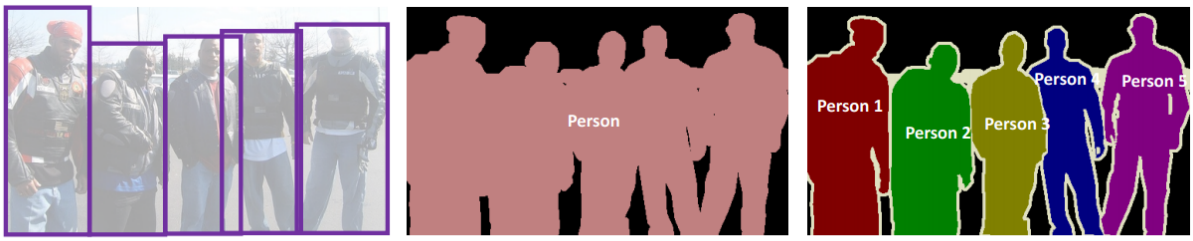


Figure 2.6: The difference between object detection, semantic segmentation and instance segmentation. In object detection the instance with a rough estimate (bounding box) is predicted, in semantic segmentation a segmentation mask for an object is predicted without considering the underlying instance and in instance segmentation the instance as well as a segmentation mask for an object is predicted [Liu].

predicted so, e.g. when two cats are present two separate masks would be predicted. Figure 2.6 illustrates the two different types of segmentation.

2.8 MobileNetV2-UNet

For the segmentation of the circuits MUnet [Jin20] is used. This network is build upon the principles of the U-Net proposed by Ronneberger et al. [Ron15]. The classical U-Net architecture consists of two main components: an encoder and a decoder. The encoder is a classical CNN which learns the features of the provided data. It uses convolutional layers and max pooling layers to downsample the feature map size. The decoder does the inverse, here the feature maps are convolved and then upsampled. Which made the U-Net unique at the proposed time is that to enhance the segmentation results, additionally after feature maps were upsampled they get concatenated with feature maps from the backbone which have the same resolution. It should be noted that this approach was probably picked up by the YOLOv4 developers and reused in their architecture shown in section 2.6.

MUnet is a combination of the MobileNetV2 [San19] which is used as the backbone and a decoder which is adapted to the backbone. In the following section the architecture of the MUnet is explained.

MobileNetV2-UNet backbone

MobileNetV2 is the backbone of the MUnet and can be decomposed into two main components:

Input	Operator	Output
$h \times w \times k$	1×1 conv2d, BatchNorm, ReLU6	$h \times w \times (tk)$
$h \times w \times (tk)$	3×3 dwise s=s, BatchNorm, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times (tk)$	1×1 conv2d, BatchNorm	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 2.11: An inverted residual block transforming from k to k' channels, with stride s and expansion factor t.

- depthwise separable convolutions
- inverted residual blocks

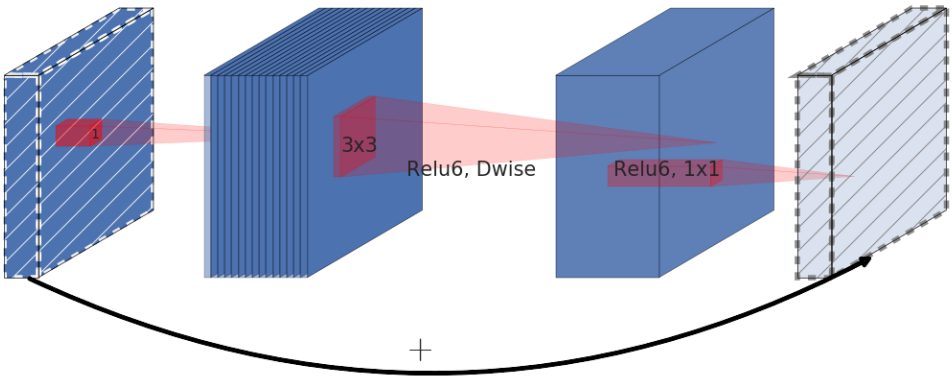
Depthwise separable convolutions were already used in the first MobileNet and are a way to factorize a standard convolution into a depthwise convolution and a 1×1 convolution. Essentially this means that a classical convolution is split into two layers, in the depthwise convolutional layer first a convolution is applied on each channel separately, i.e. given an input $I \in \mathbb{R}^{H \times W \times C}$ and C kernels $K^{h \times w \times 1}$ each channel C_i is convolved with a corresponding kernel K_i. Afterwards, to build up features lightweight 1×1 convolutions are used. This method has shown to perform almost on par with a classical convolution, but reduces the amount of computation by a factor of eight [How17]. Hence, this method is perfectly suited for resource constrained environments such as mobile phones.

The inverted residual layer is a novel layer in MobileNetV2. The idea is to first project the input feature maps into a lower dimensional subspace by using a 1×1 convolution with ReLU6 non-linearity, the resulting feature maps are then expanded inside the block by a following depthwise separable convolution. The output is again convolved with 1×1 convolution. The inverted residual has a residual connection where the input is added to the output of the last linear 1×1 convolution. The whole inverted residual block can be seen in figure 2.7. It also shows that the input gets expanded inside the inverted residual block. The expansion factor $t \in \mathbb{N}$ defines how much the input is expanded inside the inverted residual block. The expansion can be seen in table 2.11.

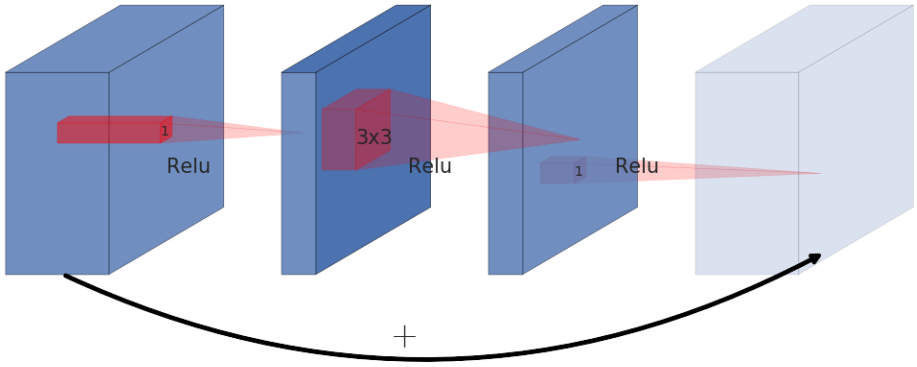
The whole MobileNetV2 architecture is given in table 2.12.

MobileNetV2-UNet decoder

The decoder of MUnet is build by the principles of U-Net [Ron15]. Feature map upsampling is done using transposed convolutions. After a feature map has been upsampled it is concatenated with a skip connection from the backbone with the same spatial dimension. Concatenation is



(a) Inverted Residual



(b) Traditional Residual

Figure 2.7: (a) The inverted residual block. A 1×1 convolution with non-linear activation followed by a depthwise separable convolution and a linear 1×1 convolution with residual connection to the input. The residual connection is here additive, i.e. the input gets added to the output of the linear 1×1 convolution. It is called inverted, due to the expansion inside the block, while in the traditional residual block (b) the input gets expanded when it leaves the block [San19].

Blocks	Operation	t	k	n	s	Output (h, w, c)
Input	-	-	-	-	-	(448, 448, 3)
	Conv+BN+ReLU6	-	3	1	2	(224, 244, 32)
$Skip_1$	InvRes	1	-	1	1	(224, 224, 16)
$Skip_2$	InvRes	6	-	2	2	(112, 112, 24)
$Skip_3$	InvRes	6	-	3	2	(56, 56, 32)
	InvRes	6	-	4	2	(28, 28, 64)
$Skip_4$	InvRes	6	-	3	1	(28, 28, 96)
	InvRes	6	-	3	2	(14, 14, 160)
	InvRes	6	-	1	1	(14, 14, 320)
$Skip_5$	Conv+BN+ReLU6	-	1	-	-	(14, 14, 1280)

Table 2.12: MobileNetV2 encoder used in MUnet Blocks marked as $Skip_*$ are outputs from the network which are used in the decoder. The parameters above define the configuration of the respective block, t is the expansion factor inside an inverted residual block as defined in 2.11, k is the kernel size for the two standard convolutions used in the backbone, n defines how often that particular block is repeated and s is the stride of a block. Note that if $s = 2$ only the first block has this stride, the others have $s = 1$.

performed along the channel axis. To unify the novel concatenated feature map it is passed to a inverted residual block. This step is repeated until no skip connections from the backbone are left. The last step in the decoder is composed of a bilinear upsampling layer, which is used to produce a prediction which has the same size as the input image. This has to be done because the backbone directly downsamples the image size in the first layer and hence there is no feature map with the same spatial dimensions as the input image. The architecture of the decoder can be found in table 2.13.

2.9 Hypergraphs

To evaluate the produced ECD topologies, topologies are represented as a hypergraph. Hypergraphs are a generalization of a graph and are formally defined as: $\mathbf{H} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is a set of $N \in \mathbb{N}$ vertices $\{v_1, ..., v_N\}$ and $\mathbf{E}$ is a set of $M \in \mathbb{N}$ hyperedges $\{\mathbf{e_1}, ..., \mathbf{e_M}\}$, where each hyperedge $\mathbf{e_i} \in \mathbf{E}$ is $\mathbf{e_i} \subseteq \mathbf{V}$ [Val21]. In matrix notation a hypergraph can be represented in an adjacency matrix [Ouv17]. An example of an adjacency matrix together with the corresponding drawing of a hypergraph can be found in figure 2.8.

Block	Inputs	Operation	t	k	s	Output (h, w, c)
$Up1$	$Skip_5$	ConvTranspose	-	4	2	(28, 28, 96)
$InvRes1$	$Skip_4 + Up1$	InvRes	6	-	1	(28, 28, 96)
$Up2$	$InvRes1$	ConvTranspose	-	4	2	(56, 56, 32)
$InvRes2$	$Skip_3 + Up2$	InvRes	6	-	1	(56, 56, 32)
$Up3$	$InvRes2$	ConvTranspose	-	4	2	(112, 112, 24)
$InvRes3$	$Skip_2 + Up3$	InvRes	6	-	1	(112, 112, 24)
$Up4$	$InvRes3$	ConvTranspose	-	4	2	(224, 224, 16)
$InvRes4$	$Skip_1 + Up4$	InvRes	6	-	1	(224, 224, 2)
$Up5$	$InvRes4$	UpBillinear	-	-	2	(448, 448, 2)

Table 2.13: MUnet decoder. The decoder uses transpose convolutions to upsample the input (ConvTranspose) and as the backbone, inverted residuals to process the upsampled input together with the skip connection. The '+' indicates a concatenation along the channel axis. Since the first block in the backbone directly downsamples the input there is no skip connection with the size of the input. Therefore the last layer of the decoder is a upsampling layer, which uses a bilinear upsampling method to increase the size of the prediction to the size of the input. As with the backbone t indicates the expansion size of the inverted residual block, k indicates the used kernel size and s indicates the used stride.

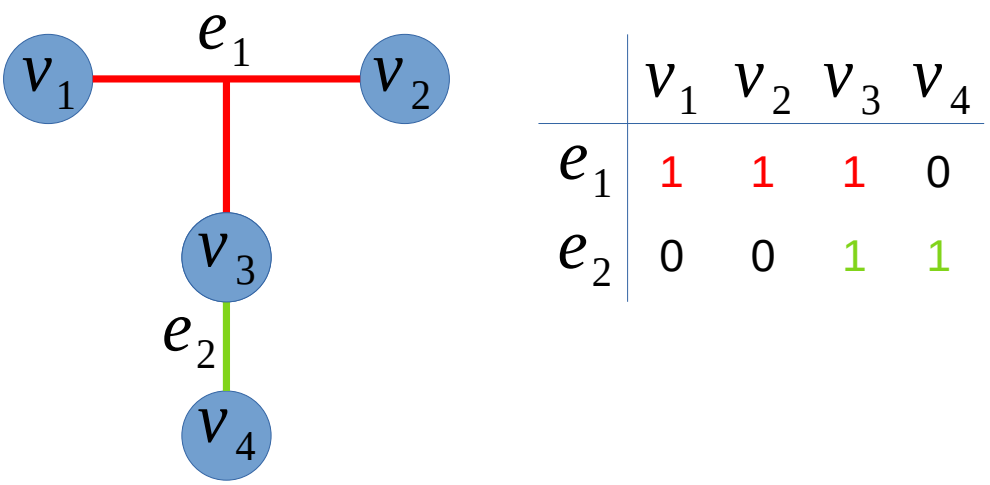


Figure 2.8: An example drawing of a hypergraph with its corresponding adjacency matrix. The hypergraph is defined as: $\mathbf{H} = (\mathbf{V}, \mathbf{E})$, $\mathbf{V} = \{v_1, v_2, v_3, v_4\}$, $\mathbf{E} = \{\mathbf{e_1}, \mathbf{e_2}\}$, with $\mathbf{e_1} = \{v_1, v_2, v_3\}$, $\mathbf{e_2} = \{v_3, v_4\}$. In the adjacency matrix a row corresponds to a hyperedge and each column to a vertex. When a vertex is present in a hyperedge it has a 1 as an entry in the matrix, when a vertex is not present it has a 0 [Kon01].

2.10 Metrics

True Positive, False Positive, True Negative, False Negative

To understand the following subsections first the notion of True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) should be explained. In a binary class environment there exists the two classes positive and negative. A TP occurs when the prediction of a network is positive and the underlying ground truth is also positive. The inverse of that is a TN, where the network predicts the negative class and the ground truth is also negative. Further, a FP occurs when the prediction of a network is positive, but the underlying ground truth is negative. Again, the inverse of that is a FN, which occurs when a network predicts a negative, but the underlying class is positive [Pow08].

For object detection the above definitions are additionally extended by an IoU threshold, i.e. a TP occurs when a class of a predicted bounding box A is the same as the ground truth bounding box B and the IoU(A, B) is greater than a certain threshold [Lew21].

Precision

The precision metric (equation 2.37) states the proportion of all correct identified samples (TP) in relation to all positive identified samples.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.37}$$

Recall

The recall metric (equation 2.38) states the proportion of all correct identified samples in relation to all possible positive samples.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.38}$$

F1-Score

The F1-Score combines precision and recall in one metric as the harmonic mean of both. It is defined as:

$$\text{F1} = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \tag{2.39}$$

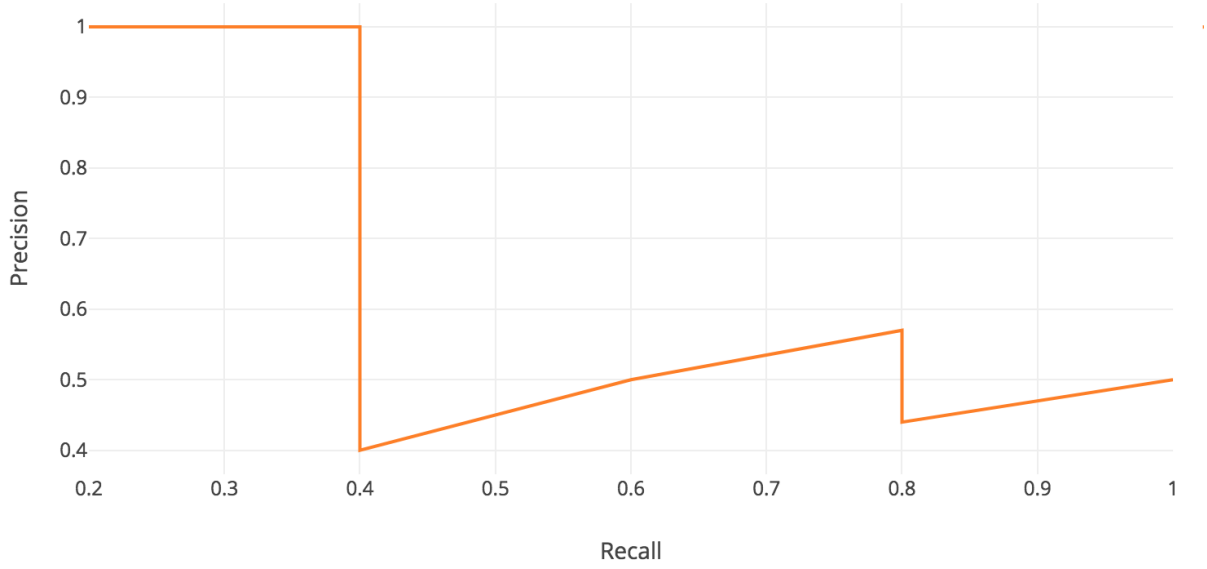


Figure 2.9: Example of a precision-recall curve, where precision and recall were calculated for different IoU thresholds and sorted and plotted by their recall values [Hui18]. The AP is the area under the curve.

Average Precision (AP)

Average Precision (AP) is the most common metric in the context of object detection. It is calculated for each class separately. It can be calculated by taking a fixed IoU threshold and calculating precision and recall with that threshold as it is done in Pascal VOC [Jun21], or by taking multiple IoU thresholds as it is done in COCO [Lew21], where the thresholds range from 0.5 to 0.95 in 0.05 steps. The resulting tuples of (recall, precision) are now sorted ascending by the recall value. The resulting precision-recall curve could then look like the one in fig. 2.9, this curve is not interpolated. Normally the precision-recall curve is further interpolated such that it is strictly monotonically decreasing. The AP is then the area under the curve and is calculated by taking the integral over the domain of the curve (equation 2.40).

$$\text{AP}(\text{Recall}, \text{Precision}) = \int_0^1 \text{Precision}(\text{Recall})\, d\text{Recall} \tag{2.40}$$

Mean Average Precision (mAP)

An extension of the AP metric is the Mean Average Precision (mAP), which measures overall classification performance for all classes combined. It is calculated as the mean of all classwise APs, where C is the number of classes (equation 2.41).

$$\text{mAP(AP)} = \frac{1}{C} \sum_{i=1}^{C} \text{AP}_i \tag{2.41}$$

Mean Intersection over Union (mIoU)

Mean Intersection over Union (mIoU) is a metric often used in segmentation tasks. As the name suggests it measures the IoU between the predicted mask and the ground truth mask. Further, the mean of all IoU values is calculated over the number of measured samples N and results in the final mIoU value (equation 2.42).

$$mIoU = \frac{1}{N} \sum_{i=0}^{N} \text{IoU}_i \tag{2.42}$$

Chapter 3

Material and Methods

3.1 Data

In this section the data which was collected for and in the scope of this thesis is presented, as well as some general information about that. The dataset is a collection of ECDs, with ECCs in German notation. Seven different ECCs with connections on two sides were chosen for the scope of this thesis and are shown in figure 2.1. The number of classes was constrained in order to allow dataset contributors to chose from a small set of symbols. Under the assumption that each contributer draws the same amount of ECCs per ECD overall the number of samples per class will be higher than with a big amount of symbols. Therefore, the individual classes are more representative. Most of the images were taken with a mobile phone, but some were directly drawn on a digital device such as a tablet. Overall 31 persons contributed to the dataset, with an average of 7.6 ECDs, a maximum of 30 ECDs and a minimum of one ECD. For the train, validation test split, ECDs of 21 persons are used in the train and validation dataset. In the test dataset those 21 person do not appear, hence 10 previously unseen persons are used for testing.

The difficulty of the task in this thesis increased gradually and more data was always acquired after the difficulty increased. At first, only images of ECDs on a white background without annotations were gathered, when that showed promising results, additionally images of ECDs without annotation but on checkered background were gathered and finally when annotations were added both images with white and checkered background and annotations were acquired. The total amount of images is shown in table 3.1. Evaluation was done with all images. When an image did not contain annotations, then the evaluation of the annotations for this particular image was skipped.

	Images	Background	Annotated	train ratio	valid ratio	test ratio
	110	white		74.66%	6.69%	18.65%
	17	checkered		17.64%	11.77%	70.59%
	89	white	✓	78.65%	11.24%	10.11%
	21	checkered	✓	9.52%	19.05%	71.43%
Total	239			45.12%	12.19%	42.69%

Table 3.1: Amount of images of ECDs used in this thesis shown with their underlying background and whether they are annotated or not. Further, the train / validation / test split of the different image types is shown. While this might seem like a big split for test, the number of bounding boxes included in the test set is way smaller and is shown in table 3.2. For the test dataset 10 persons were used, which are not present in the train and validation dataset.

Labels: Object Detection and Segmentation

The pipeline in this thesis requires the object detection network YOLOv4 and the segmentation network MUnet. Therefore, the data was labeled with bounding boxes for YOLOv4 and with segmentation masks for the MUnet.

In table 3.2 all classes used for object detection are presented. The classes have a major class which corresponds to the ECC and an orientation subclass. Some major classes like resistors have two orientations (horizontal, vertical), while others have four, like diodes (left, top, right, bottom). The only exception is the text class, which does not have an orientation, since annotations were enforced to be horizontally aligned.

Bounding boxes were annotated with the labeling tool labelme [Wad16] in the yolo format which was presented in section 2.5. To also capture parts of the wire in the prediction the bounding boxes were stretched towards the wire around an object.

The segmentation masks were created in a binary fashion, where the foreground corresponds to the drawn ECD and the background to everything else. The masks were created semi-automatically by applying a Canny Edge Detector [Can86] on the image. The resulting edge mask is dilated five times to close potential holes between the edges of a wire and eroded four times to reduce the thickness of the segmentation mask. The image processing steps become instable once gridded paper is being used, thus each mask is additionally manually fine-tuned, with a simple drawing tool build with the Python version of OpenCV [Bra00].

An example for the YOLO bounding boxes and the segmentation mask labels can be found in figure 3.1.

class	total	train ratio	valid ratio	test ratio
diode left	156	83.33%	8.97%	7.69%
diode top	210	82.38%	6.19%	11.43%
diode right	150	82.00%	12.67%	5.33%
diode bottom	102	67.65%	15.69%	16.67%
resistor horizontal	318	71.38%	6.92%	21.70%
resistor vertical	350	66.00%	6.57%	27.43%
capacitor horizontal	405	85.68%	4.94%	9.38%
capacitor vertical	268	65.30%	10.45%	24.25%
ground left	137	72.99%	10.95%	16.06%
ground top	137	81.02%	13.87%	5.11%
ground right	116	78.45%	14.66%	6.90%
ground bottom	178	73.60%	14.04%	12.36%
inductor horizontal	251	76.89%	8.37%	14.74%
inductor vertical	290	73.45%	9.31%	17.24%
source horizontal	188	77.66%	11.17%	11.17%
source vertical	238	64.71%	14.29%	21.01%
current horizontal	202	77.72%	9.41%	12.87%
current vertical	220	75.00%	12.73%	12.27%
text	877	61.92%	16.76%	21.32%
arrow left	57	70.18%	19.30%	10.53%
arrow top	77	64.94%	23.38%	11.69%
arrow right	105	70.48%	16.19%	13.33%
arrow bot	104	71.15%	15.38%	13.46%
total	5136	73.65%	12.27%	14.08%

Table 3.2: The classes present in this thesis with their major class which is an ECC and alternatively their orientation. Furthermore, the total amount of classes is shown and the train, valid, test ratio.

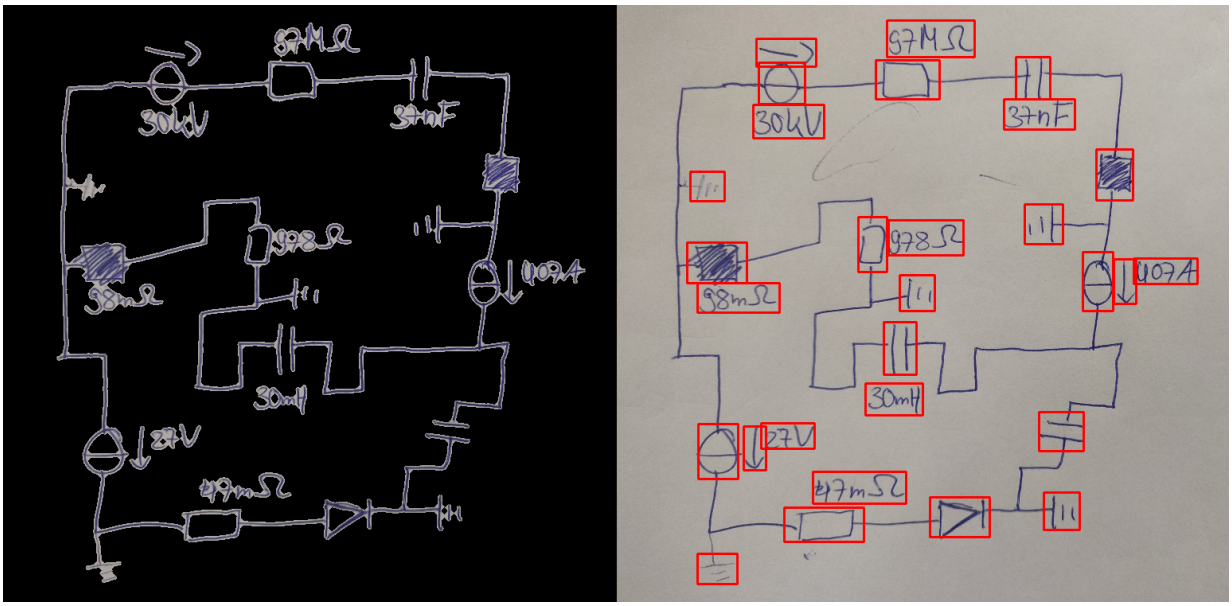


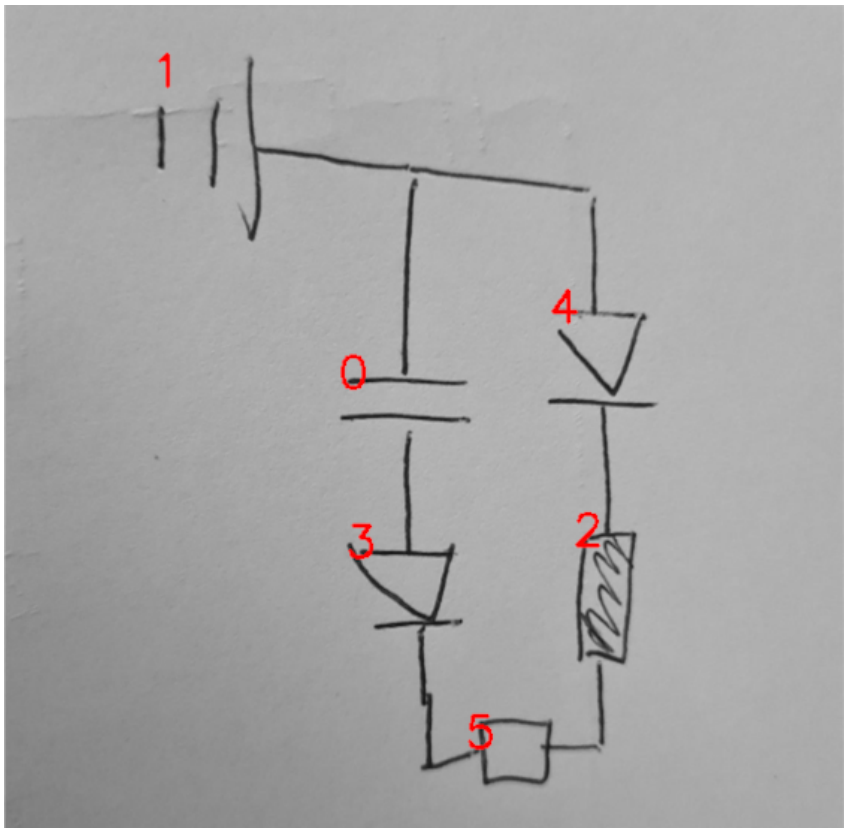
Figure 3.1: Example segmentation label mask (left) and bounding boxes labels (right).

Labels: Topology

The pipeline of this thesis produces a topology representing the underlying connections between ECCs. To quantify the amount of errors in a produced topology (prediction), a ground truth is needed, against which the prediction is compared to. The chosen prediction and ground truth format is an adjacency matrix of a hypergraph. The labeling process as well as the evaluation with proposed algorithm is time consuming, therefore only the test dataset was labeled and evaluated in this thesis. Especially due to the fact that the underlying problem, which the evaluation algorithm tries to solve, is NP-hard. More details are explained in section 5.1. Following the label format of the ECDs is explained.

First a copy of the original ECD was created, which was then populated with numbers above each component, which was labeled with a bounding box. The number above each component corresponds to the index of that bounding box in the YOLO label file. Those images were used as a visual helper in the topology labeling process. An example of such an image can be found in figure 3.2.

To now label the topology, first each index is split into to sub-indexes, each representing the orientation of the connection, where a connection is defined as the side where the wire is connected to. For instance in figure 3.2, index 1 is a ground symbol and its connection is on the right side. For index 0, which is a capacitor the connection is at the top and the bottom. The



Index	0		1		2		3		4		5		
Edge 0	1	0	0	1	0	0	0	0	1	0	0	0	1r,0t,4t
Edge 1	0	1	0	0	0	0	1	0	0	0	0	0	0b,3t
Edge 2	0	0	0	0	0	0	0	1	0	0	1	0	3b,5l
Edge 3	0	0	0	0	0	1	0	0	0	0	0	1	5r,2b
Edge 4	0	0	0	0	1	0	0	0	0	1	0	0	4b,2t

Figure 3.2: An example visual labeling helper image (top), shown with the index of the respective bounding box in the YOLO label file, together with the corresponding hypergraph adjacency matrix (bottom). Two columns in the matrix correspond to one ECC, each column to one orientation, where the first column can either be the left or top orientation and the second column be the right or bottom orientation. The last column in the table further shows the string representation of an edge, which acts as the input for the labeling tool. For instance “1r,0t,4t” means that 1 right, 0 top and 4 top are connected to each other through a hyperedge.

two orientation pairs {left, top} and {right, bottom} are fused together into one column of the hypergraph adjacency matrix. Given a hypergraph matrix, the orientation of a particular bounding box index n can be obtained with the index 2n or 2n+1, for the {left, top} or {right, bottom} orientation, respectively. A connection, where multiple ECCs are connected is represented as a row in the adjacency matrix. Figure 3.2 shows how the hypergraph adjacency can be created from a visual helper image.

For the topology labeling a small labeling tool was created, where the algorithm should be shortly presented:

1. Initialize a matrix of zeros with the size $N \times N$, where N is the number of bounding boxes in the ECD.
2. Read user input in the form: "{Index$_x$}{Orientation$_x$},...,{Index$_y$}{Orientation$_y$}", where Index$_n$ represents the index of a bounding box and orientation $\in$ {t, l, r, b} (top, left, right, bottom), which represents a complete edge in the hypergraph adjacency matrix. Do this until all edges have been entered.
3. Store the produced hypergraph adjacency matrix.

Labels: Textual Annotations and Arrow Annotations

In general all annotation types used in this work belong to an ECC and therefore act as an extension of an ECC. Therefore, matching of annotations to their respective ECC is part of this work and is presented in section 3.2. To quantify the matching performance, a representation of the ground truth is needed. The chosen label format builds on the visual helper image, presented in the topology labeling process. A map is created, where the index of an ECC is mapped against the index of the corresponding annotation. Since, two different types of annotations are present, each annotation type receives an own label file. A label file is created in the Comma Separated Values (CSV) file format, where each line represents a matching pair of ECC index and annotation index. Hence, one file contains all matching pairs for textual annotations and the second one all matching pairs for arrow annotations.

3.2 Recognition and Conversion Pipeline

In this section the pipeline is presented, which allows to convert an image of an ECD into the LTspice schematic file format. To fully convert the ECD from the Image Domain (IDom) into the LTspice Domain (LDom) the following needed conditions have been identified:

- the class and position of the ECCs
- the text and position of the annotations as well as the annotation mapping (to which ECC belongs this annotation)
- the connections between the ECCs
- the conversion into the LTspice schematic file

The above points are all embedded into a six-stage pipeline, which can be found in figure 3.3 and will be thoroughly explained throughout this section.

Stage 1: Object Detection

Predicting the class and position of an object can essentially be formulated as an object detection problem. Various object detection networks exist, which could be used for this task. In this thesis the presented YOLOv4-Tiny (section 2.6), which is from now on referred to as YOLO, is used to predict the ECCs, ECC-annotations (arrows for sources) and the text annotations in the IDom. YOLO was chosen since it has a good compromise between network size and classification performance.

Stage 2: Segmentation

Furthermore, the pipeline includes the segmentation of the circuit. Segmentation is needed because of the topology building step, which is described next requires a clean mask of the circuit. The initial clean mask was created using image binarization. While the topology building worked for circuits with white background it however failed for circuits with checkered background. Therefore, the MUnet (section 2.8) is used to segment a circuit in the IDom. The network predicts a binary classification output in the form background / not background, where everything which is unrelated to the ECD is considered background. The network was trained for both checkered and uncheckered backgrounds, such that it can be applied on both types of images. Again, MUnet was chosen, since it is a lightweight network with appropriate performance, able to be used on mobile devices.

Stage 3: Topology Creation

The next step in the pipeline is the identification of the connections between the ECCs, such that the wires from the IDom can be transformed into the LDom. In an abstract form this is topology

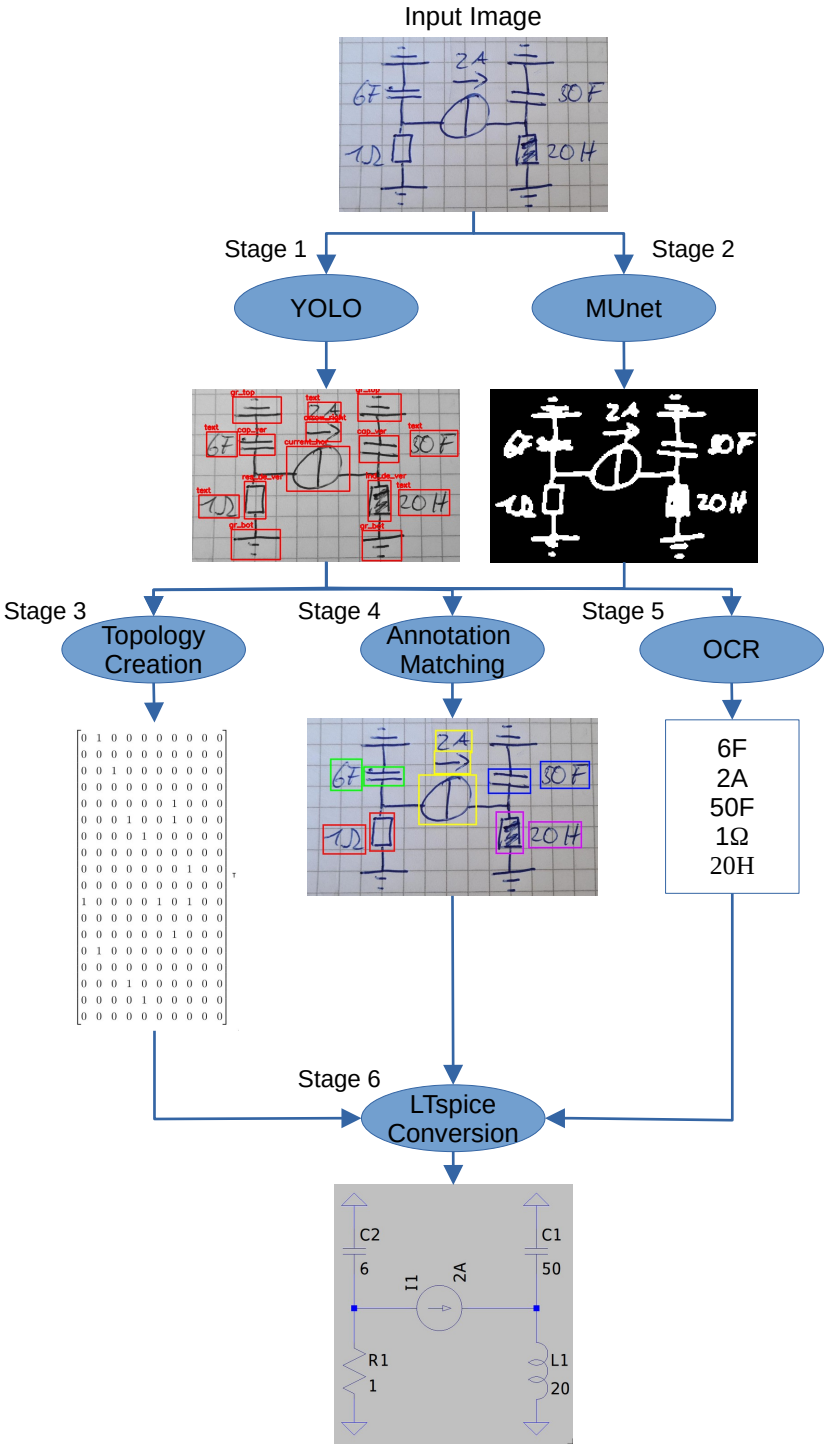


Figure 3.3: The six-stage pipeline presented in this thesis. Stage 1 shows the results of the object detection with YOLO and stage 2 the segmentation results with the MUnet. Stage 3-5 are postprocessing stages, where the topology is created, the annotations are matched to their respective ECC and the characters of the textual annotations are recognized. The last stage is the conversion stage, where the gathered information is embedded into the LTspice schematic file format.

creation. This step does not take into account the spatial positions of the components, but only the semantics of the circuit. In the following the algorithm is presented, which takes as input the predicted bounding boxes and the segmentation mask of the circuit and produces the topology of the circuit. The algorithm utilizes various OpenCV algorithms, which are first briefly explained.

Connected Components Labeling by Grana et al. [Gra10] is an 8-way connectivity algorithm used to label blob like regions in a binary image. In the topology creation process it is used to identify the wires.

Morphological Operations like erosion, dilation or the combinations of those like opening and closing [Kos20], are used to filter the used binary images and to refine results obtained from various sources in the pipeline.

After the basic methods were presented now the algorithm can be explained. The algorithm receives as input the predicted bounding boxes and the segmentation mask of the circuit.

1. Copy the **SegmentationMask** $\in \{0,1\}^{h \times w}$ into **WiresOnly**. Iterate over the bounding boxes (bbox $\in \{x_1, y_1, x_2, y_2, class\}$, where (x_1, y_1) represent the upper left corner of the bounding box and (x_2, y_2) the lower right) and remove every pixel in **WiresOnly**, which is included in a bounding box. Only the wires remain now.

2. Perform morphological closing on **WiresOnly** to close up small holes in the wires.

3. Apply connected components labeling on **WiresOnly** to label the separate wire blobs resulting in **WiresLabeled**.

4. Create a matrix **BBoxMask** of zeros with the size of **WiresOnly**. Iterate over the bounding boxes and populate **BBoxMask** with rectangles with a thin border created from the bounding box coordinates.

5. Apply the and-operator on **WiresLabeled** and **BBoxMask** and receive the intersections, where a wire is intersecting the border of a bounding box. Store the result in **Intersections**.

6. Initialize a dictionary **Topology** and populate it by iterating over **Intersections** and for each intersection index check the connected components label at this index create an entry in **Topology** with an empty array.

7. Iterate over each intersections index and find the bounding box which is “involved” in this intersection. A bounding box is “involved”, when intersection index $\in$ bounding box border and the class of the bounding box is an ECC.

8. Find the orientation of the intersection. Orientation refers to the connection orientation relative to the bounding box, i.e. is the intersection at the top, bottom, left or right border.

9. Get the connected components label at the intersection index and add a tuple of (bounding box index, orientation) at the respective spot in the **Topology** if this index with this orientation is not present.

10. After the initial **Topology** is build, iterate over it and remove each connected component label, where the involved bounding boxes array has $size = 1$ and the involved bounding box has a contradictory orientation in relation to the predicted class, i.e. classes predicted with vertical orientation can only have a connection at the top or bottom, classes predicted with horizontal orientation can only have a connection left or right.

11. Return the **Topology**

Stage 4: Annotation Matching

In this thesis different annotations are used. Arrow annotations are only used for voltage and current sources to indicate the direction of the potential difference as well as the current flow, respectively. On the other side textual annotations can be applied to any type of an ECC. To fully reflect the circuit in the LDom annotations have to be matched against their respective ECC. The algorithm used in this thesis is presented in algorithm 3.1 and 3.2 for arrow and text annotation respectively. An annotation is matched against an ECC using a simple brute force nearest neighbor approach, based on the center distance of the bounding boxes to match. Brute force in the sense that every annotation will get matched against an ECC without considering that the distance between annotation and ECC is maybe way too big, like it would be the case for example when a FP annotation is predicted, it certainly will get matched. Multiple annotations are also possible with this algorithm, but when this occurs the one with the smallest distance is taken and the other match is ignored.

Algorithm 3.1: Arrow Annotation Matching

Input: A = $\{a_1, ..., a_n\}$ (list of predicted arrow bboxes)
A = $\{e_1, ..., e_o\}$ (list of predicted ECC bboxes)
Output: A_m = $\{(a_1, e_i), ..., (a_n, e_j)\}$ (list of arrow bboxes matched against an ECC bbox)

```
A_m ← {};
for a in A do
    distances ← {};
    a_center ← get_bounding_box_center(a);
    for e in E do
        if is_source_type(e) then
            e_center ← get_bounding_box_center(e);
            dist ← euclidean_distance(a_center, e_center);
            distances ← distances + (dist, a, e);
        end
    end
    dist_min, a_m, e_m ← get_min_distance(distances);
    A_m ← A_m + (a_m, e_m);
end
```

Algorithm 3.2: Textual Annotation Matching

Input: T = $\{t_1, ..., t_m\}$ (list of predicted text bboxes)
E = $\{e_1, ..., e_o\}$ (list of predicted ECC bboxes)
Output: T_m = $\{(t_1, e_x), ..., (t_m, e_y)\}$ (list of text annotation bboxes matched against an ECC bbox)

```
T_m ← {};
for t in T do
    distances ← {};
    t_center ← get_bounding_box_center(t);
    for e in E do
        e_center ← get_bounding_box_center(e);
        dist ← euclidean_distance(t_center, e_center);
        distances ← distances + (dist, t, e);
    end
    dist_min, t_m, e_m ← get_min_distance(distances);
    T_m ← T_m + (t_m, e_m);
end
```

Stage 5: Optical Character Recognition of Textual Annotations

Interpretation of the textual annotations should normally be also part of the pipeline. Due to time constraints this step was not implemented in the scope of this thesis. But an OCR engine such as Tesseract [Smi09], could be used to detect the characters in the textual annotations.

Stage 6: LTspice Conversion

The last step in the proposed pipeline is the embedding of the gathered information into the LTspice schematic file, where the syntax was presented in section 2.2. A module was created, which utilizes this syntax and generates a LTspice schematic file, where ECCs can be directly parametrized with their properties by passing the respective values to the generator functions. Three input values can be found for the generator functions:

- ECC type
- annotation parameters (no input provided in this work)
- position of the ECC

Further the module can also generate wires to connect the ECCs, here only the start and end position of the wire have to be passed to the generator functions.

So far, the impact of the position of an ECC was not discussed, but to replicate the circuit in the LDom it is necessary to capture the positions in the IDom. Positions from the IDom were predicted by the YOLO network and are available as a bounding box, containing class, size and the center point of an ECC.

It was mentioned that LTspice relies on a 32×32 grid, where ECCs and wires are aligned to. ECCs from the IDom need to be projected into that grid. A simple approach would be to take the minimum distance between two ECCs in the IDom and use this distance as a grid normalizer. The problem is here that, the sizes and aspect ratios of bounding boxes do not correspond to the sizes in the LDom. So when just using the minimum distance it can happen, that two ECC components will overlap. To mitigate this issue an additional $stretch \in \mathbb{R}^+$ parameter is introduced, which scales the minimum distance. A $stretch < 1$ will reduce the minimum distance between the ECCs in the IDom and increase the distance in the LDom. The coordinates in the LDom can then be calculated with:

$$coord_{LDom} = \text{round}\left(\frac{coord_{IDom}}{stretch \cdot dist_{min}}\right) \cdot 32 \tag{3.1}$$

Where, $coord_*$ corresponds to the x and y coordinate respectively. A stretch of 0.3 has shown to provide a good balance between enough space between components for them to not intersect and at the same time not overscaling the distances between the ECCs too much.

Chapter 4

Training and Experiments

The following chapter describes the training process and the conducted experiments for the YOLOv4-Tiny and for the MUnet. For both network the same train, validation and test split ratio of the data was used as described in section 3.1. In the training and validation dataset ECDs of 21 persons are present, where the same person can appear in both splits. For the test dataset only persons were used which do not appear in the train nor in the validation dataset. The detailed split ratio by images can be found in table 3.1 and the class based split ratio can be found in table 3.2.

4.1 YOLOv4-Tiny

The following section deals with the training process of the YOLO network, where an overview of the performed experiments can be found in figure 4.1. All of the presented experiments were solely evaluated on the validation dataset, therefore no decision was done based on the results of the test dataset. The test dataset was only used in the final evaluation.

To perform any experiments with the YOLO network, first an initial configuration was defined. This configuration is given in table 4.1, where the default parameters of the utilized YOLO

Batch Size	64
Loss	CIoU
Optimizer	Stochastic Gradient Descent (SGD) with Momentum ($\mu = 0.9$)
Burn in	1000 steps
Input Size	$608 \times 608 \times 1$ (gray scale)

Table 4.1: The initial training configuration for the experiments performed with the YOLO network.

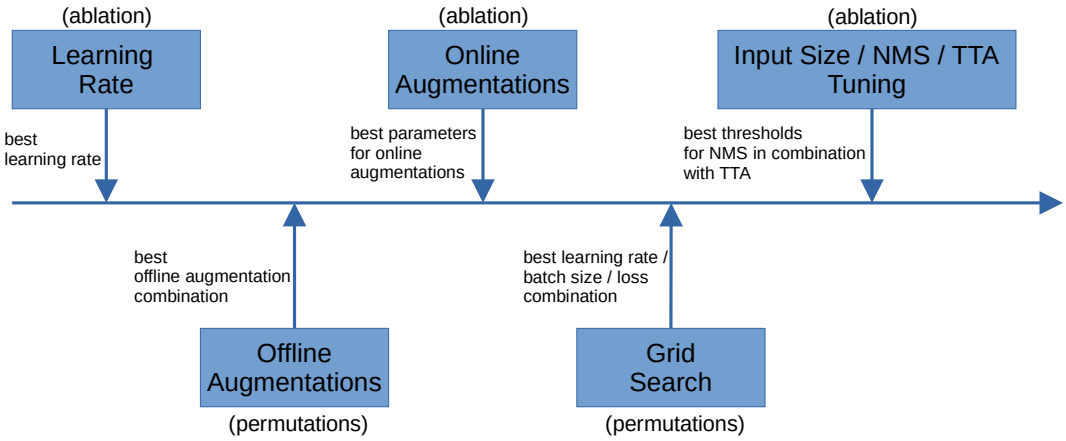


Figure 4.1: An overview of the conducted experiments with the YOLO. Best parameters which were obtained from a previous experiment were used in the following experiment.

repository were used. All networks were trained for 4000 steps (batches) and no early stopping was performed. Further, a $burn_{in}$ of 1000 steps was used for all networks.

Redmon et al. [Red16] have pointed out that training YOLO is unstable when the full learning rate is applied without proper scheduling. This was also tested in this thesis. All learning rates which were tested in the initial learning rate search diverged when used without a scheduling mechanism. The proposed scheduling function by Redmon et al., which is still used in the current YOLOv4 [Boc20], is given by the following formula:

$$lr(step) = \begin{cases} lr_{base} \cdot (\frac{step}{burn\_in})^4 & step < burn\_in \\ lr_{base} & step \geq burn\_in \end{cases} \tag{4.1}$$

All trained experiments were optimized for a slightly changed COCO mAP metric. COCO uses a mAP$0.5 : 0.95 : 0.05$, while in this thesis mAP$0.5 : 0.75 : 0.05$ is used, since an IoU of 0.75 is considered to be enough for the whole system to work properly in most circumstances. All experiments are optimized for this metric, which means that at every step the mAP is calculated for the whole validation set and if the mAP is better than the previously calculated mAP, the weights of the network are stored and used for further evaluation.

4.1.1 Learning Rate Experiment

The first experiment was executed to find an initial learning rate. The parameters for the network were set to the ones in table 4.1 and kept for all training runs. The results of the training runs can

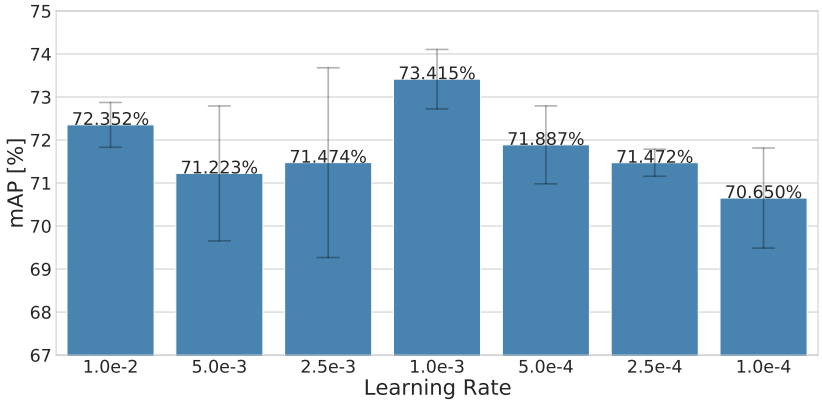


Figure 4.2: The results of the initial learning rate search shown on the validation set, with the mean and standard deviation of the mAPs over three separate training runs.

be found in figure 4.2, where the mean of mAP for the three performed runs is shown for each learning rate. The full results of the learning rate search with classwise performance can be found in table A.1. It can be observed that the learning rate 0.001 has the highest mAP, therefore it is selected as the default learning rate for further experiments. In this experiment the text class and especially the arrow classes showed consistently bad performance over all learning rates.

4.1.2 Augmentation Experiments

Offline Augmentations

This experiment was conducted to find the optimal configuration of offline augmentations, where the offline augmentations are projection (copy-paste augmentation [Ghi20]), three 90° rotations and a horizontal flip. If both the rotation and the flip augmentation are selected at the same time, then additionally the flipped image is rotated three times. Again for all runs the YOLO configuration from the previous experiment was used, but additionally now the learning rate is set to the best performing one from the learning rate search experiment, which is: $lr = 0.001$.

The full results with classwise AP can be found in table A.2. The results for this experiment are also shown in figure 4.3, where a clear trend emerges. When looking at the results as an ablation it can be seen that each offline augmentation brought an increase in mAP and the combination of those too. Rotation has an absolute increase in mAP, when compared to the baseline (the best performing learning rate), of 13.725%, flip shows an increase of 11.000% and the copy-paste augmentation shows an increase of 5.988%. The best configuration is, as expected, the one where

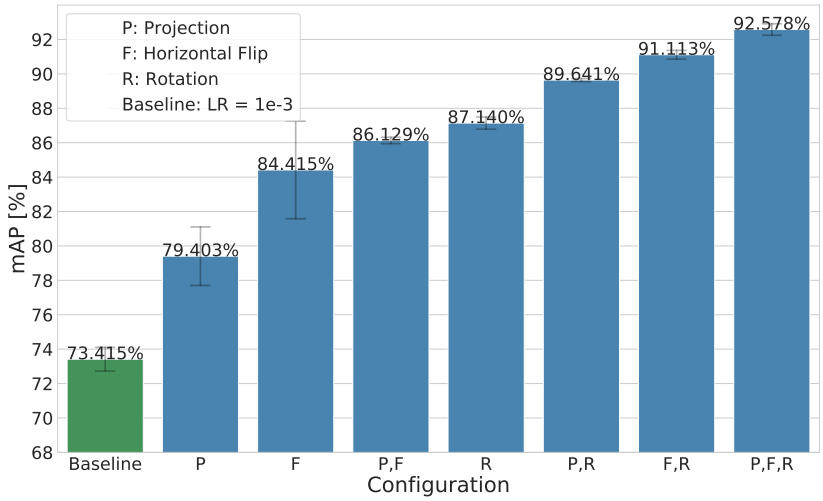


Figure 4.3: The results of the offline augmentation with the different offline augmentation configurations compared with the results of the best performing learning rate (baseline). When rotation and flip are enabled simultaneously the flipped image also gets rotated three times by 90°. Results are given with the mean and standard deviation of the mAPs of three separate training runs.

all three augmentations are used simultaneously and it has an increase in mAP of 19.163%. When comparing the classwise results of the best performing configuration with the classwise results of the baseline, it can be observed that all ECC classes have reached a mAP greater than 90%. The text and arrow classes have also greatly increased. Text shows an increase of 24.485% and the mean over the arrow classes shows an increase of 47.988%, however those two classes are still not near the desired performance observed at the ECCs. It should be pointed out that especially the increase in the text class is very interesting, because this is the only class which is not rotation and flip invariant, i.e. flipping a diode produces again a diode for human perception, just with a different orientation. However flipping a text will produce something no more longer interpretable for the human perception. An explanation why there is still an increase in recognition performance could be that the network starts learning blob like regions, which are located near ECCs, instead of specific features related to the different textual annotations.

Online Augmentations

The next presented experiment was an ablation study performed on the following augmentations:

- Rotation: 10°, 20°, 30° (maximum angle to rotate the base image +- value)
- Scale: 10%, 20%, 30% (the maximum percentage of the base scale to change +- value)
- SafeCrop: 70%, 80%, 90% (the maximum size of the crop relative to the input image size, but considers bounding boxes such that bounding boxes are never cropped)
- ColorJitter: 10%, 20%, 30% (the maximum percentage to change the brightness, contrast, saturation or the hue of the input image)

These augmentations are performed at runtime and are therefore also referred to as online augmentations. Each augmentation was applied with a probability of 50%. The configuration of this experiment now additionally utilizes the best combination of offline augmentations from the previous experiment.

The detailed results can be found in the tables A.3, A.4, A.5, A.6, for each used augmentation separately. Further, the combined results can be found in figure 4.4. The figure shows, for each augmentation and for all parameters a clear increase in mAP. The best performing combination, is the one where all offline augmentation are enabled. It achieved a mAP of 92.578% and is selected as the baseline for further experiments.

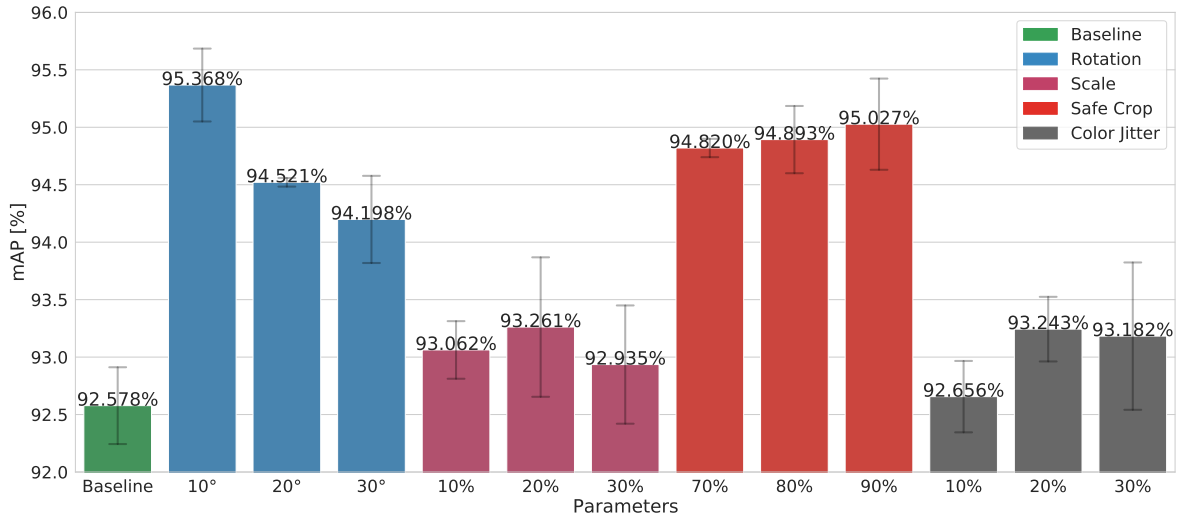


Figure 4.4: Results of the online augmentation experiment compared to the baseline which was established in the offline augmentation experiment. Each augmentation shows a clear increase in mAP in comparison to the baseline. Results are given with the mean and standard deviation of the mAPs of three separate training runs.

4.1.3 Grid Search Experiment on Hyperparameters

The last training experiment is a grid search for finding the best learning rate, batch size and most suitable loss function. The used parameters in the grid search are listed in table 4.2, seven learning rates, two batch sizes and three different types of loss functions were used. CIoU is also the default loss used throughout the previous experiments, while EIoU and Focal-EIoU are newly added for this experiment. The γ parameter in Focal-EIoU was not set arbitrary, but was selected based on the experiments of the authors of EIoU [Zha21], where that γ has shown the best results.

Learning Rates	$1.0e^{-2}, 5.0e^{-3}, 2.5e^{-3}, 1.0e^{-3}, 5.0e^{-4}, 2.5e^{-4}, 1.0e^{-4}$
Batch Sizes	$32, 64$
Losses	CIoU, EIoU, Focal-EIoU ($\gamma = 0.5$)
Offline Augmentations	*projection, flip, rotation*
Online Augmentations	*rotation = 10°, scale = 20%, safe crop = 90%, color jitter = 20%*

Table 4.2: Configuration of the YOLO grid search experiment with the tested grid parameters and the fixed offline and online augmentation parameters. The networks were trained for all possible configurations of learning rate, batch size and loss function.

In figure 4.5 first the batch sizes are compared. It can be clearly seen that a batch size of 32 performed consistently worse than a batch size of 64 independently of the underlying learning rate - loss combination. Therefore, the following evaluation of the results is only done on the networks, which were trained with a batch size of 64.

Figure 4.6 shows a heat map for all possible configurations trained with a batch size of 64. It can be observed that networks trained with EIoU performed better over all learning rates, while those trained with CIoU performed second best and networks trained with Focal-EIoU performed the worst.

4.1.4 Post-Training Fine-Tuning Experiments

Input Size Tuning

Redmon et al. [Red16] trained their YOLO networks (v2, v3) with a small input resolution and after training increased it to get a further boost in performance. Same was done in this thesis. The networks in this thesis were trained with an input size of 608×608, which is the default input size for YOLOv4-Tiny. This is now tuned by testing increasing input sizes on the validation set and selecting the best performing one. The results of this tuning can be found in figure 4.7. The

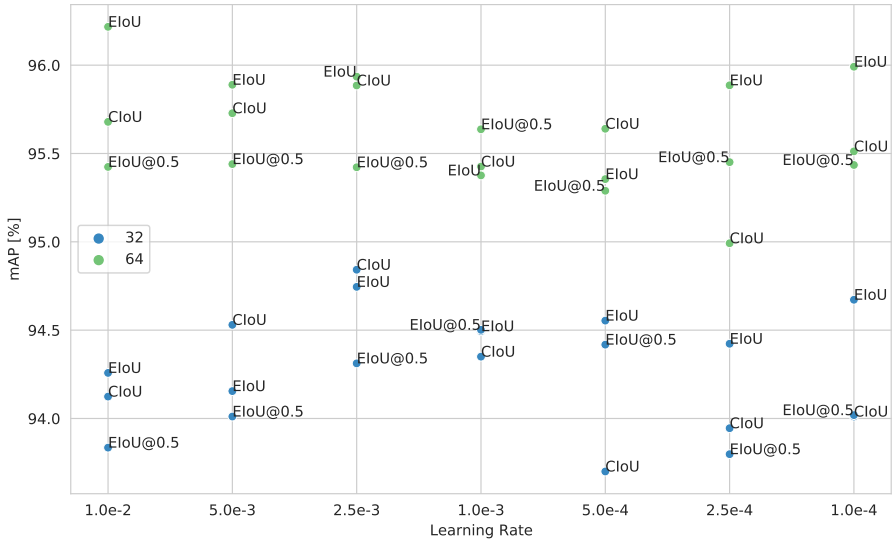


Figure 4.5: Results of the grid search shown for all loss functions, learning rates and batch sizes. Independent of the loss / learning rate combination a batch size of 32 performed consistently worse than a batch size of 64.

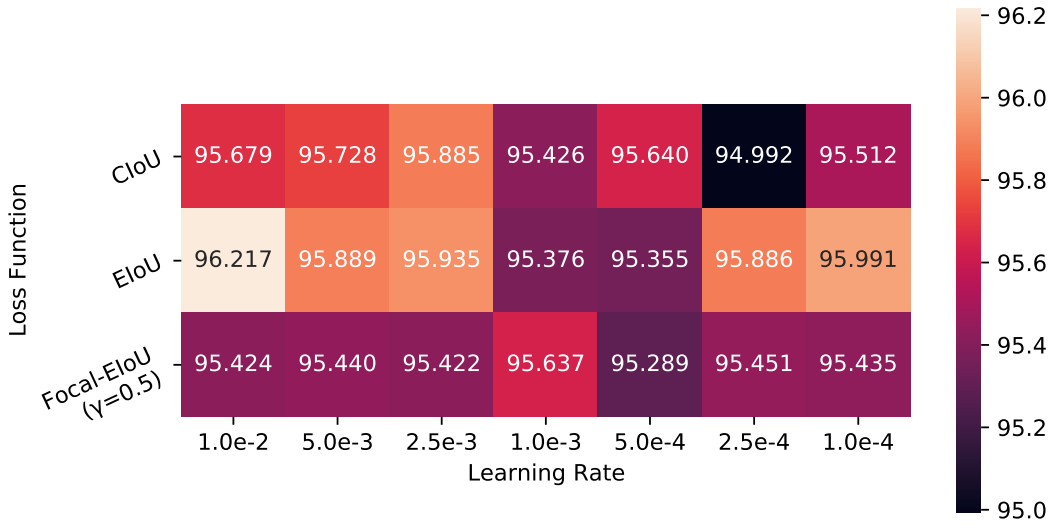


Figure 4.6: Results of the YOLO grid search for all used loss functions and learning rates shown for batch size 64 on the validation dataset.

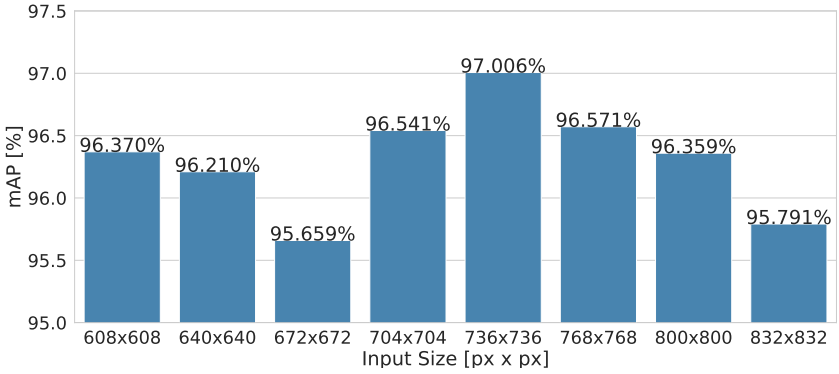


Figure 4.7: Different input sizes tested after training on the validation set.

input size 736×736, shows a maximum increase in performance and is hence selected for further tuning experiments.

Non-Maximum Suppression Tuning and Test-Time Augmentation

YOLO predicts multiple bounding boxes per object. To suppress unnecessary ones and keep only the most suitable, NMS is used. The used NMS algorithm during all previous experiments was DIoU-NMS [Zhe19]. In this subsection the results are also evaluated using the WBF algorithm [Sol19], which has shown to perform well in combination with TTA. Both algorithms were explained in section 2.6. All tunings are performed only on the validation dataset, with the new input size of 736×736.

First a tuning of the DIoU-NMS thresholds is performed. To recall, DIoU-NMS takes as input two thresholds: a score threshold, which will remove predictions with a prediction score below that one, and an IoU threshold which is used to define whether two bounding boxes A and B are the same one, i.e. if A and B have an IoU$(A, B) > IoU_{thresh}$, then only the one with the maximum prediction score is kept and the other one is removed. The tuning was performed on all possible combinations of score threshold and IoU threshold, where both have the same parameter set. The parameter set for both is ranging from 0.1 to 0.5 in 0.05 steps. The results of the tuning can be found in figure A.1. The best performing combination has a score threshold of 0.1 and a IoU threshold of 0.45.

The second tuning is done using the above setup, except that now the thresholds are tuned for the WBF algorithm. The results of the tuning are shown in figure A.2. The best performing configuration has a score threshold of 0.15 and an IoU threshold of 0.25. Further, WBF shows

slightly better results with a relative improvement of 0.15%. This improvement was expected, since in the predictions with the DIoU-NMS, some FPs were found, where multiple bounding boxes were predicted for the same text object. In WBF such predictions are fused into one bounding box, hence the slight increase in performance.

In the last tuning additionally TTA is added to the configuration. The used augmentations are, as in the offline augmentation experiment, three 90° rotations of the original image, a horizontal flip and again three 90° rotations of the horizontally flipped image. This experiment also uses WBF to merge the predicted bounding boxes. The full results of this tuning can be found in figure A.3, where the best configuration has a score threshold of 0.1 and an IoU threshold of 0.45. The relative improvement of WBF-TTA, compared to DIoU-NMS is 1.5%. For the sake of completeness it should be noted that TTA was also performed in combination with DIoU-NMS, but this actually slightly worsened the results, when comparing them to DIoU without TTA.

Improve Model Uncertainty Through Voting Based Thresholding in WBF

TTA in combination with WBF showed an improvement in performance, however while inspecting the predictions it was observed, that the amount of FPs has increased. Following a predictor is defined as a model which has predicted a certain augmentation of an image. When looking at the prediction of each predictor, it can be seen that not every predictor had predicted the same bounding box.

To incorporate the uncertainty of each predictor the idea was to create a voting based threshold mechanism, which allows WBF to reject bounding boxes, when the amount of casted votes on a bounding box is below a certain threshold $vote_{thresh}$. Obviously, $vote_{thresh}$ can range from 1 to the amount of predictors. In the case of this thesis the number of predictors is equal to the amount of images created through TTA (8 = original image + three rotations of original + horizontal flip + three rotations of flip). The voting was implemented directly in the repository of the WBF algorithm [Sol19].

Again, the tuning of the voting threshold was performed on every possible $voting_{thresh}$ and the obtained results are illustrated in figure 4.8. Using a voting threshold has not shown any improvement on the validation set, hence for further evaluation a $voting_{thresh} = 1$, which is the default behavior of WBF.

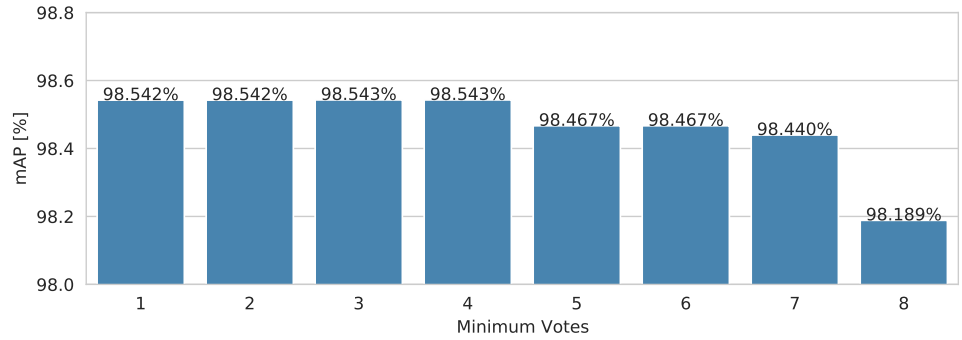


Figure 4.8: Results of the voting based thresholding approach to improve model performance, shown on the validation set for all possible voting thresholds.

Experiment	NMS	Score Thr.	IoU Thr.	Input Size	Val. mAP	Test mAP
Baseline	DIoU	0.3	0.25	608×608	96.370%	88.883%
InputSize	DIoU	0.3	0.25	736×736	97.006%	92.926%
DIoU-Tune	DIoU	0.15	0.45	736×736	97.036%	92.926%
WBF-Tune	WBF	0.15	0.25	736×736	97.187%	93.188%
WBF-TTA-Tune	WBF-TTA	0.1	0.45	736×736	**98.543%**	**95.492%**

Table 4.3: Results of the tuning experiment with the underlying threshold and input size combination, performed on the validation set, presented also with the performance on the test dataset. A visual representation can also be found in figure 4.9.

4.1.5 Final Results

Figure 4.9 and table 4.3 show the combined results of each tuning step on the validation and the test set, compared to the untuned baseline, which is the best performing network from the grid search experiment.

The impact of the input size tuning on the validation set was smaller than on the test set, which probably relies in the fact that the network was already tuned on the validation set during training and is hence already well optimized on that.

Then the two NMS algorithms DIoU-NMS and WBF were tuned, respectively. On the validation set a slight increase can be seen for both algorithms, however on the test set that increase can only be observed for WBF.

The last tuning was additionally done using WBF as NMS together with TTA. Here, a clear increase in performance can be observed and TTA benefits the recognition performance.

Overall through tuning and TTA a relative improvement of 2.173% and 6.609% could be achieved on the validation and test dataset, respectively.

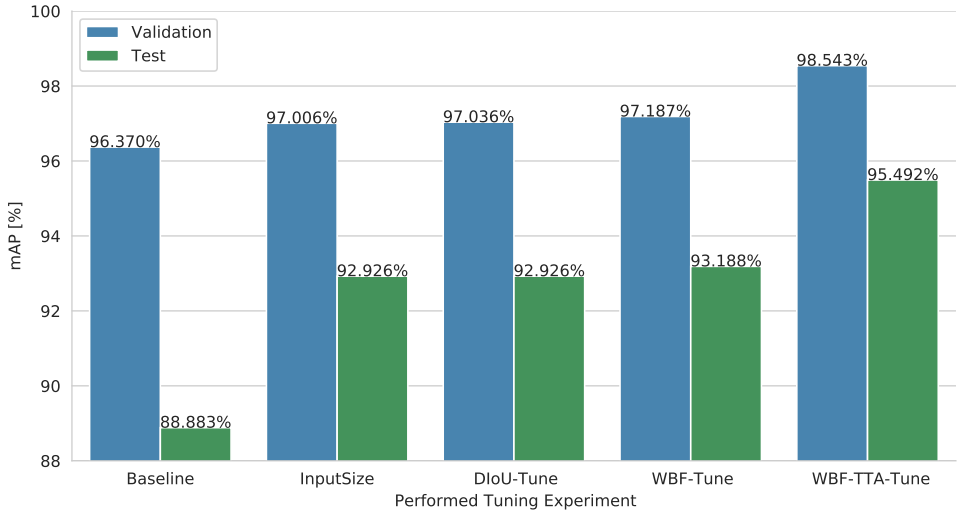


Figure 4.9: The final results of the tuning experiment performed on the validation set, presented also with the performance on the test dataset. Specific explanation of the experiments with the optimal obtained parameters can be found in table 4.3.

4.2 MobileNetV2-UNet

In this section the training and the experiments of the MUnet are presented. The experiments were conducted in the same way as with the YOLO network, as described in the beginning of this chapter and an overview of the experiments can be found in figure 4.10. The configuration for the MUnet is presented in table 4.4. The selected values correspond to the initial values from the used repository. It has been shown that transfer learning has a positive effect on the training process [Jin20], therefore the training of MUnet is performed with MobileNetV2 weights, which were pretrained on the ImageNet dataset. The backbone was pretrained on RGB images, while in this work colors are not considered to be good features. To fit the input size of the backbone network, a grayscale image is repeated three times along the channel dimension.

All MUnet training runs were executed for 7000 steps and optimized for the F1-score. Early stopping was performed, when the F1-score metric did not change for 3000 steps.

Further, a learning rate scheduler with the following formula was used:

$$lr(step) = \begin{cases} lr_{base} & step < 1000 \\ lr_{base}/5 & 1000 \leq step < 1500 \\ lr_{base}/25 & 1500 \leq step \end{cases} \tag{4.2}$$

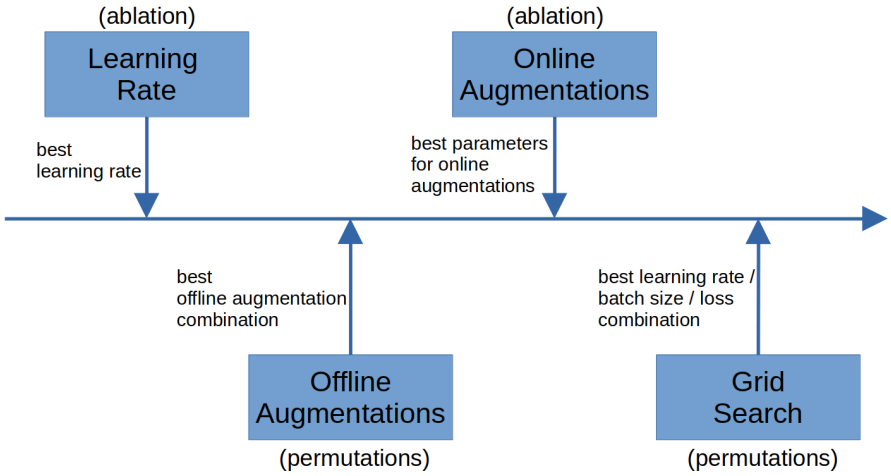


Figure 4.10: An overview of the conducted experiments with the MUnet. Best parameters which were obtained from a previous experiment were used in the following experiment.

4.2.1 Learning Rate Experiment

The first experiment was a search for an initial learning rate. The results of the learning rate search can be found in figure 4.11 The learning rate 0.01 showed the best results within a reasonable standard deviation and is therefore selected as the initial learning rate for the next experiment.

4.2.2 Augmentation Experiments

Offline Augmentations

The next experiment was done using the learning rate from the learning rate search experiment. The results can be found in figure 4.12. In contrast to YOLO this experiment shows a significantly smaller increase in F1-Score in general. Again the best performing offline augmentation combination is the one with projection, flip and rotation.

Batch Sizes	64
Loss	Focal-EIoU ($\alpha = 0.8, \gamma = 2$)
Optimizer	AMSGrad ($\mu = 0.95, \rho = 0.999$)
Input Size	$448 \times 448 \times 3$ (gray scale image with repeated channels)
Pretraining	Backbone pretrained on ImageNet

Table 4.4: Configuration of the YOLO grid search experiment. The networks were trained for all possible configurations of learning rate, batch size and loss function.

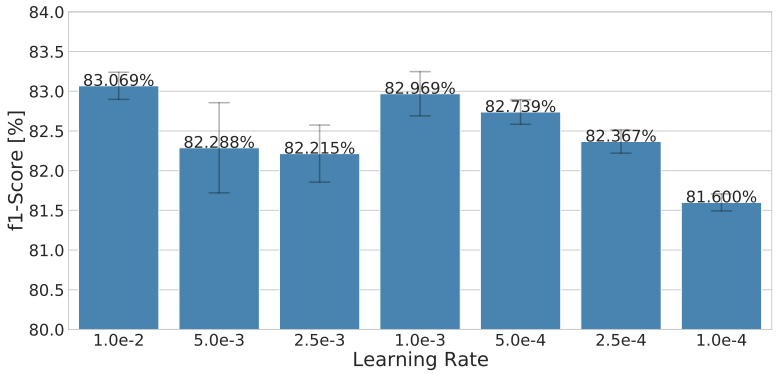


Figure 4.11: The results of the initial learning rate search shown on the validation set, with the mean and standard deviation of the F1-Score over three separate training runs.

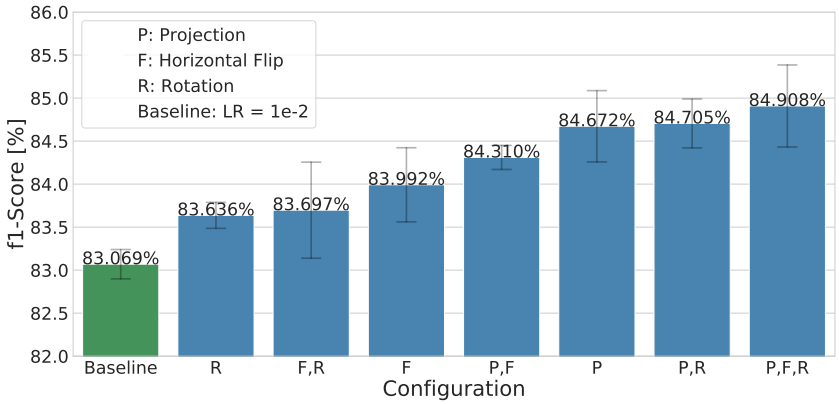


Figure 4.12: The results of the offline augmentation experiment shown on the validation set, as the mean F1-Score of three separate training runs.

Online Augmentations

The next experiment was done to find optimal parameters for the online augmentations. Here, the same augmentations were used as in the respective YOLO experiment, except for the crop augmentation, which is now not a safe crop, but just a plain random crop, which does not consider the segmentation labels. The results of this experiment can be found in figure 4.13. The only augmentation which did not increase the performance was the scale augmentation. It is hence not used in further experiments. From the other augmentations again the one with the best performing parameter was selected for the following grid search. In table 4.5 the selected augmentations with

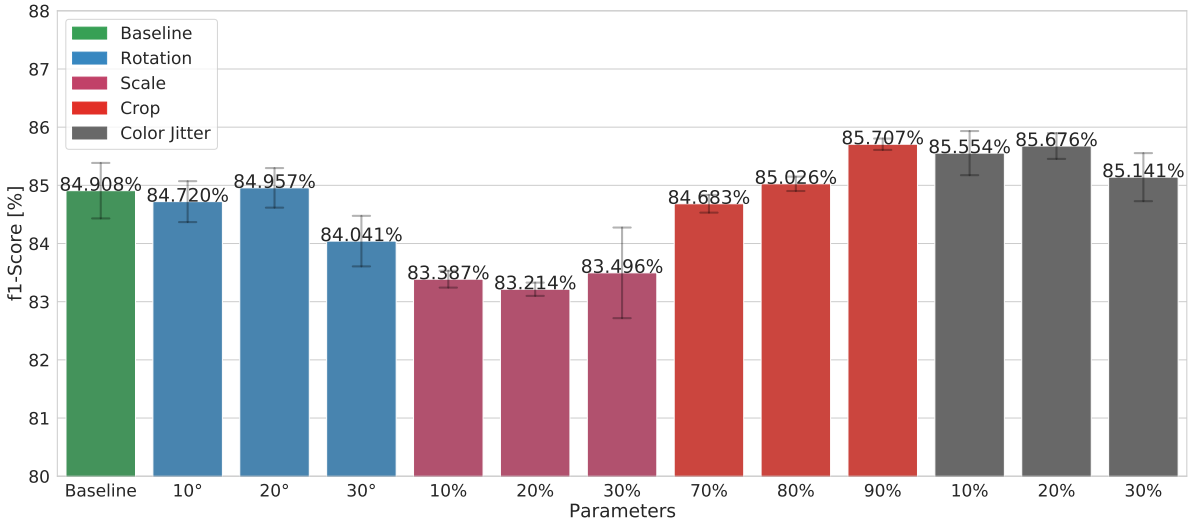


Figure 4.13: The results of the online augmentation experiment shown on the validation set, as the mean F1-Score of three separate training runs.

their respective parameters are shown.

4.2.3 Grid Search Experiment on Hyperparameters

The final training experiment of the MUnet was formed by an extensive grid search to obtain the optimal parameters for the training. The grid search was performed on the following parameters:

Learning Rates	$1.0e^{-2}, 5.0e^{-3}, 2.5e^{-3}, 1.0e^{-3}, 5.0e^{-4}, 2.5e^{-4}, 1.0e^{-4}$
Batch Size	$32, 64$
Losses	focal ($\alpha = 0.8, \gamma = 2$), focal ($\alpha = 0.1, \gamma = 2$), dice
Offline Augmentations	*projection, flip, rotation*
Online Augmentations	*rotation = 20°, crop = 90%, color jitter = 20%*

Table 4.5: Configuration of the MUnet grid search experiment, with tested grid search parameters and fixed offline and offline augmentation parameters. The networks were trained for all possible combinations of learning rate, batch size and loss function.

The results of the grid search experiment were first evaluated on the validation set and performance is shown for the three different metrics f1-score, recall and precision. Further, the evaluation was split up and the three metrics were calculated on the whole validation set and only on the checkered images in the validation set. This distinction was done to see whether the data is

biased towards non-checkered images or not. The results of the three metrics on the full validation set are shown in the figures B.1, B.2 and B.3, respectively. The results on only checkered images can be found in the figures B.4, B.5 and B.6, respectively. In the following a fold is defined as a combination of loss and batch size.

When looking at the f1-score on the full validation set (fig. B.1), no clear sign can be observed that one batch size performs better than another. Furthermore, all focal loss folds seem to be robust in terms of differing learning rates, however the same does not apply to the dice loss folds, which show with a decreasing learning rate, also a decrease in f1-score. The decrease in f1-score in the dice loss folds is related to a drop in precision (fig. B.3), but on the other hand-side an increase in recall (fig. B.2). The best performing learning rates can be found for the higher learning rates ranging from $2.5e^{-3}$ to $1.0e^{-2}$.

Inspecting now the f1-score on the validation set for checkered images only (fig. B.4), the best learning rates can be found in the middle and low range ($1.0e^{-4}$ - $1.0e^{-3}$), which leads to the assumption that higher learning rates probably perform better on non-checkered images, and lower learning rates perform better on checkered images. Comparing the different loss configurations based on their batch size shows that a higher batch size seems to perform slightly better on the checkered images. Furthermore, the dice loss against shows poor performance on small learning rates.

4.2.4 Final Results

For the final evaluation from each fold six networks are selected. This networks were chosen based on the best performing metrics, where three networks were chosen based on the best f1-score, precision and recall on the full validation set. And the other three were selected for the same metrics, based only on the checkered images from the validation set. If a network was the best performing on more than one metric no other network was selected from that fold, hence the amount of networks was reduced. The selected folds based on their best performing metric can be found in table B.1. Overall there were six folds (three loss functions · two batch sizes), each fold was tested for three metrics on two datasets ((f1-score, precision and recall) · (full validation set + checkered validation set)), therefore in theory there were a maximum of 36 network candidates. Some networks were the best on multiple metrics, therefore the actual number of selected networks was 22.

The selected networks were now used for further testing and the performance on the validation and test dataset can be found in the figures 4.14 and 4.15, for the batch size of 32 and 64 respectively. When looking at both fold types based on the batch size for the focal loss types a

decreasing trend in f1-score can be observed on the validation set, however on the test set the f1-score is increasing. This leads to the assumption that the high learning rate, chosen in the initial learning rate search experiment, leads to an overfitting on the validation set and a smaller learning rate would have been more adequate. Further, again the worst performing networks are those trained with the dice loss and a low learning rate combination, which already showed bad validation and now again showed bad test performance of around 50% f1-score. In general the best learning rates can be found in the middle and lower learning rate range.

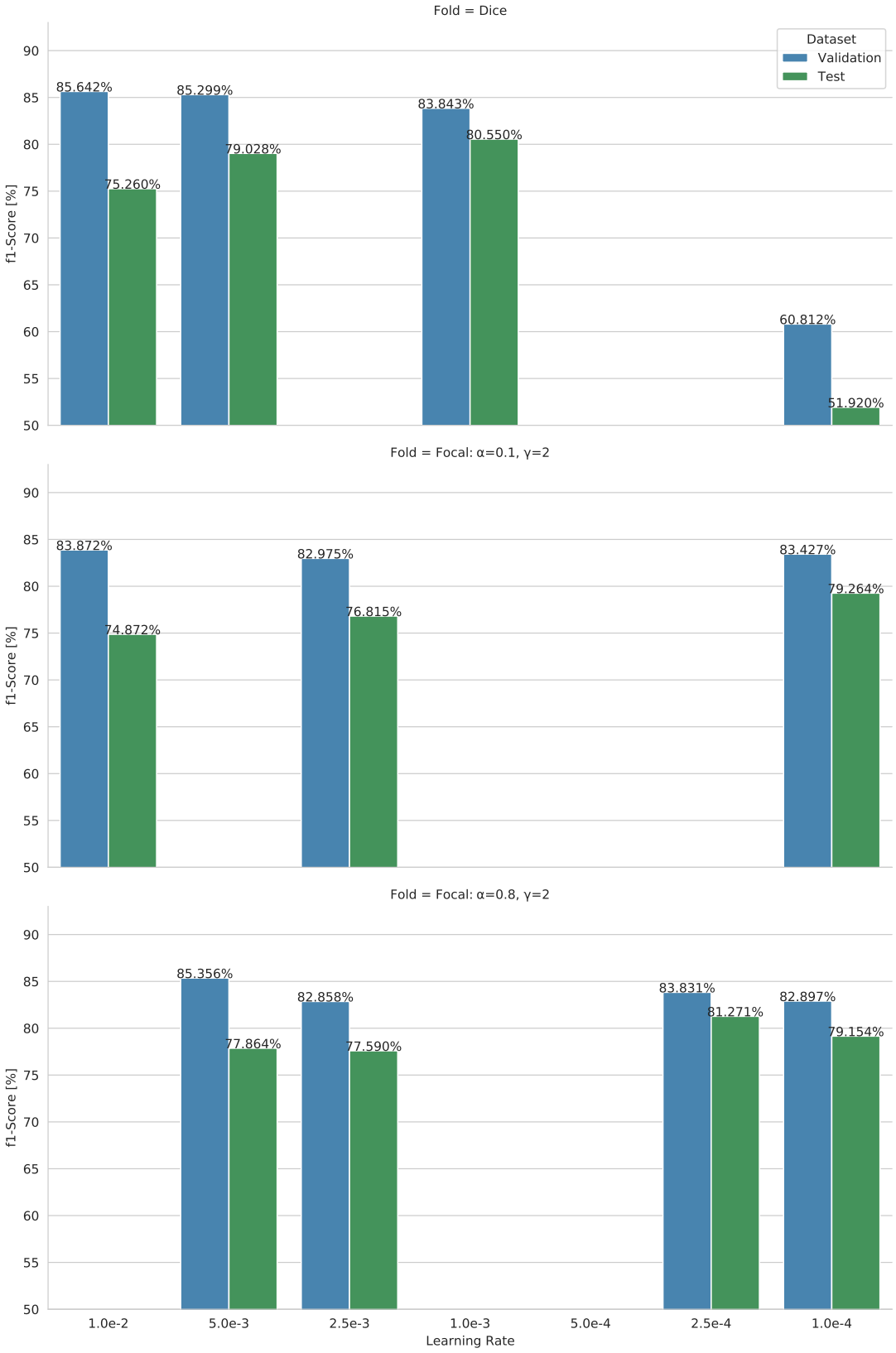


Figure 4.14: The results of the selected MUnet networks with a batch size of 32.

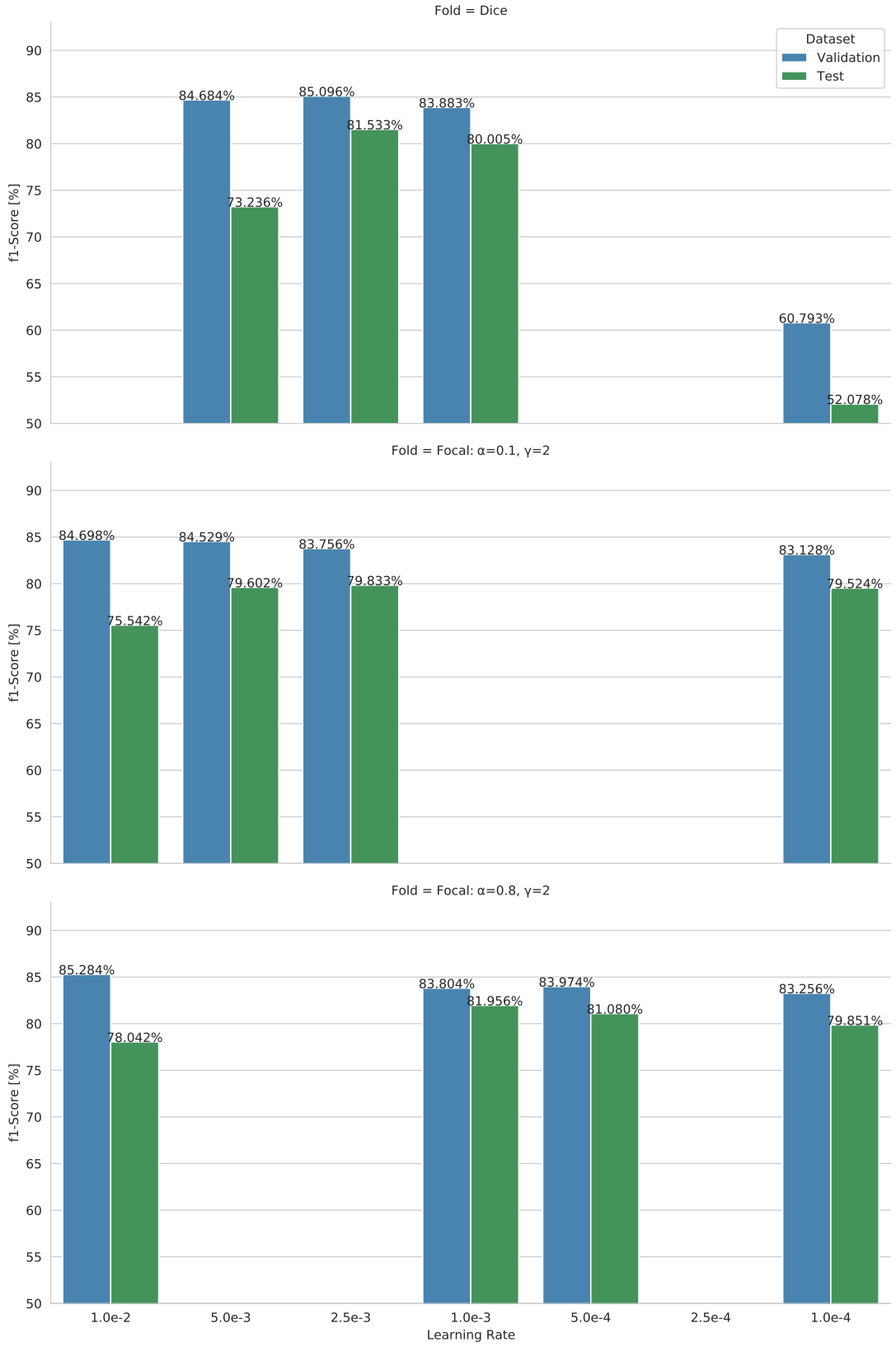


Figure 4.15: The results of the selected MUnet networks with a batch size of 64.

Chapter 5

Pipeline Evaluation

This chapter presents the evaluation of the pipeline as a whole system. First, the used evaluation algorithms are explained, together with the notions of TP, FP and FN in their respective context. For object detection this has already been explained in section 2.10. An overview of this chapter can be found in figure 5.1. At the end of this chapter, the results are presented with a thorough evaluation.

5.1 Evaluation Algorithm

5.1.1 Classification

The evaluation of the YOLO network was already done in section 4.1.5 based on the mAP metric. This metric considers multiple IoU thresholds in its calculation and provides a general detection performance. However, in the context of topology generation, a prediction with a bad IoU with its ground truth could still lead to a correct prediction of the topology. Therefore, for the evaluation of the full pipeline the matching of bounding box predictions is done using a fixed IoU threshold. To find an initial guess of the threshold, first the maximum occlusion IoU was obtained from the training and validation dataset, by calculating the IoU between each ground truth in an image. The maximum occurring IoU was found at 14%, which means that in general the dataset has low occlusion. Under the assumption that the size of the predicted bounding boxes will be greater than the size of the respective ground truth bounding boxes the IoU was chosen to be bigger than the maximum occlusion IoU. For the evaluation the IoU threshold was set to 30%. Furthermore, for the evaluation in this section, the predictions are provided with the TPs, FPs and FNs values, as well as with metrics, which can be calculated from those.

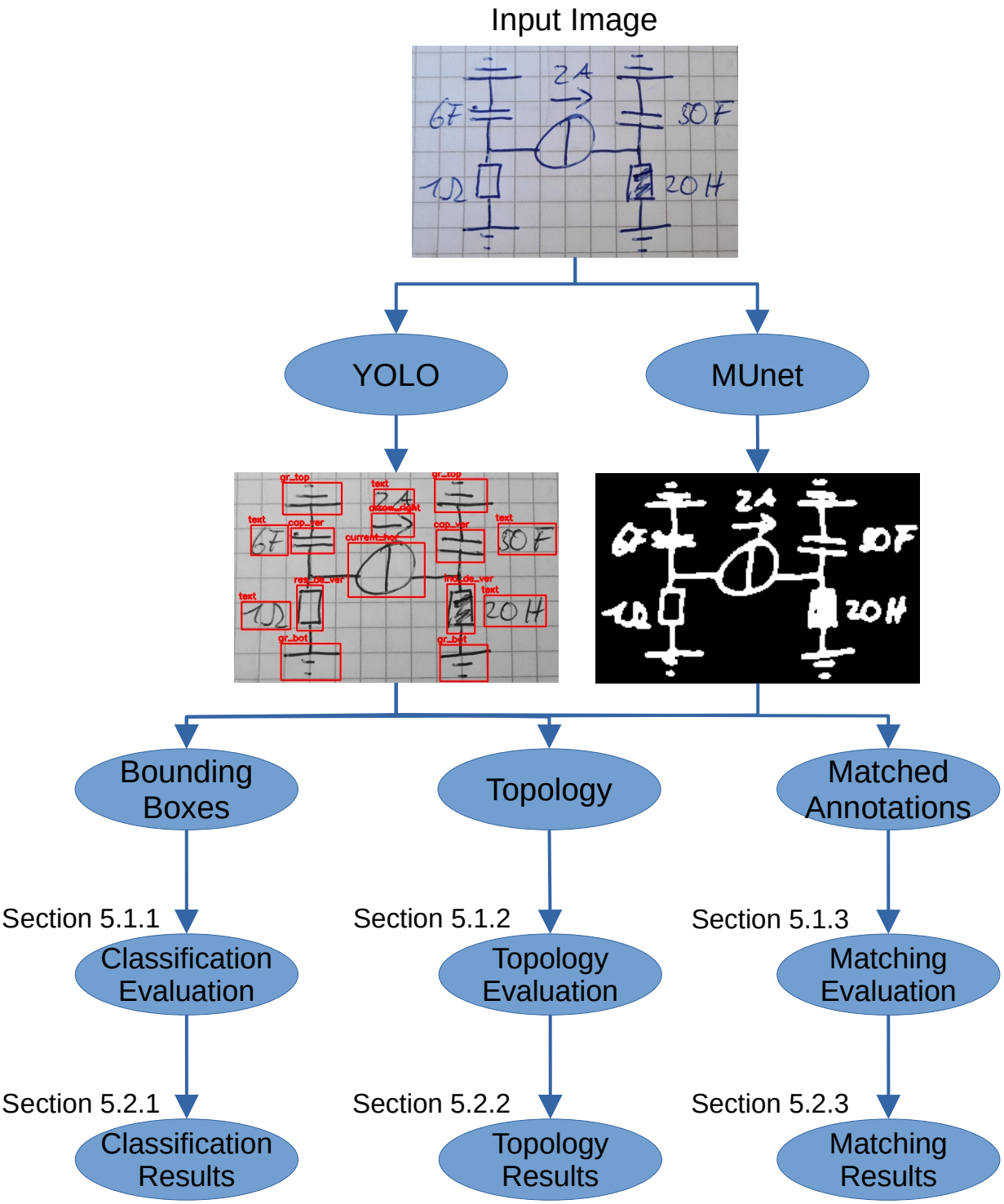


Figure 5.1: Overview of the evaluation pipeline. Evaluation is split into classification evaluation of bounding boxes, topology evaluation and matching evaluation.

The segmentation outcome is not further quantified since a re-quantification would just be a repetition of section 4.2.4, where the final results of the MUnet were presented. However, the outcome of the topology highly depends on the segmentation, hence a bad segmentation will result in a bad topology similarity between ground truth and prediction.

5.1.2 Topology

The similarity of the predicted and the ground truth topology is measured, based on a hypergraph adjacency matrix (section 3.1). To recall, a hypergraph is defined as: $\mathbf{H} = \{\mathbf{V}, \mathbf{E}\}$, with the vertices $\mathbf{V} = \{v_1, ..., v_n\}$ and the hyperedges, $\mathbf{E} = \{\mathbf{e_1}, ...\mathbf{e_n}\} = \{\{v_i, ..., v_j\}, ..., \{v_x, ..., v_y\}\}$. Further, a subedge and the amount of subedges present in an hyperedge is defined as: $\mathbf{S}(\mathbf{e_i}) = \{\{v_1, v_2\} \mid \{v_1, v_2\} \in \mathbf{e_i},\ \mathbf{e_i} \in \mathbf{E}\}$, $\#\mathbf{S}(\mathbf{e_i}) = \{\#\mathbf{e_i} - 1 \mid \mathbf{e_i} \in \mathbf{E}\}$, where # is the length operator for a set. For the presented algorithm the notation TP_h, FP_h and FN_h is used, where the $*_h$ indicates that those are quantizations related to the hypergraph evaluation.

The presented algorithm is inspired by the graph edit distance algorithm [Che18], which calculates the cost for transforming one graph into another. However, graph edit distance counts only the needed steps without considering the type of the error. Therefore, in this work the error type notions are also presented. The full algorithm is summarized in algorithm 5.1.

The subedges in prediction and ground truth are the same

This occurs when the tested ground truth hyperedge $\mathbf{e_i}$ and predicted hyperedge $\hat{\mathbf{e}}_\mathbf{i}$ have the same vertices in them. The amount of TP_h ($\#TP_h$) occurring from a perfect match is defined as:

$$\#ground\_truths = \#TP_h(\mathbf{e_i}) = \#S(\mathbf{e_i}) \tag{5.1}$$

A subedge is missing

This occurs when a perfect match can not be found directly, but only through recombination of predicted hyperedges. Consider the following example:

- Ground truth $\mathbf{H}$: $\mathbf{V} = \{v_1, v_2, v_3\}$, $\mathbf{E} = \{\{v_1, v_2, v_3\}\}$
- Predicted $\hat{\mathbf{H}}$: $\hat{\mathbf{V}} = \{v_1, v_2, v_3\}$, $\hat{\mathbf{E}} = \{\{v_1, v_2\}, \{v_3\}\}$

In this example no perfect match can be obtained between the ground truth and the prediction. However, recombining $\hat{\mathbf{e}}_\mathbf{1}$ and $\hat{\mathbf{e}}_\mathbf{2}$ would results in $\mathbf{e_1}$, which allows to calculate a perfect match.

Calculating a perfect match with equation 5.1 would result in $\#ground\_truths = 2$. However, the perfect match could only obtained through recombination. Therefore, there exists a $\#FN_h$, which is defined as:

$$\#FN_h = \#(recombined\ hyperedges\ for\ perfect\ match) - 1 \tag{5.2}$$

In this example $\#FN_h = \#(\{\{v_1, v_2\}, \{v_3\}\}) - 1 = 2 - 1 = 1$. Comparing the notion of ground truth, TP_h and FN_h to object detection, this shows a close relation and the amount of ground truths can be calculated through:

$$\#ground\_truths = TPs + FNs \tag{5.3}$$

The same notion applies here. Therefore, the $\#TP_h$ can be calculated using equation 5.3: $\#TP_h = \#ground\_truths - \#FN_h = 2 - 1 = 1$.

Naturally the equations 5.2 and 5.3 extend to the case, where the hyperedges of $\hat{\mathbf{H}}$ are $\hat{\mathbf{E}} = \{\{v_1\}, \{v_2\}, \{v_3\}\}$. Again, to have a perfect match all hyperedges have to be recombined, hence: $\#FN_h = \#(\{\{v_1\}, \{v_2\}, \{v_3\}\}) - 1 = 3 - 1 = 2$, $\#ground\_truths = 2$ and therefore $\#TP_h = \#ground\_truths - \#FN_h = 2 - 2 = 0$.

A subedge too much

This case occurs, when a perfect match can only be obtained by splitting a predicted hyperedge, due to $\mathbf{e_i} \subset \hat{\mathbf{e}}_\mathbf{j}$. Consider the following example of a ground truth and predicted hypergraph:

- Ground truth $\mathbf{H}$: $\mathbf{V} = \{v_1, v_2, v_3, v_4\}$, $\mathbf{E} = \{\{v_1, v_2\}, \{v_3, v_4\}\}$
- Predicted $\hat{\mathbf{H}}$: $\hat{\mathbf{V}} = \{v_1, v_2, v_3, v_4\}$, $\hat{\mathbf{E}} = \{\{v_1, v_2, v_3, v_4\}\}$

The hyperedge $\hat{\mathbf{e}}_\mathbf{1}$ is split into: $\mathbf{Split} = \{\{v_1, v_2\}, \{v_3, v_4\}\}$. A perfect match can now be obtained from the newly created hyperedges. The first perfect match is obtained from $\{v_1, v2\}$, with $\#TP_h = 1$ and the second one from the remaining hyperedge $\{v_3, v4\}$ also with $\#TP_h = 1$, therefore $\#TP_h = 1 + 1 = 2$. However, the perfect matches could only be obtained through splitting, hence additionally $FP_h = 1$, because it required one split to obtain the results.

Algorithm 5.1: Topology Evaluation Algorithm

Input: $\mathbf{H}$ (ground truth hypergraph)
$\hat{\mathbf{H}}$ (predicted hypergraph)
Output: TP_h, FP_h, FN_h

$\text{TP}_h \leftarrow 0$; $\text{FP}_h \leftarrow 0$; $\text{FN}_h \leftarrow 0$;
$\text{tp}_h \leftarrow$ find_all_perfect_matches($\mathbf{H}$, $\hat{\mathbf{H}}$);
$\text{TP}_h \leftarrow \text{TP}_h + \text{tp}_h$;
while $\mathbf{H} \neq \emptyset$ **do**
    tp_h, fn_h, matched_gt_idxs $\leftarrow$ find_perfect_matches_through_recombination($\mathbf{H}$, $\hat{\mathbf{H}}$);
    $\mathbf{H} \leftarrow$ remove_matched_idxs($\mathbf{H}$, matched_gt_idxs)
    $\text{TP}_h \leftarrow \text{TP}_h + \text{tp}_h$;
    $\text{FN}_h \leftarrow \text{FN}_h + \text{fn}_h$;

    tp_h, fp_h, matched_gt_idxs $\leftarrow$ find_perfect_matches_through_splitting($\mathbf{H}$, $\hat{\mathbf{H}}$);
    $\mathbf{H} \leftarrow$ remove_matched_idxs($\mathbf{H}$, matched_gt_idxs)
    $\text{TP}_h \leftarrow \text{TP}_h + \text{tp}_h$;
    $\text{FP}_h \leftarrow \text{FP}_h + \text{fp}_h$;
end
return TP_h, FN_h, FP_h;

Current Insufficiencies

At the current state, the algorithm is not able to handle mixed cases of FP_h and FN_h such as:

- Ground truth $\mathbf{H}$: $\mathbf{V} = \{v_1, v_2, v_3, v_4\}$, $\mathbf{E} = \{\{v_1, v_2\}, \{v_3, v_4\}\}$
- Predicted $\hat{\mathbf{H}}$: $\hat{\mathbf{V}} = \{v_1, v_2, v_3, v_4\}$, $\hat{\mathbf{E}} = \{\{v_1, v_3\}, \{v_2, v_4\}\}$

Here, neither through recombination, nor through splitting a perfect match can be identified. When this occurs all the remaining unmatched ground truth hyperedges are used and the worst case scenario is computed, which is that all possible remaining $\#TP_h$ are treated as $\#FN_h$.

Furthermore, the algorithm does not handle transient error cases in the topology evaluation, i.e. when a bounding box is not predicted by YOLO, this does not produce a FN_h.

5.1.3 Annotation Matching

To recreate the full topology in the LDom the textual as well as the arrow annotations have to be matched against their respective ECC. Hence, part of the evaluation is to quantify the amount of correctly and falsely matched annotations. The format of the matching labels and the matching algorithm, were already presented in section 3.1 and section 3.2, respectively. A TP being the simplest case occurs when an annotation is matched to its respective ECC. Further, a FN is always a transient error from the YOLO recognition, i.e. an annotation could not be predicted and hence there exists no annotation, which can be matched against an ECC. The last case being the FP case, which occurs when an annotation was matched with a wrong ECC.

5.2 Results

5.2.1 Classification

First the results for the plain classification based on the YOLO network are presented. The used networks were the tuned networks from figure 4.9, while in this figure the tuning increased the overall mAP, now the results are presented for an IoU threshold of 0.3 as plain TP, FP and FN (table 5.1).

NMS	**ScoreThr.**	**IoUThr.**	**InputSize**	**Vote**	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1**
DIoU	0.3	0.25	608×608	-	798	**6**	9	**99.25%**	98.88%	**99.07%**
DIoU	0.3	0.25	736×736	-	799	8	8	99.01%	99.01%	99.01%
DIoU	0.15	0.45	736×736	-	801	12	6	98.52%	99.26%	98.89%
WBF	0.15	0.25	736×736	-	801	12	6	98.52%	99.26%	98.89%
WBF-TTA	0.1	0.45	736×736	1	**804**	30	**3**	96.40%	**99.63%**	97.99%
WBF-TTA	0.1	0.45	736×736	2	**804**	21	**3**	97.45%	**99.63%**	98.53%
WBF-TTA	0.1	0.45	736×736	3	**804**	17	**3**	97.93%	**99.63%**	98.77%
WBF-TTA	0.1	0.45	736×736	4	**804**	13	**3**	98.41%	**99.63%**	99.01%
WBF-TTA	0.1	0.45	736×736	5	803	13	4	98.41%	99.50%	98.95%

Table 5.1: The results of the YOLO classification for the tuned networks presented in section 4.1. Classification was done based on a matching IoU threshold of 0.3.

Comparing the results based on the f1-score shows, that actually the baseline (first configuration) performed best. Increasing the input size added one TP, but also increased the amount of FPs by two. The further tuning of the DIoU-NMS and of the WBF algorithm also led to an

increase of two TPs, but also in a higher increase of FPs. Adding TTA shows that, the amount of TPs could be further increased, but again the trade-off is here higher increase in FPs vs. some increase in TP. Dhanushika Ranathunga The small gain by adding TTA can be explained, with the observations of [Sha20], they showed in their experiments, that TTA has especially a positive effect on networks, which are not well trained. Networks, which are trained well with enough data, as it seems to be the case here, actually started to perform worse when TTA was used. With the addition of the proposed voting mechanism in section 4.1, the amount of TPs can be maintained, while the amount of FPs can be decreased. It therefore seems to be the case that having a voting based exclusion approach, seems to work on the test dataset. When comparing the results to the ones of Dhanushika and Ranathunga [Dha19], who have obtained an f1-score of 94.85% on 478 object instances, the results in this thesis achieve a state-of-the-art performance of 99.07% on 828 object instances. Since, different datasets were used in both works the comparison is unfair and should just provide a general performance comparison.

5.2.2 Topology

The next presented results, are the results of the topology recognition. To have a comparable baseline the topology first was tested with the ground truth of the YOLO and the ground truth of the MUnet. Further, the baseline YOLO from table 5.1 was used to recognize the components. Since the evaluation of the topology currently does not consider transient errors from the YOLO recognition, the evaluation could also be performed with the ground truth, but it could be that the bounding box sizes heavily differ from the ground truth, so to make the evaluation as realistic as possible a trained YOLO is used instead of the ground truth. Further, the evaluation was performed for all MUnet networks from table B.1.

The results of the topology evaluation can be found in table 5.2, where the results are listed based on the fold configuration and the learning rate. The first listed network is the network, which was evaluated with both network ground truths. It can be observed that the evaluation with the ground truth still had 47 FNs, which are related to either mixed cases of FP and FN, as described in the evaluation algorithm section 5.1, or to false negatives produced, due to holes in the wire escaping the morphological closing. However, it overall shows the best performance over all networks with 620 TPs. In general it showed that all networks had a low amount of FPs (max. 7).

The fold trained with a batch size of 32 and the focal loss with an α of 0.1, shows generally an increase in TPs with decreasing learning rate and the smallest learning rate $1.0e^{-4}$ performed best. When looking at the metrics why this network was selected (table B.1), it can be seen that this

network had scored best on three metrics (recall full, f1-score checkered, recall checkered). This fold with the same loss function, but a batch size of 64 shows comparable performance, however the smallest learning rate was slightly better with one TP more. Interestingly the smallest learning rate was selected based on the same criteria as the networks with a batch size of 32.

The focal loss fold with an α of 0.8 and a batch size of 32 again shows that the higher learning rates had worse performance than the lower learning rates. The best performing learning rate was $2.5e^{-4}$, where this network was selected best on the best f1-score on the checkered dataset. Further, this loss function with a batch size of 64 showed the same results. Higher learning rates perform worse than lower ones. The best performing learning rate here was $5.0e^{-4}$ and was selected based on the best f1-score on the checkered dataset.

The last folds being the ones trained with the dice loss. First consider the fold with a batch size of 32. The biggest learning rate performs here poorly and the smallest exceptionally bad too. The configuration with the smallest learning rate was not able to reach half the performance of the other networks. The best network in that fold and actually overall the best, had a learning rate of $1.0e^{-3}$ with 531 TPs. This network was - as most of the others, which performed well - selected based on the best f1-score on the checkered validation set. The last remaining fold being the one with the dice loss and a batch size of 64. The smallest learning rate again showed very poor performance and could not reach half the performance of all other networks. Further, the two best performing learning rate was $2.5e^{-3}$, which is also the second best performing learning rate overall with 521 TPs. It was selected based on the best precision on the full validation set.

To summarize the presented results:

- Bigger learning rates showed generally a worse performance than smaller learning rates
- The smallest learning rates for the focal loss folds showed the best performance for those folds ($5.0e^{-4}$, $2.5e^{-4}$, $1.0e^{-4}$).
- Dice loss is exceptionally sensitive to the used learning rate and the smallest learning rates showed exceptionally poor performance.
- The dice loss folds had the two best performing learning rates overall ($2.5e^{-3}$, $1.0e^{-3}$)
- Most of the best performing networks were selected based on the best f1-score on the checkered validation set.

Batch Size	**Loss**	**Learning Rate**	$\mathbf{TP_h}$	$\mathbf{FP_h}$	$\mathbf{FN_h}$	**Precision**	**Recall**	**F1**
Ground Truth			620	0	47	100.0%	92.95%	96.35%
32	focal $\alpha = 0.1$	$1.0e^{-2}$	482	7	185	98.57%	72.26%	83.39%
32	focal $\alpha = 0.1$	$2.5e^{-3}$	488	**4**	179	99.19%	73.16%	84.21%
32	focal $\alpha = 0.1$	$1.0e^{-4}$	504	6	163	98.82%	75.56%	85.64%
64	focal $\alpha = 0.1$	$1.0e^{-2}$	464	7	203	98.51%	69.57%	81.55%
64	focal $\alpha = 0.1$	$5.0e^{-3}$	492	6	175	98.80%	73.76%	84.46%
64	focal $\alpha = 0.1$	$2.5e^{-3}$	491	**4**	176	99.19%	73.61%	84.51%
64	focal $\alpha = 0.1$	$1.0e^{-4}$	505	7	162	98.63%	75.71%	85.67%
32	focal $\alpha = 0.8$	$5.0e^{-3}$	485	5	182	98.98%	72.71%	83.84%
32	focal $\alpha = 0.8$	$2.5e^{-3}$	469	5	198	98.95%	70.31%	82.21%
32	focal $\alpha = 0.8$	$2.5e^{-4}$	498	5	169	99.01%	74.66%	85.13%
32	focal $\alpha = 0.8$	$1.0e^{-4}$	484	5	183	98.98%	72.56%	83.74%
64	focal $\alpha = 0.8$	$1.0e^{-2}$	492	5	175	98.99%	73.76%	84.54%
64	focal $\alpha = 0.8$	$1.0e^{-3}$	499	5	168	99.01%	74.81%	85.23%
64	focal $\alpha = 0.8$	$5.0e^{-4}$	511	**4**	156	99.22%	76.61%	86.46%
64	focal $\alpha = 0.8$	$1.0e^{-4}$	496	5	171	99.00%	74.36%	84.93%
32	dice	$1.0e^{-2}$	449	5	218	98.90%	67.32%	80.11%
32	dice	$5.0e^{-3}$	468	6	199	98.73%	70.16%	82.03%
32	dice	$1.0e^{-3}$	**531**	**4**	**136**	**99.25%**	**79.61%**	**88.35%**
32	dice	$1.0e^{-4}$	183	6	484	96.83%	27.44%	42.76%
64	dice	$5.0e^{-3}$	457	6	210	98.70%	68.52%	80.88%
64	dice	$2.5e^{-3}$	521	5	146	99.05%	78.11%	87.34%
64	dice	$1.0e^{-3}$	512	5	155	99.03%	76.76%	86.49%
64	dice	$1.0e^{-4}$	185	6	482	96.86%	27.74%	43.12%

Table 5.2: Results of the topology evaluation. The focal loss folds all have $\gamma = 2$. The used networks were the ones selected in table B.1.

NMS	Score Thr.	IoU Thr.	InputSize	Votes	TP	FP	FN	Precision	Recall	F1
DIoU	0.3	0.25	608	-	169	9	0	94.94%	100.00%	97.41%
DIoU	0.3	0.25	736	-	169	9	0	94.94%	100.00%	97.41%
DIoU	0.15	0.45	736	-	169	9	0	94.94%	100.00%	97.41%
WBF	0.15	0.25	736	-	169	9	0	94.94%	100.00%	97.41%
WBF-TTA	0.1	0.45	736	1	169	9	0	94.94%	100.00%	97.41%
WBF-TTA	0.1	0.45	736	2	169	9	0	94.94%	100.00%	97.41%
WBF-TTA	0.1	0.45	736	3	169	9	0	94.94%	100.00%	97.41%
WBF-TTA	0.1	0.45	736	4	169	9	0	94.94%	100.00%	97.41%
WBF-TTA	0.1	0.45	736	5	169	9	0	94.94%	100.00%	97.41%

Table 5.3: Results of the text matching, based on the YOLO configurations from table 5.1.

5.2.3 Annotation Matching

The last step in the evaluation of the whole pipeline is composed of an evaluation of the matching algorithm (algorithms 3.1 and 3.2) for the text and arrow annotations. The evaluation of the matching does not depend on the segmentation and is therefore only shown for the networks shown in table 5.1.

Table 5.3 shows the results of the text matching, where all configurations showed the same results. Every text annotation, which also had to be matched against an ECC was predicted, therefore the results have no FNs. The only error source were here FPs, which were produced either due to a FP prediction of the YOLO network, hence being a transient error, or generally due to wrong matching. Overall a matching precision of 94.94% could be achieved on the test dataset.

Further, for the matching of the arrow annotation the networks also show the same performance over all configurations. In the case of the arrow annotations the algorithm was able to match every annotation correctly to its respective ECC, therefore a precision of 100.00% was achieved. Further, since no FNs were observed in the prediction of the YOLO network, also no transient error applies here. Therefore, the arrow matching achieves perfect results.

NMS	Score Thr.	IoU Thr.	InputSize	Votes	TP	FP	FN	Precision	Recall	F1
DIoU	0.3	0.25	608	-	40	0	0	100.00%	100.00%	100.00%
DIoU	0.3	0.25	736	-	40	0	0	100.00%	100.00%	100.00%
DIoU	0.15	0.45	736	-	40	0	0	100.00%	100.00%	100.00%
WBF	0.15	0.25	736	-	40	0	0	100.00%	100.00%	100.00%
WBF-TTA	0.1	0.45	736	1	40	0	0	100.00%	100.00%	100.00%
WBF-TTA	0.1	0.45	736	2	40	0	0	100.00%	100.00%	100.00%
WBF-TTA	0.1	0.45	736	3	40	0	0	100.00%	100.00%	100.00%
WBF-TTA	0.1	0.45	736	4	40	0	0	100.00%	100.00%	100.00%
WBF-TTA	0.1	0.45	736	5	40	0	0	100.00%	100.00%	100.00%

Table 5.4: Results of the arrow matching

Chapter 6

Summary and Outlook

6.1 Discussion

YOLO

The YOLO network was with each experiment, iteratively improved on the validation set. With each experiment the previously established baseline could be improved. The highest improvement could be achieved through the offline augmentation experiment, since here, when an augmentation was applied the existing class with its orientation subclass was transformed into the same class with another orientation subclass. Especially, on the text and arrow class the offline augmentation experiment showed an improvement in mAP of 24.482% and 47.988%, respectively.

In table 5.1 the final results for the classification results for the topology were presented and the best network had a f1-score of 99.07%. This network was the baseline network from the grid search, which was not further tuned. This is an interesting observation, since the tuning results shown in figure 4.9 actually showed that the results based on the mAP metric improved. However, it seems that the mAP metric is probably not the best choice for the task of topology classification.

MUnet

The results of the MUnet experiments showed that a lower learning rate, would have been a better choice for the experiments. The choice for the best learning rate, was done based on the best f1-score. The f1-score was calculated on the whole validation set, where ECDs drawn on white paper background as well as on checkered paper background were used. Comparing the f1-score on the validation set with white paper background and checkered paper background, showed that the results were better on white paper background, therefore the f1-score was more biased towards

ECDs drawn on white paper background. This makes sense, since the segmentation of ECDs on white paper background is probably simpler than the segmentation of ECDs on checkered paper background. In general the segmentation did not perform as well as the object detection with YOLO and the results still require improvement.

Pipeline

The current pipeline algorithm (section 3.2) shows an approach how an image of ECDs can be converted into the LTspice format. In the pipeline the detection of the characters inside the detected text class were not considered and hence also not evaluated. However, some possible problems could be:

- The bounding boxes for the text class do not cover the text fully
- The characters can not be identified correctly

This would additionally introduce new error sources. When observing the results presented in table 5.2 the topology creation algorithm (section 3.2) could not produce perfect results with the ground truth from the YOLO and the ground truth from the MUnet. The root cause is here manifold. On the one hand, the evaluation algorithm can not handle mixed cases of FN_h and FP_h and when those occur the remaining unmatched ground truth hyperedges are taken as a worst case approximation of remaining errors, therefore introducing a high amount of FN_h. On the other hand, when inspecting the results of the topology creation, it could be observed that most FN_h were not related to a insufficient segmentation of the wires, but rather to already existing holes in the wires that were drawn that way.

The matching of the text classes in table 5.3 showed for each configuration the same results. This is due to the fact that, the TPs, FPs and FNs for the text class were the same in each configuration. Further, the matching relies on the center point of the predicted bounding box, which here did not influence the matching outcome. The same applies to the arrow matching in table 5.4.

Evaluation Algorithm

The results presented in 5.2 should be treated with care, since the current state of the topology evaluation algorithm has multiple drawbacks:

- Classification errors from the YOLO are not treated as errors in the topology

- Mixed cases of FN_h and FP_h can not be solved and induce a worst case error, where the remaining unmatched ground truth hyperedges are treated as FN_h

6.2 Future Work

The current classification stages of the presented pipeline (section 3.2) require two separate networks, an object detection network to identify the components of an ECDs and a segmentation network to segment the ECDs. As a future work these two stages of the pipeline could be unified through an instance segmentation model such as Mask-RCNN [He17], which would simplify the classification process due to reduced computational overhead.

Furthermore, at the current stage neither the object detection model, nor the segmentation model are able to identify holes between wires. While the preprocessing steps of the pipeline close some possible holes through morphological operations, still a non-negligible amount remains which is reflected in the evaluation. Possible solutions for that could be to either, to detect holes as a node with the object detection network, or try to make the segmentation network learn the notion of holes and make it predict a mask, where the holes are closed.

As mentioned, OCR was introduced as a necessary step in the pipeline, however it was not implemented. As a future work this step could be included in the pipeline.

6.3 Summary

Hand-drawn Electrical Circuit Diagrams (ECDs), consist of Electrical Circuit Components (ECCs) and are a way to represent an electrical circuit. While for humans the recognition of a circuit is simple, it is a challenging task for computers [Edw00] [Rab16]. Circuit simulation software, such as LTspice, are used in electrical science to model and simulate circuits. In LTspice the simulation is fast, however the modeling of the circuits can take much time. Especially when considering that the circuit to simulate already exists in the form of a drawing on paper. So to ease the use of applications like LTspice, the overall goal of this thesis was to develop a pipeline, which converts an image of a hand-drawn ECD into the LTspice schematic file format. The components of an ECD can be decomposed into the ECCs, the connections between ECCs, and annotations which describe an ECC. All those components have to be recognized to reflect a hand-drawn ECD in the digital world.

Recent works regarding this topic have mostly examined parts of the problem. In [Gü20], [Rab16] and [Roy20] only the classification of samples of ECCs was studied. The works by

[AD18] and [Moe18] extended the problem further and additionally included the detection of ECCs from an ECDs image. Afterwards, the detected ECCs were also classified. The most relation to this thesis has the work by Dhanushika and Ranathunga [Dha19]. In their work they detected the components of logical gates and recreated a boolean expression from the hand-drawn diagram. The detection of the components was done with an object detection network and the connections between the components were detected by utilizing various computer vision methods. What all works have in common is that checkered paper background, which is a common paper type used for drawing such diagrams, was not considered at all.

In this thesis, the hand-drawn ECDs are converted into the LTspice schematic file format, which is a human readable file format. It contains commands which build an ECD with its various components and annotations in the simulation software.

Deep learning methods are used in this thesis to detect ECC and to segment the ECD, where deep learning refers to the training of neural networks. Classification in neural networks is performed through a chain of trainable and non-trainable functions. The behavior of the trainable functions, which are also called layers, is defined by a weight matrix, which is optimized during the training process of an ANN. During training the error with respect to the prediction is calculated and the weights of the last layer are updated based on the gradient with respect to the weights. Utilizing the chain rule this is done for all previous layers of the network.

CNNs are networks which have shown to perform well on image data since spatial dependencies are captured better than in MLPs [LeC99]. In such networks, convolutions are used to perform calculations, where an input gets convolved with a trainable weight kernel. As in classical ANNs the weight kernels are optimized during training.

Object detection is one problem domain where CNNs are commonly used [Red15]. It is an extension of the classification task, where additionally to the class also a bounding box is predicted for an object in an image, where a bounding box is defined by the corner coordinates of the box enclosing an object. Two different types of object detectors exist: the two-stage detectors and the one-stage detectors. In two-stage detectors the prediction pipeline is split into two parts. First, probable regions are obtained where an object could be present and afterwards those regions are classified [Gir13]. Obtaining probable regions introduces an overhead and therefore one-stage detectors were invented, which internally model this process.

In this thesis the one-stage object detection architecture YOLOv4-Tiny [Wan21] was used to detect the ECCs and ECC annotations in the hand-drawn ECD. YOLO uses an encoding network to create features, which are then taken by the decoder network for further processing. The decoder upsamples the input feature maps and outputs the prediction of the bounding boxes. To

detect various object sizes the YOLO architecture has multiple output branches, which utilize different feature map resolutions from the encoder [Lin17a]. The prediction of YOLO often contains multiple bounding boxes per object, therefore NMS methods have to be applied in order to suppress unnecessary bounding boxes. The two NMS algorithms used in this thesis were the DIoU-NMS [Zhe19], which suppresses bounding boxes based on the DIoU, and the WBF algorithm [Sol19], which does not suppress bounding boxes, but rather fuses those together which have a certain IoU. Therefore, WBF is well suited for environments in which TTA is used and can enhance recognition performance.

Another problem in the image domain is the segmentation problem, which essentially is a classification problem, however instead of predicting the class of the whole image, in segmentation the class is predicted pixel-wise. Two types of segmentation exist. In instance segmentation the target is to predict the class and the instance of an object [He17], so when multiple instances of the same class are present in the prediction, they can be distinguished. On the other hand in semantic segmentation only the class in general is of interest [Net21].

The pipeline of this thesis requires the segmentation of the ECD. To accomplish this task, the MobileNetV2-UNet (MUnet) architecture [Jin20] was selected. As with YOLO the MUnet architecture has an encoder network to create features from the input image and a decoder which outputs a prediction map. The MUnet encoder is composed of the MobileNetV2 architecture [San19], which is a CNN architecture for resource constrained environments. The efficiency of the network comes through the usage of depthwise separable convolutions, which are a way to factorize a convolution. Depthwise separable convolutions in combination with skip connections and a linear bottleneck layer form the inverted residual block, which is the main building block of the MobileNetV2 architecture.

The topologies created in this thesis are represented as a hypergraph [Val21]. A hypergraph is a generalization of a graph, where one edge can connect more than two vertices. Hypergraphs can be represented as an adjacency matrix, where each row represents a hyperedge and the columns represent a vertex in the hypergraph [Ouv17].

Training of the two architectures YOLO and MUnet requires data. Therefore, in the scope of thesis a dataset was created, which contains 239 images of hand-drawn ECDs, created by 31 different persons. The drawings contain nine different ECC types and annotations, and were drawn on checkered as well as on white paper background. Further, the images were labeled with bounding boxes for the YOLO and with segmentation masks for the MUnet.

The full pipeline of this thesis can be summarized as follows:

1. Object detection of ECCs and related components (textual and arrow annotations)

2. Segmentation of the ECD
3. Creation of a topology
4. OCR of the textual annotations (not implemented in the scope of this thesis)
5. Matching of textual and arrow annotations against their respective ECC
6. Conversion into the LTspice format

As mentioned, for the object detection of the ECCs and annotations the YOLO architecture was used. The topology creation algorithm performed well on white paper background, however failed on checkered paper background. Therefore, segmentation with the MUnet was introduced to provide a clean mask of the underlying ECD. The topology creation algorithm uses as input the bounding boxes provided by the YOLO and the segmentation mask provided by the MUnet. First, the bounding boxes were removed from the segmentation mask, which results in an image containing only the wires. Using a connected components analysis the individual wires were labeled. This labeled mask was then intersected with a plain bounding box mask and the ECC, which were connected to each other were obtained, resulting in the topology of the ECD. Under normal circumstances the detected textual annotations, would have been processed by an OCR algorithm, however this was not implemented in the scope of this thesis. Further, the pipeline also contained the matching of the annotations to their respective ECC, which was done by calculating the nearest neighbor ECC. Afterwards, the gathered information is embedded into the LTspice schematic file format. The created module would also allow embedding the text of the annotations, but since they were not detected dummy values were embedded.

The aim of this thesis was to provide the best possible configuration for the YOLO and MUnet architecture. Therefore, an experimental setup was established in which the two networks were systematically parameterized. In the first step an initial learning rate search was performed. This was followed by experiments on various augmentations, such as rotation, flip, crop, color jitter and the copy-paste augmentation [Ghi20]. In the last step, a grid-search was performed, where various loss functions, learning rates and batch sizes were tested to find the optimal hyperparameters. For YOLO additionally a post training fine-tuning was performed on the DIoU-NMS, WBF algorithm and the WBF algorithm in combination with TTA. The results for both networks were measured on the test dataset, where for YOLO a mAP@0.5:0.75 of 95.492% was obtained. Comparing the classification results of the YOLO to the results of the related works (including plain classification), the YOLO trained in this thesis obtains state-of-the-art performance in the detection of ECCs.

Obviously, this comparison is not perfectly fair since different datasets were used and the used classes differ in shape. Furthermore, for the MUnet an f1-score of 81.9% was obtained.

The final step in this thesis was the evaluation of the whole system. Classification results of the YOLO were measured with a fixed IoU threshold of 0.3, which was obtained as the maximum occlusion appearing in the train and validation dataset. Here, the YOLO obtained an f1-score of 99.07% (precision 99.25%, recall 98.88%) on the test dataset. The results of the segmentation can not be measured directly, however the segmentation outcome is reflected in the topology results, i.e. a bad segmentation will lead to a bad topology outcome. For the evaluation of the topology an algorithm was implemented, which is closely related to the graph edit distance [Che18]. The graph edit distance measures the steps, which are needed to transform one graph into another. However, it does not consider the error type of the step (FP, FN), i.e. is this edge missing or is this edge too much. Therefore, the notions of TP_h, FN_h and FP_h, where $*_h$ refers to hyperedge, were introduced in order to classify the type of the error. The algorithm could not be finished and therefore the highest degree of error is assumed, in cases where the algorithm fails. Such evaluation has not been done so far and is one of the contributions of this thesis. During the evaluation of the topology an f1-score of 88.35% (precision 99.25%, recall 79.61%) was obtained. This results were compared to the baseline, which was established by running the topology evaluation with the ground truth of the YOLO and the ground truth of the MUnet. With the baseline an f1-score of 96.35% (precision 100.00%, recall 92.95%) was obtained. In the last step of the evaluation pipeline the annotation algorithm was evaluated. Textual and arrow annotations were evaluated separately. For the textual annotations an f1-score of 97.41% (precision 94.94%, recall 100.00%) was obtained and for the arrow annotations an f1-score of 100.00% (precision 100.00%, recall 100.00%) was obtained.

6.4 Conclusion

A pipeline was presented, which can convert an image of a hand-drawn ECD into the LTspice schematic file format. The pipeline utilizes deep learning methods to detect ECCs and annotations (object detection), and to segment the ECD (segmentation). The two utilized network architectures YOLOv4-Tiny and MUnet were evaluated on a test dataset, where the YOLO obtained state-of-the-art performance in ECC classification. Such a pipeline could be used as a plugin for LTspice to directly generate an ECD inside the software, from the image of a hand-drawn ECD, omitting the need of manual circuit modeling.

Abbrevations

ANN Artificial Neural Network

AP Average Precision

BatchNorm Batch Normalization

CAD Computer-Aided Design

CE Cross Entropy

CIoU Complete IoU

CNN Convolutional Neural Network

CSV Comma Separated Values

DIoU Distance IoU

ECC Electrical Circuit Component

ECD Electrical Circuit Diagram

EIoU Efficient IoU

FCOS Fully Convolutional One Stage Detector

FC Fully-Connected

FN False Negative

FP False Positive

GIoU Generalized IoU

HOG Histogram of Oriented Gradients

IDom Image Domain

IoU Intersection over Union

KNN K-Nearest-Neighbor

LDom LTspice Domain

LReLU Leaky ReLU

MLP Multilayer Perceptron

MSE Mean Squared Error

MUnet MobileNetV2-UNet

NMS Non-Maximum Suppression

OCR Optical Character Recognition

PANet Path Aggregation Network

R-CNN Regions with CNN features

ReLU Rectified Linear Unit

RoI Region of Interest

SGD Stochastic Gradient Descent

SMO Sequential Minimal Optimization

SSD Single Shot Multibox Detector

SVM Support Vector Machine

TN True Negative

TP True Positive

TTA Test-Time Augmentation

WBF Weighted Bounding Box Fusion

YOLOv1 You Only Look Once Version 1

YOLOv4 You Only Look Once Version 4

YOLO You Only Look Once

mAP Mean Average Precision

mIoU Mean Intersection over Union

List of Figures

List of Tables

Bibliography

[Aba15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AD18] A. Dhole A. Dewangan. KNN based hand drawn electrical circuit recognition. *International Journal for Research in Applied Science and Engineering Technology*, 6(6):1111–1115, jun 2018.

[Boc20] A. Bochkovskiy, C Wang, and H. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[Bod17] N. Bodla, B. Singh, R. Chellappa, and L. Davis. Improving object detection with one line of code. *CoRR*, abs/1704.04503, 2017.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[Bri90] J. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Françoise Fogelman Soulié and Jeanny Hérault, editors, *Neurocomputing*, pages 227–236, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[Bus20] A. Buslaevnder, V. Iglovikov, E. Khvedchenya, A Parinov, M. Druzhinin, and A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[Cao] R. Cao and C. Tan. Line primitive extraction by interpretation of line continuation.

[Che18] X. Chen, H. Huo, J. Huan, and J. Vitter. An efficient algorithm for graph edit distance computation. *Knowledge-Based Systems*, 10 2018.

[Dha19] T. Dhanushika and L. Ranathunga. Fine-tuned line connection accompanied boolean expression generation for hand-drawn logic circuits. In *2019 14th Conference on Industrial and Information Systems (ICIIS)*, pages 436–441, 2019.

[Dig19] M. Diganta. Mish: A self regularized non-monotonic neural activation function. *CoRR*, abs/1908.08681, 2019.

[Dum16] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[Edw00] B. Edwards and V. Chandran. Machine recognition of hand-drawn circuit diagrams. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, volume 6, pages 3618–3621 vol.6, 2000.

[Fel10] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[Ger15] R. Gershick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[Ghi20] G. Ghiasi, Y. Cui, A. Srinivas, R. Qian, T. Lin, E. Cubuk, Q. Le, and B. Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation, 2020.

[Gir13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[Goo16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[Gra10] C. Grana, D. Borghesani, and R. Cucchiara. Optimized block-based connected components labeling with decision trees. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 19(6):1596–609, Jun 2010.

[Gro73] S. Grossberg. Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, 52(3):213–257, 1973.

[Gü20] M. Günay, M. Köseoglu, and Ö. Yildirim. Classification of hand-drawn basic circuit components using convolutional neural networks. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–5, 2020.

[He15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[He17] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[How17] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[Hua16] G. Huang, Z. Liu, and K. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[Hui18] J. Hui. map (mean average precision) for object detection. `https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173`, 2018. Accessed: 20.04.2021.

[IEC] IEC. IEC-60617. `https://webstore.iec.ch/publication/2723`. Accessed: 21.12.2020.

[Iof15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[Jad20] S. Jadon. A survey of loss functions for semantic segmentation. *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, Oct 2020.

[Jin20] J. Jing, Z. Wang, M. Rätsch, and H. Zhang. Mobile-unet: An efficient convolutional neural network for fabric defect detection. *Textile Research Journal*, page 004051752092860, may 2020.

[Jun21] W. Jun. Two-phase weakly supervised object detection with pseudo ground truth mining, 2021.

[Kai19] D. Kaiwen, B. Song, X. Lingxi, Q. Honggang, H. Qingming, and T. Qi. Centernet: Keypoint triplets for object detection. *CoRR*, abs/1904.08189, 2019.

[Kin17] D. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[Kon01] E. Konstantinova and V. Skorobogatov. Application of hypergraph theory in chemistry. *Discrete Mathematics*, 235(1-3):365–383, may 2001.

[Kos20] R. Kosti, V. Christlein, and M. Stamminger. *Computer Vision Lecture*. Friedrich-Alexander-University, 2020.

[Law18] H. Law and J. Deng. Cornernet: Detecting objects as paired keypoints. *CoRR*, abs/1808.01244, 2018.

[LeC99] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In D. Forsyth, editor, *Feature Grouping*. Springer, 1999.

[Lew21] Y. Lewei, P. Renjie, X. Hang, Z. Wei, L. Zhenguo, and Z. Tong. Joint-detnas: Upgrade your detector with nas, pruning and dynamic distillation, 2021.

[Lin17a] T. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[Lin17b] T. Lin, P. Goyal, R. Girshick, K. He, and P. Doll. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[Liu] P. Liu. Single stage instance segmentation - a review. `https://towardsdatascience.com/single-stage-instance-segmentation-a-review-1eeb66e0cc49`. Accessed: 04.06.2021.

[Liu15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[Liu18] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534, 2018.

[Mai20] A. Maier, V. Christlein, K. Breininger, and S. Vesal. *Deep Learning Lecture*. Friedrich-Alexander-University, 2020.

[McC43] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophysics*, 5:115–143, 1943.

[Moe18] M. Moetesum, S. Younus, M. Warsi, and I. Siddiqi. Segmentation and recognition of electronic components in hand-drawn circuit diagrams. *EAI Endorsed Transactions on Scalable Information Systems*, 5(16), 4 2018.

[Mos20] N. Moshkov, B. Mathe, A. Kertesz-Farkas, R. Hollandi, and P. Horvath. Test-time augmentation for deep learning-based cell segmentation on microscopy images. *Scientific Reports*, 10(1), mar 2020.

[Net21] P. Neto. Deep learning based analysis of prostate cancer from mp-mri, 2021.

[Ouv17] X. Ouvrard, J. Le Goff, and S. Marchand-Maillet. Adjacency and tensor representation in general hypergraphs part 1: e-adjacency tensor uniformisation using homogeneous polynomials. *CoRR*, abs/1712.08189, 2017.

[Pas19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[Pow08] D. Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[Pri20] M. Pritt. Deep learning for recognizing mobile targets in satellite imagery, 2020.

[Rab16] M. Rabbani, R. Khoshkangini, H.S. Nagendraswamy, and M. Conti. Hand drawn optical circuit recognition. *Procedia Computer Science*, 84:41 – 48, 2016. Proceeding of the Seventh International Conference on Intelligent Human Computer Interaction (IHCI 2015).

[Ram21] A. Ramezani-Kebrya, A. Khisti, and B. Liang. On the generalization of stochastic gradient descent with momentum. *CoRR*, abs/2102.13653, 2021.

[Red15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[Red16] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger, 2016.

[Red18] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[Ref08] K. Refaat, W. Helmy, A. Ali, M. AbdelGhany, and A. Atiya. A new approach for context-independent handwritten offline diagram recognition using support vector machines. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 177–182, 2008.

[Ren15] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[Rez19] S. 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. *CoRR*, abs/1902.09630, 2019.

[Rod18] P. Rodríguez, M. Bautista, J. Gonzàlez, and S. Escalera. Beyond one-hot encoding: lower dimensional target embedding. *CoRR*, abs/1806.10805, 2018.

[Ron15] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[Roy20] S. Roy, A. Bhattacharya, and N. Sarkar et al. Offline hand-drawn circuit component recognition using texture and shape-based features. *Springer Science+Business Media*, August 2020.

[San19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[Sha20] D. Shanmugam, D. Blalock, G. Balakrishnan, and J. Guttag. When and why test-time augmentation works. *CoRR*, abs/2011.11156, 2020.

[She15] A. Shehata, S. Mohammad, M. Abdallah, and M. Ragab. A survey on hough transform, theory, techniques and applications. 02 2015.

[Sho19] C. Shorten and T. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), jul 2019.

[Smi09] R. Smith, D. Antonova, and D. Lee. Adapting the tesseract open source ocr engine for multilingual ocr. In *Proceedings of the International Workshop on Multilingual OCR*, page 1, 2009.

[Sol19] R. Solovyev and W. Wang. Weighted boxes fusion: ensembling boxes for object detection models. *CoRR*, abs/1910.13302, 2019.

[Sze14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[Tia20] Z. Tian, C. Shen, H. Chen, and T. He. FCOS: A simple and strong anchor-free object detector. *CoRR*, abs/2006.09214, 2020.

[Val21] P. Valdivia, P. Buono, C. Plaisant, N. Dufournaud, and J. Fekete. Analyzing Dynamic Hypergraphs with Parallel Aggregated Ordered Hypergraph Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(1):1–13, Jan 2021.

[vdS11] K. van de Sande, J. Uijlings, T. Gevers, and A. Smeulders. Segmentation as selective search for object recognition. In *2011 International Conference on Computer Vision*. IEEE, nov 2011.

[Wad16] K. Wada. labelme: Image Polygonal Annotation with Python. `https://github.com/wkentaro/labelme`, 2016.

[Wan21] C. Wang, A. Bochkovskiy, and H. Liao. Scaled-yolov4: Scaling cross stage partial network, 2021.

[Yu16] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. Huang. UnitBox. In *Proceedings of the 24th ACM international conference on Multimedia*. ACM, oct 2016.

[Zha21] Y. Zhang, W. Ren, Z. Zhang, Z. Jia, L. Wang, and T. Tan. Focal and efficient iou loss for accurate bounding box regression, 2021.

[Zhe19] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren. Distance-iou loss: Faster and better learning for bounding box regression, 2019.

Appendix

A YOLOv4-Tiny Experiments

Learning Rate	0.01		0.005		0.0025		0.001	
	mean	std	mean	std	mean	std	mean	std
mAP@0.5:0.75:0.05	72.352%	0.637%	71.223%	1.921%	71.474%	2.702%	**73.415%**	0.846%
diode left	74.175%	7.430%	77.938%	7.303%	75.364%	7.540%	75.587%	16.873%
diode top	93.976%	1.531%	97.446%	2.364%	77.711%	14.357%	87.505%	10.368%
diode right	79.058%	4.788%	79.734%	3.301%	86.005%	2.689%	87.168%	3.649%
diode bottom	74.403%	6.324%	69.694%	4.530%	78.748%	7.862%	79.644%	4.437%
resistor horizontal	80.652%	0.872%	77.821%	1.867%	76.449%	5.747%	75.920%	7.097%
resistor vertical	83.037%	4.325%	83.848%	5.982%	86.968%	3.789%	90.844%	2.174%
capacitor horizontal	92.595%	1.867%	90.544%	6.632%	88.616%	7.781%	87.420%	3.080%
capacitor vertical	79.086%	6.620%	78.762%	8.044%	70.272%	1.997%	78.861%	7.128%
ground left	85.183%	4.212%	85.352%	1.159%	82.464%	6.487%	87.967%	9.064%
ground top	61.676%	3.277%	56.253%	6.616%	64.206%	7.733%	63.528%	12.543%
ground right	79.280%	3.662%	82.551%	9.184%	85.100%	1.603%	78.510%	2.877%
ground bottom	89.499%	0.693%	86.624%	3.800%	86.354%	2.998%	82.913%	2.825%
inductor horizontal	84.975%	3.678%	83.396%	7.024%	85.205%	7.116%	89.902%	7.664%
inductor vertical	79.985%	6.390%	76.593%	6.189%	81.495%	1.126%	80.216%	7.083%
source horizontal	84.176%	6.806%	86.053%	3.953%	83.602%	4.300%	91.016%	3.320%
source vertical	88.539%	6.835%	90.478%	1.975%	90.274%	3.176%	92.630%	2.238%
current horizontal	87.637%	2.207%	86.639%	3.838%	86.958%	3.620%	89.566%	4.880%
current vertical	89.294%	3.717%	88.397%	4.205%	87.456%	2.205%	90.530%	4.228%
text	55.651%	7.837%	55.285%	2.696%	52.627%	4.047%	59.403%	1.547%
arrow left	21.396%	7.794%	13.283%	9.587%	18.018%	4.558%	13.468%	11.740%
arrow top	21.377%	9.568%	26.515%	8.849%	25.981%	3.613%	24.602%	9.146%
arrow right	44.958%	5.011%	37.513%	2.471%	39.517%	8.467%	44.401%	8.481%
arrow bottom	33.487%	11.453%	27.415%	10.686%	34.509%	12.261%	36.938%	4.996%

Learning Rate	0.0005		0.00025		0.0001			
	mean	std	mean	std	mean	std		
mAP@0.5:0.75:0.05	71.887%	1.109%	71.472%	0.385%	70.650%	1.425%		
diode left	82.741%	5.128%	81.244%	11.701%	84.043%	1.531%		
diode top	86.939%	5.819%	92.482%	5.291%	78.414%	12.199%		
diode right	84.688%	6.574%	79.312%	7.430%	79.763%	8.054%		
diode bottom	75.452%	12.993%	69.201%	5.885%	72.131%	11.203%		
resistor horizontal	85.571%	2.927%	85.971%	6.467%	84.247%	10.270%		
resistor vertical	85.043%	4.834%	91.121%	4.439%	85.381%	5.604%		
capacitor horizontal	88.360%	6.029%	90.408%	5.995%	83.446%	9.121%		
capacitor vertical	68.056%	4.011%	64.589%	7.098%	75.924%	11.799%		
ground left	84.352%	4.618%	85.688%	10.394%	86.833%	5.818%		
ground top	64.161%	1.523%	72.774%	6.100%	73.761%	5.064%		
ground right	84.012%	5.192%	81.670%	6.488%	79.685%	5.496%		
ground bottom	84.644%	2.516%	85.902%	1.206%	88.425%	3.036%		
inductor horizontal	86.034%	6.448%	89.681%	7.568%	78.955%	10.867%		
inductor vertical	81.418%	3.476%	81.187%	2.645%	83.662%	7.021%		
source horizontal	83.394%	5.462%	78.192%	0.585%	80.298%	5.585%		
source vertical	88.979%	6.943%	91.701%	3.748%	92.244%	2.296%		
current horizontal	87.417%	3.124%	86.055%	3.948%	84.849%	5.598%		
current vertical	90.420%	3.305%	87.512%	6.048%	86.889%	0.886%		
text	59.627%	1.322%	56.887%	4.524%	57.941%	4.932%		
arrow left	15.556%	4.516%	20.191%	3.788%	9.498%	3.472%		
arrow top	20.918%	0.593%	11.650%	2.983%	19.318%	7.161%		
arrow right	38.044%	7.739%	33.895%	10.660%	32.585%	4.073%		
arrow bottom	27.575%	6.488%	26.545%	0.680%	26.672%	12.922%		

Table A.1: The results of the initial learning rate search shown on the validation set, with mAP over all classes as well as the classwise AP. The mean and standard deviation are taken over three separate runs.

Offline Augmentation	P0F0R1		P0F1R0		P0F1R1		P1F0R0	
	mean	std	mean	std	mean	std	mean	std
mAP@0.5:0.75:0.05	87.140%	0.432%	84.415%	3.471%	91.113%	0.313%	79.403%	2.083%
diode left	92.234%	1.321%	91.294%	11.021%	90.803%	2.624%	85.666%	6.769%
diode top	94.578%	5.028%	97.637%	2.344%	97.840%	1.975%	93.787%	1.538%
diode right	91.243%	1.669%	93.301%	4.324%	94.277%	0.624%	94.306%	1.645%
diode bottom	90.823%	1.872%	84.067%	10.039%	95.018%	3.899%	73.765%	16.099%
resistor horizontal	97.046%	3.405%	94.047%	4.202%	98.570%	1.540%	84.394%	1.470%
resistor vertical	95.147%	5.910%	95.420%	3.709%	96.094%	4.142%	90.873%	6.573%
capacitor horizontal	93.635%	2.761%	96.318%	1.487%	97.084%	1.654%	94.134%	3.223%
capacitor vertical	91.722%	6.025%	91.730%	3.580%	94.169%	2.593%	88.688%	2.366%
ground left	94.520%	2.395%	89.788%	2.521%	94.771%	1.759%	92.257%	0.852%
ground top	89.806%	4.827%	83.703%	10.054%	89.804%	5.464%	79.250%	9.105%
ground right	87.742%	2.412%	90.646%	6.170%	94.216%	1.081%	82.389%	4.109%
ground bottom	90.629%	2.506%	97.206%	1.084%	96.884%	1.538%	91.053%	2.011%
inductor horizontal	97.763%	2.858%	86.493%	9.296%	97.326%	2.945%	85.473%	2.084%
inductor vertical	93.382%	3.552%	90.954%	4.338%	96.304%	2.290%	92.229%	6.615%
source horizontal	95.718%	5.079%	91.954%	4.404%	97.915%	1.475%	90.277%	0.305%
source vertical	98.336%	0.313%	96.963%	1.952%	98.584%	0.396%	87.670%	6.401%
current horizontal	95.852%	2.444%	96.302%	2.777%	98.804%	0.470%	92.569%	1.106%
current vertical	97.209%	2.439%	94.627%	2.585%	97.680%	2.010%	93.514%	5.561%
text	69.993%	1.410%	71.140%	5.502%	77.819%	3.669%	67.715%	2.369%
arrow left	42.737%	1.960%	33.867%	2.496%	57.432%	5.763%	24.293%	3.328%
arrow top	75.880%	7.437%	56.755%	11.619%	83.454%	1.074%	40.397%	8.439%
arrow right	59.931%	14.305%	55.652%	5.557%	70.008%	9.051%	45.171%	7.226%
arrow bottom	68.291%	3.777%	61.672%	3.133%	80.737%	5.176%	56.408%	11.852%

Offline Augmentation	P1F0R1		P1F1R0		P1F1R1		Baseline	
	mean	std	mean	std	mean	std	mean	std
mAP@0.5:0.75:0.05	89.641%	0.106%	86.129%	0.229%	**92.578%**	0.409%	73.415%	0.846%
diode left	93.597%	4.014%	91.181%	7.667%	92.333%	4.550%	75.587%	16.873%
diode top	91.681%	1.473%	97.798%	2.010%	96.948%	1.737%	87.505%	10.368%
diode right	91.757%	3.252%	89.021%	4.762%	93.518%	4.222%	87.168%	3.649%
diode bottom	94.280%	4.056%	81.750%	2.703%	95.016%	3.342%	79.644%	4.437%
resistor horizontal	95.806%	3.506%	93.281%	3.849%	97.322%	0.526%	75.920%	7.097%
resistor vertical	97.179%	1.734%	99.284%	0.649%	97.359%	1.835%	90.844%	2.174%
capacitor horizontal	99.293%	0.670%	97.800%	0.587%	98.232%	1.589%	87.420%	3.080%
capacitor vertical	94.622%	1.469%	87.713%	3.641%	94.118%	4.588%	78.861%	7.128%
ground left	96.220%	4.299%	94.495%	3.422%	96.770%	1.317%	87.967%	9.064%
ground top	93.892%	2.475%	83.231%	9.478%	93.935%	3.310%	63.528%	12.543%
ground right	91.405%	2.666%	83.262%	4.277%	90.997%	0.904%	78.510%	2.877%
ground bottom	98.080%	1.169%	94.569%	5.678%	98.260%	1.757%	82.913%	2.825%
inductor horizontal	97.153%	2.481%	93.622%	0.712%	99.342%	0.585%	89.902%	7.664%
inductor vertical	95.204%	3.354%	93.947%	1.989%	97.193%	1.906%	80.216%	7.083%
source horizontal	97.212%	1.229%	94.548%	3.643%	97.966%	1.765%	91.016%	3.320%
source vertical	96.283%	1.995%	94.320%	0.996%	96.674%	2.675%	92.630%	2.238%
current horizontal	97.694%	2.175%	97.485%	0.323%	100.000%	0.000%	89.566%	4.880%
current vertical	98.231%	1.539%	94.763%	3.435%	98.055%	2.632%	90.530%	4.228%
text	77.341%	0.219%	77.322%	1.650%	83.885%	1.505%	59.403%	1.547%
arrow left	54.932%	6.911%	50.885%	0.934%	65.458%	5.535%	13.468%	11.740%
arrow top	72.627%	2.817%	63.662%	2.807%	85.952%	3.736%	24.602%	9.146%
arrow right	63.998%	3.719%	54.102%	3.586%	79.383%	3.489%	44.401%	8.481%
arrow bottom	73.246%	6.320%	72.935%	4.459%	80.567%	1.846%	36.938%	4.996%

Table A.2: The results of the offline augmentation configuration search. The letter "P" stands for projection, "F" for flip and "R" means rotation. Rotation always includes a 90° 180° and a 270° rotation. Flip is a horizontal flip. All results are given with the mean and standard deviation over three runs and the results are compared against the baseline, which is the best performing learning rate from the learning rate search experiment (table A.1).

Rotation	10°		20°		30°		Baseline	
	mean	std	mean	std	mean	std	mean	std
mAP@0.5:0.75:0.05	**95.368%**	0.388%	94.521%	0.046%	94.198%	0.464%	92.578%	0.409%
diode left	93.276%	2.454%	93.879%	1.966%	95.265%	3.030%	92.333%	4.550%
diode top	99.327%	1.165%	98.394%	1.564%	99.436%	0.977%	96.948%	1.737%
diode right	99.198%	0.762%	96.956%	1.386%	97.650%	2.048%	93.518%	4.222%
diode bottom	95.149%	2.444%	94.065%	2.119%	91.071%	2.280%	95.016%	3.342%
resistor horizontal	100.000%	0.000%	100.000%	0.000%	98.245%	0.775%	97.322%	0.526%
resistor vertical	97.537%	2.241%	97.330%	2.359%	98.698%	2.255%	97.359%	1.835%
capacitor horizontal	97.545%	1.114%	97.043%	0.570%	96.277%	3.629%	98.232%	1.589%
capacitor vertical	94.609%	1.019%	92.022%	2.187%	95.924%	3.782%	94.118%	4.588%
ground left	98.785%	1.311%	98.399%	0.406%	94.828%	2.016%	96.770%	1.317%
ground top	96.769%	0.811%	95.385%	4.197%	93.485%	2.550%	93.935%	3.310%
ground right	98.152%	0.929%	96.896%	1.449%	93.781%	0.777%	90.997%	0.904%
ground bottom	97.540%	0.740%	97.735%	0.791%	98.538%	0.611%	98.260%	1.757%
inductor horizontal	99.449%	0.955%	98.613%	1.814%	99.441%	0.968%	99.342%	0.585%
inductor vertical	96.846%	1.177%	95.766%	2.795%	97.990%	1.334%	97.193%	1.906%
source horizontal	99.352%	0.575%	98.567%	1.195%	97.685%	2.474%	97.966%	1.765%
source vertical	99.188%	0.273%	98.998%	1.289%	98.932%	1.302%	96.674%	2.675%
current horizontal	100.000%	0.000%	100.000%	0.000%	100.000%	0.000%	100.000%	0.000%
current vertical	96.787%	0.755%	99.667%	0.577%	98.788%	1.118%	98.055%	2.632%
text	89.967%	5.288%	88.700%	3.258%	86.481%	3.616%	83.885%	1.505%
arrow left	86.635%	10.788%	78.791%	9.348%	78.060%	7.895%	65.458%	5.535%
arrow top	90.350%	3.327%	89.965%	1.384%	91.907%	3.234%	85.952%	3.736%
arrow right	82.225%	14.839%	81.451%	10.183%	84.894%	4.084%	79.383%	3.489%
arrow bottom	84.769%	3.433%	85.361%	6.232%	79.179%	5.651%	80.567%	1.846%

Table A.3: The results of the rotation augmentation, performed on different parameters and compared against the best offline augmentation configuration. The results are given with the mean and standard deviation over three runs.

Random Scale	0.1		0.2		0.3		Baseline	
	mean	std	mean	std	mean	std	mean	std
mAP@0.5:0.75:0.05	93.062%	0.307%	**93.261%**	0.743%	92.935%	0.630%	92.578%	0.409%
diode left	90.376%	3.263%	92.683%	1.615%	97.339%	1.487%	92.333%	4.550%
diode top	96.439%	3.219%	94.953%	6.245%	95.501%	2.195%	96.948%	1.737%
diode right	95.813%	1.919%	97.094%	0.732%	95.301%	0.948%	93.518%	4.222%
diode bottom	92.050%	1.504%	94.765%	2.123%	91.933%	0.849%	95.016%	3.342%
resistor horizontal	98.677%	1.859%	99.795%	0.355%	98.895%	1.914%	97.322%	0.526%
resistor vertical	98.205%	0.802%	97.411%	3.210%	96.905%	5.279%	97.359%	1.835%
capacitor horizontal	95.953%	4.320%	97.299%	2.102%	97.991%	2.703%	98.232%	1.589%
capacitor vertical	93.371%	2.172%	93.726%	2.925%	91.555%	2.602%	94.118%	4.588%
ground left	96.456%	2.114%	98.522%	1.368%	98.220%	2.505%	96.770%	1.317%
ground top	95.687%	3.178%	96.022%	2.692%	94.681%	0.516%	93.935%	3.310%
ground right	90.381%	6.591%	94.929%	3.106%	91.290%	3.461%	90.997%	0.904%
ground bottom	97.467%	1.141%	98.836%	1.141%	99.128%	0.865%	98.260%	1.757%
inductor horizontal	99.429%	0.990%	98.846%	1.999%	100.000%	0.000%	99.342%	0.585%
inductor vertical	98.271%	0.912%	96.650%	0.566%	96.546%	1.742%	97.193%	1.906%
source horizontal	98.650%	1.209%	99.322%	0.587%	98.884%	1.126%	97.966%	1.765%
source vertical	98.010%	1.461%	99.089%	0.603%	99.313%	0.417%	96.674%	2.675%
current horizontal	98.786%	1.183%	97.716%	1.907%	98.102%	1.603%	100.000%	0.000%
current vertical	99.239%	0.704%	98.560%	1.215%	96.507%	1.300%	98.055%	2.632%
text	85.191%	0.360%	85.457%	0.102%	83.989%	1.161%	83.885%	1.505%
arrow left	79.751%	1.490%	73.498%	8.638%	74.671%	13.037%	65.458%	5.535%
arrow top	80.715%	8.463%	82.648%	2.584%	86.412%	4.003%	85.952%	3.736%
arrow right	77.748%	6.507%	77.027%	6.180%	74.048%	12.438%	79.383%	3.489%
arrow bottom	83.764%	3.405%	80.166%	5.373%	80.294%	2.372%	80.567%	1.846%

Table A.4: The results of the random scale augmentation, performed on different parameters and compared against the best offline augmentation configuration. The results are given with the mean and standard deviation over three runs.

Bounding Box Safe Crop	0.7		0.8		0.9		Baseline	
	mean	std	mean	std	mean	std	mean	std
mAP@0.5:0.75:0.05	94.820%	0.098%	94.893%	0.358%	**95.027%**	0.486%	92.578%	0.409%
diode left	93.084%	1.078%	96.176%	3.373%	97.706%	1.229%	92.333%	4.550%
diode top	98.095%	2.406%	97.037%	3.262%	96.658%	2.062%	96.948%	1.737%
diode right	96.525%	0.898%	97.442%	0.936%	96.620%	2.598%	93.518%	4.222%
diode bottom	89.817%	9.159%	90.210%	3.847%	94.340%	2.061%	95.016%	3.342%
resistor horizontal	99.498%	0.870%	99.392%	1.053%	99.293%	0.626%	97.322%	0.526%
resistor vertical	98.882%	1.507%	99.382%	1.071%	97.768%	3.866%	97.359%	1.835%
capacitor horizontal	98.572%	0.565%	98.617%	1.381%	97.819%	1.962%	98.232%	1.589%
capacitor vertical	93.857%	1.789%	94.765%	5.410%	89.785%	5.612%	94.118%	4.588%
ground left	98.626%	1.246%	100.000%	0.000%	100.000%	0.000%	96.770%	1.317%
ground top	94.710%	2.108%	97.340%	2.478%	94.895%	3.148%	93.935%	3.310%
ground right	95.895%	0.476%	97.663%	2.088%	97.053%	1.536%	90.997%	0.904%
ground bottom	99.132%	0.889%	97.534%	2.976%	98.981%	1.764%	98.260%	1.757%
inductor horizontal	100.000%	0.000%	98.907%	1.023%	99.728%	0.471%	99.342%	0.585%
inductor vertical	98.317%	1.615%	97.755%	2.366%	97.282%	1.006%	97.193%	1.906%
source horizontal	100.000%	0.000%	100.000%	0.000%	100.000%	0.000%	97.966%	1.765%
source vertical	97.335%	3.013%	99.419%	0.083%	98.859%	1.230%	96.674%	2.675%
current horizontal	98.865%	1.059%	99.724%	0.477%	99.778%	0.385%	100.000%	0.000%
current vertical	95.113%	4.134%	96.791%	0.923%	97.675%	2.113%	98.055%	2.632%
text	91.133%	2.779%	90.303%	2.514%	87.787%	2.882%	83.885%	1.505%
arrow left	78.028%	0.953%	79.199%	2.284%	80.299%	6.205%	65.458%	5.535%
arrow top	93.190%	1.924%	90.054%	4.971%	92.811%	1.214%	85.952%	3.736%
arrow right	82.417%	9.390%	79.780%	4.322%	84.389%	5.589%	79.383%	3.489%
arrow bottom	89.762%	2.138%	85.042%	3.452%	86.092%	3.233%	80.567%	1.846%

Table A.5: The results of the safe bounding box crop augmentation, performed on different parameters and compared against the best offline augmentation configuration. The results are given with the mean and standard deviation over three runs.

Color Jitter	0.1		0.2		0.3		Baseline	
	mean	std	mean	std	mean	std	mean	std
mAP@0.5:0.75:0.05	92.656%	0.380%	**93.243%**	0.344%	93.182%	0.785%	92.578%	0.409%
diode left	90.749%	8.335%	89.807%	4.257%	91.810%	6.853%	92.333%	4.550%
diode top	97.418%	1.230%	98.221%	1.976%	97.977%	3.505%	96.948%	1.737%
diode right	96.042%	1.293%	95.140%	2.250%	96.634%	2.942%	93.518%	4.222%
diode bottom	93.131%	3.232%	94.387%	1.707%	91.644%	5.242%	95.016%	3.342%
resistor horizontal	97.539%	2.276%	100.000%	0.000%	99.154%	1.465%	97.322%	0.526%
resistor vertical	99.156%	1.033%	98.329%	1.549%	99.584%	0.375%	97.359%	1.835%
capacitor horizontal	95.483%	2.615%	98.257%	2.093%	94.332%	2.224%	98.232%	1.589%
capacitor vertical	93.641%	1.915%	97.297%	0.249%	96.518%	2.333%	94.118%	4.588%
ground left	100.000%	0.000%	96.796%	2.775%	99.110%	0.771%	96.770%	1.317%
ground top	97.822%	1.284%	96.667%	1.637%	94.929%	3.812%	93.935%	3.310%
ground right	93.658%	2.735%	94.323%	3.961%	95.661%	0.906%	90.997%	0.904%
ground bottom	97.196%	1.605%	96.826%	2.433%	97.260%	1.107%	98.260%	1.757%
inductor horizontal	99.524%	0.587%	98.293%	2.956%	98.193%	2.574%	99.342%	0.585%
inductor vertical	93.929%	3.609%	98.109%	1.610%	98.220%	1.397%	97.193%	1.906%
source horizontal	100.000%	0.000%	97.994%	1.753%	94.880%	7.009%	97.966%	1.765%
source vertical	97.856%	2.019%	98.334%	0.820%	98.891%	1.004%	96.674%	2.675%
current horizontal	98.664%	1.284%	99.644%	0.616%	97.508%	0.350%	100.000%	0.000%
current vertical	98.544%	1.361%	97.644%	0.543%	97.690%	1.207%	98.055%	2.632%
text	83.194%	1.732%	85.632%	0.741%	81.753%	4.080%	83.885%	1.505%
arrow left	66.081%	9.052%	69.506%	7.001%	74.520%	4.022%	65.458%	5.535%
arrow top	82.928%	2.215%	80.955%	4.039%	85.429%	1.046%	85.952%	3.736%
arrow right	79.251%	5.624%	78.987%	6.857%	82.125%	2.520%	79.383%	3.489%
arrow bottom	79.281%	5.008%	83.450%	3.019%	79.374%	4.998%	80.567%	1.846%

Table A.6: The results of the color jitter augmentation, performed on different parameters and compared against the best offline augmentation configuration. The results are given with the mean and standard deviation over three runs.

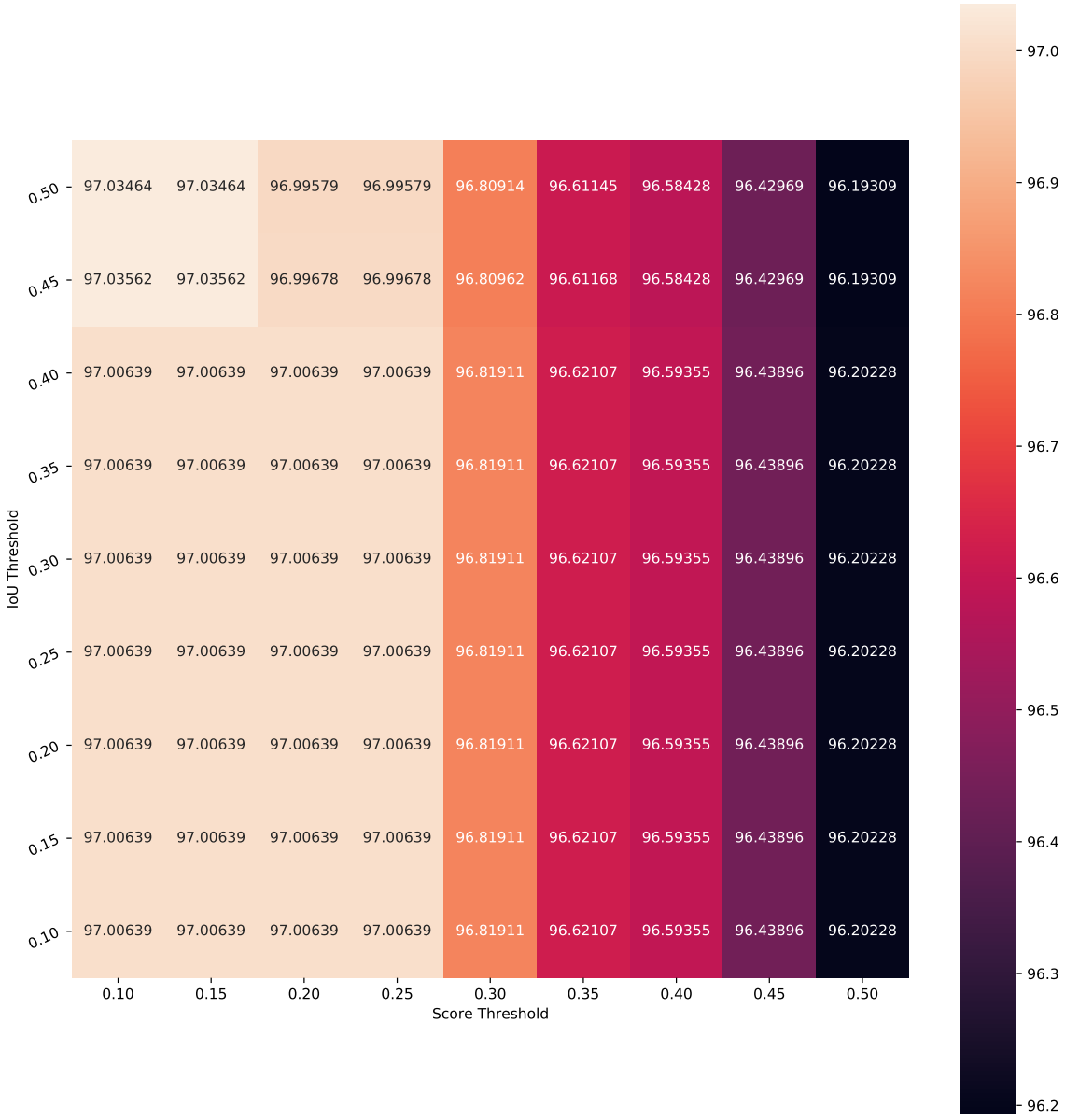


Figure A.1: The results of the DIoU-NMS parameter tuning. All possible combinations of score threshold and IoU threshold were evaluated on the validation set. The best performing combination is the one with a score threshold of 0.1 and a IoU threshold of 0.45.

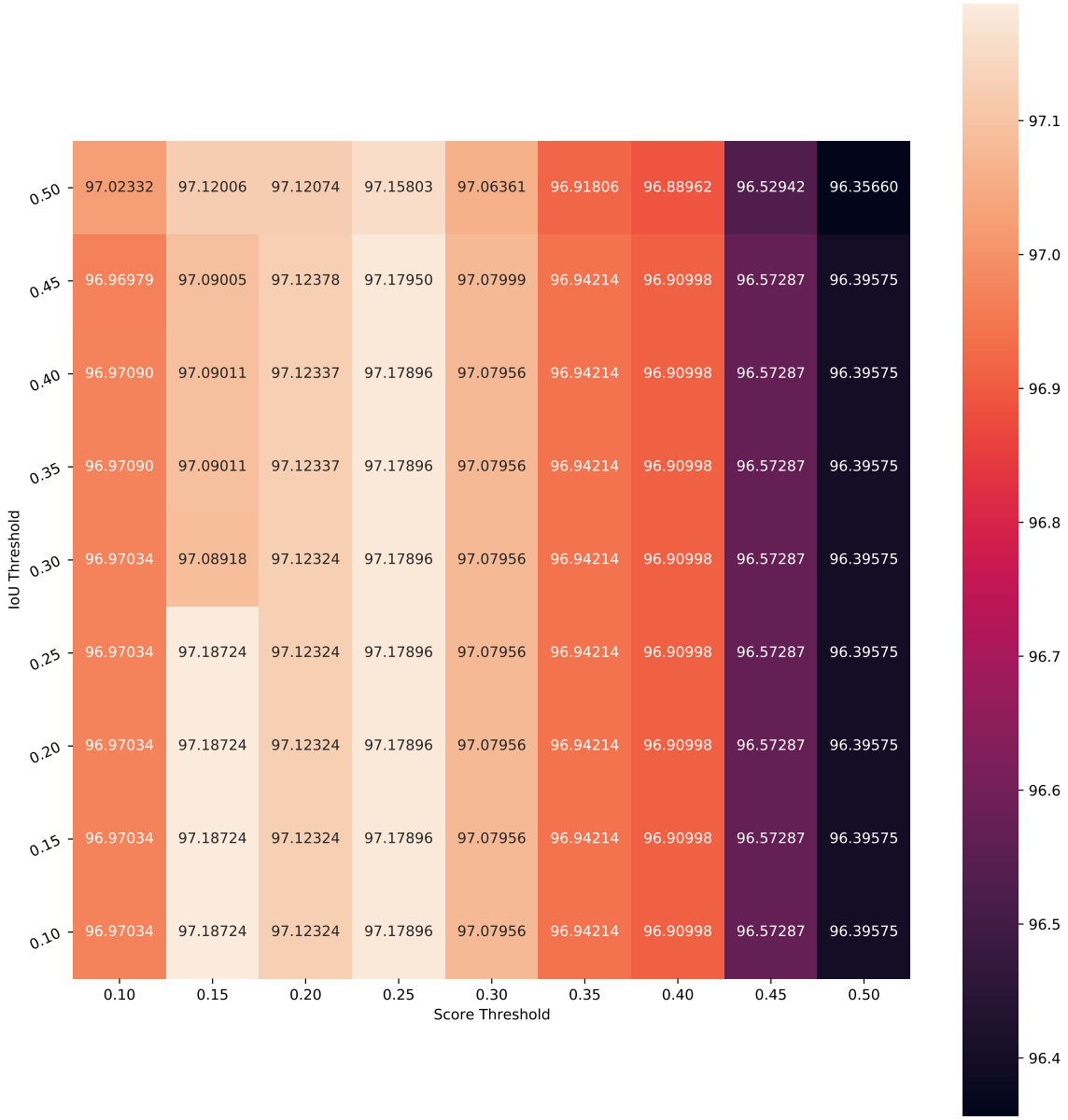


Figure A.2: The results of the WBF parameter tuning. All possible combinations of score threshold and IoU threshold were evaluated on the validation set. The best performing combination is the one with a score threshold of 0.15 and an IoU threshold of 0.25.

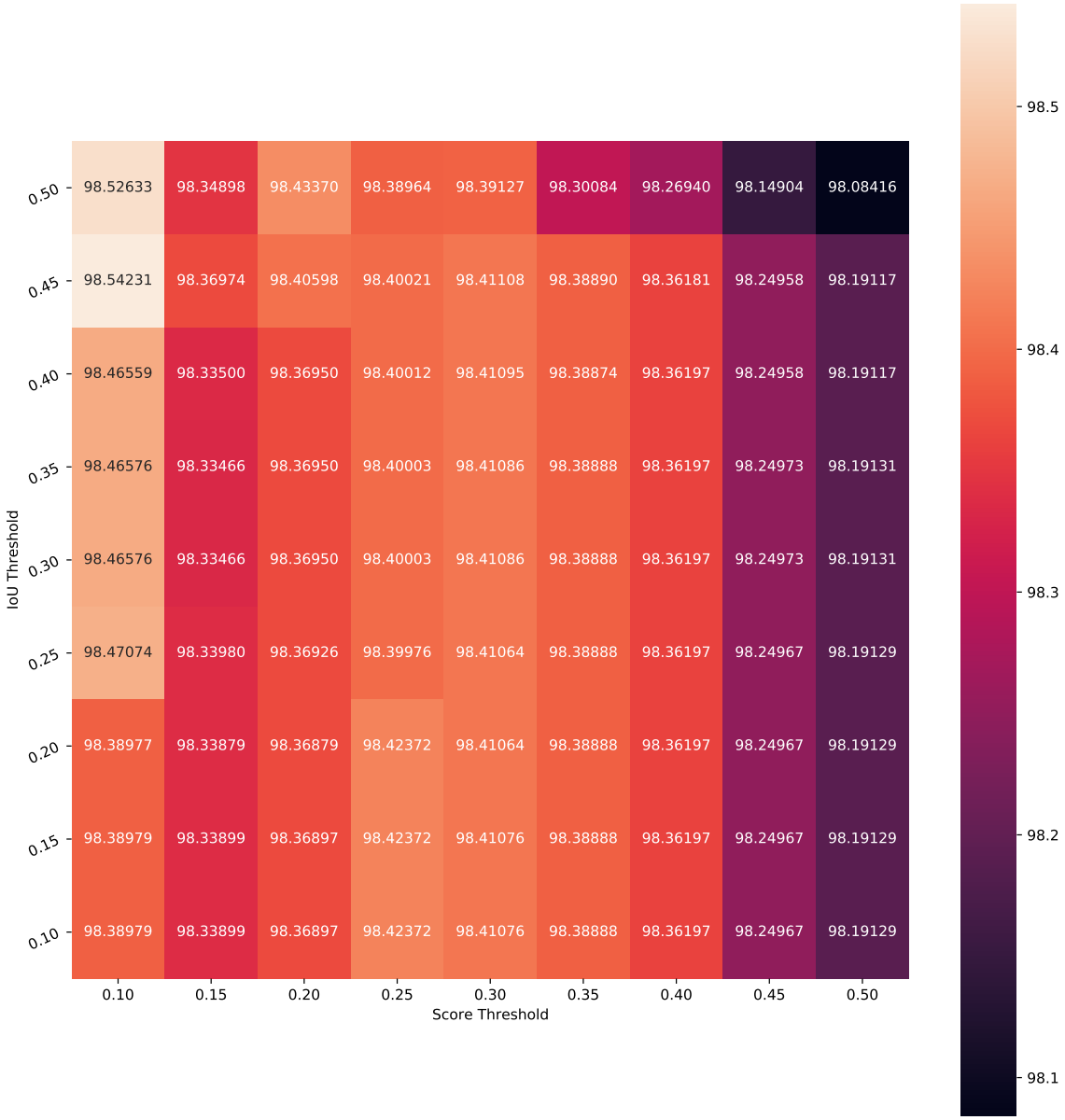


Figure A.3: The results of the WBF tuning with TTA. All possible combinations of score threshold and IoU threshold were evaluated on the validation set. The best performing combination is the one with a score threshold of 0.1 and an IoU threshold of 0.45.

B MobileNetV2-UNet Experiments

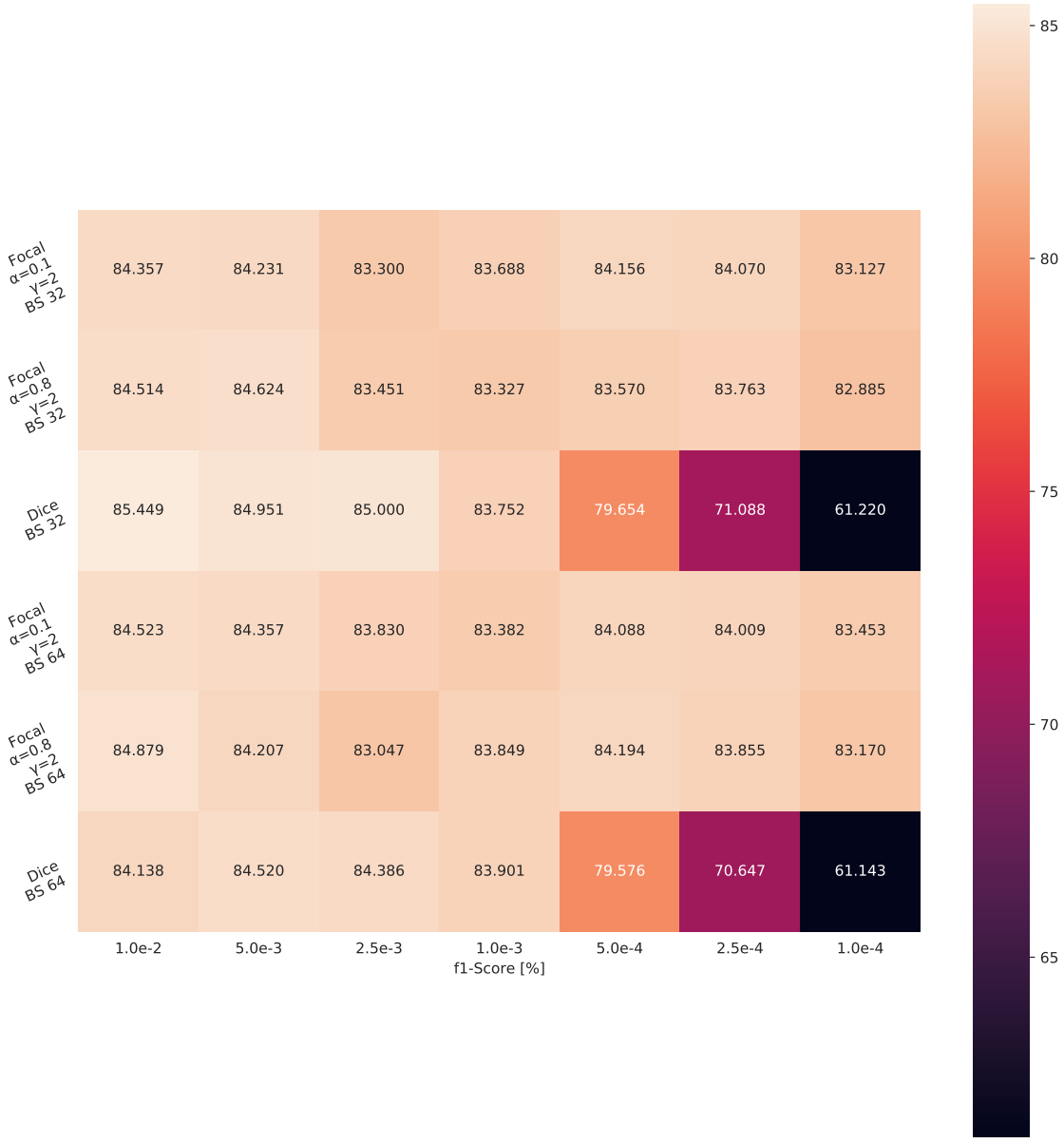


Figure B.1: F1-Score results of the MUnet grid search experiment shown on the full validation data. Results are shown combined for the loss and batch size against the learning rate.

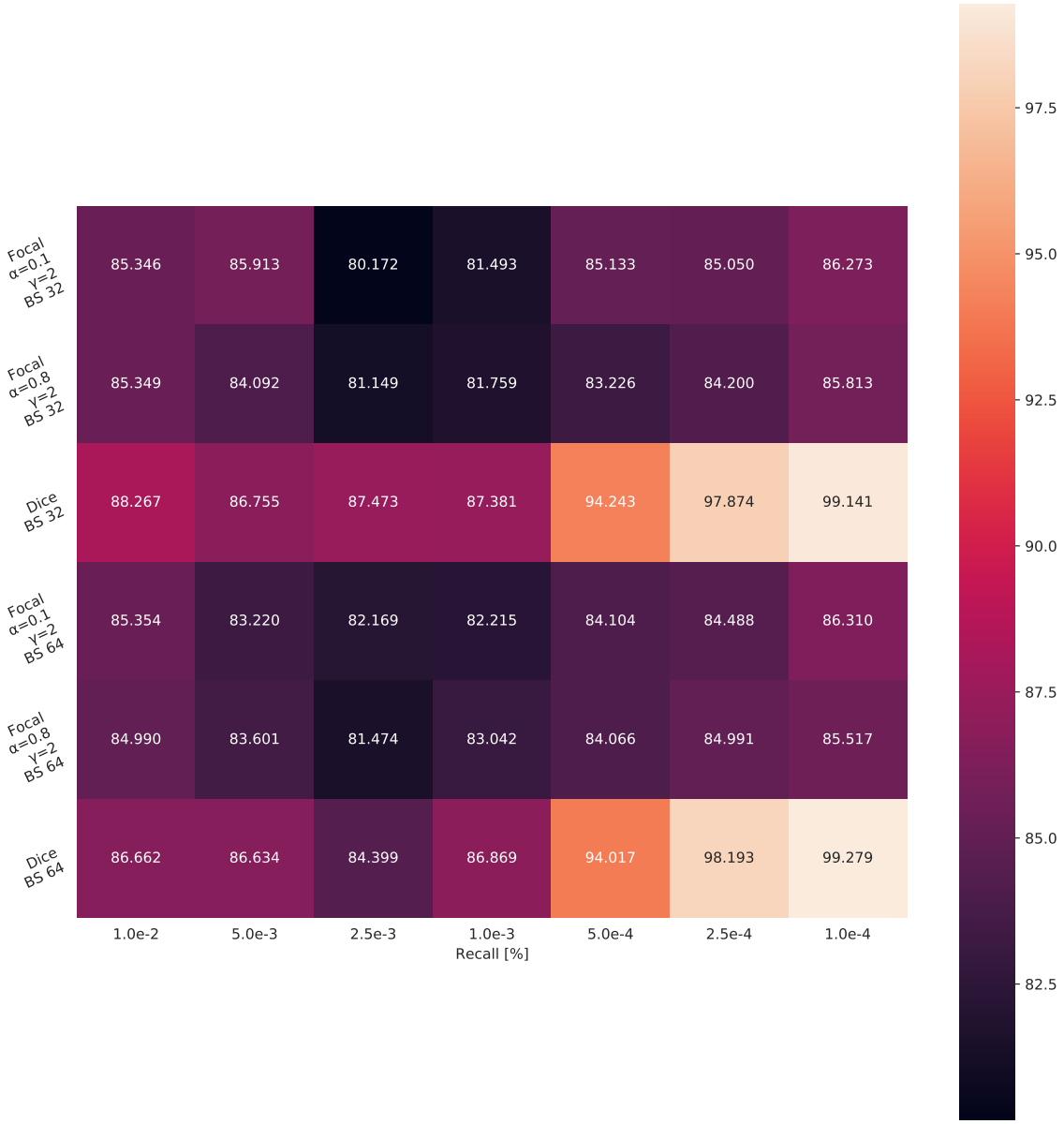


Figure B.2: Recall results of the MUnet grid search experiment shown on the full validation data. Results are shown combined for the loss and batch size against the learning rate.

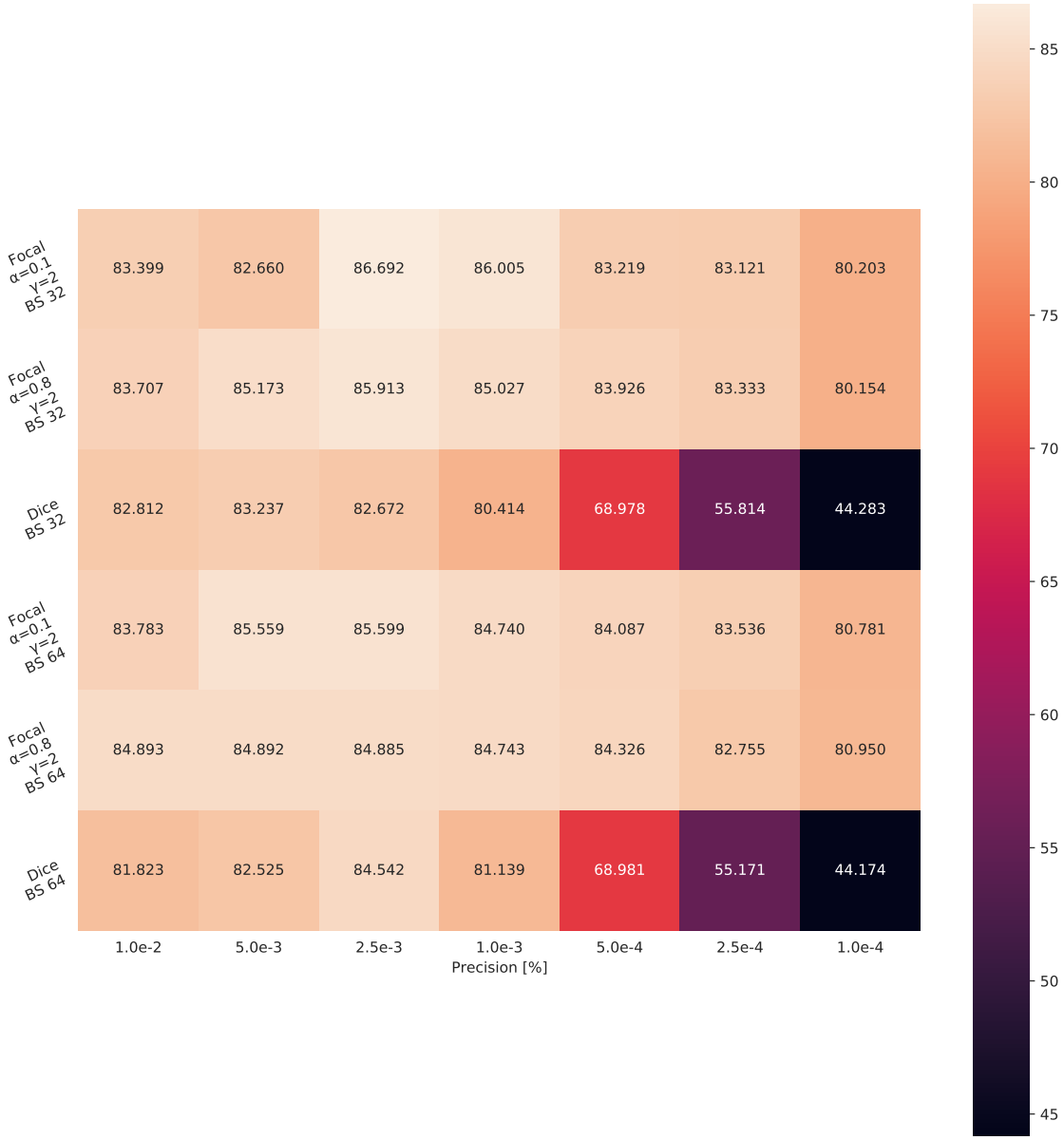


Figure B.3: Precision results of the MUnet grid search experiment shown on the full validation data. Results are shown combined for the loss and batch size against the learning rate.

Figure B.4: F1-Score results of the MUnet grid search experiment shown validation data, for checkered images only. Results are shown combined for the loss and batch size against the learning rate.

Figure B.5: Recall results of the MUnet grid search experiment shown validation data, for checkered images only. Results are shown combined for the loss and batch size against the learning rate.

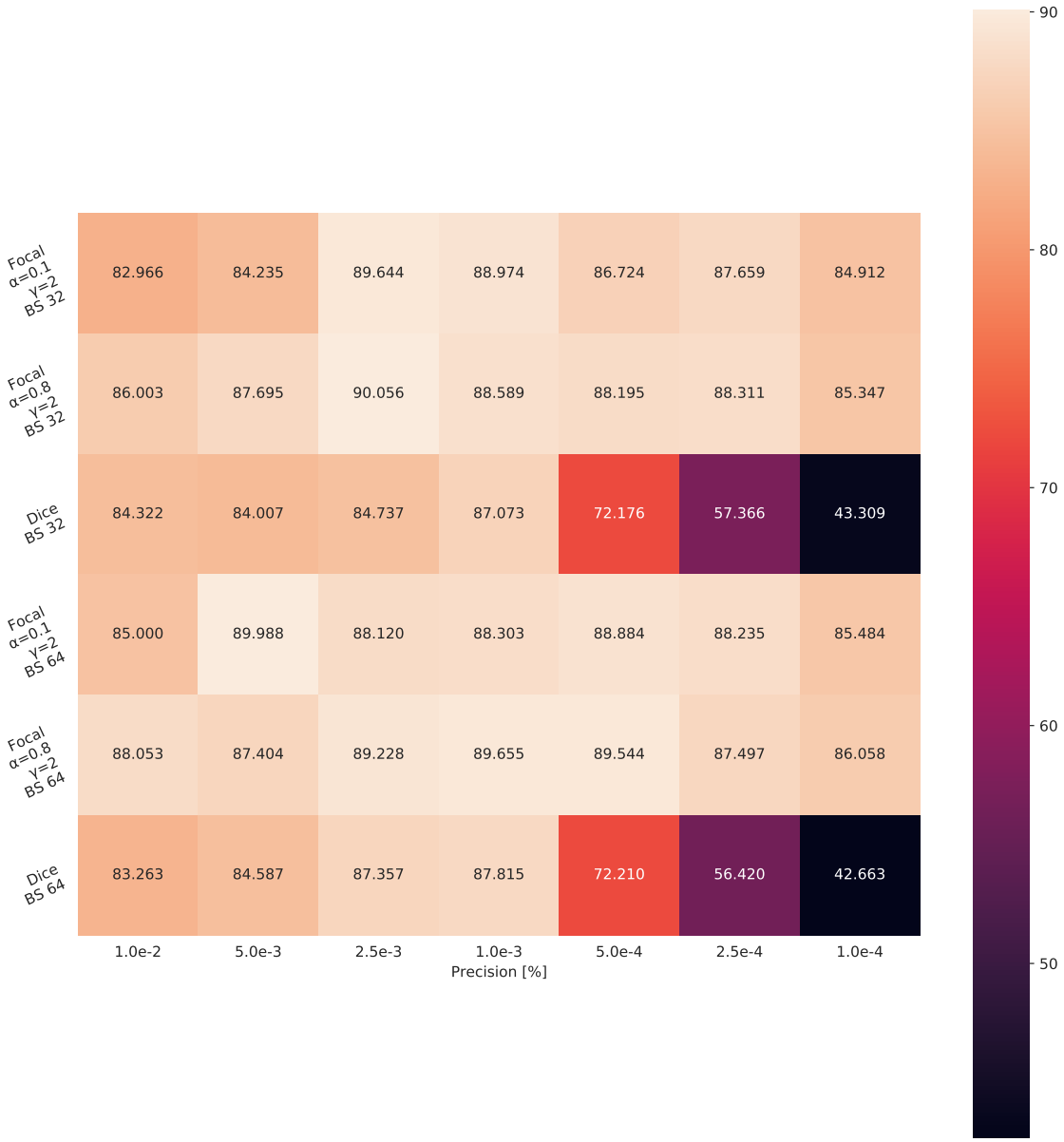


Figure B.6: Precision results of the MUnet grid search experiment shown validation data, for checkered images only. Results are shown combined for the loss and batch size against the learning rate.

Batch Size	Loss	Learning Rate	Dataset Type	Metric	Metric Value
32	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-2}$	Full	F1	84.3573
32	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-4}$	Full	Recall	86.273
32	Focal $\gamma = 2$, $\alpha = 0.1$	$2.5e^{-3}$	Full	Precision	86.6923
32	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-4}$	Checkered	F1	83.779
32	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-4}$	Checkered	Recall	82.6813
32	Focal $\gamma = 2$, $\alpha = 0.1$	$2.5e^{-3}$	Checkered	Precision	89.6437
32	Focal $\gamma = 2$, $\alpha = 0.8$	$5.0e^{-3}$	Full	F1	84.624
32	Focal $\gamma = 2$, $\alpha = 0.8$	$1.0e^{-4}$	Full	Recall	85.8127
32	Focal $\gamma = 2$, $\alpha = 0.8$	$2.5e^{-3}$	Full	Precision	85.9127
32	Focal $\gamma = 2$, $\alpha = 0.8$	$2.5e^{-4}$	Checkered	F1	83.8423
32	Focal $\gamma = 2$, $\alpha = 0.8$	$1.0e^{-4}$	Checkered	Recall	82.273
32	Focal $\gamma = 2$, $\alpha = 0.8$	$2.5e^{-3}$	Checkered	Precision	90.0563
32	Dice	$1.0e^{-2}$	Full	F1	85.449
32	Dice	$1.0e^{-4}$	Full	Recall	99.1407
32	Dice	$5.0e^{-3}$	Full	Precision	83.2367
32	Dice	$1.0e^{-3}$	Checkered	F1	84.9903
32	Dice	$1.0e^{-4}$	Checkered	Recall	98.5847
32	Dice	$1.0e^{-3}$	Checkered	Precision	87.0733
64	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-2}$	Full	F1	84.523
64	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-4}$	Full	Recall	86.31
64	Focal $\gamma = 2$, $\alpha = 0.1$	$2.5e^{-3}$	Full	Precision	85.599
64	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-4}$	Checkered	F1	84.114
64	Focal $\gamma = 2$, $\alpha = 0.1$	$1.0e^{-4}$	Checkered	Recall	82.8
64	Focal $\gamma = 2$, $\alpha = 0.1$	$5.0e^{-3}$	Checkered	Precision	89.9877
64	Focal $\gamma = 2$, $\alpha = 0.8$	$1.0e^{-2}$	Full	F1	84.879
64	Focal $\gamma = 2$, $\alpha = 0.8$	$1.0e^{-4}$	Full	Recall	85.5173
64	Focal $\gamma = 2$, $\alpha = 0.8$	$1.0e^{-2}$	Full	Precision	84.8933
64	Focal $\gamma = 2$, $\alpha = 0.8$	$5.0e^{-4}$	Checkered	F1	84.133
64	Focal $\gamma = 2$, $\alpha = 0.8$	$1.0e^{-4}$	Checkered	Recall	81.942
64	Focal $\gamma = 2$, $\alpha = 0.8$	$1.0e^{-3}$	Checkered	Precision	89.655
64	Dice	$5.0e^{-3}$	Full	F1	84.52
64	Dice	$1.0e^{-4}$	Full	Recall	99.279
64	Dice	$2.5e^{-3}$	Full	Precision	84.5423
64	Dice	$1.0e^{-3}$	Checkered	F1	85.2167
64	Dice	$1.0e^{-4}$	Checkered	Recall	98.911
64	Dice	$1.0e^{-3}$	Checkered	Precision	87.815

Table B.1: The selected MUnet folds (combination of batch size and loss) from the grid search experiment. From each fold the best performing metrics were selected, based on the full validation set and the validation set with checkered backgrounds only. The metric values indicate the mean score of that particular fold. From each fold then the best performing training run has been selected as a network for further testing. Overall 36 folds have been selected. Some combinations appeared more then once, hence overall 26 networks have been in the end selected.