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 23. Juni 2021

Übersicht

Abstract

Contents

Chapter 1

Introduction

1.1 Motivation

1.2 Related Works

The aim of this thesis is to provide a conversion pipeline for hand-drawn Electrical Circuit Diagrams (ECDs) into the LTspice schematic file format. To provide some context for the task, the following subsections present approaches related to the goal of this thesis.

- TODO general structure

1.2.1 Classification of Eletrical Circuit Components

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 Electrical Circuit Components (ECCs). The dataset consisted of four classes (resistor, capacitor, inductor, voltage source) and had overall a size of 863 images. best cnn 83%

1.3 Goals of the Thesis

1.3.1 Task Description

1.3.2 Contribution

Chapter 2

Theory

2.1 Electrical Circuit Diagrams

An ECD consists of ECCs where for each ECC an 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 degree and would still result in a correct notation. Further, 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.

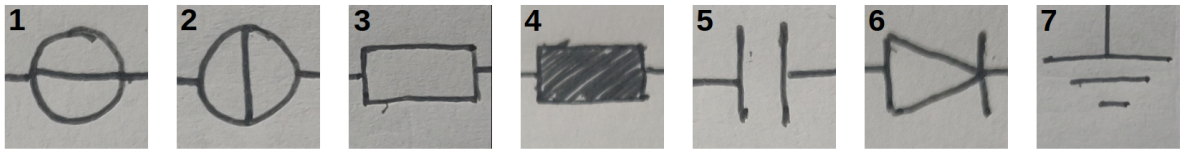

Figure 2.1: All used ECCs in this thesis in German notation: 1. Voltage Source, 2. Current Source, 3. Resistor, 4. Inductor, 5. Capacitor, 6. Diode, 7. Ground

2.2 LTspice

LTspice is a circuit simulation software, where circuits can be modeled, parametrized and simulated. A modeled circuit is stored in a 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 a LTspice schematic file, the file structure and syntax were analyzed and are presented in this section.

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 $32x32$ units, where a unit is an abstract measure inside of LTspice.

2.2.1 LTspice Schematic File Syntax

Header

Each schematic file starts with a header, which defines the used version of the syntax. Throughout this thesis the forth 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 symbols image used to represent the symbol 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 degree. 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 degree.

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 to declare a name for the symbol and the value therefore is a string. The *Value* attribute allows to declare 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 second 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. Biological neurons receive a signal via dendrites and output the processed signal through the axon [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 $V \in \mathbb{R}^n$ with an input vector $x \in \mathbb{R}^n$. The decision boundary of this function is a linear function, which means non-linear problems like the XOR-problem, can't be solved with the perceptron.

$$\hat{y} = sign(V^T x) \tag{2.1}$$

Multi Layer Perceptron

To tackle this insufficiency the Multi Layer Perceptron (MLP) was introduced [Gro73], which is able to solve non-linear problems. Generally, a 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 $x \in \mathbb{R}^n$ without change to the following hidden layer.

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

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

$$\hat{y} = (V_1^T x, \cdots, V_m^T x)^T + b \tag{2.3}$$

The calculation can further be simplified by using a weight matrix, which combines all weights V and the bias b. The output $\hat{y}$ is then simply the matrix multiplication of the weight matrix $W \in R^{(n+1)\times m}$ with the input vector x. Note that, to fit the dimensionality x has to be extended with a *1* at the end hence results in $x \in \mathbb{R}^{(n+1)}$ [Mai20]. The current output vector $\hat{y}$ is just a linear transformation of the input vector 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:

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

The last missing layer type is the output layer, which in a classification setting outputs a vector of conditional class probabilities $\hat{y} \in \mathbb{R}^n$, where n 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 by taking

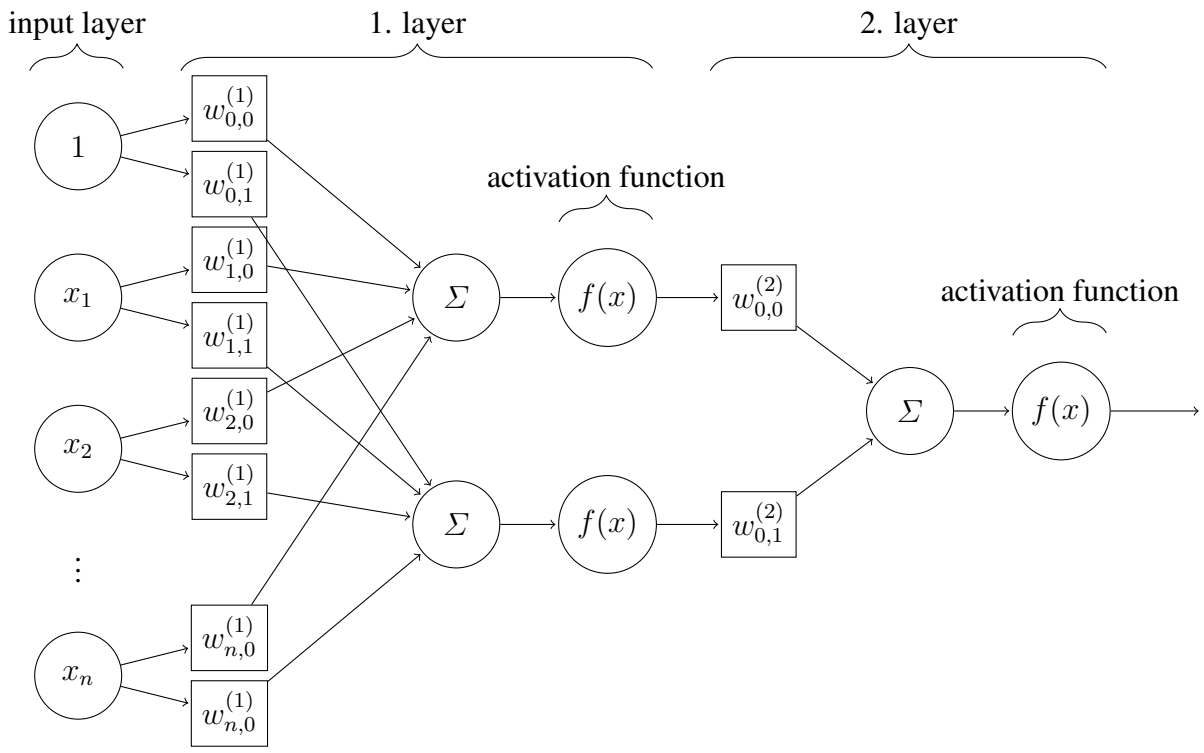

Figure 2.2: A MLP with two hidden layers (two neurons in the first and one in the last), each input gets multiplied with each weight of each neuron, summed up and fed into an activation function to produce the input for the next layer, where this process is repeated.

the fraction of the exponential function applied to an element of the input vector, divided by the sum of all exponential function outputs applied to all elements of the input vector. A pseudo probability element in the output vector is defined as follows:

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

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 $f^{(O)}(f^{(n)}(\cdots f^{(1)}(f^{(I)}(x))))$, where $f^{(I)}$ is the input layer, $f^{(O)}$ the output layer and $f^{(1)} \cdots f^{(n)}$ are the amount of n hidden layers, respectively.

Learning Procedure

In the first step the input data is fed to the network producing an output $\hat{y}$. 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.

Afterwards, the output $\hat{y} \in \mathbb{R}^n$ is compared to the desired output $y \in \mathbb{R}^n$ using a loss function

$L(y, \hat{y}) : (\mathbb{R}^n, \mathbb{R}^n) \rightarrow \mathbb{R}$. A common loss function is the Mean Squared Error (MSE) [Red15], which calculates the mean sum of the squared differences between the inputs y and $\hat{y}$ (eq. 2.6).

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

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

$$CE(y, \hat{y}) = \begin{cases} -log(\hat{y}), & \text{if } y = 1 \\ -log(1 - \hat{y}), & \text{else} \end{cases} \tag{2.7}$$

The resulting loss value is used to calculate the gradients with respect to the last layer weights. 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:

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

Here, $w^{(k+1)}$ denotes the updated weights, and $w^{(k)}$ the current state of the weights. η is the learning rate of the network, which is essentially the amount of the gradient which should be used to update the weights. Typically an $0 < \eta < 1$ is chosen, since big values have shown to make the training process unstable, resulting in divergence of the loss. Further, $\nabla L(w^{(k)})$ should denote 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:

$$v^{(k)} = \mu v^{(k-1)} - \eta \nabla L(w^{(k)}) \tag{2.9}$$

$$w^{(k+1)} = w^{(k)} + v^{(k)} \tag{2.10}$$

μ denotes here the momentum value which is typically set to 0.9, 0.99 respectively [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 function approximation [Dig19]. In the Perceptron the *sign* function was used, but due to its non-differentiable property it isn't suited for the use in ANNs, since back propagation 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 $x \to \pm\infty$, 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.

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

Rectified Linear Unit (ReLU)

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

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

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:

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

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]

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

2.3.3 Convolutional Neural Networks

While MLPs perform pretty 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 CNNs, which have shown to perform pretty 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 (eq. 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]

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

In simpler terms that just means there is an image $I \in \mathbb{R}^{WxHxC}$, where W and H are the width and height of the image and C being the number of channels. Further, there is a kernel $K \in \mathbb{R}^{W_K x H_K x C}$. The image and the kernel are now convolved by moving the kernel over the

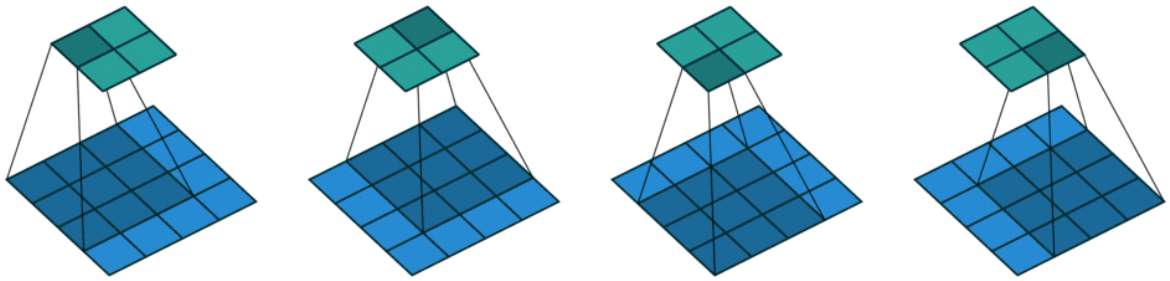

Figure 2.3: Example convolution of a 4x4 input (blue) with a 3x3 kernel (dark blue) and a stride of 1, resulting in a 2x2 output (cyan) [Dum16]

image and at each position an element-wise multiplication of the overlapping area of the image and the kernel is taken. Afterwards, 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 MAX 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 big amount of data to learn the desired objective and to prevent the problem of overfitting, where a network perfectly learns the underlying data and isn't 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 normally are 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 or tensorflow, 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 additionally 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 average, or weighted average for example is taken over the predictions to form the final prediction.

	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.

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 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

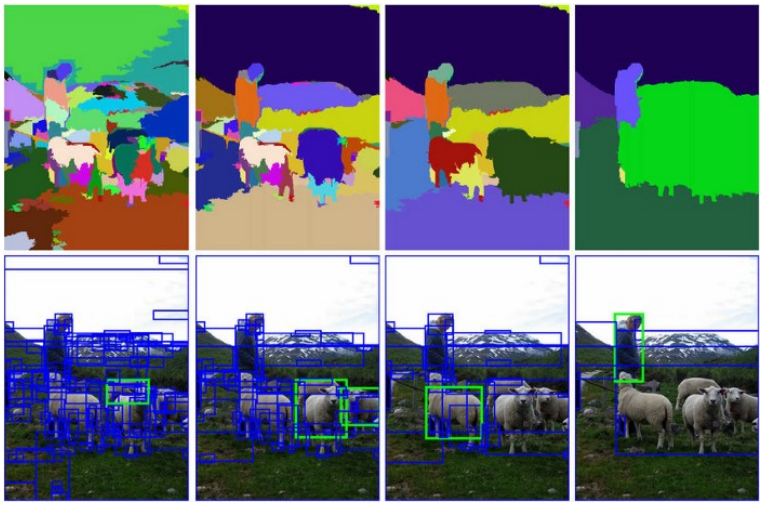

Figure 2.4: Example of results obtained through the Selective Search algorithm (top) with increasing region scale from left to right and bounding boxes drawn around those regions (bottom). Selective Search produces sub-segmentations of objects in an image, considering size, color, texture and shape based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

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 Support Vector Machines (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 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], You Only Look Once (YOLO) [Red15], Single Shot Multibox Detector (SSD) [Liu15] and RetinaNet [Lin17] 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 predict bounding box regression offsets based on a certain anchor box. [Red18] The selection of good anchor box sizes is a hyper parameter 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 and also to construct various loss functions as well as metrics.

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

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 (eq. 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.

$$L_{IoU} = 1 - IoU(A, B) \tag{2.18}$$

Further, to solve these drawbacks Rezatofighi et al. [Rez19] proposed the Generalized IoU (GIoU) Loss (eq. 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].

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

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 (eq. 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.

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

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 (eq. 2.21, 2.22, 2.23). The penalty in CIoU is split into α and ν. 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 the 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].

$$L_{CIoU} = DIoU(A, B) + \alpha(A, B) * \nu(A, B) \tag{2.21}$$

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

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

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.

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

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 You Only Look Once Version 4 (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 as input an image 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 (tab. 2.9), which is sequentially build out of a convolutional layer, a batch normalization layer and a leaky ReLU activation function, following the principles of [Iof15].

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 $3x3$ convolutions, followed by one $1x1$ convolution and a max pooling layer to half the feature map size. The number of filters for the four convolutions is always $N * (K, K/2, K/2, K)$, where N denotes the layer number and K the number of filters beginning with 64. So as can be seen in 2.8, 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 ($Skip_S$, $Skip_M$, $Skip_L$), 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).

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	$Skip_S$
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	$Skip_M$
m2	c10 & c13	MaxPool		2x2	2	same	
Block 4							
c14	m2	YOLOConv	512	3x3	1	same	$Skip_L$

Table 2.8: The CSPDarknet53Tiny Architecture is a scaled version of the original CSPDarknet53 architecture [Boc20]. The definition for YOLOConv can be found in table 2.9. 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.

Sequence	Parameter
Convolution	see tab. 2.8; l2 kernel regularization $5x10^{-3}$
BatchNorm	
LeakyReLU	$\alpha = 0.1$

Table 2.9: Convolutional basic building block in YOLOv4 (YOLOConv). A convolutional layer, followed by a batch normalization layer, followed by a leaky ReLU activation function.

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 as input the outputs of the backbone 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 ($Pred_L$), which has the size of the last backbone output block. Therefore, $Pred_L$ has a size of $S \times S$, $Pred_M$ has a size of $2S \times 2S$ and $Pred_S$ has a size of $4S \times 4S$, where $S \times S$. Generally, the PANet-tiny is build 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 $3\dot{(}5 + C)$, where 3 are the number of anchor boxes in that particular scale, 5 are the number of bounding box parameters (4 regression offsets, 1 objectness score) and C are the number of classes. So the final output tensor of the network is $(2^{O-1} * S) \times (2^{O-1} * S) \times (3 * (5 + C))$.

Head

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

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

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:

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

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 most lowest skip connection $Skip_L$ 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.

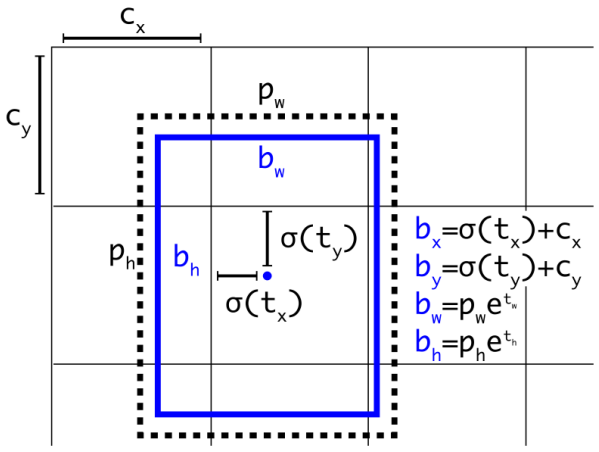

Figure 2.5: Bounding box calculation in the YOLOv4 network based on a prior anchor box [Red16]. The b_* indicate the final prediction, p_* indicate parameters of the prior bounding box and c_* indicate the prior bounding box spatial offset based on the current cell. The final center coordinate offset 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].

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 (eq. 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.

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

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.

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

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

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 (eq. 2.29) defines the loss which occurs when an object is present and the latter (eq. 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 L_{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 and any ground truth bounding box 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.

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

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

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

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

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 B, also called candidates, as well as the corresponding prediction scores S and a suppression threshold ϵ. The algorithm first selects the bounding box B_m with the highest score S_m and compares it to every other bounding box b_i in the candidate set B. If the DIoU measure between B_m and b_i is above the threshold ϵ, the bounding box b_i gets suppressed by removing it from the set B. This process is repeated until no bounding boxes are present in B.

Algorithm 2.1: DIoU-NMS Algorithm TODO caption to bottom and format

```
input:  B = {b_1 ,... ,b_N}, S = {s_1 ,... ,s_N}, ε
        B: list of bounding box proposals
        S: list of bounding box scores
        ε: NMS threshold
output: B_S ⊆ B, B_S = {b_1,...,b_M}
        B_S: list of bounding boxes after suppression
        S_S: list of bounding box scores after suppression

begin
B_S ← {}; S_S ← {}
while B ≠ empty do
    m ← argmax(S)
    B_m ← B[m]; S_m ← S[m]

    B_S ← B_S ∪ B_m; S_S ← S_S ∪ S_m
    B ← B − B_m; S ← S − S_m

    foreach 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. In TODO SECTION WHERE 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 will be 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 ϵ. 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:

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

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

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

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:

$$C = C * \frac{min(T, N)}{N} \tag{2.35}$$

or

$$C = C * \frac{T}{N} \tag{2.36}$$

Here, T is the number of bounding boxes in a cluster and N 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't 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.

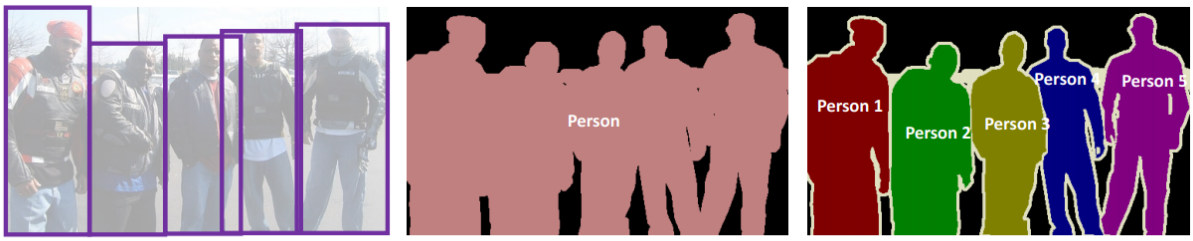

Figure 2.6: The difference between object detection, semantic segmentation and instance segmentation. In object detection the instance with a rough estimate (bounding box) is predicted, in semantic segmentation a segmentation mask for an object is predicted without considering the underlying instance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

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 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 MobileNetV2-UNet (MUnet) [Jin20] is used. This network is build upon the principles of the famous 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 a feature map was upsampled it gets concatenated with a feature map from the backbone which has the

same resolution. It should be noted that this approach was probably picked up by the YOLOv4 developers and reused in their architecture as has been 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:

- 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. Afterwards, the output is again convolved with 1×1 convolution. Further, 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

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.

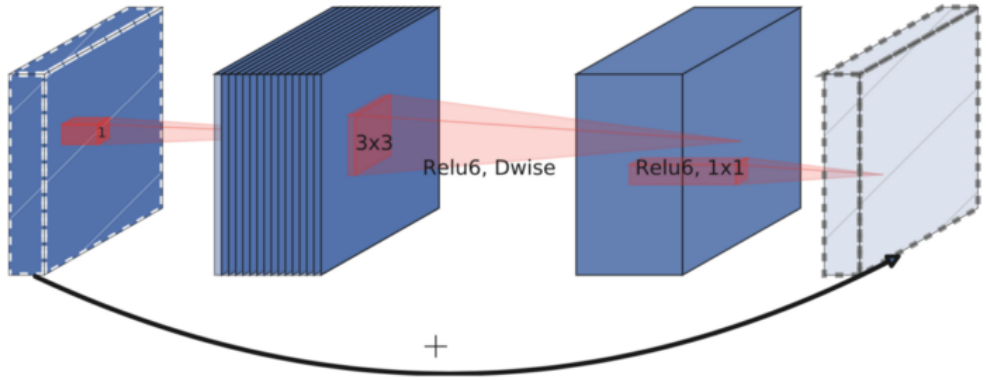

Figure 2.7: The inverted residual block. A 1×1 convolution with non-linear activation followed by a depthwise separable convolution and a linear 1×1 convolution with residual connection to the input. The residual connection is here additive, i.e. the input gets added to the output of the linear 1×1 convolution.

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 backbone 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 $s = 1$.

with a skip connection from the backbone with the same spatial dimension. Concatenation is 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 backbone can be found in table 2.13.

2.9 Hypergraphs

To evaluate the produced ECD topologies, topologies are represented as a hypergraph, therefore the basic theory of hypergraphs first should be explained. Hypergraphs are a generalization of a graph and are formally defined as: $H = (V, E)$, where V is a set of $N \in \mathbb{N}$ vertices $\{v_1, ..., v_N\}$ and E is a set of $M \in \mathbb{N}$ hyperedges $\{e_1, ..., e_M\}$, where each hyperedge $e_i \in E$ is $e_i \subseteq V$. [Val21] In matrix notation a hypergraph can be represented in a so called adjacency matrix. 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.

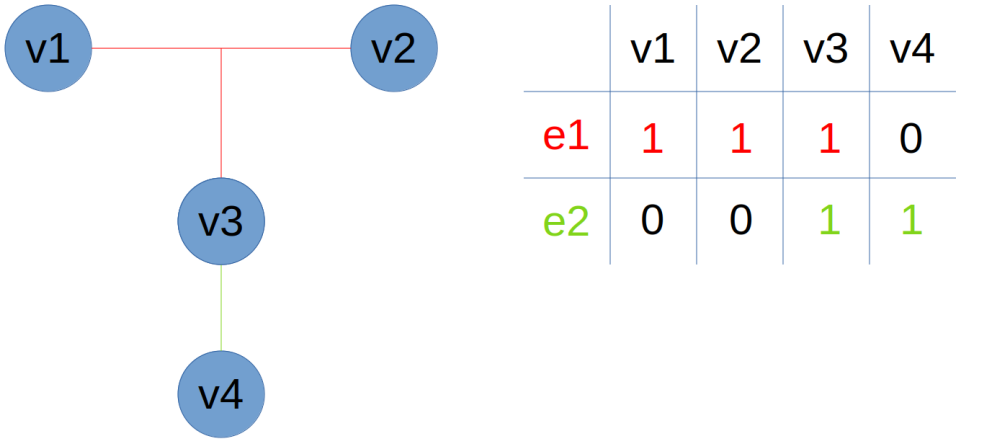

Figure 2.8: An example drawing of a hypergraph with its corresponding adjacency matrix. The hypergraph is defined as: $H = (V, E)$, $V = \{v1, v2, v3, v4\}$, $E = \{e1, e2\}$, with $e1 = \{v1, v2, v3\}$, $e2 = \{v3, v4\}$. In the adjacency matrix a row corresponds to a hyperedge and each column to a 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.

2.10 Metrics

True Positive, False Positive, False Negative

To understand the following subsections first the notion of True Positive (TP), False Positive (FP) and False Negative (FN) should be explained. A TP occurs when the prediction of a network is equal to the underlying ground truth. Further, a FP occurs when the prediction of a network is not equal to the underlying ground truth, normally in terms of that the predicted class differs from the ground truth class, this also includes predictions where a ground truth class is not present at all. Finally, a FN occurs when a prediction is missing completely, so no prediction exists for a ground truth. In object detection the above definition can't be applied directly. Since multiple bounding boxes are predicted there has to be a measure of similarity for bounding boxes, to match a ground truth bounding box against a predicted one. So for example for the TP case not only the class of predicted box A and ground truth box B has to match, but also the $IoU(A, B)$ has to be above a certain threshold.

Precision

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

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

Recall

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

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

F1-Score

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

$$F1 = \frac{2 * recall * precision}{recall + precision} \tag{2.39}$$

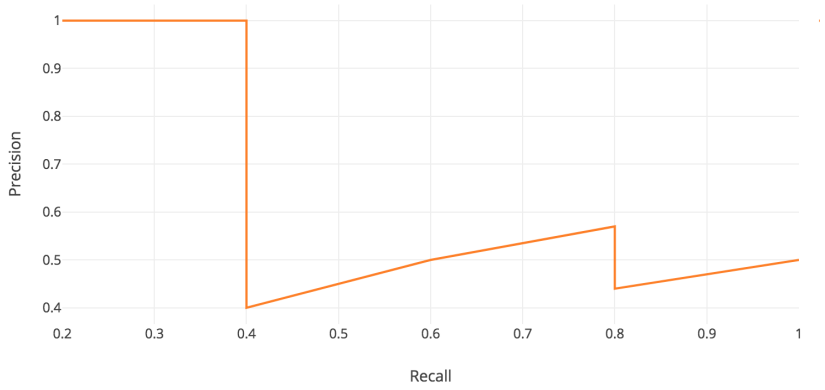

Figure 2.9: Example of a precision-recall curve, where precision and recall were calculated for different IoU thresholds and sorted and plotted by their recall values [Hui18]. The AP is the area under the curve.

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 (eq. 2.40).

$$AP = \int_0^1 Precision(Recall) dRecall \tag{2.40}$$

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 N is the number of classes (eq. 2.41).

$$mAP = \frac{1}{N} \Sigma_{i=1}^{N} AP_i \tag{2.41}$$

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 (eq. 2.42).

$$mIoU = \frac{1}{N} \Sigma_{i=0}^{N} IoU_i \tag{2.42}$$

Chapter 3

Material and Methods

3.1 Data

In this section the data which was used in this thesis is presented, as well as some general information about that. Generally, the dataset is a collection of ECDs, with ECCs in German notation. The task was constrained to seven different ECCs, which are shown in figure 2.1. Most of the images were taken with a mobile phone, but some were also directly drawn on a digital device such as a tablet.

The difficulty of the task in this thesis increased gradually and more data was always acquired when 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 show in table 3.1.

Labeling

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.

	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

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 is 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 the holes between the edges of a wire and afterwards eroded four times to reduce the thickness of the segmentation mask. The labels are not perfect because sometimes it is hard to remove the gridded background without removing parts of the wire with this technique, therefore each mask is additionally manually fine-tuned, with a simple drawing tool build with the Python version of OpenCV [Bra00].

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

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.

The above points are all embedded into a three-stage pipeline, which will be presented throughout this section.

Recognition

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), or just 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.

Furthermore, the pipeline also utilizes segmentation to segment the circuit. Segmentation is needed because the topology building step, which will be 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 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. Note that 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.

An example prediction of both networks can be seen in figure 3.1.

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 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.

Adaptive Binary Thresholding TODO? maybe I won't use it any more

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

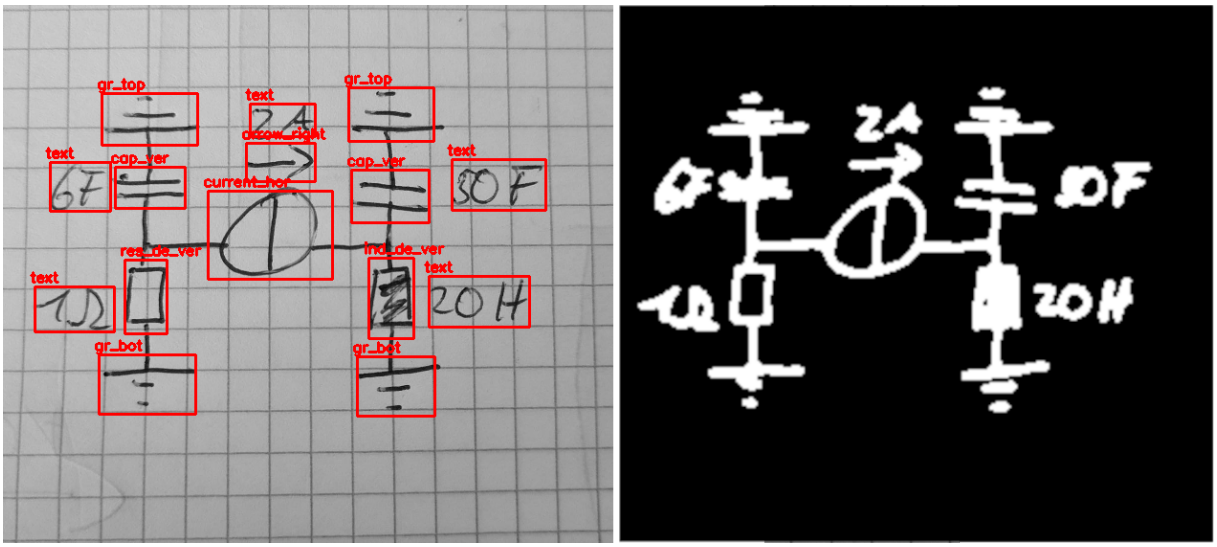

Figure 3.1: Example predictions of YOLO (left) and MUnet (right). YOLO predicts a bounding box around each component with its corresponding class, while MUnet predicts a segmentation mask a binary fashion of the whole ECD drawing, including ECC annotations. Primary segmentation is needed to remove the checkered background from the image.

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 $\in \{x1, y1, x2, y2, class\}$, where $(x1, y1)$ represent the upper left corner of the bounding box and $(x2, y2)$ 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**

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, it takes as input a list of arrow bounding boxes, a list of text bounding boxes and a list of ECCs. 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: Annotation Matching TODO caption to bottom and format

```
input:  A = {a_1, ..., a_n}, T = {t_1, ..., t_m}, E = {e_1, ..., e_o}
        A: list of predicted arrow bounding boxes
        T: list of predicted text bounding boxes
        E: list of predicted ECC bounding boxes
output: A_m = {(a_1, e_i), ..., (a_n, e_j)}
        T_m = {(t_1, e_x), ..., (t_m, e_y)}
        A_m: list of arrow bounding boxes matched against source bounding boxes
        T_m: list of text bounding boxes matched against ECC bounding boxes

begin
    A_m = {}
    foreach a in A; do
        distances = {}

        a_center = get_bounding_box_center(a)
        foreach 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

        min_dist, a_matched, e_matched = sort_ascending_by_distance(distances)[0]
        A_m = A_m ← (a_match, e_match)
    end

    T_m = {}
    foreach t in T; do
        distances = {}

        t_center = get_bounding_box_center(t)
        foreach e in E; do
            e_center = get_bounding_box_center(e)
            dist = euclidean_distance(t_center, e_center)
            distances ← distances + (dist, t, e)
        end

        min_dist, t_matched, e_matched = sort_ascending_by_distance(distances)[0]
        T_m = T_m ← (t_match, e_match)
    end

    return A_m, T_m
end
```

LTspice Conversion

The last step in the proposed pipeline is the embedding of the gathered information into the LTspice schematic file.

TODO not 100% sure how this will be written

3.3 Training

The following section deals with the training process of YOLO and MUnet. For both networks similar experiments were performed each sub-experiment was repeated three times each with a different random seed. The used seeds were: {42, 1337, 0xDEADBEEF}

First an initial learning rate search was performed. Here six to seven learning rates were tested with the basic dataset. The basic dataset is the default train / valid / test split (table 3.1 and 3.2) without any pre-augmentations of the training set. The best performing learning rate was taken as the baseline for following experiments.

The second conducted experiment was a search for the optimal configuration of so called offline augmentations. Offline augmentations are a 90°, 180° and a 270° rotation of the original image, as well as a horizontal flip and again the three rotations of the flipped image. Furthermore, as has been described in section 3.1, for some images a mask has been created, which is projected on different checkered background images to increase the amount of those. This augmentation is referred to as projection or copy-paste augmentation. The search for the optimal configuration was performed by training every possible combination of the three described offline augmentations. Again, the best performing combination was used as the baseline for the next experiment.

Afterwards, an ablation study was performed for various augmentations, with different parameters. The augmentations in that study are also referred to as online augmentations, since they were performed at training time. The occurrence probability of those augmentations was set to 50%. Since the augmentations slightly differ for each network they are explained in the respective subsection.

The last experiment step was formed by a fine tuning step, where a grid search was performed. The grid search always utilized the best performing offline and online augmentations and consisted of various learning rates, batch sizes and loss functions.

Finally after training each network, a threshold fine-tuning was performed for each network and other improvement strategies were tested, such as TTA.

3.3.1 YOLOv4-Tiny

To perform any experiments with the YOLO network, first an initial configuration was defined. This configuration is given in table 3.3, where the default parameters of the utilized YOLO

repository were used. All networks were trained for 4000 steps (batches) and no early stopping was performed.

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

Redmon et al. have pointed out that training YOLO is unstable, when the full learning rate is applied without proper scheduling [Red16]. This was also tested in this thesis and can be confirmed. 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:

$$lr(step) = \begin{cases} lr_{base} * (\frac{step}{burn\_in})^4 & \textbf{if } step < burn\_in \\ lr_{base} & \textbf{else} \end{cases} \tag{3.1}$$

All trained experiments were optimized for a slightly changed COCO mAP metric. COCO uses a $mAP0.5 : 0.95 : 0.05$, while in this thesis $mAP0.5 : 0.75 : 0.05$ 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.

Experiment: Initial Learning Rate Search

The first experiment was executed to find an initial learning rate. The parameters for the network were set to the ones in table 3.3 and kept for all training runs. The results of the training runs can be found in figure 3.2, 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.

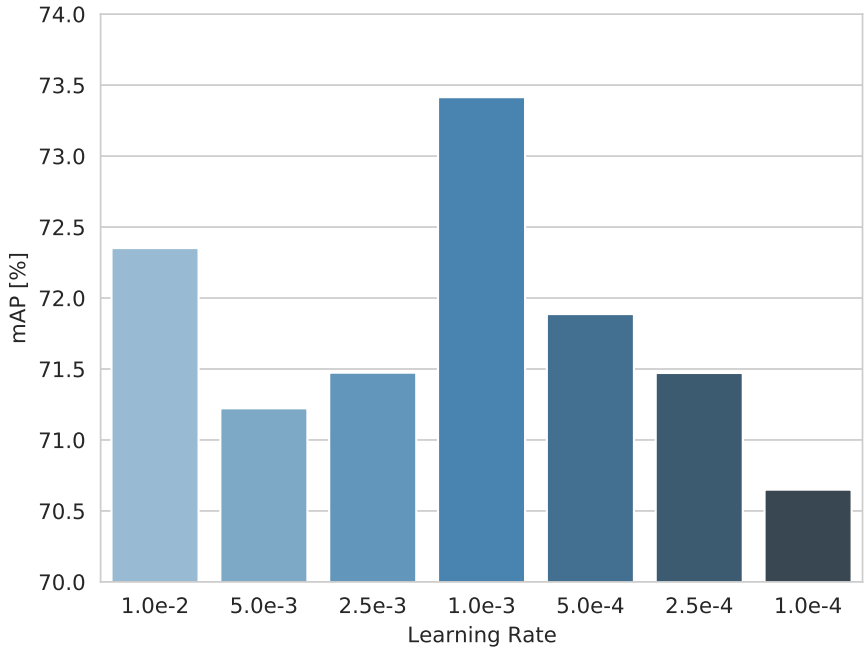

Figure 3.2: The results of the initial learning rate search shown on the validation set, as the mean of the mAPs of three separate training runs.

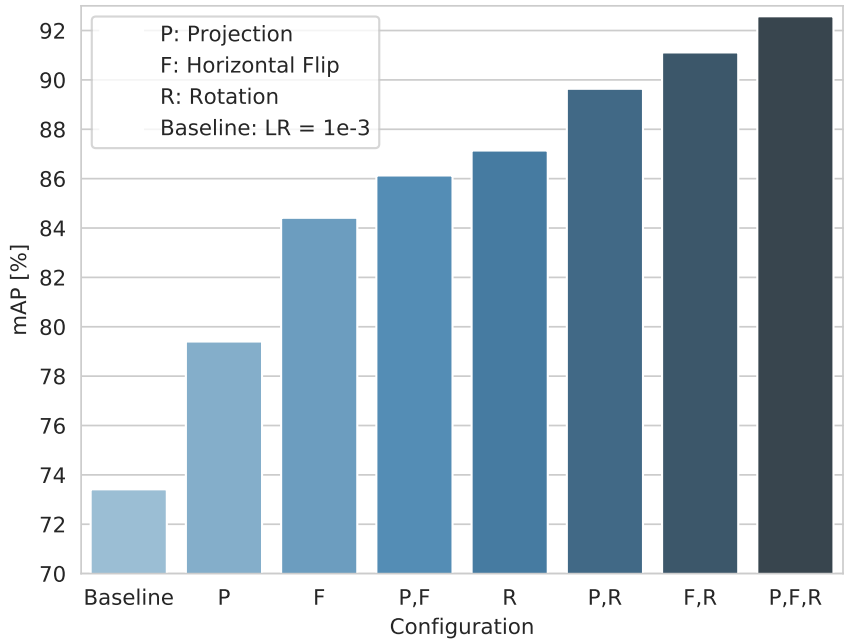

Figure 3.3: The results of the offline augmentation with the different offline augmentation configurations compared with the results of the best performing learning rate (baseline). When rotation and flip are enabled simultaneously the flipped image also gets rotated three times by 90°. Results are given as the mean of the mAP of three separate training runs.

Experiment: 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 3.3, 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

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 really 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.

Experiment: 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 3.4. The figure shows, for each augmentation and for all parameters a clear increase in mAP. The best performing parameters are selected to be used in the following grid search experiment.

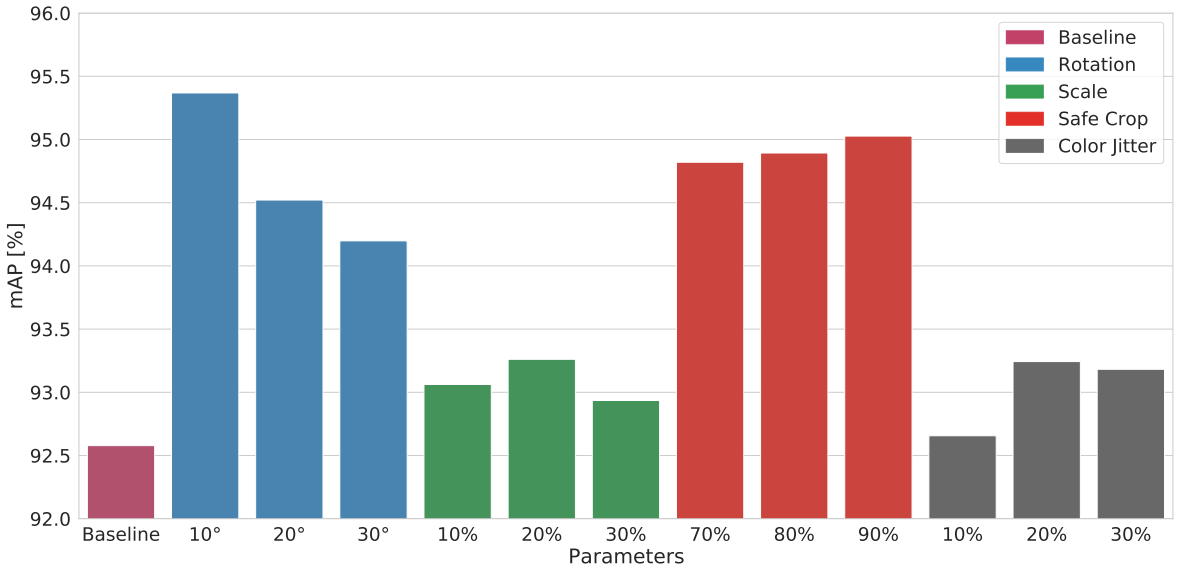

Figure 3.4: Results of the online augmentation experiment compared to the baseline which was established in the offline augmentation experiment. Each augmentation shows a clear increase in mAP in comparison to the baseline.

Experiment: Grid Search

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 3.4, 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 3.4: Configuration of the YOLO grid search experiment. The networks were trained for all possible configurations of learning rate, batch size and loss function.

The full results of the grid search can be found in appendix in the tables TODO REF TABLES.

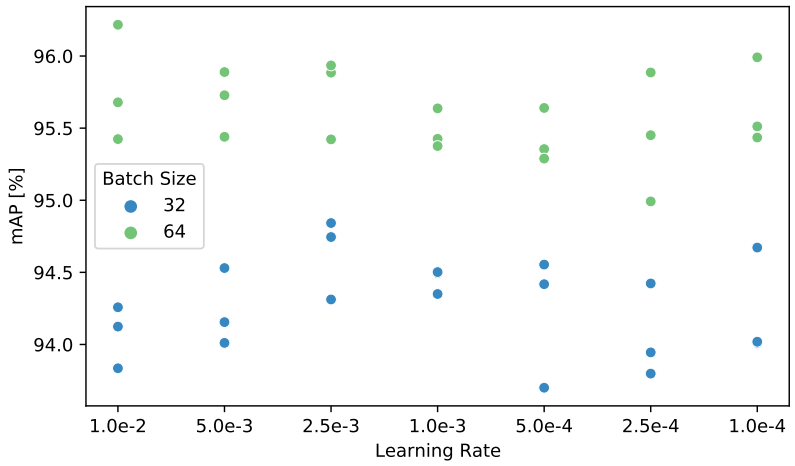

Figure 3.5: Results of the grid search shown for all loss functions, learning rates and batch sizes.

In figure 3.5 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 3.6 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.

Tuning: Input Size

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 is 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 3.7. The input size 736×736, shows a maximum increase in performance and is hence selected for further tuning experiments.

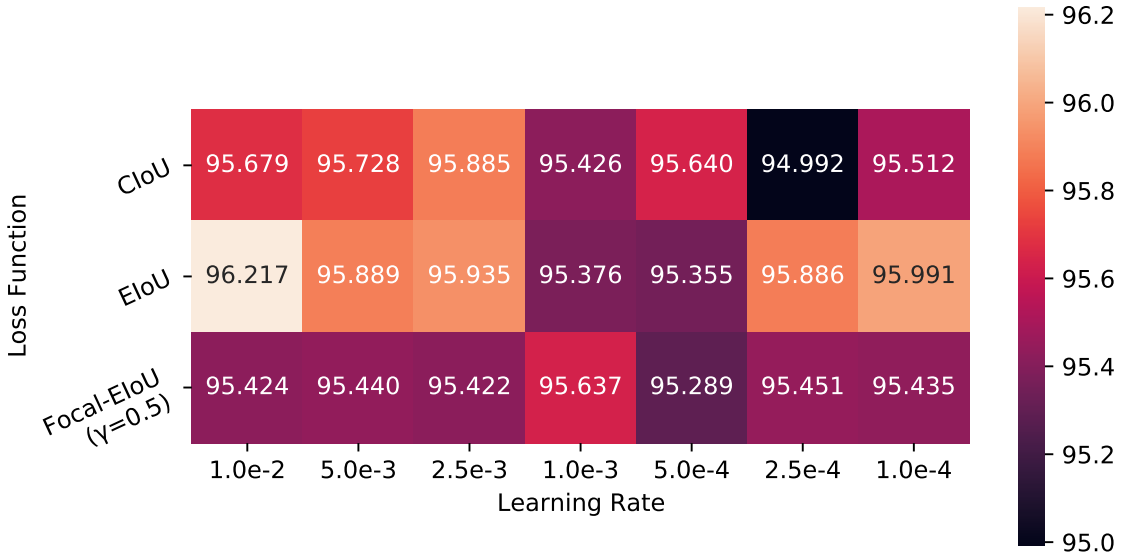

Figure 3.6: Results of the YOLO grid search for all used loss functions and learning rates shown for batch size 64.

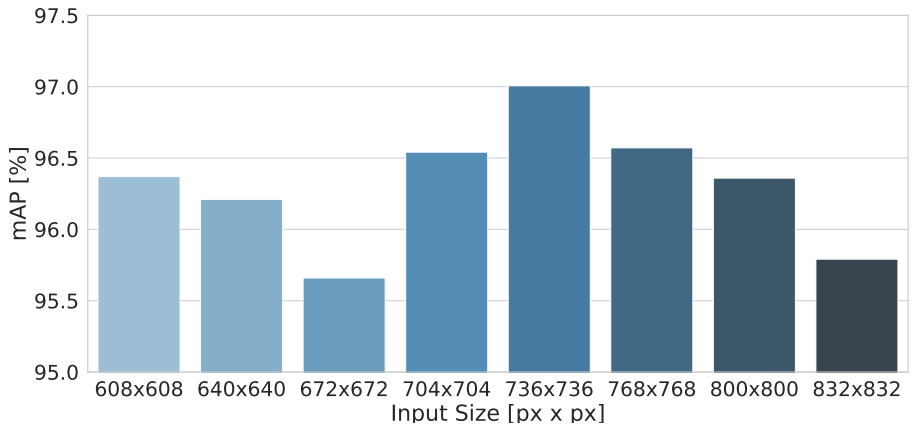

Figure 3.7: Different input sizes tested after training on the validation set.

Tuning: Non-Maximum Suppression 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 are the same one, i.e. if two bounding boxes have an $IoU > IoU_{thresh}$, 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.

Experiment: 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.

So 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.

Again, the tuning of the voting threshold was performed on every possible $voting_{thresh}$ and the obtained results are illustrated in figure 3.8. 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.

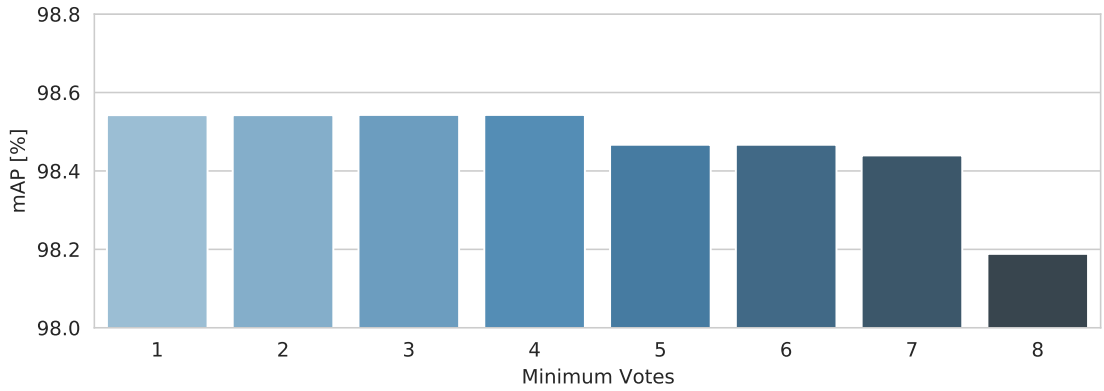

Figure 3.8: Results of the voting based thresholding approach to improve model performance, shown on the validation set for all possible voting thresholds.

Final Results

Figure 3.9 shows the combined results of each tuning step on the validation and the test set. Note that all thresholds were tuned on the validation set only.

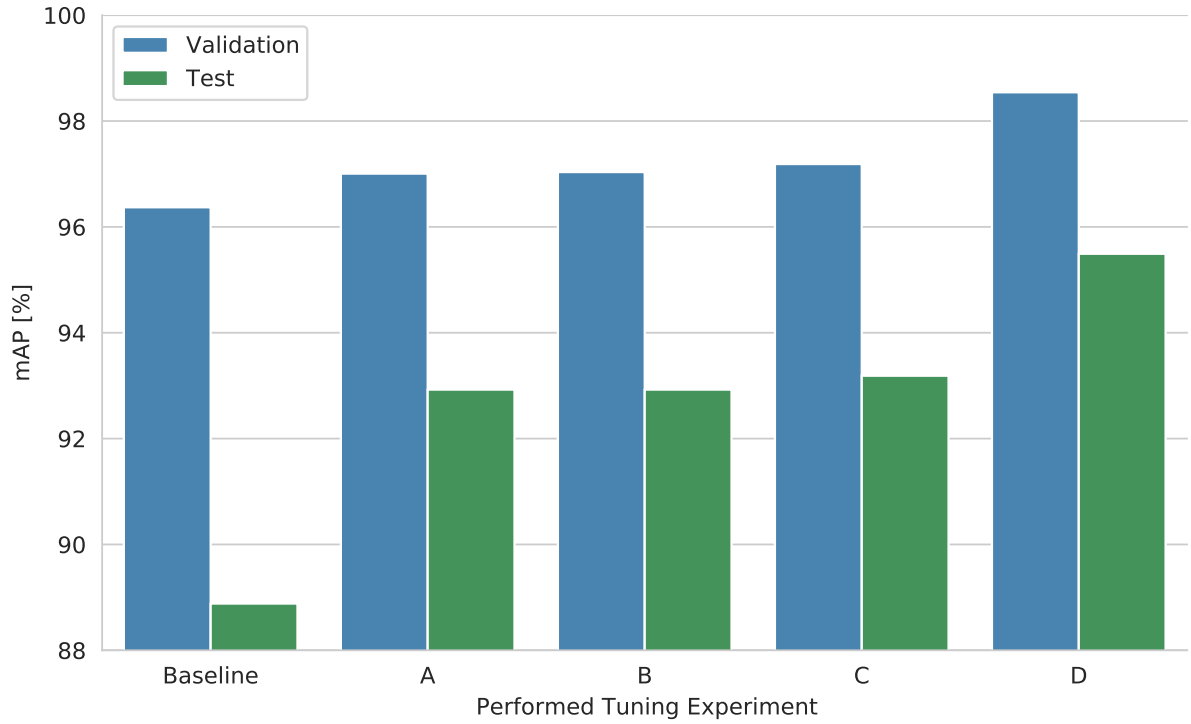

Figure 3.9: The final results of the tuning experiment performed on the validation set, presented also with the performance on the test dataset. Specific explanation of the experiments with the optimal obtained parameters can be found in table 3.5.

Experiment in fig. 3.9	NMS	Score Thr.	IoU Thr.	Input Size	Val. mAP	Test mAP
Baseline	DIoU	0.3	0.25	608×608	96.370%	88.883%
A	DIoU	0.3	0.25	736×736	97.006%	92.926%
B	DIoU	0.15	0.45	736×736	97.036%	92.926%
C	WBF	0.15	0.25	736×736	97.187%	93.188%
D	WBF-TTA	0.1	0.45	736×736	98.543%	95.492%

Table 3.5: Results of the tuning experiment performed on the validation set, presented also with the performance on the test dataset. A visual representation can also be found in figure 3.9.

3.3.2 MobileNetV2-UNet

In this section the training and the tuning of thresholds of the MUnet is presented. The experiments were conducted in the same way as with the YOLO network, as described in the beginning of this section 3.3. The configuration for the MUnet is presented in table 3.6. The selected values correspond to the initial values from the used repository. TODO transfer learning

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

Table 3.6: Configuration of the YOLO grid search experiment. The networks were trained for all possible configurations of learning rate, batch size and loss function.

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

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

$$lr(step) = \begin{cases} lr_{base}/10 & \textbf{if } step > 1500 \\ lr_{base}/5 & \textbf{else if } step > 1000 \\ lr_{base} & \textbf{else} \end{cases} \tag{3.2}$$

Experiment: Initial Learning Rate Search

Experiment: Offline Augmentations

Experiment: Online Augmentations

Experiment: Grid Search

Tuning: Input Size

Chapter 4

Results

Chapter 5

Discussion

aklsdfj alksdfj

Chapter 6

Abbrevations

ANN Artificial Neural Network

CCA Connected Component Analysis

CNN Convolutional Neural Network

ECC Electrical Circuit Component

ECD Electrical Circuit Diagram

MLP Multi Layer Perceptron

MSE Mean Squared Error

NMS Non-Maximum Suppression

OCR Optical Character Recognition

R-CNN Regions with CNN features

YOLO You Only Look Once

YOLOv1 You Only Look Once Version 1

YOLOv4 You Only Look Once Version 4

SVM Support Vector Machine

RoI Region of Interest

RPN Region Proposal Network

IoU Intersection over Union

GIoU Generalized IoU

DIoU Distance IoU

CIoU Complete IoU

EIoU Efficient IoU

CE Cross Entropy

SSD Single Shot Multibox Detector

ReLU Rectified Linear Unit

LReLU Leaky ReLU

PANet Path Aggregation Network

mAP Mean Average Precision

AP Average Precision

mIoU Mean Intersection over Union

TP True Positive

TN True Negative

FP False Positive

FN False Negative

FC Fully-Connected

BatchNorm Batch Normalization

FCOS Fully Convolutional One Stage Detector

WBF Weighted Bounding Box Fusion

TTA Test-Time Augmentation

LDom LTspice Domain

IDom Image Domain

SGD Stochastic Gradient Descent

MUnet MobileNetV2-UNet

List of Figures

List of Tables

Bibliography

[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.

[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.

[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.

[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.

[Lin17] 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.

[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.

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

[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.

[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.

[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.

[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.

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

[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%		

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 configuration should be interpreted in a way that a 1 behind a letter means that particular augmentation is enabled. Further, 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. Additionally, when flip and

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.

B MobileNetV2-UNet experiments

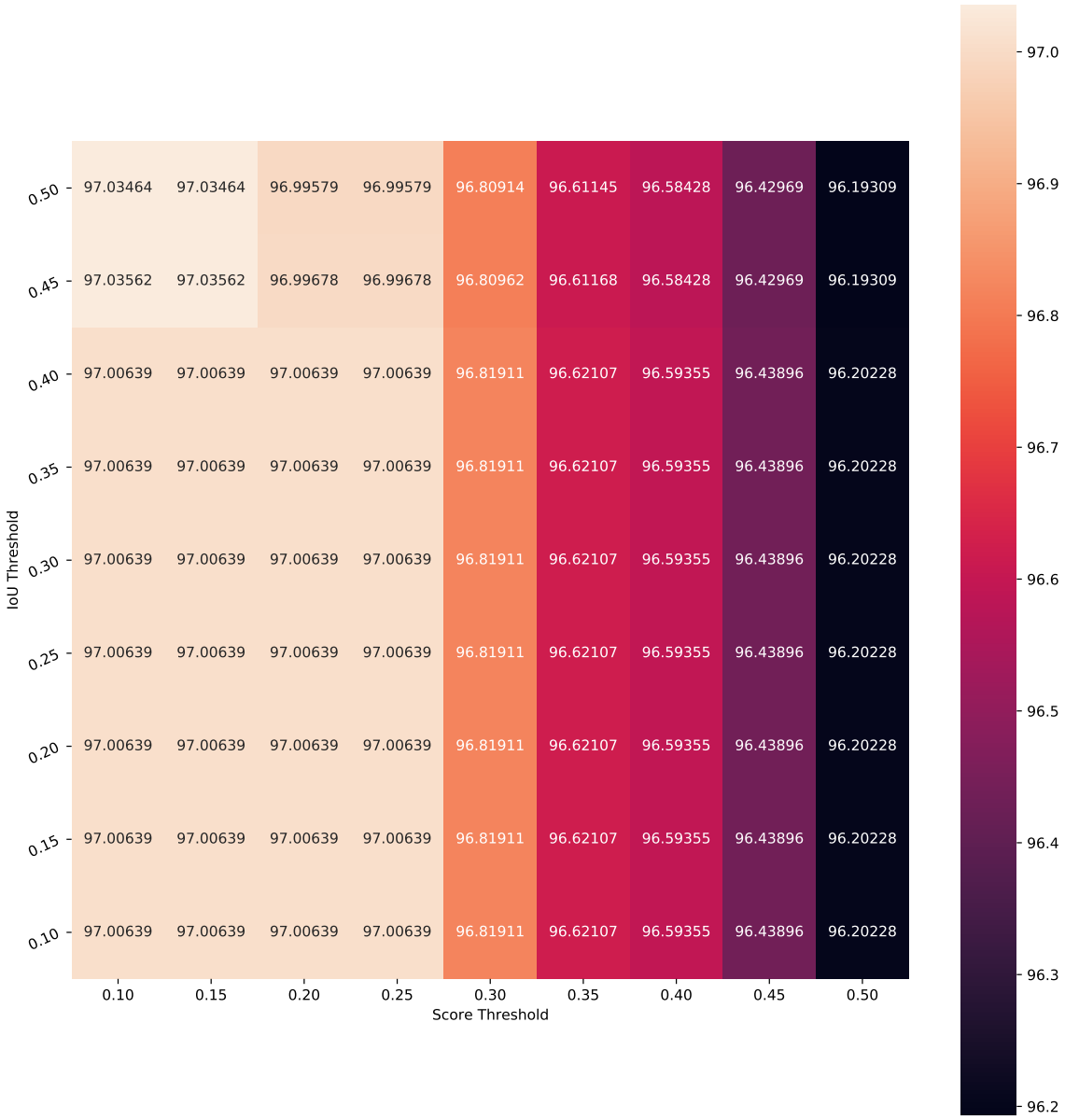

Figure A.1: The results of the DIoU-NMS parameter tuning. All possible combinations of score threshold and IoU threshold were evaluated on the validation set. The best performing combination is the one with a score threshold of 0.1 and a IoU threshold of 0.45.

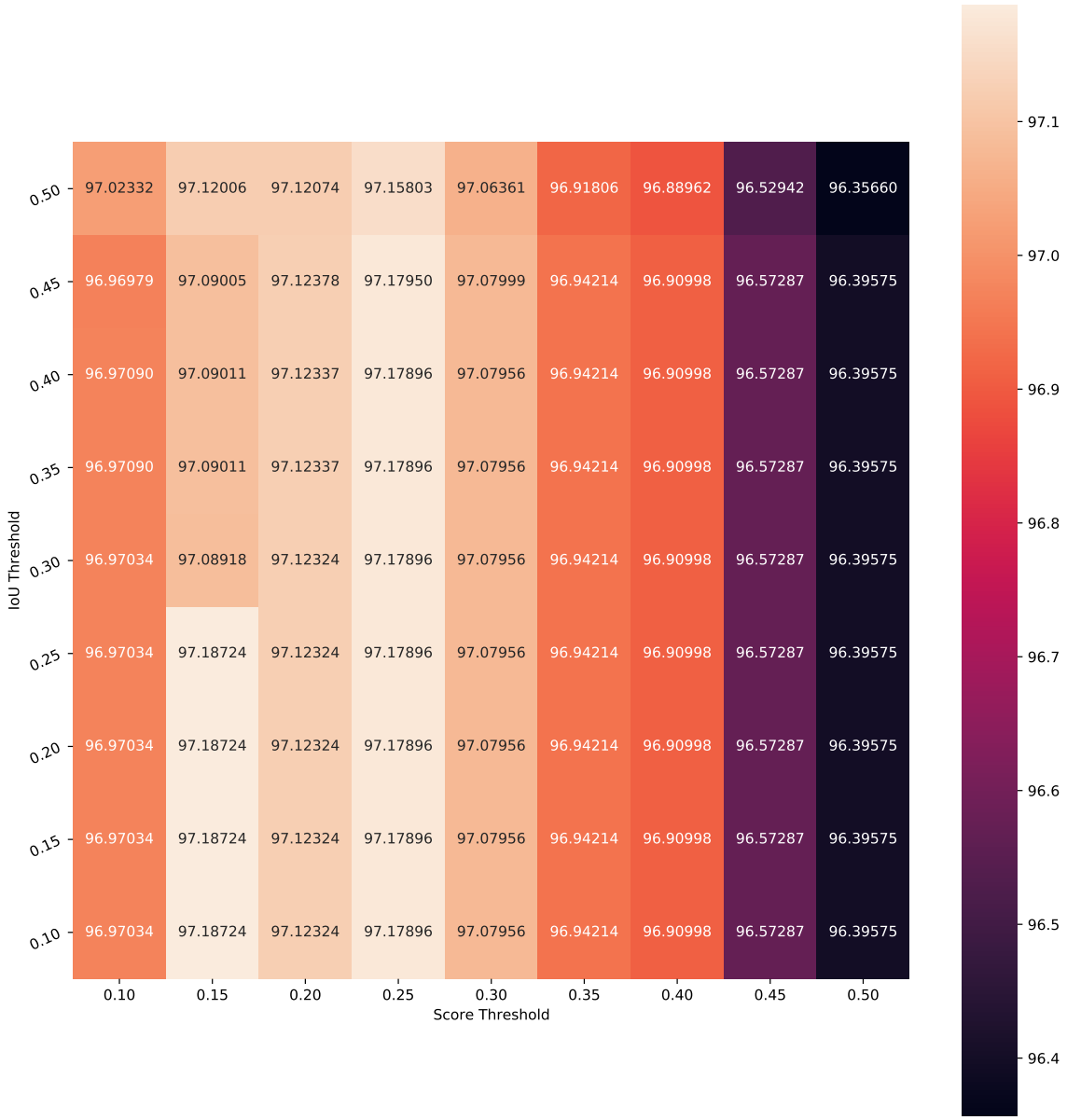

Figure A.2: The results of the WBF parameter tuning. All possible combinations of score threshold and IoU threshold were evaluated on the validation set. The best performing combination is the one with a score threshold of 0.15 and an IoU threshold of 0.25.

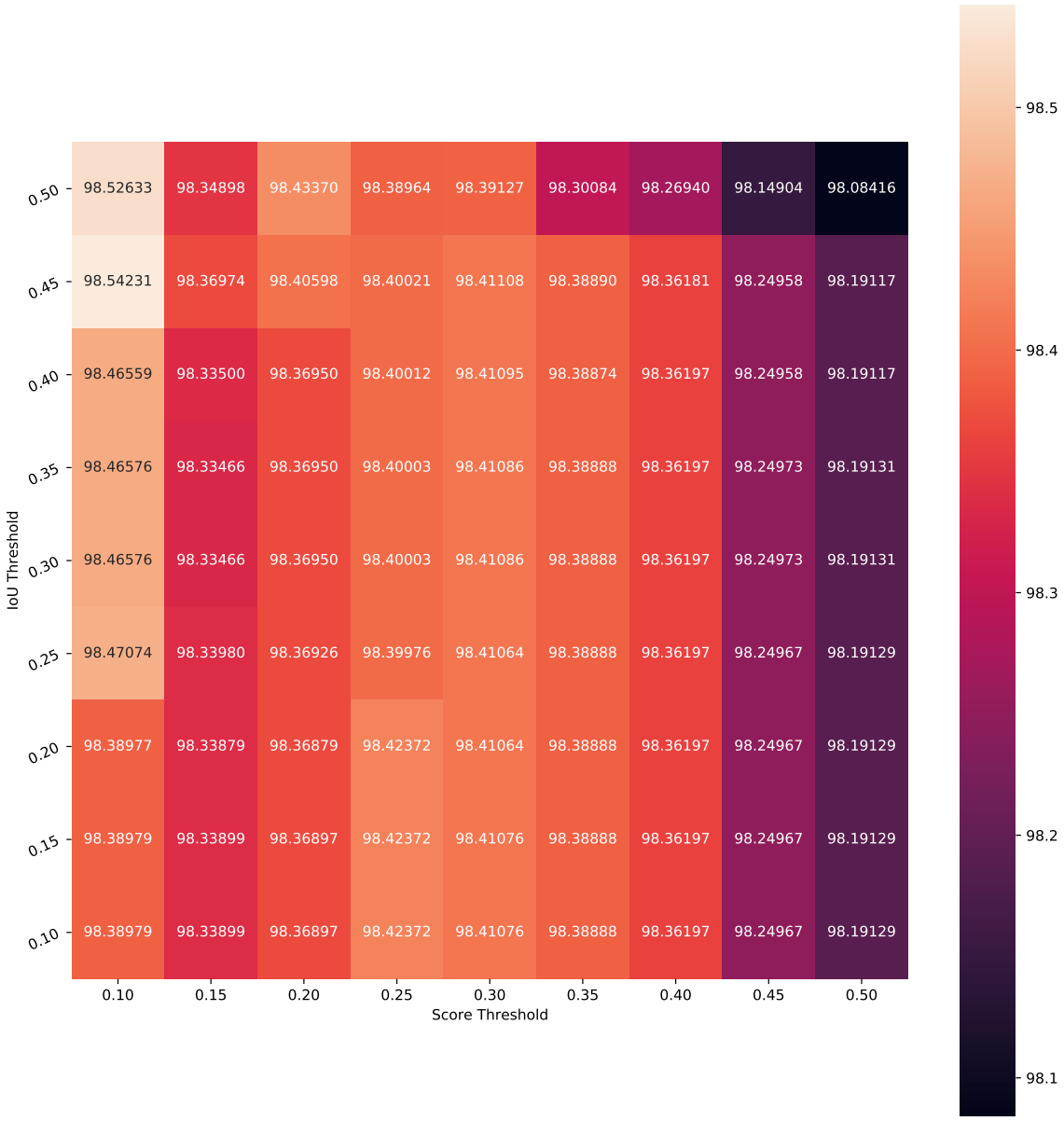

Figure A.3: The results of the WBF tuning with TTA. All possible combinations of score threshold and IoU threshold were evaluated on the validation set. The best performing combination is the one with a score threshold of 0.1 and an IoU threshold of 0.45.