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 1. 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.

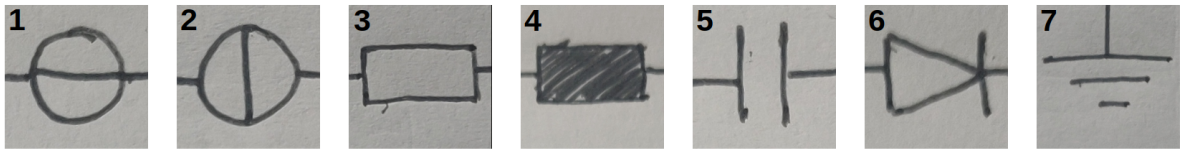


Figure 2.1: All used ECCs in this thesis in German notation: 1. Voltage Source, 2. Current Source, 3. Resistor, 4. Inductor, 5. Capacitor, 6. Diode, 7. Ground

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

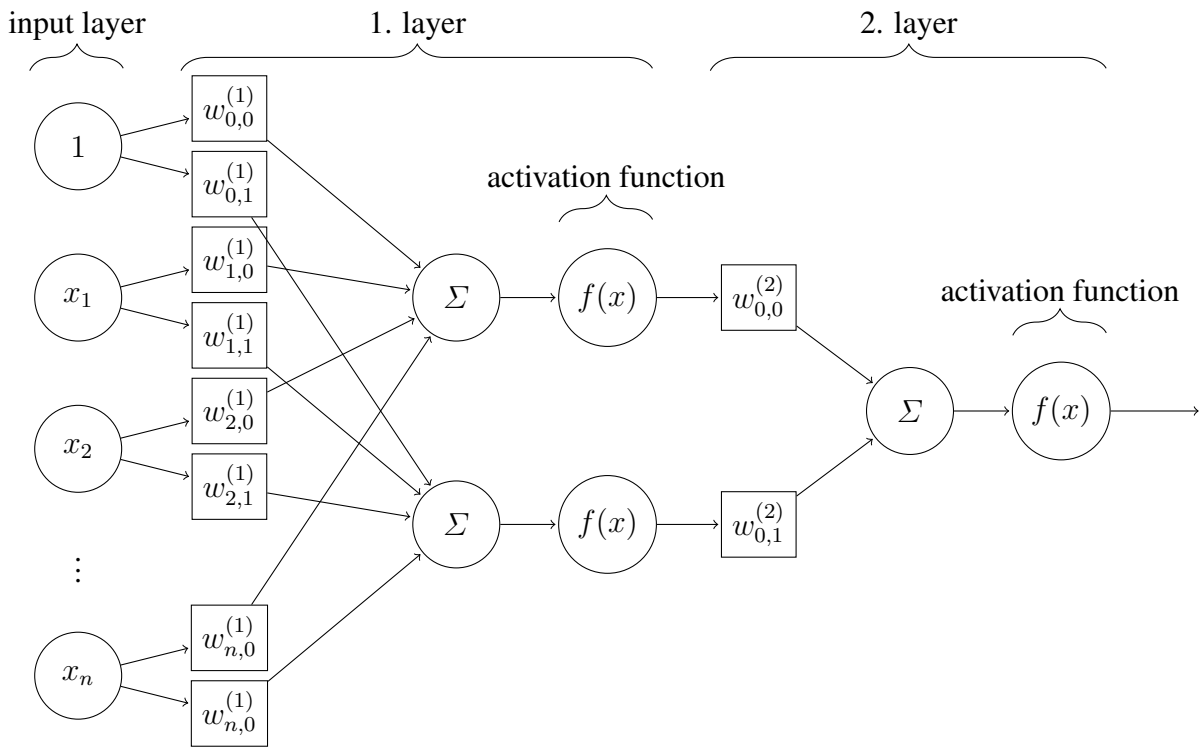


Figure 2.2: A MLP with two hidden layers (two neurons in the first and one in the last), each input gets multiplied with each weight of each neuron, summed up and fed into an activation function to produce the input for the next layer, where this process is repeated.

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:

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

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. 2.14), 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]

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau \tag{2.14}$$

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 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

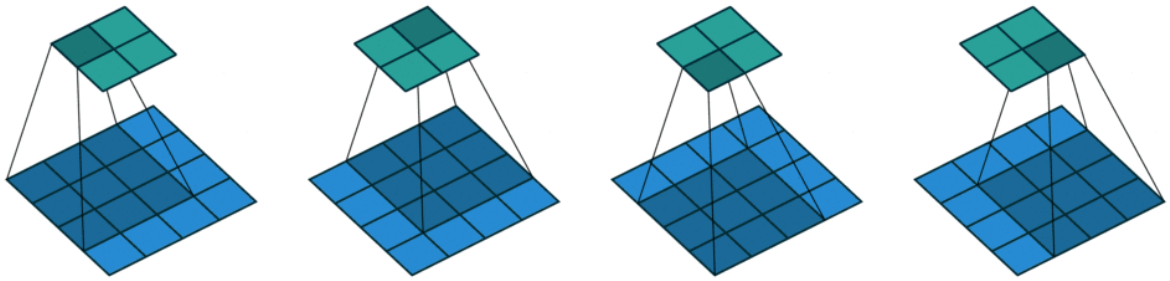


Figure 2.3: Example convolution of a 4x4 input (blue) with a 3x3 kernel (dark blue) and a stride of 1, resulting in a 2x2 output (cyan) [Dum16]

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 ANN. 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.

	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.

2.4 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.

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:

$$bbox = (x_{rel}, y_{rel}, w_{rel}, h_{rel}) \tag{2.15}$$

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 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

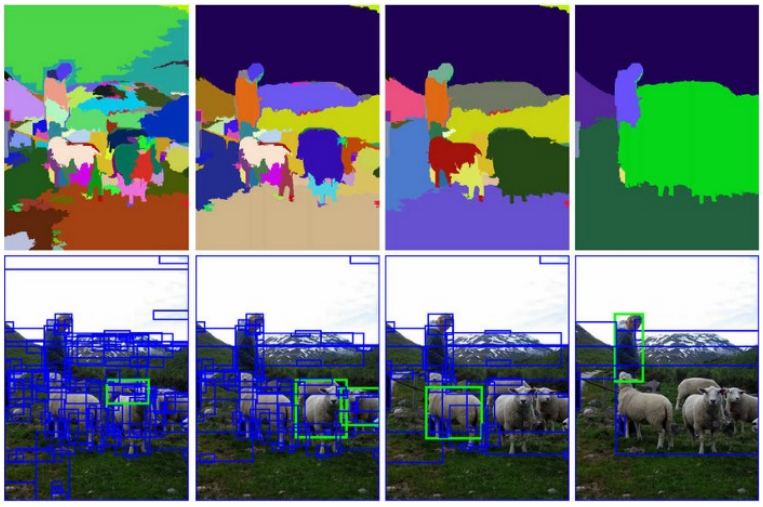


Figure 2.4: Example of results obtained through the Selective Search algorithm (top) with increasing region scale from left to right and bounding boxes drawn around those regions (bottom). Selective Search produces sub-segmentations of objects in an image, considering size, color, texture and shape based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

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.

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2.16}$$

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. 2.17). 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.

$$L_{IoU} = 1 - IoU(A, B) \tag{2.17}$$

Further, to solve these drawbacks Rezatofighi et al. [Rez19] proposed the Generalized IoU (GIoU) Loss (eq. 2.18). 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].

$$L_{GIoU} = 1 - IoU(A, B) + \frac{|C - (A \cup B)|}{|C|} \tag{2.18}$$

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. 2.19) 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.

$$L_{DIoU} = 1 - IoU(A, B) + \frac{\|(A_{center} - B_{center})\|^2}{\|C_{diag}\|^2} \tag{2.19}$$

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. 2.20, 2.21, 2.22). 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].

$$L_{CIoU} = DIoU(A, B) + \alpha(A, B) * \nu(A, B) \tag{2.20}$$

$$\nu(A, B) = \frac{4}{\pi^2}[arctan(\frac{A_w}{A_h}) - arctan(\frac{B_w}{B_h})]^2 \tag{2.21}$$

$$\alpha(A, B) = \frac{\nu}{1 - IoU(A, B) + \nu'} \tag{2.22}$$

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.

$$L_{EIoU} = 1 - IoU(A, B) + \frac{\|(A_{center} - B_{center})\|^2}{\|C_{diag}\|^2} + \frac{\|(A_w - B_w)\|^2}{\|C_w\|^2} + \frac{\|(A_h - B_h)\|^2}{\|C_h\|^2} \tag{2.23}$$

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).

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 featuremap 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

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.

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.

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:

$$Pred_X = \{t_x, t_y, t_w, t_h, q_{conf}, q_{C1}, ..., q_{CC}\} \tag{2.24}$$

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

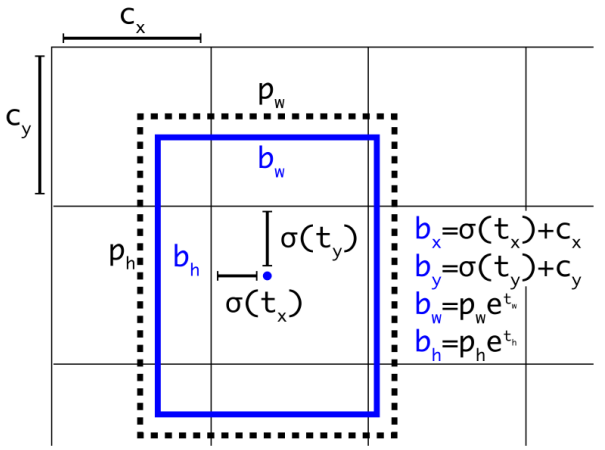


Figure 2.5: Bounding box calculation in the YOLOv4 network based on a prior anchor box [Red16]. The b_* indicate the final prediction, p_* indicate parameters of the prior bounding box and c_* indicate the prior bounding box spatial offset based on the current cell. The final center coordinate offset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

$*$ denotes x and y respectively, is calculated by applying the sigmoid activation function on t_* and adding the grid cell offset c_* to the results. Further, the final size b_+, where $+$ denotes w and h respectively, is calculated by multiplying the prior assigned anchor box sizes p_+ with the exponential function applied to the predicted spatial offset t_+. Finally, the objectness score as well as the class probabilities are also calculated through the sigmoid activation function.

Loss

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

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

In the following 1_{sb}^{obj} denotes that the anchor box b in grid cell s is responsible for detecting the object. Before training, responsibility is assigned by making that anchor box responsible which has the highest IoU with the ground truth bounding box. Furthermore, S^2 denotes the number of

grid cells in a particular scale and B the number of anchor boxes present in a grid cell (three in YOLOv4).

The class loss (eq. 2.25) 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.

$$L_{class} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} \sum_{c \in classes} 1_{sb}^{obj} * CE(P_{sbc}, \hat{P}_{sbc}) \tag{2.25}$$

The bounding box regression loss (eq. 2.26) 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.

$$L_{IoU} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} 1_{sb}^{obj} * w_{scale} * XIoU(bbox, \hat{bbox}) \tag{2.26}$$

$$w_{scale} = 2 - w * h \tag{2.27}$$

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. 2.28) defines the loss which occurs when an object is present and the latter (eq. 2.29) 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.

$$L_{obj} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} CE(1_{sb}^{obj}, \hat{P}_{sb}) \tag{2.28}$$

$$L_{noobj} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} CE(1_{sb}^{noobj}, 1 - \hat{P}_{sb}) * \{max(IoU(\forall bbox, \hat{b}box_{sb})) < t_{ignore}\} \tag{2.29}$$

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

$$L_{YOLO} = \lambda_{class} * L_{class} + \lambda_{IoU} * L_{IoU} + \lambda_{objectness} * (L_{obj} + L_{noobj}) \tag{2.30}$$

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
```

```
20                B ← B − b_i; S ← S − s_i
21            end
22        end
23
24        return B_S , S_S
25    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 $\boldsymbol{\epsilon}$. If no match can be found for a bounding box add it to $\mathbf{L}$ as a cluster of size 1.

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

$$C = \frac{\sum_{i=1}^{T} C_i}{T} \tag{2.31}$$

$$X_{1,2} = \frac{\sum_{i=1}^{T} C_i * X_{1,2}}{\sum_{i=1}^{T} C_i} \tag{2.32}$$

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

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:

$$C = C * \frac{min(T, N)}{N} \tag{2.34}$$

or

$$C = C * \frac{T}{N} \tag{2.35}$$

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 2.34, because there the confidence can't be greater than one. In equation 2.35 the confidence can exceed one, when the number of predicted bounding boxes is higher than the number of predictors, which is often the case.

2.7 Segmentation

general segmentation

mobilenetv2

2.8 Hypergraphs

2.9 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. 2.36) states the proportion of all correct identified samples (TP) in relation to all positive identified samples.

$$Precision = \frac{TP}{TP + FP} \tag{2.36}$$

Recall

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

$$Recall = \frac{TP}{TP + FN} \tag{2.37}$$

F1-Score

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

$$F1 = \frac{2 * recall * precision}{recall + precision} \tag{2.38}$$

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. 2.6, 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. 2.39).

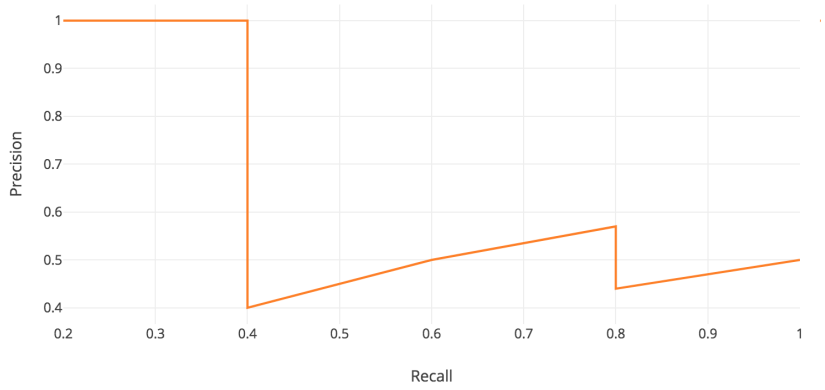


Figure 2.6: Example of a precision-recall curve, where precision and recall were calculated for different IoU thresholds and sorted and plotted by their recall values [Hui18]. The AP is the area under the curve.

$$AP = \int_0^1 Precision(Recall) dRecall \tag{2.39}$$

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. 2.40).

$$mAP = \frac{1}{N} \Sigma_{i=1}^{N} AP_i \tag{2.40}$$

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. 2.41).

$$mIoU = \frac{1}{N} \Sigma_{i=0}^{N} IoU_i \tag{2.41}$$

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 MobilenetV2-UNet. Therefore, the data was labeled with bounding boxes for YOLOv4 and with segmentation masks for the MobilenetV2-UNet

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

3.2.1 General Overview

predict: bboxes, seg

postprocess: generate topology

convert: LTspice

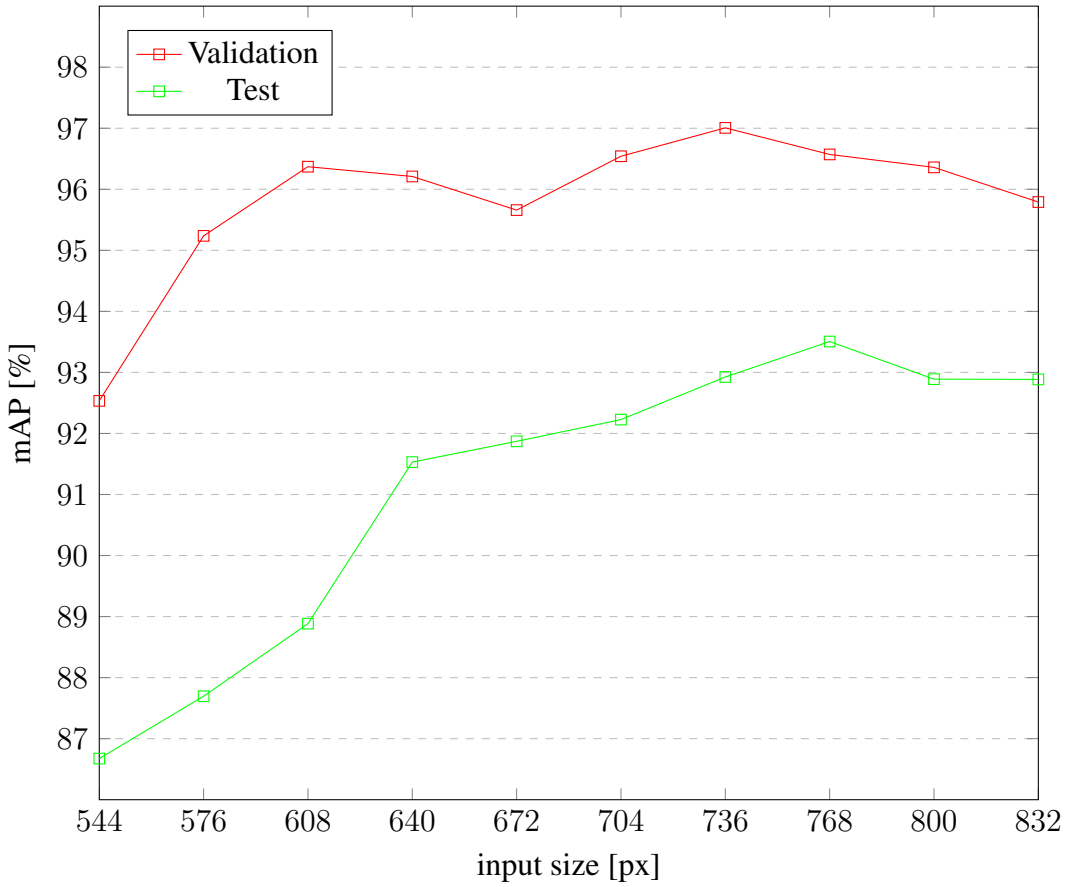
class	**total**	**train ratio**	**valid ratio**	**test ratio**
diode left	156	83.33%	8.97%	7.69%
diode top	210	82.38%	6.19%	11.43%
diode right	150	82.00%	12.67%	5.33%
diode bottom	102	67.65%	15.69%	16.67%
resistor horizontal	318	71.38%	6.92%	21.70%
resistor vertical	350	66.00%	6.57%	27.43%
capacitor 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.

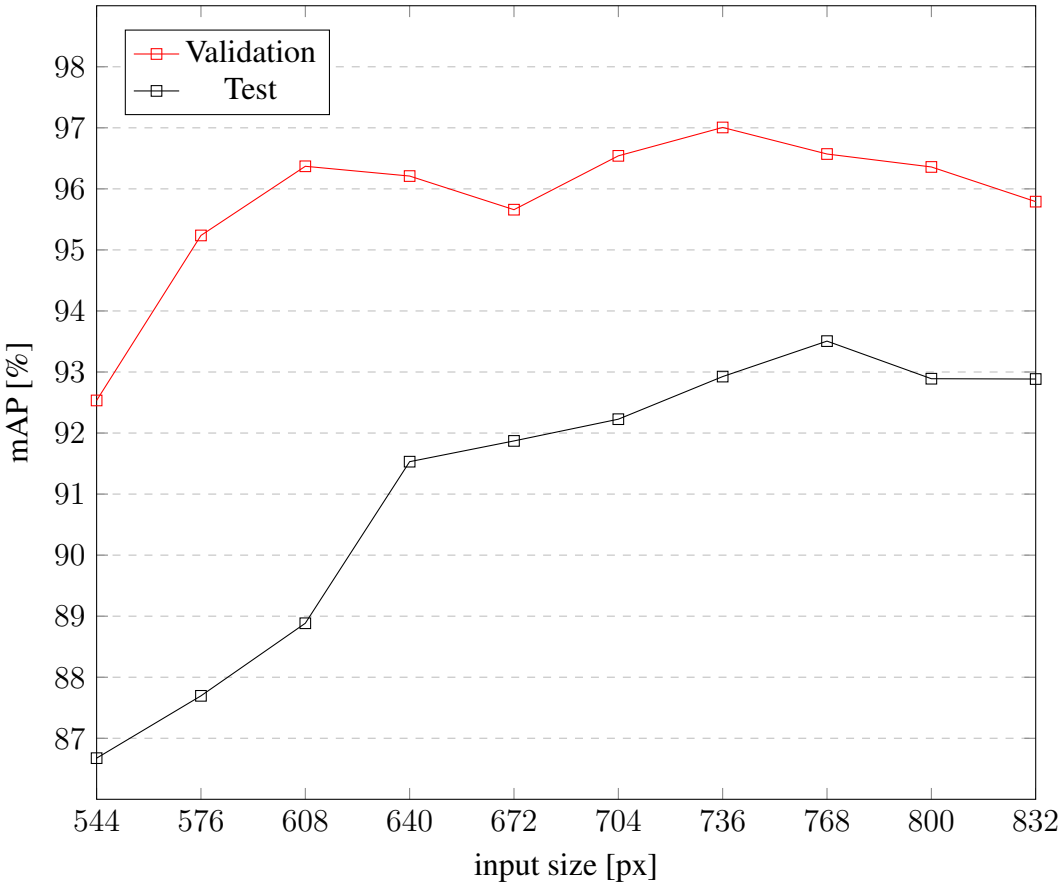
3.3 Training

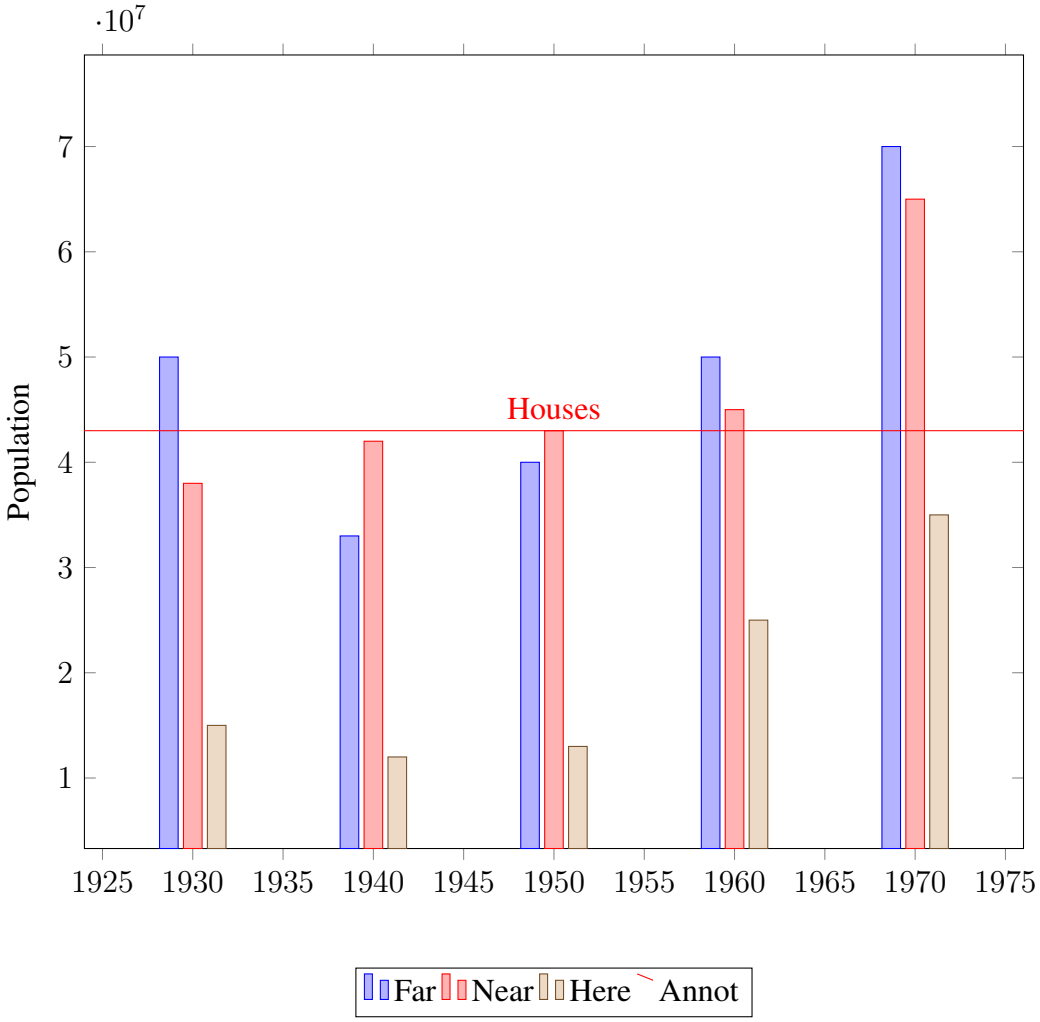
3.3.1 YOLOv4-Tiny

TODO entry YOLO evaluation for different input sizes, on validation and test set.



TODO entry YOLO evaluation for different input sizes, on validation and test set.





MobileNetV2-UNet

Chapter 4

Results

Chapter 5

Discussion

aklsdfj alksdfj

Appendix A

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

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`.

[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.

[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.

[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.

[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.

[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), March 2020.

[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.

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

[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), July 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.

[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.