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 15. April 2021

Übersicht

Abstract

Contents

Chapter 1

Introduction

1.1 Motivation

1.2 Related Works

1.3 Goals of the Thesis

1.3.1 Task Description

1.3.2 Contribution

Chapter 2

Theory

2.1 Electrical Circuit Diagrams

An Electrical Circuit Diagram (ECD) consists of Electrical Circuit Components (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. In this thesis

- TODO introduce symbols
- TODO sources current with arrows
- TODO image
- TODO parallel reihen schaltung
- TODO introduce notation of each used ecc
- TODO say that eccs are not rotation invariant

2.2 LTspice File

- TODO program itself?

Since the hand drawn ECDs are to be converted into a LTspice format, the general structure, as well as the syntax for LTspice schematic files (file extension *.asc*), is presented here.

For this thesis the basic syntax was reverse engineered by creating a schematic in LTspice, reading the file, changing a value and analyzing the effect it had on the schematic inside of LTspice.

2.2.1 General

LTspice files are written in plain text and are human readable. The file structure has to 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 provided to 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.

2.2.2 File Definition

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.

- TODO Note that there is no constraint which point has to be first or second as long as the wire endpoint overlaps with a component it is connected.

	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]. TODO How the signal is processed depends on the biological structure of the neuron. In general a biological neuron produces an output signal, when a certain activation potential is reached.

The first artificial neuron was proposed in 1958 by Rosenblatt [Ros58] and is known as the Rosenblatt Perceptron. 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 $V \in \mathbb{R}^{n \times 1}$ with an input vector $x \in \mathbb{R}^{n \times 1}$. 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 TODO CITE.

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

To tackle this insufficiency the Multi Layer Perceptron (MLP) was introduced [Gro73], which is able to solve non-linear problems. The MLP is created by using the output of multiple perceptrons as an input to another perceptron. In fig. 2.1 the general structure of a MLP is shown. This particular MLP has two neurons in the first layer and one in the second layer. In general the number for neurons inside a layer, as the number of layers is not bounded.

Learning Procedure

The learning procedure of an ANN is as follows:

1. The input data gets fed into the network and produces an output.
2. The similarity of the output and the desired output (label) are measured using a loss function.
3. The loss is used to calculate the gradients by propagating it back through the network.
4. The gradients are used in a gradient descent algorithm to update and optimize the weights of the network.
5. The above is performed until the weights don't change anymore.

Forward Propagation

As with the perceptron the input vector x gets multiplied with the corresponding weights $W_{n,i}$, where n denotes the index of the neuron in the layer and i the index of the input vector x. After the multiplication the results are summed up and are forwarded to an activation function. The results of the activation function are now inputs for the second layer and the above is repeated. In the end, the second layer activation function produces an output $\hat{y}$.

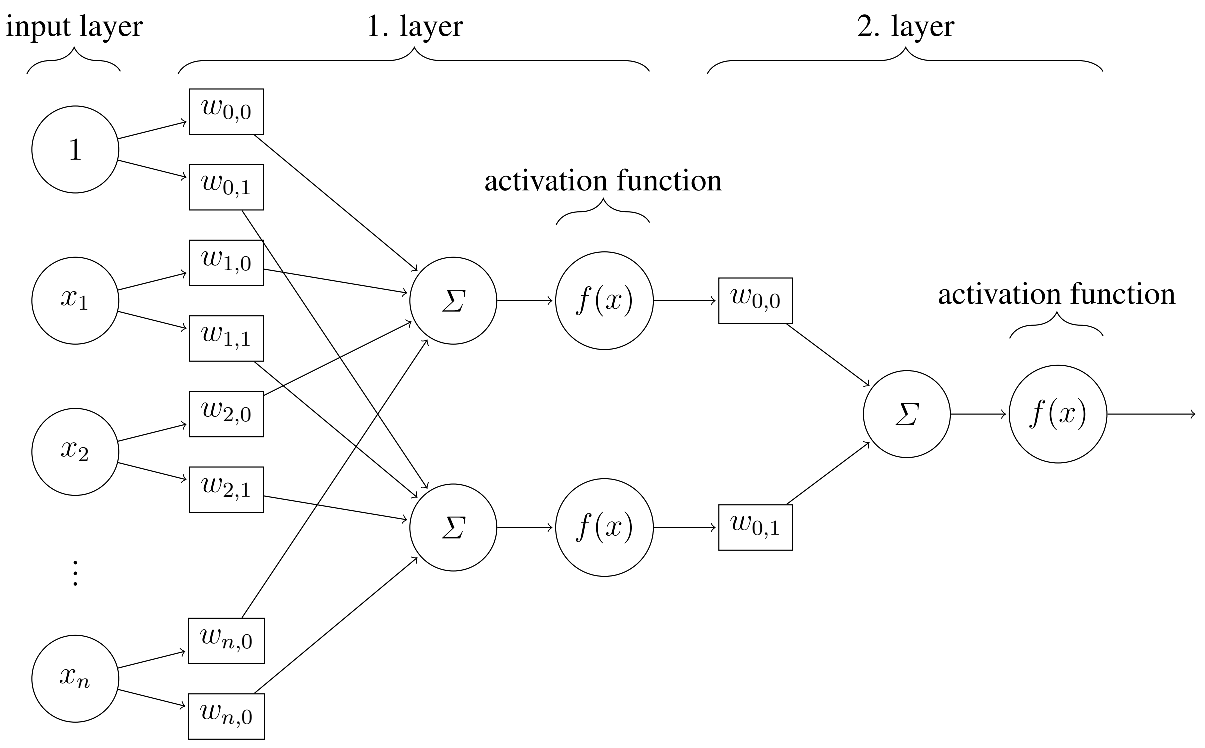

Figure 2.1: Multi Layer Perceptron with two layers

Loss

After the output $\hat{y}$ was calculated a loss function is applied to $\hat{y}$ and the label y. The most prominent of these loss functions is the Mean Squared Error (MSE) (see eq. 2.2). The MSE takes the labels and the output and calculates the sum of the differences between the two vectors. Finally, the mean is taken of the resulting sum, where n here denotes the dimensionality of y and $\hat{y}$.

$$MSE(y, \hat{y}) = \frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.2}$$

Backward Propagation

To make an ANN learn the

While in the perceptron the sign function is used as an activation function, for MLPs this is not a good choice, because the sign function is non-differentiable. In particular, this is important because a gradient is needed to update the weights of an MLP during training.

Optimization

2.3.2 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 $100x100$, this would already require the input size of the network to have 10.000 weights per neuron in the next layer. This increases drastically the capacity of the network and hence requires a larger training set. Additionally nearby pixels in images are often highly correlated and classical unstructured ANN fails to capture such spatial dependencies. [LeC99]

- TODO transforms and bla

The proposed alternative therefor are Convolutional Neural Networks (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
- 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. The equation for a convolution (see eq. 2.3), states that a function f convolved with another function g, is the same as 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]

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \tag{2.3}$$

In simpler terms that just means there is an image I with $I \in \mathbb{N}^{W_I x H_I x C_I}$, where W_I and H_I are the width and height of the image and C_I being the number of channels. Furthermore, there is a kernel K with $K \in \mathbb{N}^{W_K x H_K x C_I}$. 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

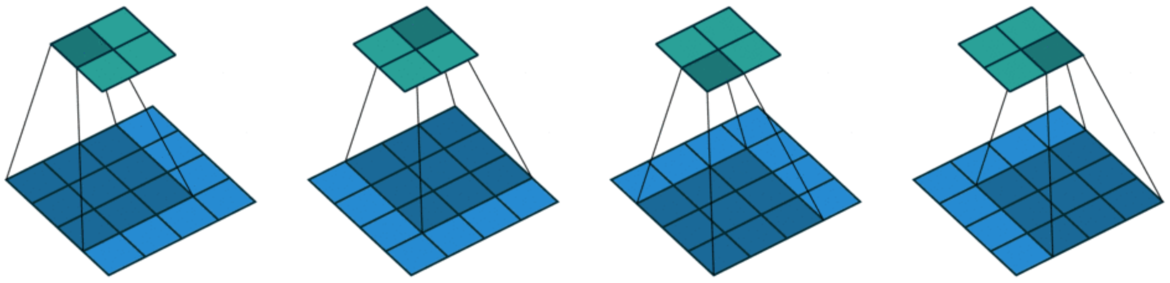

Figure 2.2: Example convolution of a 4x4 input (blue) with a 3x3 kernel (dark blue) and a stride of 1, resulting in a 2x2 output (cyan) [Dum16]

been convolved. 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 $4x4$ input with a $3x3$ kernel and a stride of 1 is given in figure 2.2.

- TODO size of output?
- TODO padding?

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 $2x2$ or $3x3$) 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]

- TODO average pooling?
- TODO global average?
- TODO maybe Vincents learnable pooling

Fully-Connected Layer

The fully-connected layer as described in 2.3.1 is the final layer inside a CNN. Before the calculations of this layer are applied, the input tensor is flattened into a vector. The input corresponds to some high level features which were previously build through the convolutional and pooling layers. The last operation in this output layer in a multi-class classification task is normally a softmax activation function [Bri90], which produces a probability vector. Each element in the vector corresponds to the probability of a class in the task. The softmax function is defined as:

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{2.4}$$

Which basically means that the output probability $softmax(x)_i$ is defined as the fraction of the exponential function applied to an element of the vector, divided by the sum of all exponential function outputs applied to all elements of the vector.

2.3.3 Activation Functions

Rectified Linear Unit (ReLU)

ReLU6

Loss Functions

- TODO CE

Skip Connections

2.3.4 Batch Normalization

2.3.5 Dropout

2.4 Data Augmentation

2.5 Object Detection

Object detection is one of the subtasks in the image domain. It is an extension of the classical classification task, where additionally to the predicted class, the location of the object should be provided. The location is normally given as a bounding box. Various formats for the bounding box definition exist. One common format is $bbox = (x1, y1, x2, y2)$, where $(x1, y1)$ being the coordinate of the upper left corner of the bounding box and $(x2, y2)$ the lower right corner TODO cite. Another format, used in the coco dataset is $bbox = (x, y, w, h)$, where again (x, y) define the upper left corner and (w, h) the width and the height of the bounding box TODO cite.

TODO this is not fully correct just define it later in YOLO chapter.

In this thesis the format of Redmon et al. [Red15] is used. Which is defined as $bbox = (x_{rel}, y_{rel}, w_{rel}, h_{rel})$. Here, (x_{rel}, y_{rel}) define the relative center of the bounding box and (w_{rel}, h_{rel}) the relative width and height of the bounding box. Relative means that each coordinate is normalized over its corresponding axis. E.g. x_{rel} would be calculated through $x_{rel} = \frac{x_{abs}}{max_x}$, where max_x being the image size in x direction. The advantages of this format are, that the definition of the bounding box becomes invariant to the image size. One can resize the image without having to recalculate the bounding box, as it is the case with an absolute format.

2.5.1 History of Object Detection

Sliding Window

The simplest algorithm to detect objects in an image is the sliding window approach. Before an object detection can be performed on an image, a classifier has to be trained. This classifier is normally trained on image patches, where a patch has roughly the size of the objects it should classify. The object detection phase starts by dividing the input image into patches. Those patches are now fed to the classifier and when the predicted probability exceeds a predefined threshold the patch is considered to have an object in it. It should be noted that classification accuracy can be improved by feeding overlapping patches into the classifier. The resulting predicted bounding boxes now look like the image on the left side in fig. 2.3. It contains multiple bounding boxes for the same object. To have only one prediction per object, a Non-Maximum Suppression (NMS) algorithm is applied on the overlapping bounding boxes. The most common NMS algorithm is the greedy-NMS. Here bounding boxes are grouped together, when their overlap exceeds a certain threshold. Rejection of neighboring bounding boxes is done by using the predicted class probability i.e. using only those bounding boxes with the highest prediction score and rejecting the others [Hos17]. The results of a NMS can be seen on the right side in fig. 2.3.

- TODO add tackle scale by using patches of different size

Regions with CNN Features (R-CNN)

While the sliding window approach is effective, it is also highly inefficient, since every generated has to be processed 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. Selective Search produces sub-segmentations of objects in an image, considering size, color, texture and shape based features for the grouping of the regions.

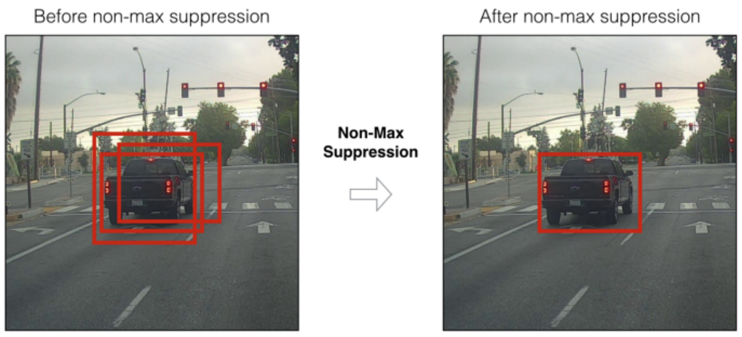

Figure 2.3: Predicted bounding boxes before and after a non-maximum suppression was applied [Sam]

How the algorithm performs and what kind of bounding boxes are produced, can be seen in fig. 2.4, where the size of the regions is increased from left to right. The second row of the fig. additionally shows the proposed bounding boxes. 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 4096-dimensional feature vector. In the next step the 4096-dimensional feature vector is fed into $N + 1$ binary Support Vector Machines (SVMs), where N is the number of classes to predict plus one background class. To further improve the bounding box additionally a bounding box regressor is trained for each class as describe in [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. First a CNN had to be fine tuned. One SVM had to be trained for each existing class, which involved saving a lot of CNN produced features to a hard drive. Lastly a bounding box regressor for each class had to be trained. Furthermore, the inference process took around 47s per image with a VGG16 [Sim14] backbone. [Ger15]

With his work on Fast R-CNN Girshick [Ger15] improved on the training and inference time by a large margin in contrast to R-CNN. He was able to perform the training process with a VGG16 network 9 times faster and the inference 213 times faster.

The main difference to R-CNN is that instead of generating region proposals, warping them to the input size of the CNN 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

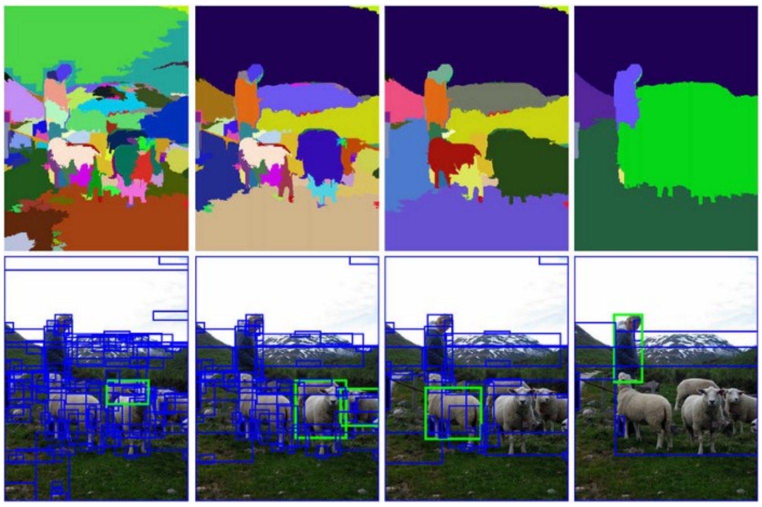
Figure 2.4: Example of results obtained through the Selective Search algorithm with increasing region scale from left to right [vdS11]

Interest (RoI) pooling layer. Meaning that now the CNN computations are now 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, since it showed to perform on par with the SVMs. The second branch is comprised of a bounding box regression layer, which outputs bounding box regression offsets as in R-CNN [Gir13]. Due to the one-stage nature of the pipeline a novel multi-task loss was required, which the author defined as follows:

$$L(\hat{y}, y, \hat{b}, b) = L_{cls}(\hat{y}, y) + \lambda L_{loc}(\hat{b}, b) \tag{2.5}$$

$\hat{y}$ being the predicted class probabilities by the softmax layer and y the ground truth class.

- TODO should I even do that further?

Faster R-CNN

In Fast R-CNN the prediction time was decreased by compressing the multi-stage pipeline into a single-stage pipeline. The remaining non-learnable part of the pipeline became the region proposal algorithm. It still took around $2s$ to propose bounding boxes with Selective Search. Ren et. al therefore proposed Faster R-CNN [Ren15] to further increase the performance of the overall algorithm. In Faster R-CNN the performance is boosted through the novel Region Proposal

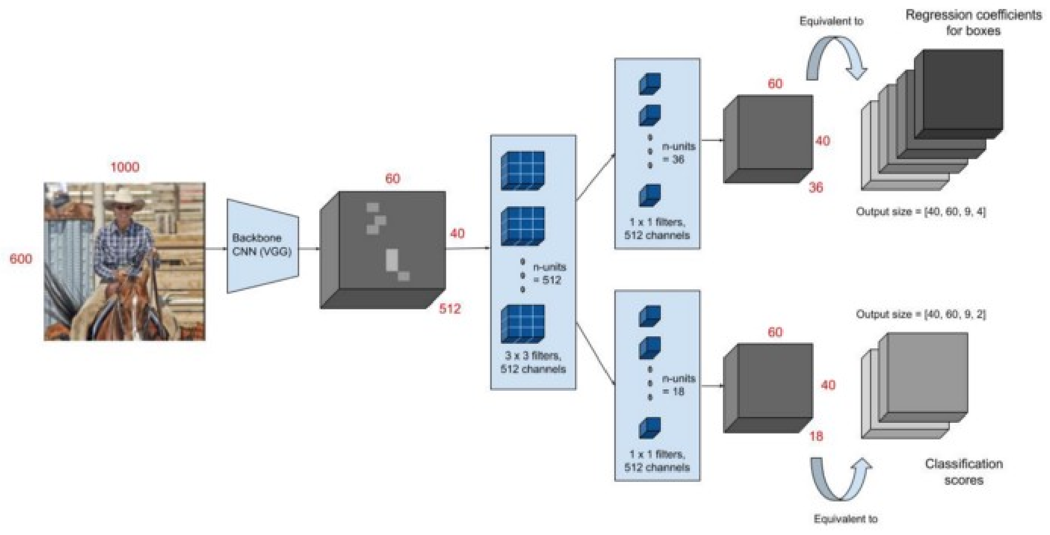
Equivalent to
Regression coefficients for boxes
1000
600
Backbone CNN (VGG)
60
40
512
n-units = 512
3 x 3 filters, 512 channels
n-units = 36
1 x 1 filters, 512 channels
60
40
36
Output size = [40, 60, 9, 4]
Output size = [40, 60, 9, 2]
n-units = 18
1 x 1 filters, 512 channels
60
40
18
Classification scores
Equivalent to

Figure 2.5: The Faster R-CNN Network Architecture [Ana] TODO img quality sucks

Network (RPN). As with Fast R-CNN an image is first fed to the CNN to produce feature maps. Afterwards, the feature maps are used as an input to the RPN. The RPN operates in a sliding window fashion, where a window size of nxn ($n = 3$) is used to traverse the whole feature map and produce features for the following network.

Single Shot Detector

TODO? Anchor Box Free Detection?

2.5.2 Intersection Over Union (IoU)

The Intersection over Union (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. TODO a bit more; maybe also cite last sentence; last sentence sucks

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2.6}$$

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 each regressed variable (x, y, w, h) is independent and can be optimized separately [Yu16].

To take the correlation of those variables into account, Yu et al. [Yu16] proposed the IoU Loss (eq. 2.7). 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.

$$L_{IoU} = 1 - IoU(A,B) \tag{2.7}$$

Further, to solve these drawbacks Rezatofighi et al. [Rez19] proposed the Generalized IoU (GIoU) Loss (eq. 2.8). 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].

$$L_{GIoU} = 1 - IoU(A,B) + \frac{|C - (A \cup B)|}{|C|} \tag{2.8}$$

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. 2.9) 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.

$$L_{DIoU} = 1 - IoU(A,B) + \frac{||(A_{center} - B_{center})||^2}{||C_{diag}||^2} \tag{2.9}$$

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. 2.10, 2.11, 2.12). 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].

$$L_{CIoU} = DIoU(A,B) + \alpha(A,B) * \nu(A,B) \tag{2.10}$$

$$\nu(A,B) = \frac{4}{\pi^2}[arctan(\frac{A_w}{A_h}) - arctan(\frac{B_w}{B_h})]^2 \tag{2.11}$$

$$\alpha(A,B) = \frac{\nu}{1 - IoU(A,B) + \nu'} \tag{2.12}$$

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.

- TODO they also incorporated focal loss into that, maybe say that too, at least when I use it

$$L_{EIoU} = 1 - IoU(A,B) + \frac{\|(A_{center} - B_{center})\|^2}{\|C_{diag}\|^2} + \frac{\|(A_w - B_w)\|^2}{\|C_w\|^2} + \frac{\|(A_h - B_h)\|^2}{\|C_h\|^2} \tag{2.13}$$

2.5.4 Anchor Boxes

- TODO meh

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. Anchor boxes are used to assign a ground truth to a prediction of a network, which is then used to apply a loss on the prediction and hence make the network learn. For example in Faster R-CNN the ground truth bounding box is assigned to an anchor when the IoU of the bounding box and the anchor box is greater than a certain threshold. Anchor boxes can be seen as predictors, who over time get better and better of predicting objects of certain size and aspect ratios [Red15]. (TODO actually more the layer outputs which are multiplied with that anchor) The scale and aspect ratio is often selected by using the k-means clustering algorithm on the dataset, to obtain optimal scales and aspect ratios representing the underlying dataset [Red16]. The selection of good anchor box sizes can improve the prediction quality [Ren15].

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

- TODO backbone is a smaller scale of the original Darknet53

The backbone of the YOLOv4 network is mostly formed out of YOLOConv layers (tab. 2.8), which is sequentially build out of a convolutional layer, a batch normalization layer and a leaky Rectified Linear Unit (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

Layer	Input	Type	Filters	Size	Stride	Padding	Output
Reduction Layer							
c0	img	YOLOConv	32	3x3	2	valid	
c1	c0	YOLOConv	64	3x3	2	valid	
Layer 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	
Layer 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	
Layer 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	
Layer 4							
c14	m2	YOLOConv	512	3x3	1	same	$Skip_L$

Table 2.7: CSPDarknet53Tiny Architecture

K the number of filters beginning with 64. So as can be seen in 2.7, 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. Outputs have a size of $(2^{L-1} * S)x(2^{L-1} * S)$, where L is the output layer number. Hence, the spatial size of the output is doubled with increasing

Sequence	**Parameter**
Convolution	see tab. 2.7; l2 kernel regularization $5x10^{-3}$
BatchNormalization	
LeakyReLU	$\alpha = 0.1$

Table 2.8: Convolutional Base Block in YOLOv4 (YOLOConv)

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 2.9: PANetTiny Architecture

network depth. Therefore, $Pred_L$ has a size of SxS, $Pred_M$ has a size of $2Sx2S$ and $Pred_S$ has a size of $4Sx4S$. 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 $1x1$ convolution and hence acts as a fully-connected layer. The output size of each output layer is $3 * (5 + C)$, 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 $(2^{L-1} * S)x(2^{L-1} * S)x(3 * (5 + C))$.

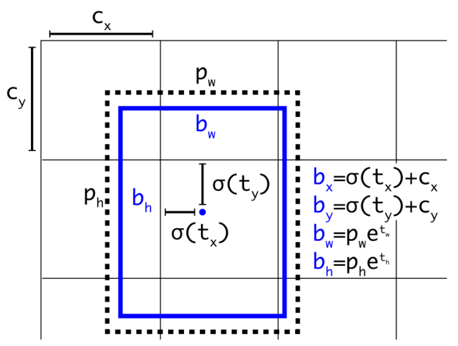

Figure 2.6: Final bounding box calculation in the YOLOv4 network based on an anchor box [Red16]

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:

$$Pred_X = \{t_x, t_y, t_w, t_h, q_{conf}, q_{C1}, ..., q_{CC}\} \tag{2.14}$$

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

Loss

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

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

In the following 1_{sb}^{obj} denotes that the anchor box b in grid cell s is responsible for detecting the object. Before training responsibility is assigned by making that anchor box responsible which has the highest IoU with the ground truth bounding box. Furthermore, S^2 denotes the number of grid cells in a particular scale and B and the number of anchor boxes present in a grid cell. All predicted values are denoted with a “hat” (e.g. $\hat{P}$).

The class loss (eq. 2.15) is calculated by taking the sum of all possible anchor boxes and the sum of all independent class losses, which is calculated with the Cross Entropy (CE) Loss. An anchor box only contributes to the loss, when an object is labeled to be present in it.

$$L_{class} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} \sum_{c \in classes} 1_{sb}^{obj} * CE(P_{sbc}, \hat{P}_{sbc}) \tag{2.15}$$

The bounding box regression loss (eq. 2.16) 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.

$$L_{IoU} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} 1_{sb}^{obj} * w_{scale} * XIoU(bbox, \hat{bbox}) \tag{2.16}$$

$$w_{scale} = 2 - w * h \tag{2.17}$$

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. 2.18) defines the loss which occurs when an object is present and the latter (eq. 2.19) 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.

$$L_{obj} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} CE(1_{sb}^{obj}, \hat{P}_{sb}) \tag{2.18}$$

$$L_{noobj} = \sum_{s=0}^{S^2} \sum_{b=0}^{B} CE(1_{sb}^{noobj}, 1 - \hat{P}_{sb}) * \{max(IoU(\forall bbox, \hat{b}box_{sb})) < t_{ignore}\} \tag{2.19}$$

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

$$L_{YOLO} = \lambda_{class} * L_{class} + \lambda_{IoU} * L_{IoU} + \lambda_{objectness} * (L_{obj} + L_{noobj}) \tag{2.20}$$

Non-Maximum Suppression

During prediction time to further increase the detection performance a NMS algorithm is applied. More precisely DIoU-NMS [Zhe19], since it has shown to perform better than classical NMS [Bod17], especially in occlusion cases.

2.7 Segmentation

2.8 Connected Components Analysis

- TODO write why I need CCA?

- TODO only general

A Connected Component Analysis (CCA) describes the process of labeling connected pixels in a binary image.

In the most simple case a structuring element such as a cross (four-connection-labeling) or a rectangle (eight-connection-labeling) is moved over an image and if two pixels are neighbors and their value is the same they are considered to have the same label. In this thesis the eight-connection-labeling algorithm by Grana et al. [Gra10] is used.

2.9 Optical Character Recognition

2.10 Hypergraphs

2.11 Metrics

Chapter 3

Material and Methods

3.1 Data

3.1.1 Statistics

3.1.2 Label Format

3.1.3 Training Split TODO

3.2 Recognition and Conversion Pipeline

3.2.1 General Overview

3.2.2 Component Detection

3.2.3 ECD TODO Segmentation

3.2.4 Training

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

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

PANet Path Aggregation Network

List of Figures

List of Tables

Bibliography

[Ana] S. Ananth. Faster r-cnn for object detection. `https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46`. Accessed: 07.04.2021.

[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.

[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.

[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.

[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.

[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.

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

[Hos17] J. Hosang, R. Benenson, and B. Schiele. Learning non-maximum suppression. *CoRR*, abs/1705.02950, 2017.

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

[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.

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

[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.

[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.

[Sam] K. Sambasivarao. Non-maximum suppression (NMS). `https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c`. Accessed: 02.04.2021.

[Sim14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[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.

[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.

[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.