Bibliography

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

[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.

[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.

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

[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.

[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.

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 2. 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 Electrical Circuit Component (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. For instance a resistor can be denoted as “100 mΩ” (Milliohm).

- TODO introduce more symbols?

- TODO generally create hyperref to abbrevations

- TODO sources current with arrows?

- TODO image?

2.2 LTspice File

- TODO program itself?

Since the hand drawn Electrical Circuit Diagrams (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 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

2.2.3 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

2.2.4 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

2.2.5 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

2.2.6 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 the formula 2.1. It states that the output $\hat{y}$ is defined by the sign of the dot product of a weight vector W with an input vector x. 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.

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

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 $\hat{y}$.

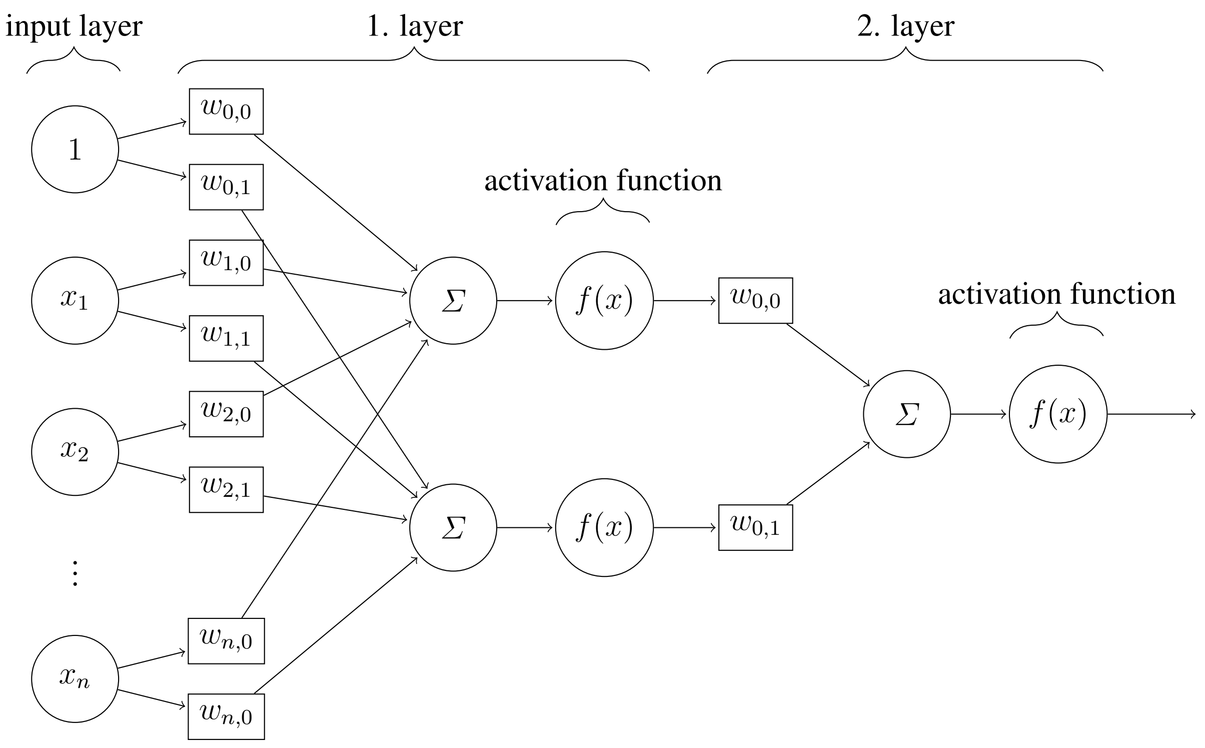

Figure 2.1: Multi Layer Perceptron with two layers

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 Multi Layer Perceptrons (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 Convolutional Neural Network (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. Futhermore, 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

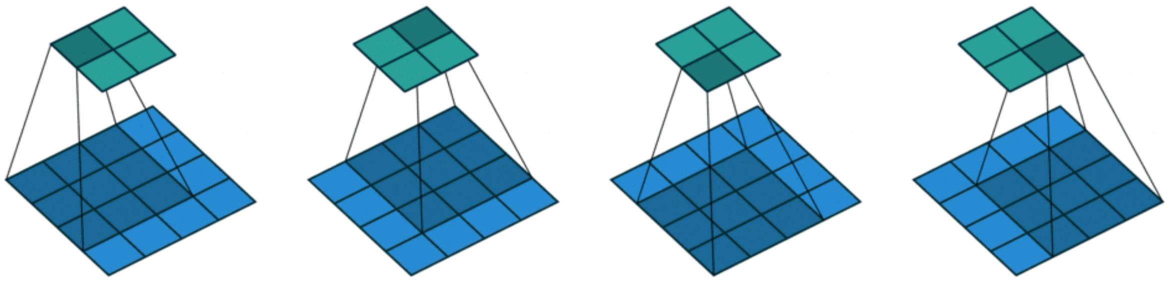

Figure 2.2: Example of a convolution of a 4x4 image (blue) with a 3x3 kernel (dark blue), resulting in a 2x2 output (cyan). [Dum16]

pixel for the convolution result. Finally, the kernel is shifted further until the whole image has 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 $4x4$ image with a $3x3$ kernel and a stride of 1 is given in figure 2.2.

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?

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 is often in a multiclass task a softmax function, which produces some pseudo output probabilities.

- TODO softmax?

2.3.3 Batch Normalization

2.3.4 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 TOOD cite. 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)

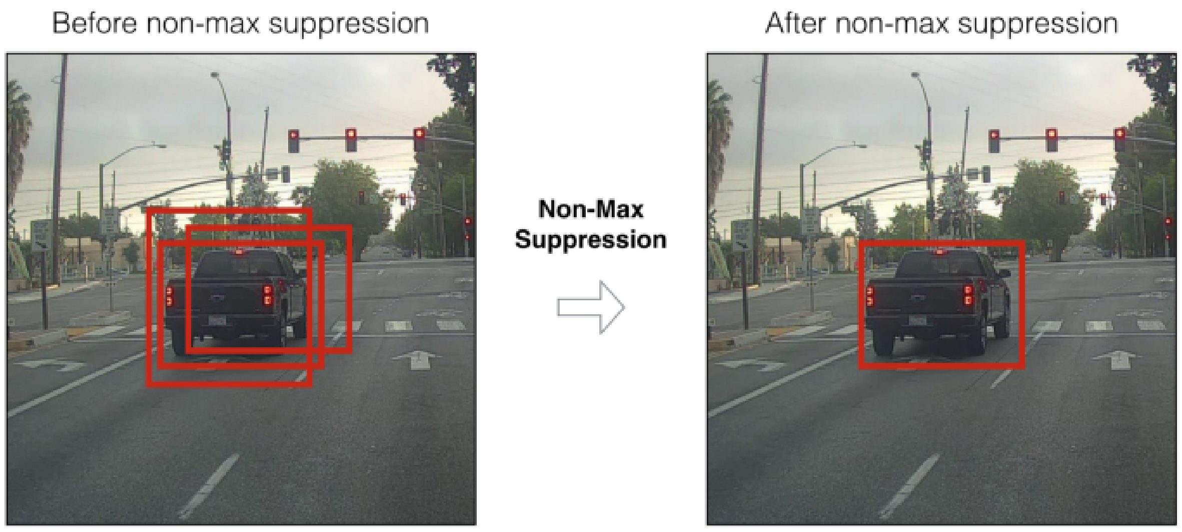

Figure 2.3: Predicted bounding boxes before and after a non-maximum suppression was applied [Sam]

algorithm is applied on the overlapping bounding boxes. The results of the 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 all generated patches have to be processed in order to obtain the bounding boxes of the 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 probabel regions. In contrast to brute forcing each possible region in an image this was a major improvement in performance. In their work the selective search algorithm [vdS11] was used to generate region proposals. The selective search algorithm produces sub-segmentations of objects in an image, considering size, color, texture and shape based features, for the grouping of the regions. How the algorithm performs and what kind of bounding boxes are produces can be seen in fig. 2.4. The size of the regions is increased from left to right, additionally the proposed bounding boxes can be seen in the second row. 2000 of those proposed bounding boxes are now taken from different scales and warped into the input shape of the following CNN, disregarding the size or aspect ratio of the proposed region. Each region proposal is passed through the CNN and yields a 4096-dimensional feature vector. The final step

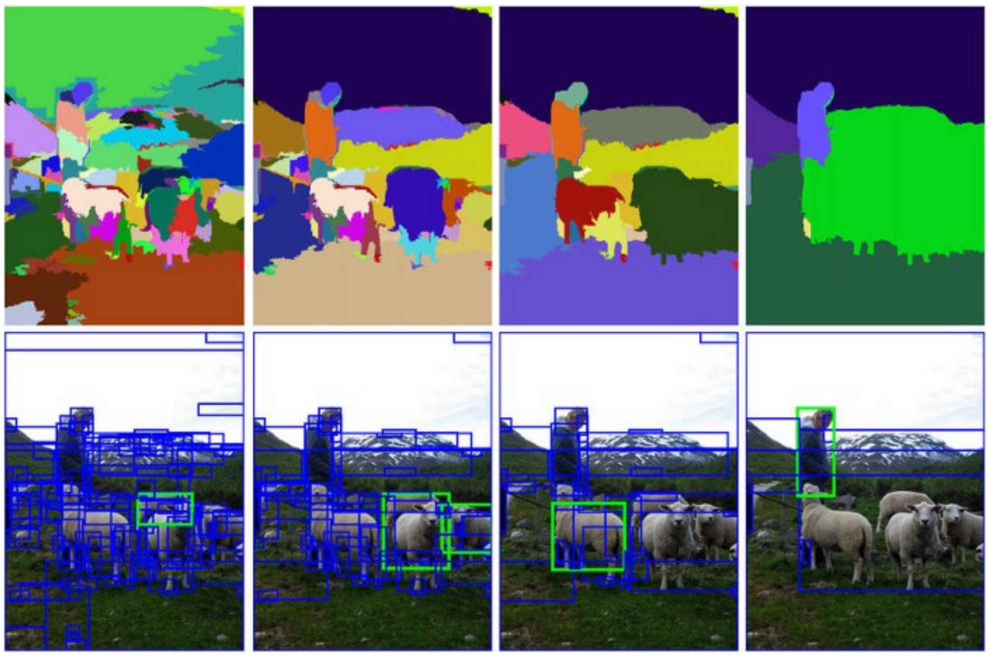
Figure 2.4: Example of results obtained through the selective search algorithm with increasing region scale from left to right [vdS11]

in the detection pipeline is comprised of feeding the 4096-dimensional feature vector into $N + 1$ binary-Support Vector Machines (SVMs), where N is the number of classes to predict plus one background class.

- TODO bbox regression fine tune

Fast R-CNN

- faster through sharing computation
    - spatial pyramid pooling to tackle arbitrary image sizes
    - region proposal performed on feature maps
    - joint multi task loss combining bbox reg and classification

Faster R-CNN

Singleshot Detector

2.5.2 You Only Look Once

2.6 Segmentation

2.7 Connected Components Analysis

- TODO write why I need CCA?

- TODO later

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.

2.8 Optical Character Recognition

2.9 Hypergraphs

2.10 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

Chapter 4

Results

Chapter 5

Discussion

aklsdfj alksdfj

Appendix A

Abbrevations

ANNs Artificial Neural Networks

ANN Artificial Neural Network

CCA Connected Component Analysis

CNNs Convolutional Neural Networks

CNN Convolutional Neural Network

ECCs Electrical Circuit Components

ECC Electrical Circuit Component

ECDs Electrical Circuit Diagrams

ECD Electrical Circuit Diagram

MLPs Multi Layer Perceptrons

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

SVMs Support Vector Machines

List of Figures

List of Tables

Bibliography

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

[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.

[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.

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

[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.

[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.

[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.