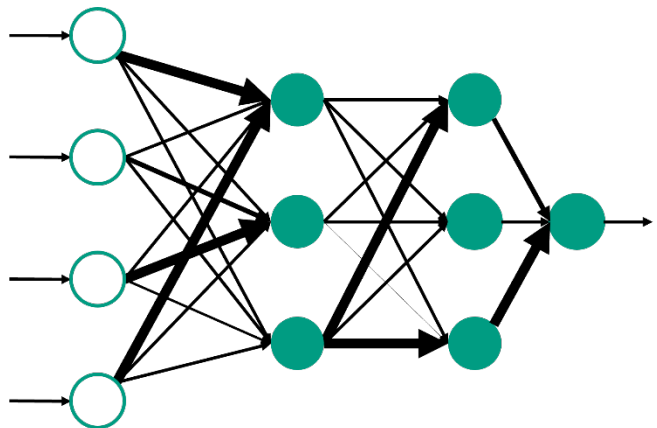


# DEEP LEARNING FOR COMPUTER VISION

A MANUAL FOR THE IT-MINDED ENGINEER



**Author**

Jeroen Linssen

**Date of publication**

2022-05-20

**Ambient  
Intelligence**

*Enabling IT for a smart world*



**Mechatronics**

## COLOPHON

Name	Email	Website
<b>Jeroen Linssen</b> Associate Lector Manual author	<a href="mailto:j.m.linssen@saxion.nl">j.m.linssen@saxion.nl</a>	<a href="http://saxion.nl/ami">saxion.nl/ami</a>
<b>Roy de Kinkelder</b> Project leader	<a href="mailto:r.dekinkelder@saxion.nl">r.dekinkelder@saxion.nl</a>	<a href="http://saxion.nl/mechatronics">saxion.nl/mechatronics</a>

## AVAILABILITY

This manual is published by the Saxion University of Applied Sciences. This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/). All source material is referenced with the required licensing.

An accompanying webpage is available at <https://github.com/SaxionAMI/Saxion-AI-Training/>. All resources are archived there, including a copy of this manual.

Some material is gratefully borrowed and adapted from <https://github.com/glouppe/info8010-deep-learning> under a BSD-3 license, and from the Saxion HBO-ICT specialization Big Data Technologies.

In the tutorial of this manual (Section 6), use is made of code available under the MIT License and the Apache License, Version 2.0. These licenses are copied below as part of their requirements.

Licensed under the Apache License, Version 2.0 (the "License");  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at

<https://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.

## MIT License

Copyright (c) 2017 François Chollet

Permission is hereby granted, free of charge, to any person obtaining a  
 copy of this software and associated documentation files (the  
 "Software"),  
 to deal in the Software without restriction, including without limitation  
 the rights to use, copy, modify, merge, publish, distribute, sublicense,  
 and/or sell copies of the Software, and to permit persons to whom the  
 Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## LIST OF ACRONYMS

Acronym	Description
AI	Artificial intelligence
Aml	Ambient intelligence
API	Application programming interface
ANN	Artificial neural network
CNN	Convolutional neural network
CRISP-DM	CRoss-Industry Standard Process for Data Mining
DL	Deep learning
LSTM	Long short-term memory
ML	Machine learning
MNIST	Modified National Institute of Standards and Technology, database for handwritten digits
MT	Mechatronics
RNN	Recurrent neural network

# 1 CONTENTS

Colophon .....	2
Availability .....	2
List of acronyms .....	3
1 Contents .....	4
2 Introduction to the manual .....	6
2.1 Goal & scope .....	6
2.2 Outline .....	7
3 Fundamentals of deep learning .....	8
3.1 What deep learning is (not) .....	8
3.2 Origins of deep learning .....	9
3.3 Applications of deep learning for computer vision .....	11
3.3.1 Classification .....	11
3.3.2 Object detection .....	12
3.3.3 Semantic segmentation .....	12
3.3.4 Instance segmentation .....	12
3.4 Data-driven machine learning .....	12
3.5 Concepts in deep learning .....	13
4 Contemporary use cases .....	15
4.1 Quality control of coffee filter holders .....	15
4.2 Wi-Fi for predictive maintenance .....	16
4.3 Point cloud segmentation in railway infrastructure .....	16
4.4 Additional examples .....	17
5 Tools & techniques .....	18
5.1 Software frameworks .....	18
5.2 Mathematical principles of deep learning .....	19
5.3 Labelling tools .....	20
6 Tutorial .....	22
6.1 Requirements .....	22
6.2 Approach .....	23
6.2.1 Business & data understanding .....	24
6.2.2 Data preparation .....	26
6.2.3 Modelling .....	27
6.2.4 Evaluation .....	28
6.2.5 Deployment .....	29
6.3 Recap and your second tutorial .....	30
7 Learning deep learning .....	32
7.1 More reading .....	32

7.2	Exemplary courses.....	32
7.3	Courses on frameworks .....	33
7.4	Integration and automation .....	34
7.5	Advanced techniques.....	34
7.6	Additional resources.....	35
7.7	Closing notes.....	36
	Acknowledgements .....	37
	References .....	37

## 2 INTRODUCTION TO THE MANUAL

This is a manual for your first steps into deep learning for computer vision. It is a shallow overview of fundamentals, best practices, and tips & tricks for the starting deep learning enthusiast. It is short and simple, and focuses on bringing across the intuition behind the concept, finishing with a hands-on example.

### 2.1 GOAL & SCOPE

Deep learning (DL) is not a new field: much has been written about it, from its origins in 1967 (Ivakhnenko & Lapa, 1967) to its revolution from 2012 onward. The goal of this manual is not to provide a new paradigm or to be used as an alternative to the wealth of sources available. Instead, this manual provides the first steppingstones to understand the basics of deep learning, with a hands-on tutorial to create a functional deep learning solution. The interested reader is referred to the works of Haenlein & Kaplan (2019) and McCorduck (2004) which provide ample insights into the development of artificial intelligence (AI) in the twentieth century. Plenty has been written on the rise of deep learning already and therefore, this manual will not dive deep into its history, yet focus instead on its fundamentals, possibilities, and hands-on challenges.<sup>1</sup>

There is a big disclaimer on this manual: **you will not become a DL expert on having completed it.**

Instead, you will become acquainted with the key concepts and methods in the field. When you have worked through this manual, you should be able to orient yourself in the field of DL, be aware of drawbacks, disadvantages, and opportunities of deep learning, and you will find the footing to continue your development with the help of this technology. At the end, you should be able to replicate the example use case and have the insights to apply this method to a use case of your choosing.

This manual is aimed at people who have some IT experience; basically, you should be able to read code. All concepts are explained to a degree that their intuition becomes clear. Code is explained and documented, and additional sources are mentioned.

If you still want to become a DL expert after having completed this manual, there are plenty of study programs at universities (of applied sciences) and online courses (by the likes of Stanford and DeepMind). For this, please refer to Section 6 of this manual.

Finally, another disclaimer is particularly important: **deep learning is not a goal, but a means to an end.** Please do not delve into deep learning because it sounds promising, innovative or in any sense fashionable, but do it for a proper cause. Section 6 also provides handholds to structure this properly. To emphasize this point: most of the time you work on deep learning will not be spend on running a specific algorithm, but on preparing the data, constructing the right network, tuning it, and evaluating the results.

In short, read this manual if you want to have a quick understanding of deep learning (for computer vision), but consult the resources mentioned for a more complete view of the field and support in developing solutions.

For more material on data mining, machine learning, and deep learning, Saxion has a three-part training available as open access material. This manual serves as an extension of those resources and is included as part of all the training material related to AI through the following link.

**Saxion AI Training**

<https://github.com/SaxionAMI/Saxion-AI-Training/>

<sup>1</sup> <https://cerncourier.com/a/the-rise-of-deep-learning/> and <https://youtu.be/uawLjkSI7Mo>

## 2.2 OUTLINE

This manual first delves into the fundamentals of deep learning ([Section 2](#)), briefly explaining how it came to be. Building on this, [Section 3](#) details exemplary use cases in which DL has proved its merit. Available tools and required skills are discussed in [Section 4](#), providing an overview of available software tooling and techniques from mathematics and computer science. [Section 5](#) is where the magic happens: it comprises a hands-on tutorial with a dataset, tooling, and a step-by-step explanation on how to make DL work. Finally, this manual closes by providing additional material to guide you to design your own DL solution with more advanced and helpful techniques ([Section 7](#)).

### 3 FUNDAMENTALS OF DEEP LEARNING

This section quickly brings the reader up to speed with the basics of deep learning. Section 3.1 first addresses some misconceptions about the field of DL, Section 3.2 details the origins of the field, and Section 3.3 treats the most common applications of DL.

#### 3.1 WHAT DEEP LEARNING IS (NOT)

A prevailing, common perspective on deep learning is that it requires large amounts of data, makes computers 'intelligent', and involves difficult algorithms that can tell you what is in a picture. This is all partially true to a certain extent, so it is wise to get possible misconceptions out of the way first.

Let's forget about pictures that come up when you use your favorite search engine and look for 'deep learning': humanoid robots, electronic brains, smart chips, etc. The concept of intelligence is too broad and philosophical to address in this manual, yet an often-quoted aphorism by Dijkstra is: 'The question of whether Machines Can Think (...) is about as relevant as the question of whether Submarines Can Swim,' (Dijkstra, 1984).

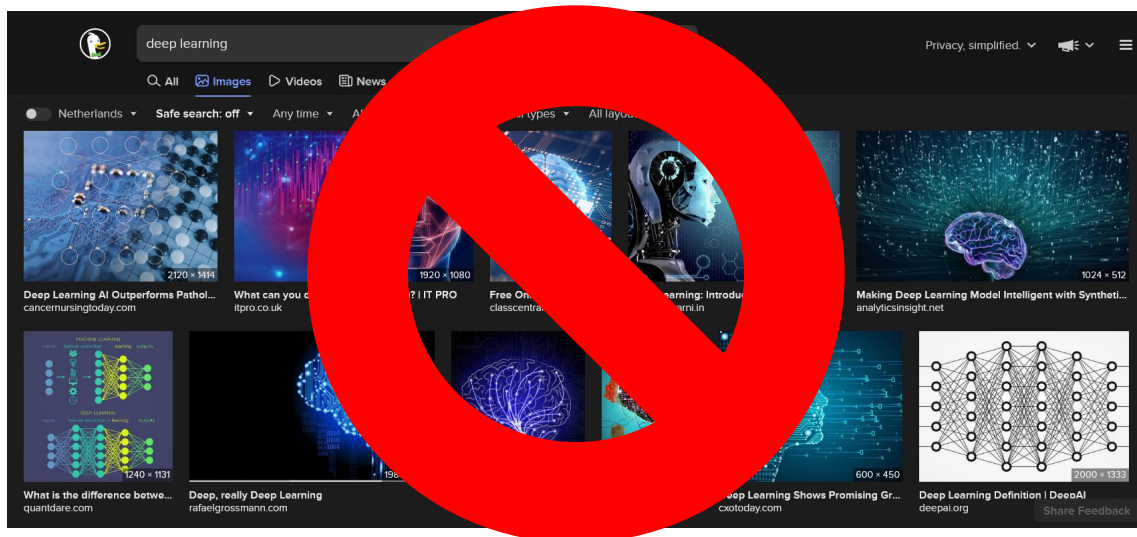


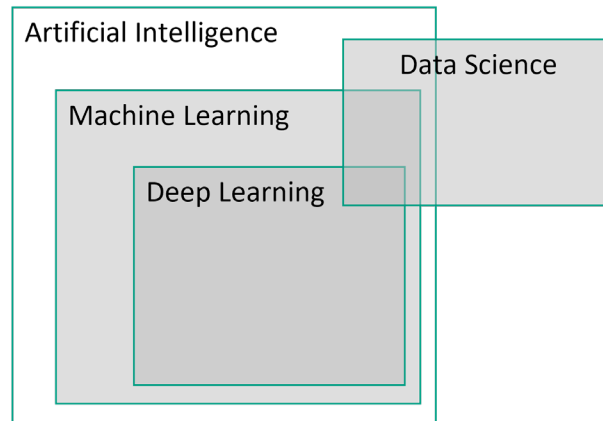
Figure 1: Common visualizations of deep learning which miss the point.

Furthermore, please also discard ideas about the necessity of **big data** and its connotations. By most proper definitions, 'big data' comprises complex data structures. Most notable for this is the 'V' model, which has three main 'V's that constitute big data (Sagiroglu & Sinanc, 2013):

- Volume: the quantity of the generated, processed, or stored data. Often upwards of terabytes to be considered big data.
- Variety: the type of data. For example, whether data is structured (spreadsheets) or unstructured (images, video, audio).
- Velocity: the speed at which data is generated or processed. Elementary to big data is that there is a stream of data which keeps delivering new data.
- Three other Vs can also be included, namely:
  - Veracity: the reliability of the data, which needs to be high to be processed properly.
  - Value: the worth of profitability of the data.
  - Variability: the consistency of data formats, structures, or storage.

The relation between DL and Artificial Intelligence (AI) is a commonly confused topic as well. A schematic diagram explaining the overlap between the various fields is shown below. In short, DL is a subfield of **Machine Learning** (ML), which is itself a subfield of AI. The field of **Data Science** lends techniques from all these fields to get insights into data.

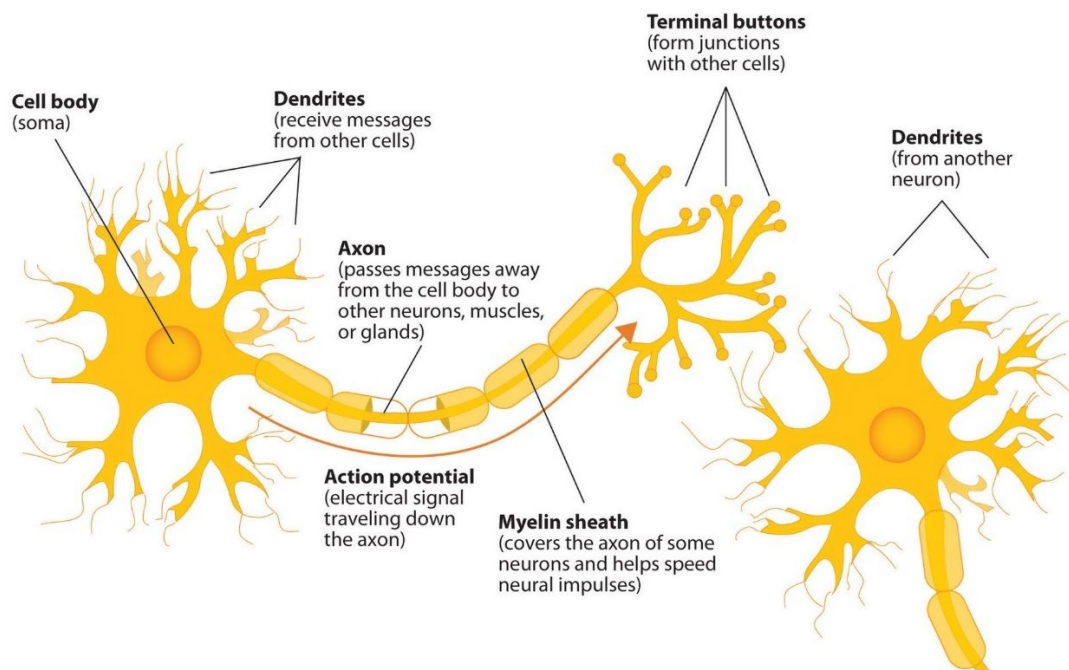




**Figure 2: The relation between deep learning and other fields.**

A basic definition of deep learning is: a machine learning method based on **artificial neural networks**, with the goal of **pattern recognition**. The next section explains the key concepts underlying this definition.

### 3.2 ORIGINS OF DEEP LEARNING



**Figure 3: Components of a neuron.<sup>2</sup>**

The origins of deep learning can be traced back to the 1940s, even before AI as a field had been ‘officially’ started (Haenlein & Kaplan, 2019). It was then that the term neural nets was coined, based on insights from biology into the functioning of the human brain: the collection of neurons, synapses, and axons in our grey matter which constitute our natural neural networks that enable us to think and function, see Figure 3. Indeed, a simple interpretation of a neuron is that it provides a certain **function**

<sup>2</sup> CC BY-SA 4.0 Jennifer Walinga, <https://opentextbc.ca/introductiontopsychology/chapter/3-1-the-neuron-is-the-building-block-of-the-nervous-system/>.

which can transform an input signal into an output signal. Given the **connections** between multiple (artificial) neurons, increasingly elaborate functions can be constructed, for example, the basic constructs of logic operators (AND, OR, XOR, NOT). Moreover, the learning capabilities of neural networks are enabled by adjustable **weights** tied to the neurons and connections. In essence, the learning process determines these weights and hence creates a **model**: a certain configuration of neurons, connections, and weights, which carries out a certain task (namely producing a desired output given an input).

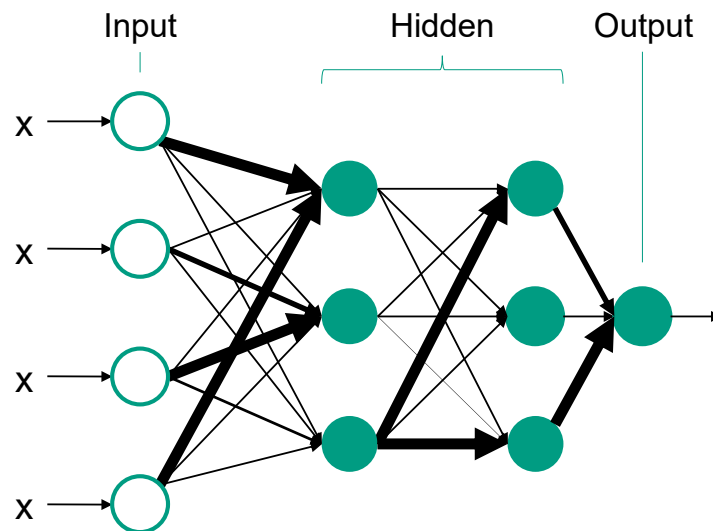


Figure 4: Schematic view of an artificial neural network.

An example, schematic artificial neural network (ANN) is shown in Figure 4. The round nodes are neurons, and the edges are the connections, with weights represented as arrows with varying thickness. As described above, all ANNs feature input and output layers. The layers in between these are called hidden layers. The configuration of this network, including the number, size, and type of the layers, is modifiable. Specifically, the term **deep learning** comes from the number of hidden layers: if there are more than two, an ANN can be called a deep learning network. To give an impression of the complexity and size of contemporary ANNs, the architecture of AlexNet is shown in Figure 5. Many different layers and connections are used to create this specific architecture for the task of image classification (Section 3.3.1).

The process of learning comprises reconfiguring the weights tied to the connections. However, first the architecture of the ANN needs to be determined: how many input and output neurons do we need, how many layers, what type of connections between the layers, etc. In Section 6, this process is explained in more detail. Important for the efficient learning of ANNs is the automated, optimized way of updating the weights. This is done by evaluating the output of the network with the goal of minimizing the **error**; in other words, the quantified distance of the given output to the desired output is to be kept as low as possible. For this, **cost functions** are defined that determine this quantified distance. Updating the weights in the ANN is performed through **backpropagation**, a technique which calculates the gradient of the cost function to determine whether a weight should be increased or decreased (applying **gradient descent** to find minima). To influence this, the **learning rate** defines this step size. Through these algorithms, the learning process eventually finds a certain configuration which minimizes the error as much as possible.

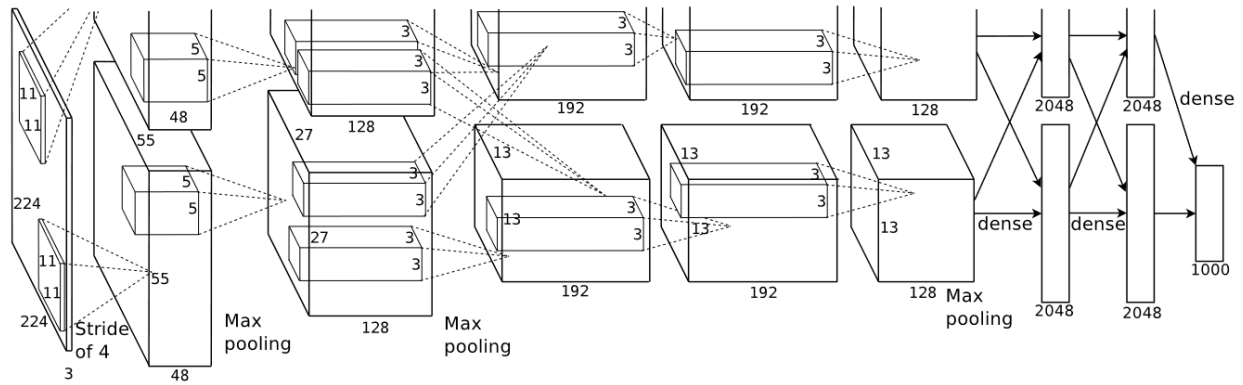


Figure 5: The architecture of AlexNet (Krizhevsky, Sutskever, & Hinton, 2012).

Over the years, various implementations, and designs of deep learning networks, including the underlying activation functions and methods for evaluating them. However, the basics have been founded in the 1940s, with the first general, working algorithm being described in 1967 (Ivakhnenko & Lapa, 1967). A seminal work of recent advancements in the field, with a taxonomy of the different approaches and types of deep learning is provided by Schmidhuber (2015). In the following section, specific applications, including related deep learning types, are discussed for exemplary purposes.

### 3.3 APPLICATIONS OF DEEP LEARNING FOR COMPUTER VISION

There are four typical tasks for deep learning in computer vision, which call for different approaches and deep learning architectures, see Figure 6. In this example, all of them are based on a picture being the input, yet with the desired outputs markedly different.

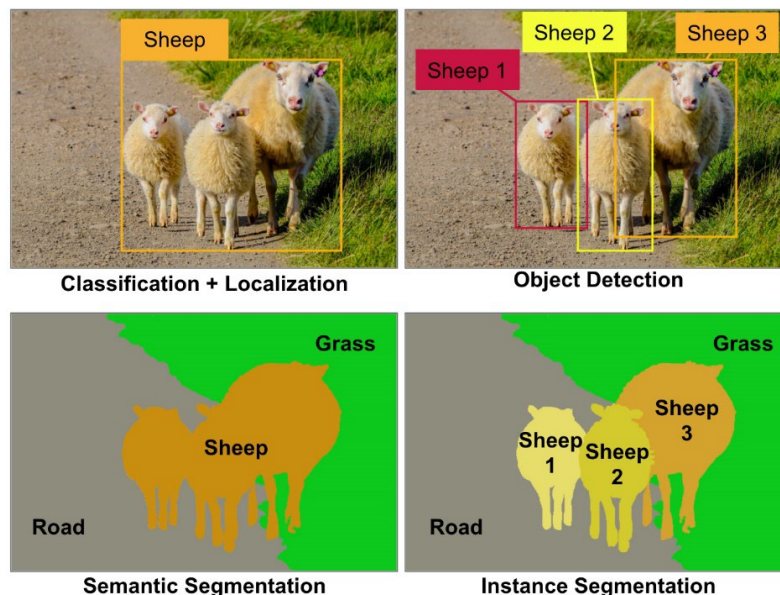


Figure 6: The four tasks for deep learning in computer vision.

#### 3.3.1 CLASSIFICATION

The desired output is a description of the object in the picture, including a so-called **bounding box**, which is the smallest possible upright rectangle which contains the classified object. One of the most well-known deep learning architectures which gave yield to the growth and start of the deep learning resurgence of the 2010s is **AlexNet**, a **Convolutional Neural Network (CNN)** architecture specifically constructed for classifying images (Krizhevsky, Sutskever, & Hinton, 2012). CNNs are proficient at

finding structure in large data (such as visual data). Each of the layers in a CNN learns a filter for a specific feature, such as textures and shapes.

---

### 3.3.2 OBJECT DETECTION

The goal is to perform classification for all the recognizable objects at any scale and location in the picture and detect them as unique objects. There is a line of two types of typical approaches to this. First, the 'sliding window' approach was investigated, which involves a CNN that employs many bounding boxes with different scales and locations to evaluate specific classes (Sermanet et al., 2013). However, this remains very computationally intensive. A second method which approached things from a different angle is **YOLO**: You Only Look Once (Redmon, Divvala, Girshick, & Farhadi, 2016). YOLO is a highly optimized algorithm which treats detection as a regression problem by classifying initially defined bounding boxes on pictures, yet which in turn connects adjacent bounding boxes to approximate the center of the detected object with a certain accuracy. In this way, it considers more contextual information about the entire picture to detect objects. Various implementations of YOLO have shattered successive records in different challenges, and it remains a go-to approach for object detection.

---

### 3.3.3 SEMANTIC SEGMENTATION

Providing even more fine-grained information, the goal is to partition an image into regions of different semantic categories; in other words, per pixel, label which type of object it belongs to. Classic computer vision methods for performing this rely on determining similarity between adjacent pixels, yet this approach does not incorporate semantic content. Building on advances in object detection, semantic segmentation is reframed by deep learning as 'pixel segmentation'. Here, the idea is to leverage the capabilities of **Fully Convolutional Networks** (FCN) to first attempt to process all pixels of the picture, but quickly down-sample the picture using smaller layers to make further calculations possible; otherwise, the networks would remain infeasibly large to process (Long, Shelhamer, Darrell, 2015). At the end, the trick is that the down-sampled classifications are up-sampled again to the full resolution, so that regions of the input picture are segmented on a pixel basis.

---

### 3.3.4 INSTANCE SEGMENTATION

Similar to the distinction between 1 and 2, perform semantic segmentation while distinguishing between different instances, i.e., unique objects. The breakthrough approach for this application was Mask R-CNN for Region-based Convolutional Neural Networks (He, Gkioxari, Dollár, & Girshick, 2017). As its name implies, this approach employs 'regions of interest' to predict segmentation masks, in parallel with classification and bounding box regression (as in YOLO). In short, it is an extension of existing methods, yet easily surpassed many approaches by combining efforts by performing the parallel mask representation that does not interfere with the other tasks.

---

## 3.4 DATA-DRIVEN MACHINE LEARNING

In machine learning, **datasets** are used to train models. A key thing is that such datasets need to fit the information that is to be classified or predicted. For example, to recognize a dog in a picture, we would need a collection of pictures with dogs. The of such a dataset varies strongly depending on the objective and the degree to which something needs to be determined. A good dataset accounts for the variety in data points that can be expected. For example, for recognition of handwritten digits, there need to be samples of all digits from 0 to 9 to train a proper model.

In **supervised learning**, training data are **labelled**: the samples have been annotated with information, for example, a label identify that there is a dog in the picture in a certain location. With the help of machine learning, a **model** can then be trained on these labelled data which iteratively learns properties that identify dogs in pictures. For **unsupervised learning**, the data are not labelled. With such data, a

model can be trained to identify commonalities in the data, for example, to cluster similar pictures based on shapes or colors.

When training a model, the data need to be split into two separate sets: a **training set** and a **test set**. The training set is shown to the model that is being trained, after which the model is validated using the test set. That is, the model has never seen the test set before, so it can hence be used to provide an unbiased evaluation of the model. Splitting the original dataset into such sets is usually done with an 80/20 or 90/10% split.

A classical mistake is to train a model data from both sets, which leads to **overfitting**: the model will be tuned to data which are also used to evaluate it. This means that it becomes overly sensitive to the specific data in the dataset and the results from the validation will not be trustworthy. For example, the evaluation might reveal that the model performs with close to 100% accuracy. However, when the model is afterwards tested on new data, it will turn out that the model is way less robust to variations in samples.

### 3.5 CONCEPTS IN DEEP LEARNING

Below, an overview of the most frequently occurring concepts in deep learning follows, as a handy resource and guide.

- **Artificial neural network**: connected collection of synthetic representations of neurons which can transmit signals.
  - **Neuron**: a mathematical function that can ingest multiple inputs and outputs a single signal.
  - **Activation**: a neuron outputs ('fires') an activation signal when a weighted sum of the incoming connections crosses a minimum threshold.
  - **Connection**: a link between two neurons, with a **weight** determining the factor to which it is considered when calculating the activation of a neuron.
  - **Propagation function**: a technique which calculates the gradient of the cost function to determine whether a weight should be increased or decreased (often, **gradient descent** is used for this).
  - Types: diverse designs for deep learning ANNs have been developed for specific purposes.<sup>3</sup>
    - **Feedforward**: the most common type of ANN, features only connections that propagate signals in one direction, without any loops (a Directed Acyclic Graph). An example is the **Convolutional Neural Network** (CNN), which are proficient at finding structure in large data (such as visual data). Each of the layers in a CNN learns a filter for a specific feature, such as textures and shapes.
    - **Recurrent**: features connections that allow loops in the network. This allows for a certain 'memory' to be incorporated, enabling the processing of sequences of inputs of varying length. This makes such networks well-suited for speech recognition and temporal signal processing. An example is the **long short-term memory** architecture, which is designed for handling time series data.
- **Hyperparameter**: a constant parameter which is set before the learning process is initiated. Examples: learning rate, number of hidden layers.
- **Layer**: a collection of neurons which outputs to a following layer, based on the configured connections and weights.
  - **Input**: the layer of neurons which ingests the original data.
  - **Hidden**: the layer(s) of neurons which process the activation signals, and between which the weights of the connections are updated through the learning process.

<sup>3</sup> See Schmidhuber (2015) and [https://en.wikipedia.org/wiki/Types\\_of\\_artificial\\_neural\\_networks](https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks) for more complete overviews.

- **Output**: the layer of neurons which outputs data in the desired format.
- **Fully connected**: every neuron in a layer connects to all the neurons in the next layer.
- **Pooling**: a group of neurons connects to a single neuron in the next layer, thereby reducing the number of neurons in that layer.
- **Learning**: the process of updating the configuration of an ANN.
  - **Learning rate**: defines the step size to update weights.
  - **Cost function**: determines the error of the output.
  - **Backpropagation**: see propagation function.
  - **Overfitting**: creating a model which is highly specific for one dataset, and thereby being less robust for new data.
  - **Paradigms**: these are basic paradigms for most forms of machine learning, and hence, deep learning.
    - **Supervised**: a function is learned to map an input to a certain output, given example training data which is labeled.
    - **Unsupervised**: a function is learned to find patterns in data, given example training data which is unlabeled.
    - **Semi-supervised**: a combination of the two previous approaches, which attempts to generalize from a small set of labeled training data.
    - **Reinforcement learning**: an approach in which an artificial actor (an **agent**) attempts to learn a task, based on certain reward which is influenced by, e.g., the number and cost of actions, time.
- **Model**: a certain configuration of neurons, connections, and weights, which conducts a certain task (namely producing a desired output given an input).
- **Test set**: the subset of the original dataset, based on which a model is evaluated. It should not include any samples from the training set.
- **Training set**: the subset of the original dataset, based on which a model is trained. It should not include any samples from the test set.



## 4 CONTEMPORARY USE CASES

This section includes descriptions of three exemplary use cases from projects run by the research groups Mechatronics and Ambient Intelligence.

### 4.1 QUALITY CONTROL OF COFFEE FILTER HOLDERS

With industrial mass production of items, quality control is one key area in which automation is necessary. In the RAAK-mkb project [Focus op Vision](#), a use case on plastic coffee filter holders was investigated to verify the possibilities for deep learning to detect deficiencies on the plastic surface. For example, Figure 7 shows a coffee filter holder which has a haze covering an area. This product should be rejected during production.



Figure 7: A coffee filter holder (left) and samples images of deficient and satisfactory areas (right, up and down, resp.).

Using an image acquisition system specifically designed for this use case, a series of images was captured and labelled to create a dataset. This was then used to train a model based on an approach taken by Wang et al. (2018) which leverages a CNN architecture. This model is capable of detecting, with high accuracy, the type and location of deficiencies on specific areas of coffee filter holders.

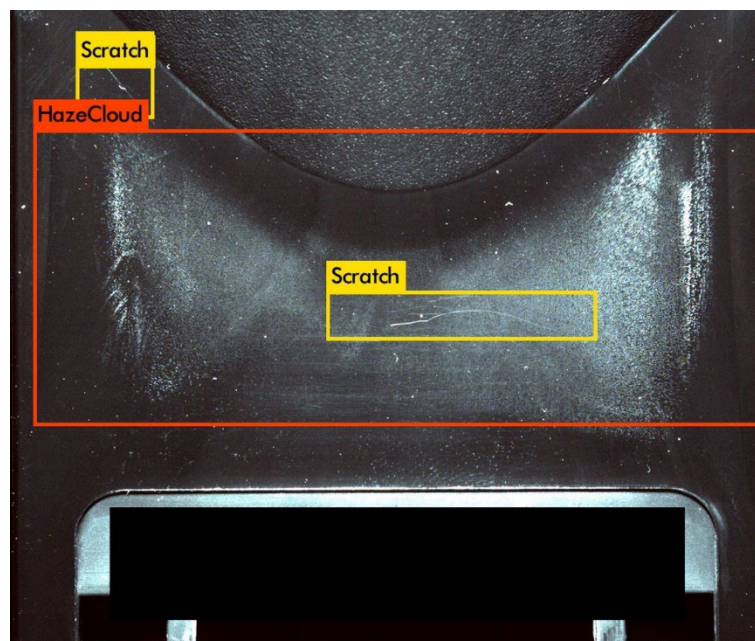
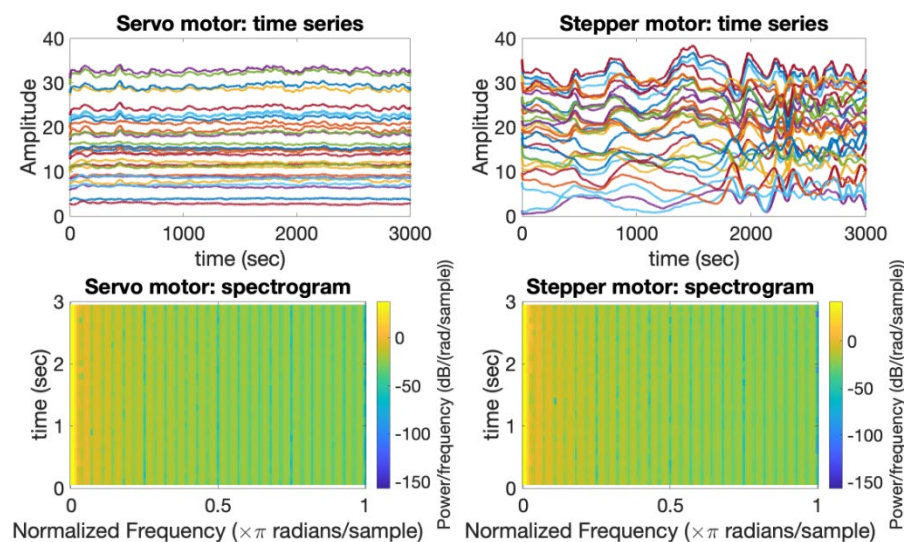


Figure 8: Detected defects on a coffee filter holder.

## 4.2 WI-FI FOR PREDICTIVE MAINTENANCE

To support the fourth industrial revolution; commonly framed as **Industry 4.0**, various ways investigated to acquire data from industrial systems in unobtrusive ways. That is, the functioning of such systems needs to be done so that the process at hand is not disturbed and would subsequently lead to suboptimal behavior. For this reason, researchers from the **Pervasive Systems** group at the University of Twente collaborated with the research group **Ambient Intelligence** of Saxion to investigate the use of Wi-Fi for remote monitoring (Bagave, Linssen, Teeuw, Brinke, & Meratnia, 2019).

In a laboratory setting, two motors with different behaviors (a stepper and servo motor) were activated, and a Wi-Fi transmitter and receiver were placed at opposing ends of the respective motors. The experiments relied on the Channel State Information (CSI) of the Wi-Fi, which comprises different channels operating at slightly different frequencies. The concept here is that this information can be leveraged to detect changes in the environment, as dynamic behavior interferes with the transmitted signals. Figure 9 shows this influence, accompanied by spectrograms of these data, which are visual representations of the spectrum of frequencies in the data.



**Figure 9: Influence of the two motor behaviors on CSI.**

Spectrograms are a well-known way of remediating information, with the advantage that, by transforming spectrum frequency information to visual information, typical deep learning approaches can be employed. Indeed, in this case a relatively simple Convolutional Neural Network (with six layers) was trained to detect different behavior types of the motors, for example, whether the movement of the motor was hindered or unhindered by some added weight meant to symbolize wear or malfunction. The best-performing classifier reached over 80 and 90 % accuracy for the behaviors of the separate motors. Nonetheless, this direction of research remains challenging due to many uncertain and interfering factors, such as environmental integrity and dynamic behavior of unknown systems.

## 4.3 POINT CLOUD SEGMENTATION IN RAILWAY INFRASTRUCTURE

There are over 3,000 km of railroad tracks in the Netherlands. The digitization of this infrastructure is aimed at the improvement of maintenance and construction activities. Currently, inspections are done manually, with a domain expert classifying objects along the railway, such as the catenary arches, poles, and insulators.

**Strukton Rail** is an engineering company which supplies services for railway infrastructure. One of the technologies they employ is **point clouds**, which are sets of spatial data points captured by 3D scanning techniques such as lidar. These point clouds contain many millions of points of data, resulting in 3D



representations of the railway environment. The research group **Ambient Intelligence** has been collaborating with Strukton Rail to investigate deep learning methods to do automatic object recognition and segmentation.<sup>4</sup> Several models have been evaluated on a dataset of a 1 km long railway track, and experiments are being carried out with additional datasets.

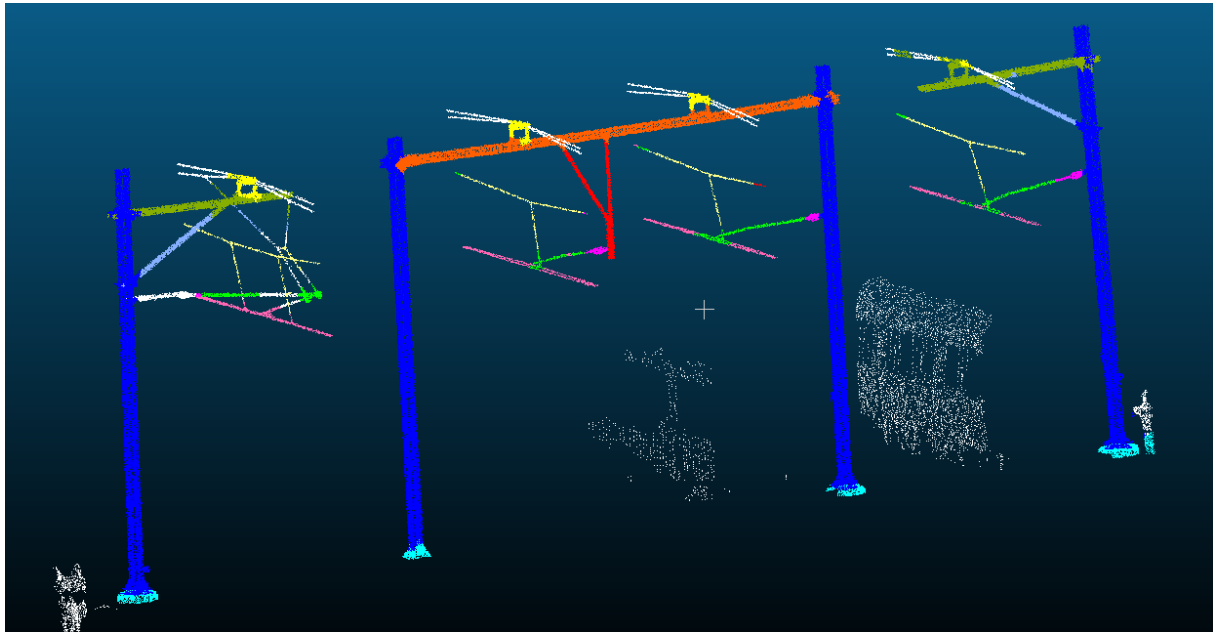


Figure 10: Semantic segmentation of a catenary arch in a point cloud.

Figure 10 shows the result of the segmentation of a so-called catenary arch which has been sampled from the complete dataset. Distinct colors indicate the various types of objects present in this subset, among which the poles (blue), crossbeams (orange and green), and insulators (yellow). This project involves research into proper pre-processing of the available data, as the methods used for segmentation are notably memory-consuming given the size of the dataset. Modifications comprised efforts such as slicing the data into smaller subsets and downsampling the data while retaining sufficient resolution (density) of the point cloud for analyses. Various implementations of deep learning methods for point cloud segmentation (Landrieu & Simonovsky, 2018; Qi, Yi, Su, & Guibas, 2017; Zhao, Jiang, Jia, Torr, & Koltun, 2021) were successfully implemented and evaluated on the dataset; a measure of accuracy used was the mean Intersection over Union, which reached up to 70 % in initial experiments. Especially the difference in size between present objects proves to be a challenging factor. Continued research in this collaboration currently focuses on further integrating the generated models and developed approach into a pipeline connecting this output to databases of CAD drawings to assist in digital design.

#### 4.4 ADDITIONAL EXAMPLES

In the past decade, many implementations of machine learning algorithms and applications to domain-specific use cases have been created. Two of the more famous ones are **AlexNet**, which was the key ANN architecture that the possibilities of boosted deep learning, and **YOLO**, which improved the speed at which detections on images could occur. Example implementations of these architectures are available through sites such as [paperswithcode.com](https://paperswithcode.com), which hosts thousands of software implementations of scientific literature.<sup>5</sup>

<sup>4</sup> See <https://www.saxion.edu/business-and-research/research/smart-industry/ambient-intelligence/digitalisatie-bovenleidingen-en-draagconstructies>.

<sup>5</sup> See <https://paperswithcode.com/method/alexnet> and <https://paperswithcode.com/method/yolov4> for implementations of AlexNet and YOLOv4, respectively.

## 5 TOOLS & TECHNIQUES

This section details available tools and techniques before the tutorial is addressed in Section 6.

### 5.1 SOFTWARE FRAMEWORKS

There are ample software frameworks available for deep learning. Notable examples are Google's [TensorFlow](#) and Facebook's [PyTorch](#). These are but two examples of generic frameworks, with many more software libraries available for specific applications. One overview of the landscape of software related to AI & data in a broad sense is given by the [LF AI & Data Foundation](#).<sup>6</sup> Specifically for deep learning, a variety of frameworks, platforms, libraries, and tools is shown as a subset of this in Figure 11.

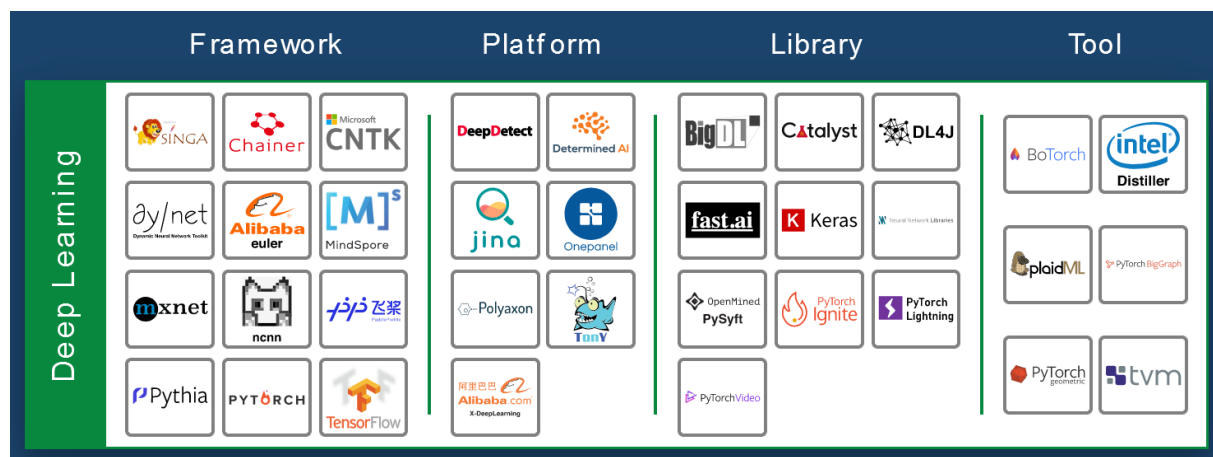


Figure 11: An overview of software related to deep learning.

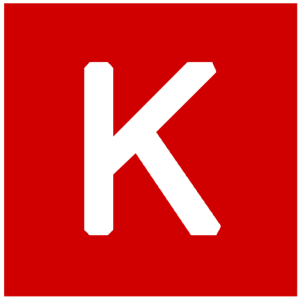


When choosing a specific framework, platform, library, or tool, its specific requirements and limitations should be considered. These typically are:

- The applied licensing, and hence whether it is usable in research or commercial contexts.
- The supported programming language; Python is well-supported, almost de facto standard.
- The technical challenge certain software offers: some require more technical knowledge than others. This mostly comes with a trade-off in configurability, yet this does not necessarily have to impede results.
- The support and documentation that is available. This depends to a large degree on the active maintenance on the software.

Currently, TensorFlow and PyTorch are the two main frameworks for deep learning that undergo active development and based on which most research and implementations are based. PyTorch is based on the Torch library and is mainly used for applications in computer vision and natural language processing, being primarily developed by Facebook's AI Research lab. TensorFlow precedes its existence, being developed by Google Brain, and with a more generic approach to problems related to deep learning. A third contender, [Keras](#), is actually a layer on top of TensorFlow, acting as an interface to with by aiming to be more user-friendly and modular. A simplified comparison between the three is shown in Table 1.

<sup>6</sup> <https://landscape.lfai.foundation>

Table 1: Comparison between Keras, PyTorch and TensorFlow frameworks.

Property	Keras	PyTorch	TensorFlow
Logo			
API level	High	Low	High and Low
Architecture	Simple	Complex	Not easy to use
Datasets	Small	Large	Large
Debugging	Often not needed	Good	Difficult
Trained models available	Yes	Yes	Yes
Popularity (rank)	1	3	2
Speed	-	+	+
Written in	Python	Lua	C++, CUDA, Python
Interfaces	Python	Python, C++	Python, C++, Javascript, Java

The tutorial adopts Keras for its simplicity to show how the basics of deep learning networks can be implemented. These translate well to other frameworks, and resources for similar tutorials for PyTorch are given in the corresponding section. For descriptions of these frameworks and other software, consult the LF AI & Data Foundation landscape.

## 5.2 MATHEMATICAL PRINCIPLES OF DEEP LEARNING

As explained in Section 3.1, deep learning is not an isolated field. It builds on many concepts from mathematics (linear algebra and matrices), statistics, computer science, and biology, to name the most prominent ones. Explaining these basics is out of the scope of these manual, yet several noteworthy resources on the underlying mathematics are as follows.

Table 2: Resources for mathematics of deep learning.

Title	Type	Description	Link
Mathematics for Machine Learning	Book, with code	A book on the mathematical principles underlying many ML techniques. From the basics of linear algebra and matrices to linear regression and principal component analysis.	<a href="https://mml-book.github.io">https://mml-book.github.io</a>
The Mathematical Engineering of Deep Learning	Online course, with code	A collection of algorithms, models, and methods that allow the statistician, mathematician, or machine learning professional to use deep learning methods effectively.	<a href="https://deeplearningmath.org">https://deeplearningmath.org</a>
Python for Probability, Statistics, and Machine Learning	Book	An explanation of many statistical concepts related to machine learning, using the Python library numpy.	<a href="https://link.springer.com/book/10.1007%2F978-3-030-18545-9">https://link.springer.com/book/10.1007%2F978-3-030-18545-9</a>
The Matrix Calculus You Need For Deep Learning	Article	A concise summary of all the matrix calculus required to understand deep learning.	<a href="https://arxiv.org/abs/1802.01528">https://arxiv.org/abs/1802.01528</a>

### 5.3 LABELLING TOOLS

While doing deep learning and all its related (and required) activities, specifically labelling tools will prove to be useful for ensuring that your dataset is complete and reliable. These are all geared towards annotation of images (and videos), which can then be used as input for DL networks.

Table 3: Tools for data labelling.

Name	Description	Supported formats	Link
CVAT: Computer Vision Annotation Tool	A free, online, interactive video and image annotation tool for computer vision.	COCO, CVAT, VOC, VGG, YOLO, many others	<a href="https://github.com/openvinotoolkit/cvat">https://github.com/openvinotoolkit/cvat</a> and <a href="https://cvat.org">https://cvat.org</a>
Onepanel	An end-to-end enterprise computer vision platform with built-in modules for model building, automated labeling, data processing, model training, hyperparameter tuning, workflow orchestration and serverless inference. Employs CVAT.	See CVAT.	<a href="https://docs.onepanel.ai">https://docs.onepanel.ai</a>

Name	Description	Supported formats	Link
Make Sense	A free, online tool for image annotation.	COCO, CSV, VOC, VGG, YOLO	<a href="https://www.makesense.ai">https://www.makesense.ai</a>
Universal Data Tool	An open-source tool and library for creating and labeling datasets of images, audio, text, documents, and video in an open data format.	CSV, JSON, conversion to COCO and VOC through convertor tool	<a href="https://universaldatatool.com">https://universaldatatool.com</a>
VIA: VGG Image Annotator	A standalone image annotator application packaged as a single HTML file that runs on most modern web browsers.	VIA3, COCO, JSON, converters available	<a href="https://www.robots.ox.ac.uk/~vgg/software/via/">https://www.robots.ox.ac.uk/~vgg/software/via/</a>
Img Lab	A free, online tool for image annotation.	COCO, VOC	<a href="https://imglab.in">https://imglab.in</a>
Labelme	Image Polygonal Annotation with Python (polygon, rectangle, circle, line, point and image-level flag annotation). Based on MIT's original LabelMe tool.	COCO, VOC	<a href="https://github.com/wkentaro/labelme">https://github.com/wkentaro/labelme</a>

## 6 TUTORIAL

There are many tutorials available on deep learning for computer vision. The example described in this section simply offers a handy abbreviated version to function as a gentle introduction to the field. This tutorial is based on an existing Keras tutorial as written by Google.<sup>7</sup>

### 6.1 REQUIREMENTS

Most deep learning frameworks, libraries and platforms are either written in the programming language Python or provide versions in this programming language. A basic of Python is therefore very much advised. Example resources to learn Python are:

- <https://python-10-minutes-a-day.rocks>, which is a low-level entry point to quickly understand the most common concepts and get started.
- <https://www.kaggle.com/learn/python>, which provides online notebooks.
- <https://www.learnpython.org>, which has interactive online exercises.

Next to this, a software environment is necessary to work in. This allows programmers to work on projects in such a way that changes, dependencies, and various installations of libraries do not conflict with each other. An entry level and easy to set up way of working with a simple environment is Google Colab. This is a Python-based, web-based environment in which code can be programmed and run. It is available through the link below and only requires a Google account. It is also what we recommend for carrying out this tutorial. A 5-minute tutorial on Google Colab is also available to get acquainted with its general functions.<sup>8</sup>

**Google Colab**

<https://colab.research.google.com>

For more advanced work where data and code are hosted on your local computer, we advise Anaconda, a distribution of Python focused on managing environments and packages related to data science. It includes a graphical user interface to do so and comes with a suite of other software that is able to be installed when desired. Anaconda is available through the link below and it also has a rich database of tutorials.<sup>9</sup>

**Anaconda Data  
Science Platform**

<https://www.anaconda.com>

For this tutorial, we will use keras, a high-level API for TensorFlow. Section 5.1 compares keras to TensorFlow and PyTorch. Here, keras is chosen for its ease of use and readability.

As a dataset for which we will develop a model in this tutorial, we use the Fashion-MNIST dataset (Xiao et al., 2017). This dataset is available through the link below.

**Fashion-MNIST dataset**

<https://github.com/zalandoresearch/fashion-mnist>

This is an alternative to the 'classic' MNIST dataset, which is a database of 70,000 handwritten digits (from 0 to 9) to test computer vision and machine learning approaches to classify these. The Fashion-MNIST dataset was developed by Zalando Research to provide a more challenging, refreshing dataset for machine learning and that is why it is also used in this tutorial as it is considered more future-proof. It also contains 70,000 examples divided over 60,000 training items and 10,000 test items and is 30 megabytes in size. The official repository for this dataset describes it as follows:

<sup>7</sup> Following <https://www.tensorflow.org/tutorials/keras/classification>, under MIT and Apache 2.0 Licenses.

<sup>8</sup> [https://colab.research.google.com/notebooks/basic\\_features\\_overview.ipynb](https://colab.research.google.com/notebooks/basic_features_overview.ipynb)

<sup>9</sup> <https://docs.anaconda.com/anaconda/user-guide/>



*Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct **drop-in replacement** for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits. Here's an example of how the data looks (each class takes three-rows):*

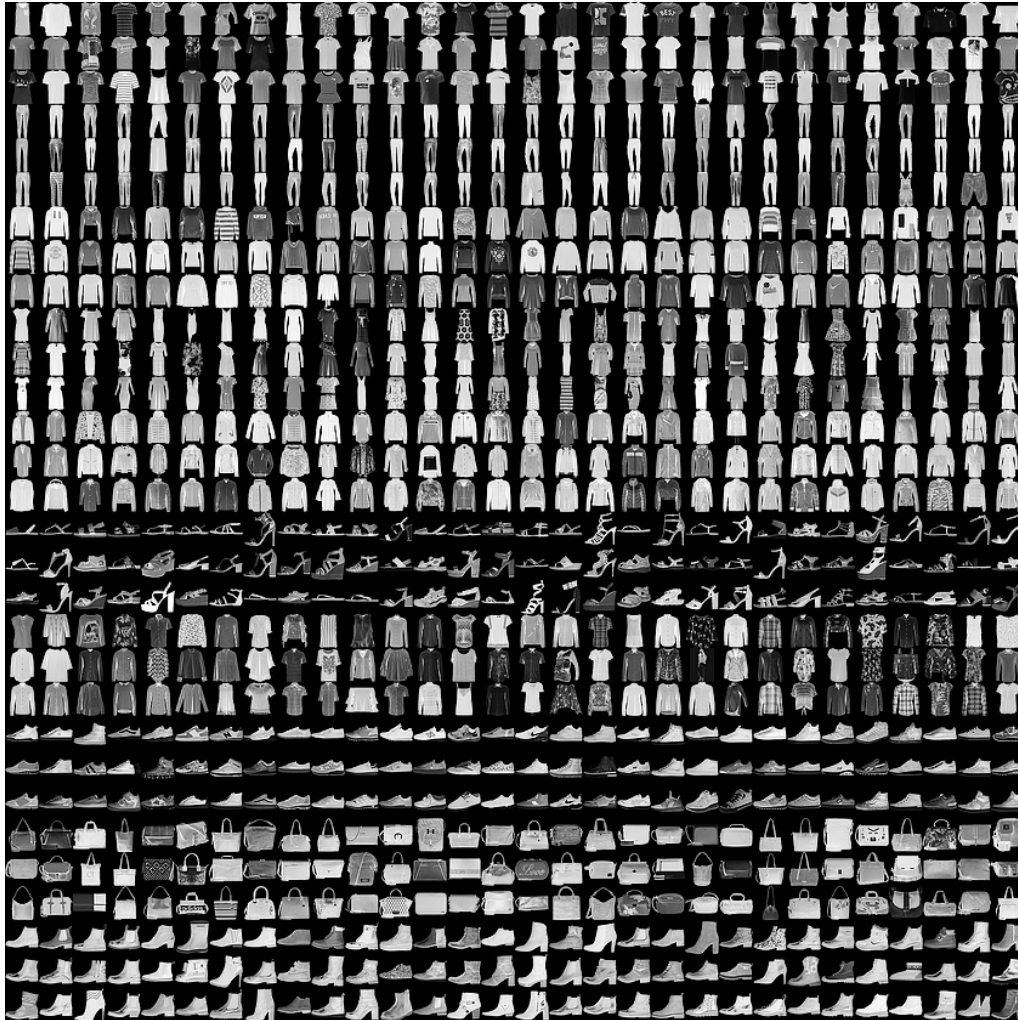


Figure 12: An example selection of the 10 different classes in the Fashion-MNIST dataset.

To recap the requirements:

1. Have some elementary understanding of Python;
2. We employ keras, a high-level abstraction for TensorFlow;
3. Use Google Colab (<https://colab.research.google.com>) with the supplied notebook (saxion\_tutorial\_dl\_classification.ipynb);
4. Use the Fashion-MNIST dataset (<https://github.com/zalandoresearch/fashion-mnist>); we will download this during the tutorial.

## 6.2 APPROACH

To structure this tutorial and provide a generic, best practice approach, it follows the six phases of the **CRoss-Industry Standard Process for Data Mining (CRISP-DM)**. This is a standard process model to derive information from data and integrate it within existing processes. A complete description of this model is out of the scope of this tutorial, but in short, it can be described as follows. Through six phases, CRISP-DM structures data mining projects by explicitly addressing, among other things, requirements,

constraints, and activities in high detail. Figure 13 provides a schematic overview of these phases, highlighting the overall iterative nature and possibility of iterations between the phases. Each phase is structured according to tasks, activities, and outcomes to cover all facets, map responsibilities, and produce realistic plans. A full description is available through the official documentation<sup>10</sup> and a training session on this model has been organized by Saxion as well.<sup>11</sup>

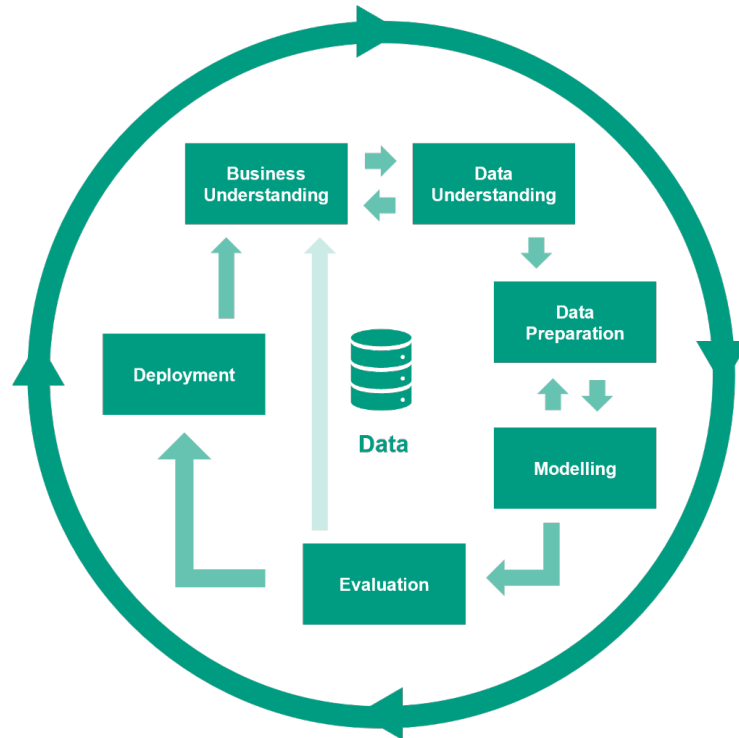


Figure 13: The six phases of CRISP-DM.

### 6.2.1 BUSINESS & DATA UNDERSTANDING

Any type of data-related project should start with a proper understanding of why a certain effort is undertaken, according to CRISP-DM. For this tutorial, the primary business (research) question (Phase 1) is whether we can optimize the process of cataloging fashion items, like shoes, dresses, coats, etc. (a perspective we borrow from Zalando, who have created the Fashion-MNIST dataset). Based on this question and the available data, a collection of pictures of such items, we can construct the data mining goal: how can we classify an image of a fashion item?

For Phase 2, to understand the data, we need to be able to open and inspect it. Here, we start with the tutorial proper. First, open the following notebook in Google Colab in a web browser of your choice:

**Fashion-MNIST  
classification  
notebook**

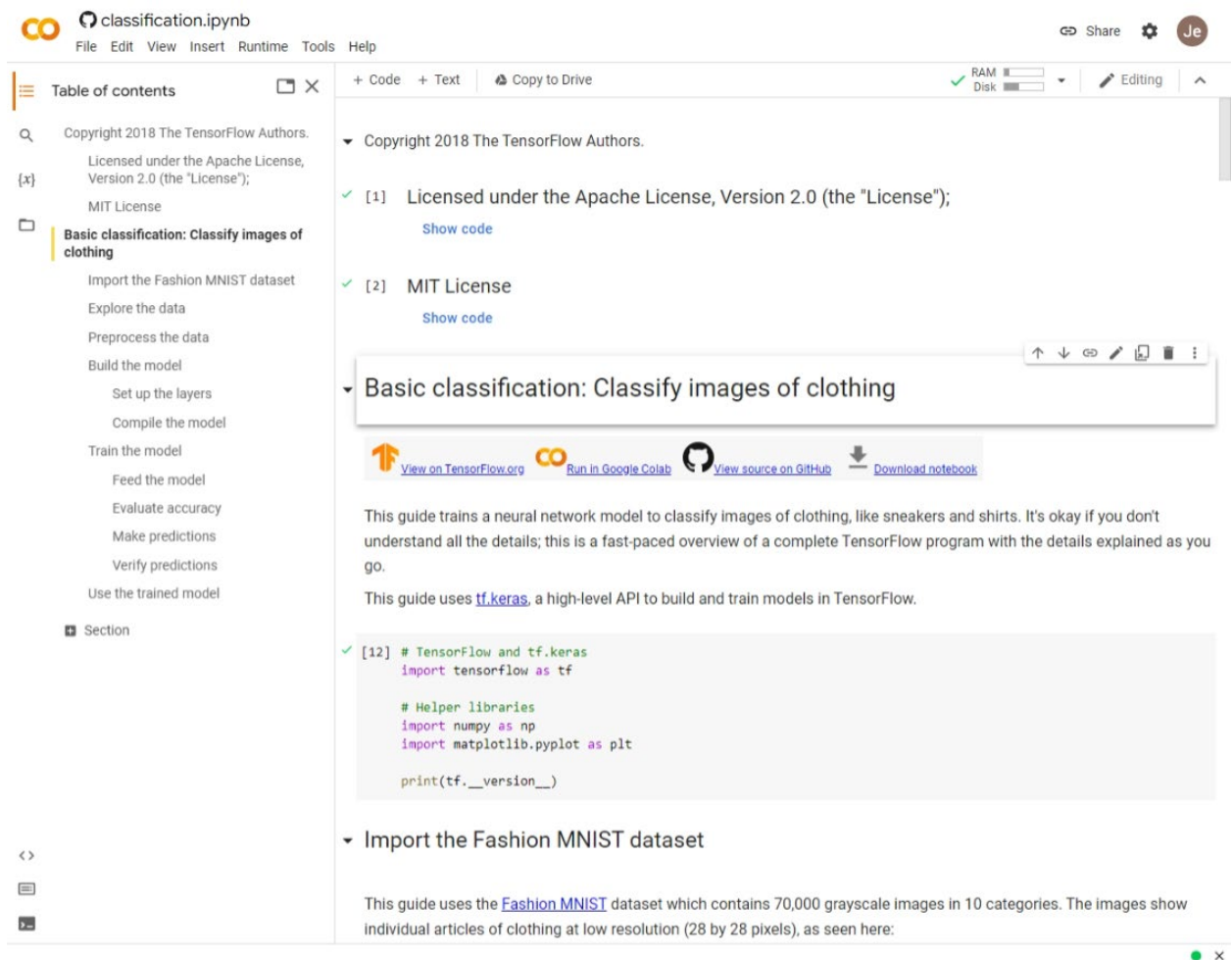
[https://github.com/SaxionAMI/Saxion-AI-Training/tree/main/Resources/saxion\\_tutorial\\_dl\\_classification.ipynb](https://github.com/SaxionAMI/Saxion-AI-Training/tree/main/Resources/saxion_tutorial_dl_classification.ipynb)

Your browser should show a screen that looks similar to the screenshot below (Figure 14). On the left of your screen, an outline of the notebook contents is shown; on the right, the so-called cells which either contain text or code are shown. The cells with a dark background contain code which can be run by hovering over them until a play button appears. When it is clicked, the contents of that cell are run by a server from Google's cloud platform.

<sup>10</sup> <https://the-modeling-agency.com/crisp-dm.pdf>

<sup>11</sup> <https://github.com/SaxionAMI/Saxion-AI-Training/>





The screenshot shows a Jupyter Notebook interface. On the left, the 'Table of contents' sidebar lists sections: Copyright 2018 The TensorFlow Authors., Licensed under the Apache License, Version 2.0 (the "License");, MIT License, Basic classification: Classify images of clothing, Import the Fashion MNIST dataset, Explore the data, Preprocess the data, Build the model, Set up the layers, Compile the model, Train the model, Feed the model, Evaluate accuracy, Make predictions, Verify predictions, Use the trained model, and Section. The main area shows the first code cell, which is titled 'Basic classification: Classify images of clothing'. The code cell contains the following code:

```
[12] # TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

Below the code cell, there is a section titled 'Import the Fashion MNIST dataset' with a description: 'This guide uses the [Fashion MNIST](#) dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels), as seen here:'.

Figure 14: Screenshot of the Python notebook used for classification of the Fashion-MNIST dataset.

The first code cell contains the following code which is intended to import the correct libraries and print the version of TensorFlow on which keras runs. Clicking the play button in the top left of the cell should run the code and print a version number (such as '2.8.0').

```
# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

Loading the Fashion-MNIST dataset is up next, so that we can actually get an understanding of its contents. As is explained in Section 3.4, the dataset is split up into a training and a test set. This means that the test data are never used to train the model, only to verify how good it performs.

Running the following code cell loads the dataset as an object 'fashion\_mnist'. Furthermore, it carries out splitting the dataset into train and test sets.

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Now, to include the actual names of the different classes of labels for the items included in the dataset, these need to be paired with the classes 0 to 9, as the following cell does.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

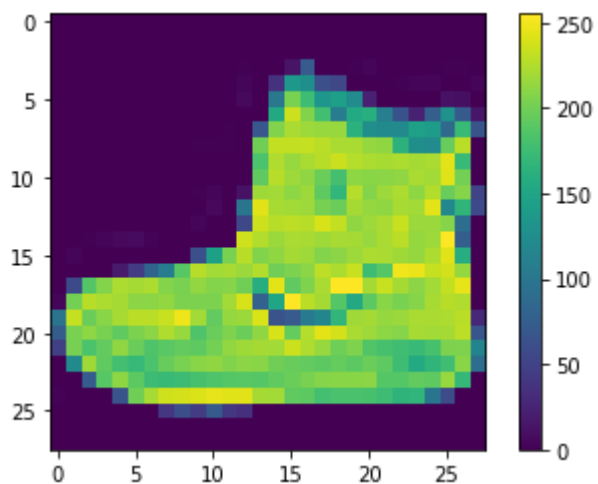
In the section 'Explore the data', the notebook contains several cells that show the number of images in the training and test sets, and the fact that each of the images is 28 by 28 pixels in size.

### 6.2.2 DATA PREPARATION

Phase 3 of CRISP-DM is reserved for preparation of the data. Usually, this is carried out to overcome issues in the data, for example, missing values, low quality of certain data points or redundancies in the data.

In the case of the Fashion-MNIST dataset, the images are made up of grayscale pixels with values running from 0 to 255. The code cell below performs operations to show the picture on the right, which indeed shows the range of pixel values for the first item in the training images.

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



The only necessary step for this dataset to be prepared for modelling is scaling the pixel values to a range of 0 to 1 so that a neural network can ingest them. This is done simply by carrying out the following code.

```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

Having done so, the following code cell shows the first 25 items in grayscale values to verify that the operation was successful.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i],
               cmap=plt.cm.binary)
    plt.xlabel(class_names
               [train_labels[i]])
plt.show()
```



### 6.2.3 MODELLING

Now that the data has been prepared, modelling can commence (Phase 4 of CRISP-DM). This involves two steps: building a neural network by configuring its layers and compile functions and training the model with the training data.

#### 6.2.3.1 BUILDING A NEURAL NETWORK

As discussed in Section 3.2, deep learning relies on artificial neural networks that can be trained by iteratively letting them approach a target output. Firstly, they require to be constructed by defining their layers and how they are chained together. In its simplest form, this is done by defining a sequential model so that each of the layers follows on the previous one. For this tutorial, we define three layers in Keras, as shown in the code below. The first and last layers are the input and output layers, respectively. The input layer takes the 28 by 28 pixel images and flattens them into one long one-dimensional array so that it can be accepted by the network. The second layer is a dense (fully connected) layer with 128 neurons, and with the ReLu activation function.<sup>12</sup> The output layer provides an array of length 10 with scores relating to the predicted classes, one for each of the 10 classes in the dataset.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

After defining the model, the model needs to be informed how it should be compiled. This is done with three settings:

1. A **loss function**, which measures the accuracy of the model during training, and which needs to be minimized to result in as little loss as possible.
2. An **optimizer**, which updates the model based on the loss function and the available data.

<sup>12</sup> This defines how signals yield activations of neurons. For more information on activation functions, consult Schmidhuber (2015).

3. **Metrics**, which are used to monitor training; in this case, a simple form of accuracy is used, but more advanced metrics can be chosen instead.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(
                  from_logits=True),
              metrics=['accuracy'])
```

---

### 6.2.3.2 TRAINING A NEURAL NETWORK

The actual work of instructing a neural network to be trained is betrayingly simple and can be done with one line of code.

```
model.fit(train_images, train_labels, epochs=10)
```

Here, the model is trained using the training dataset for 10 epochs, during each of which all the training items are shown once. When this code is run, the loss and accuracy for each epoch are shown. When training is complete with this configuration, an accuracy of about 91% should be reached.

---

### 6.2.4 EVALUATION

Deciding how well a model performs relies on evaluation with data which has not been used to train the model: the test set.

```
test_loss, test_acc = model.evaluate(test_images,
                                     test_labels,
                                     verbose=2)

print('\nTest accuracy:', test_acc)
```

Often, the result is that the model performs slightly worse (for example, 88% accuracy) on the test set than on the training set: this is called **overfitting** and is a result of the model being too fine-tuned for the training data. In that way, it can be said to be less robust to new data.

The following step is to verify label predictions by gaining insight into model predictions.<sup>13</sup> We define a probability model which is then used to create a complete list of 10,000 predictions on the test dataset. Then, we store the prediction labels in a new array.

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])

predictions = probability_model.predict(test_images)
pred_labels = np.array(predictions).argmax(1)
```

To get a complete overview of how our trained model performs, a **confusion matrix** is helpful to show the actual labels of the training set versus the predicted labels. This allows use to inspect whether the model performs below expectations on certain classes, for instance.

---

<sup>13</sup> We use a softmax layer to show probabilities instead of the model's linear outputs.

```
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_labels, pred_labels)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=class_names)

disp.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=45)
plt.show()
```

The code above yields a confusion matrix with a heatmap: the more intense the color, the higher the count of items in that cell. From this, we can deduce that, for example, shirts are predicted to be pullovers in 100 test cases out of 1,000 and are only classified correctly 73.9% of the time (739/1,000).<sup>14</sup>

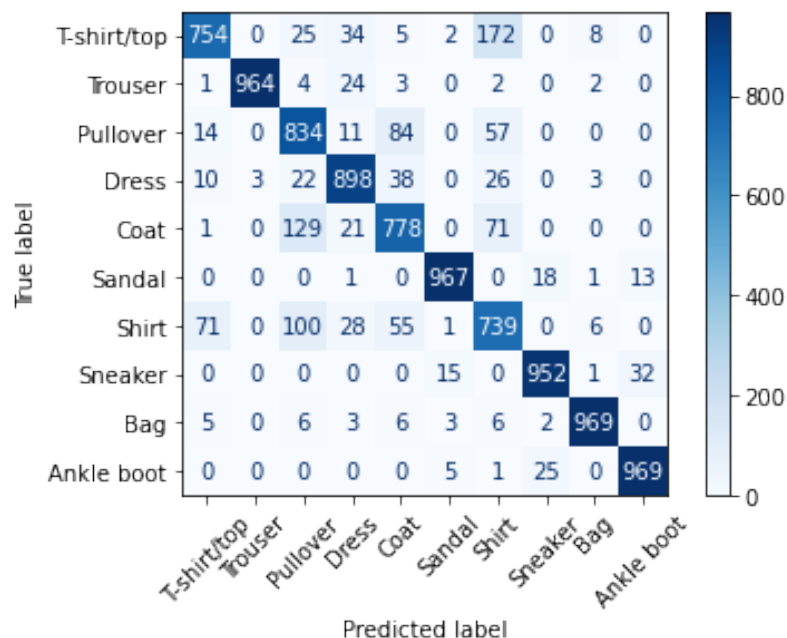


Figure 15: A confusion matrix for the trained model.

To delve deeper into this, other metrics need to be considered, namely sensitivity, specificity, precision, and recall. These concepts stem from statistics and describe the ratios between correctly and incorrectly predicted labels. Further discussion of these is out of the scope of this manual, but other resources and courses are available on this.<sup>15</sup>

The remaining code cells in the following section of the notebook ('Additional visualizations') contain more examples to visualize the model results.

### 6.2.5 DEPLOYMENT

When CRISP-DM treats the final phase of Deployment, it focuses on using a trained and validated model to be used in production. In this case, say we have a picture of an unknown item and would like

<sup>14</sup> These values will vary with different instantiations of the trained model.

<sup>15</sup> For example, [https://en.wikipedia.org/wiki/Sensitivity\\_and\\_specificity](https://en.wikipedia.org/wiki/Sensitivity_and_specificity) and [https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall), and <https://online.stat.psu.edu/stat507/lesson/10/10.3>.

to know its class. For ease of use, we take an image from the test dataset. Keras models are designed to operate on a batch of examples at once, so the single picture is added to a list.

```
img = test_images[1]

print(img.shape)

img = (np.expand_dims(img, 0))

print(img.shape)
```

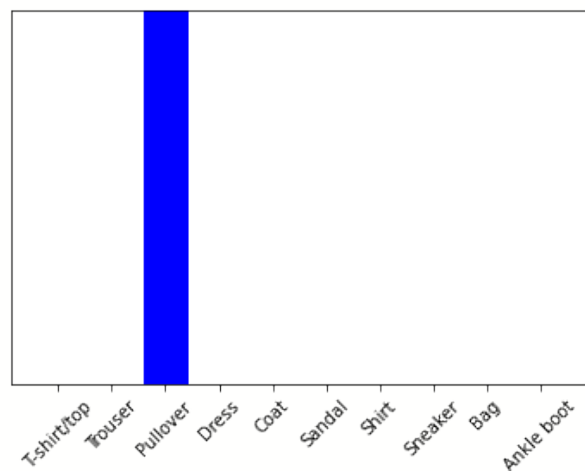
Now, the label for the picture can be predicted, which outputs a list of 10 values which in turn can be visualized as a bar chart with the different classes.

```
predictions_single =

probability_model.predict(img)

print(predictions_single)
plot_value_array(1,

predictions_single[0],
                test_labels)
_ = plt.xticks(range(10),
                class_names,
                rotation=45)
plt.show()
```



To get a direct result, namely the label of the class for the picture that was fed to the model, the final line in the notebook can be used. It takes the maximum value from the list of predicted classes and uses this to retrieve the correct label 'Pullover'.

```
class_names[predictions_single[0].argmax(0)]
```

### 6.3 RECAP AND YOUR SECOND TUTORIAL

The above tutorial is intended to provide a structured overview of the six phases that have to be passed to set up, train and evaluate a deep learning neural network for computer vision. This is a basic, straightforward example using a use case on classification of images. The take home message is to structure every approach like this, in a staged manner, for example, following CRISP-DM. Additionally, the evaluation phase cannot be overestimated: understanding metrics like accuracy and the actual output of trained models should never be skipped.

Several steps in the tutorial are simplified for demonstration purposes. For example, little data pre-processing was necessary, and there were no missing values in the data. In real-life use cases, these are rule rather than exception, so much effort should be expected to properly handle these issues.



As a follow-up step, a similar approach to a different type of challenge can be a next step for the reader. For example, object detection is an extension of classification, as discussed in Section 3.3. One of the most well-known deep neural networks is YOLO: *You Only Look Once*. It is able to efficiently localize objects in images with high accuracy and has a still-growing number of successors.

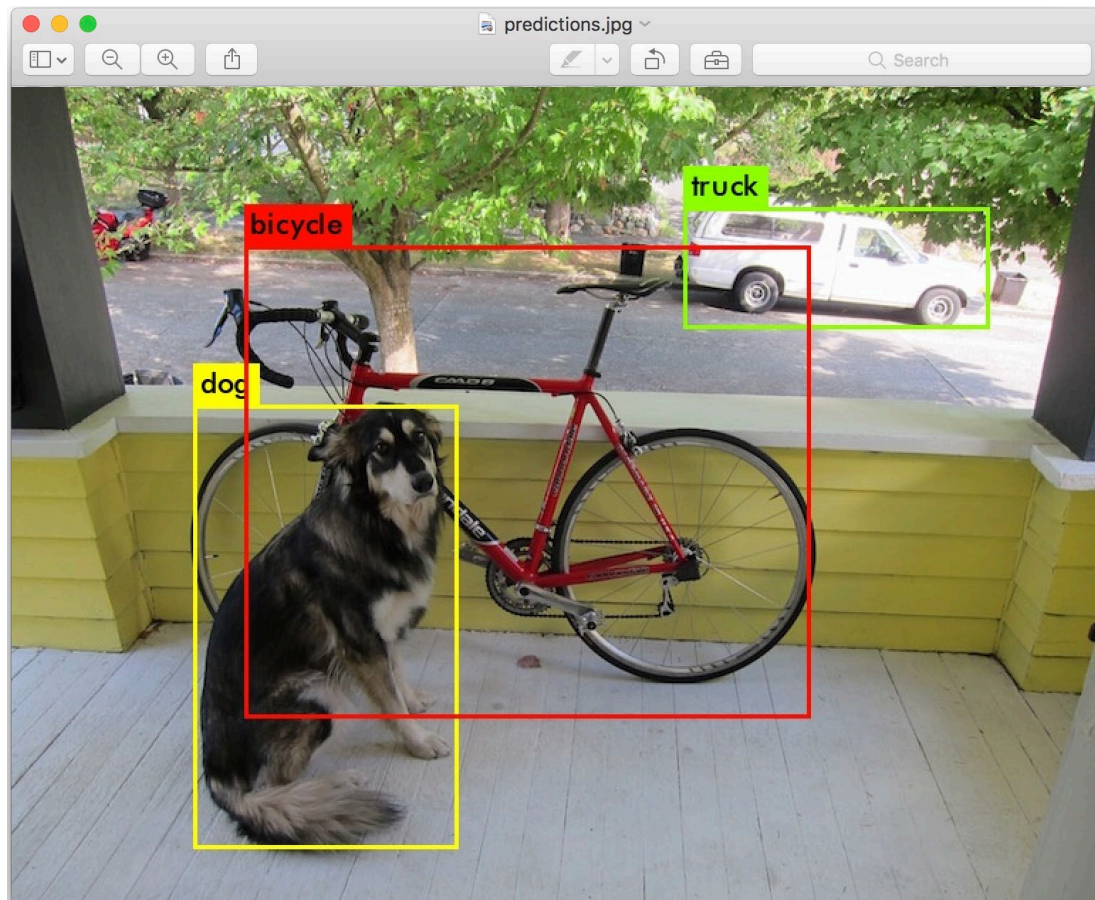


Figure 16: Object detection using YOLO (Redmon et al., 2016).

To get acquainted with YOLO, there are many tutorials available. One of these details YOLOv4, the fourth version of YOLO with improved speed and accuracy and details the approach of an open-source implementation of it.<sup>16</sup> An archived version of this tutorial is available through Archive.org via the following URL.

**Object detection with  
YOLO: hands-on  
tutorial**

<https://web.archive.org/web/20210904114914/https://neptune.ai/blog/object-detection-with-yolo-hands-on-tutorial>

<sup>16</sup> Available through <https://github.com/taipingeric/yolo-v4-tf.keras>.

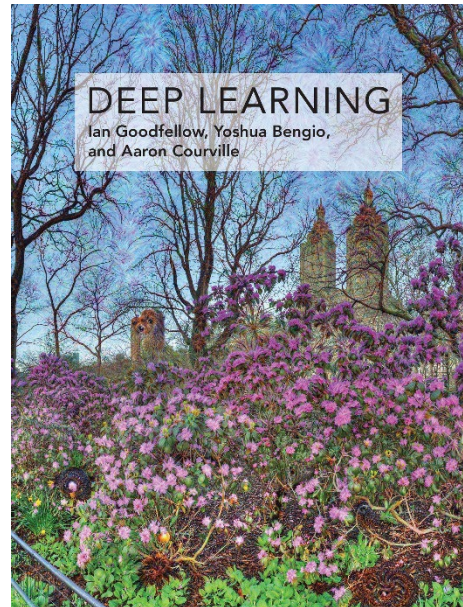
## 7 LEARNING DEEP LEARNING

If this manual has inspired you or you are otherwise interested in the topic of deep learning: congratulations, there are many resources, courses, and challenges to be found. This section serves as a springboard for you to delve deeper into all the material that is available. In the GitHub repository of the Saxion AI Training, a more expansive list of resources is available.<sup>17</sup>

### 7.1 MORE READING

There are several contemporary books on deep learning that are freely available and feature in-depth explanations of basic and complex topics. A seminal work which this manual already referred to is Schmidhuber's (2015) article '[Deep learning in neural networks: An overview](#)'.<sup>18</sup> It is a fairly complete overview of all the main topics relevant to deep learning, yet very in-depth and hard to digest for uninformed readers. As such, it functions better as a resource than as an entry point.

A more low-level entry point is the online book '[Neural networks and deep learning](#)' by Nielsen (2015).<sup>19</sup> This book features code snippets in a tutorial-like fashion. A more recent book, simply titled '[Deep learning](#)' by Goodfellow et al. (2016) takes a similar approach but is more focused on the underlying mathematics.<sup>20</sup> A final recommendation goes to '[Dive into deep learning](#)', an interactive online book by Zhang et al. (2021) with code implementations in various frameworks and recent updates to its contents and additional material.<sup>21</sup>



### 7.2 EXEMPLARY COURSES

Deep learning presupposes knowledge of various fields, among which statistics, machine learning and software engineering. Below, noteworthy courses on these topics are listed.

Name	Description	Organizer	Topic	URL
Introduction to Statistics	Fundamentals of statistics.	Stanford University	Statistics	<a href="https://www.coursera.org/learn/stanford-statistics">https://www.coursera.org/learn/stanford-statistics</a>
Improving Your Statistical Questions	Foundations of statistics.	Eindhoven University of Technology	Statistics	<a href="https://www.coursera.org/learn/improving-statistical-questions">https://www.coursera.org/learn/improving-statistical-questions</a>
Data science courses	A variety of short courses with code examples.	Kaggle	Machine learning, deep learning, and others	<a href="https://www.kaggle.com/learn">https://www.kaggle.com/learn</a>
Machine Learning	The classic machine learning course by Andrew Ng.	Stanford University	Machine learning	<a href="https://www.coursera.org/learn/machine-learning">https://www.coursera.org/learn/machine-learning</a>

<sup>17</sup> See <https://github.com/SaxionAMI/Saxion-AI-Training/tree/main/Additional%20materials>.

<sup>18</sup> Available through <https://doi.org/10.1016/j.neunet.2014.09.003>.

<sup>19</sup> Available through <http://neuralnetworksanddeeplearning.com>.

<sup>20</sup> Available through <https://www.deeplearningbook.org>.

<sup>21</sup> Available through <http://d2l.ai>.



Name	Description	Organizer	Topic	URL
Machine Learning Crash Course	A fast-paced introduction to machine learning.	Google	Machine learning	<a href="https://developers.google.com/machine-learning/crash-course/">https://developers.google.com/machine-learning/crash-course/</a>
Deep Learning Specialization	Follow-up to Andrew Ng's Machine Learning course.	Deeplearning.ai	Deep learning	<a href="https://www.coursera.org/specializations/deep-learning">https://www.coursera.org/specializations/deep-learning</a>
Deep Learning for Computer Vision	Official university course with online material.	Stanford University	Deep learning	<a href="http://cs231n.stanford.edu">http://cs231n.stanford.edu</a>
Practical Deep Learning for Coders	To-the-point course, focused on coding.	Fast.ai	Deep learning	<a href="https://course.fast.ai">https://course.fast.ai</a>

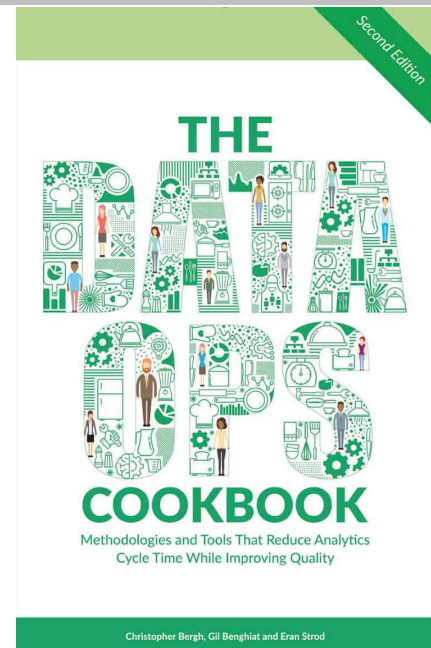
### 7.3 COURSES ON FRAMEWORKS

The three most well-known frameworks PyTorch, TensorFlow and the extension of the latter, Keras, have well-developed courses readily available. In addition to these and the courses listed in the previous section, there are more courses available that focus on specific frameworks.

Name	Description	Organizer	Topic	URL
PyTorch Tutorials	Official resource for PyTorch.	Facebook	PyTorch	<a href="https://pytorch.org/tutorials/">https://pytorch.org/tutorials/</a>
Introduction to TensorFlow	Official resource for TensorFlow.	Google	TensorFlow	<a href="https://www.tensorflow.org/learn">https://www.tensorflow.org/learn</a>
Learning resources for Keras	Official resource for Keras.	Google	Keras	<a href="https://keras.io/getting_started/learning_resources/">https://keras.io/getting_started/learning_resources/</a>
Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning	The name of the course says it all.	Deeplearning.ai	TensorFlow	<a href="https://www.coursera.org/learn/introduction-tensorflow">https://www.coursera.org/learn/introduction-tensorflow</a>
Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python	Small tutorial similar to the tutorial discussed in this manual	EliteDataScience	Keras	<a href="https://elitedatascience.com/keras-tutorial-deep-learning-in-python">https://elitedatascience.com/keras-tutorial-deep-learning-in-python</a>

## 7.4 INTEGRATION AND AUTOMATION

Training models using deep learning is only a part of fully embedding these results into practice. For this, a variety of steps needs to be taken to properly integrate and automate deep learning models. CRISP-DM, the CRoss-Industry Standard Process for Data Mining, is a helpful handhold for this process. In addition to models such as this, the term **DataOps** has gained traction in recent years. This term captures the idea of Data + Operations, building on concepts from DevOps (Development and Operations, in software engineering), lean manufacturing and an agile methodology. The '**DataOps Cookbook**' provides a high-level, partially managerial, overview of these practices and how to align businesses with data-driven objectives.<sup>22</sup> From a technical point of view, this process requires prolonged alignment with IT, most prominently by automating the process of data analytics. This requires proper data pipelines to be constructed so that newly acquired data can be used to continuously develop improved models with automated testing and validation processes.



## 7.5 ADVANCED TECHNIQUES

With the attention deep learning has received in the past decade, many researchers and other professionals have worked on various techniques. A few of the most notable and directly usable of these should be considered to save yourself lots of effort, as they tackle common problems in deep learning.

First, as deep learning revolves around the construction of increasingly complex neural networks, it has been investigated how existing networks can be leveraged for other purposes. Simply put, the technique of **transfer learning** relates to using parts, often the first layers of an ANN, in a new network and only training the last couple of layers.<sup>23</sup> If a certain model has proved its worth and its first few layers are robust to detect typical features, such as lines and shapes in the case of computer vision, it can be built upon for a new purpose. TensorFlow and PyTorch have dedicated tutorials for this.<sup>24</sup>

Another common issue in data-related research is the lack of data or lacking variety of data. For example, what would we do if we only had 10 examples of shirts for our tutorial whereas we have 1,000 examples of the other categories? To tackle this challenge, **data augmentation** comes into play. This technique can be used to synthetically generate additional data by modifying the existing data with small tweaks. Typical examples are mirroring, rotating and skewing pictures so that the overall data is more varied, and the trained model hence becomes more robust. Whereas this process can be carried out manually<sup>25</sup>, there are various frameworks and libraries available to automate this task. Well-known examples are:

- **Albumentations** and **imgaug**, which are used specifically for image augmentation;<sup>26</sup>
- **AugLy**, which also features methods for other media (audio, text, and video).<sup>27</sup>

<sup>22</sup> Available through <https://datakitchen.io/the-dataops-cookbook/>.

<sup>23</sup> See <https://www.coursera.org/lecture/machine-learning-projects/transfer-learning-WNPap> for a more complete explanation.

<sup>24</sup> See [https://www.tensorflow.org/tutorials/images/transfer\\_learning](https://www.tensorflow.org/tutorials/images/transfer_learning) and [https://pytorch.org/tutorials/beginner/transfer\\_learning\\_tutorial.html](https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html), respectively.

<sup>25</sup> [https://www.tensorflow.org/tutorials/images/data\\_augmentation](https://www.tensorflow.org/tutorials/images/data_augmentation) provides a tutorial for TensorFlow.

<sup>26</sup> Available through <https://albumentations.ai> and <https://github.com/aleju/imgaug>, respectively.

<sup>27</sup> Available through <https://github.com/facebookresearch/AugLy>.

To make machine learning techniques more accessible, the topic of **automated machine learning (AutoML)** has gained attention in recent years. Here, the goal is to abstract away as much as possible of the internals of machine learning pipelines, for example, by testing variations of applicable algorithms and selecting (hyper)parameters. The work of Hutter et al. (2019) is a valuable resource on the fundamentals and implementations of various AutoML algorithms.<sup>28</sup> Several libraries and platforms that feature such options are:

- **AutoKeras**, which is based on Keras and provides several methods to automatically instantiate a machine learning model.<sup>29</sup>
- **FLAML**, a library developed by Microsoft, which has functionality similar to AutoKeras.<sup>30</sup>
- **Google AutoML**, which is an online service hosted by Google which simplifies many of the steps related to machine and deep learning.<sup>31</sup>

A final topic that merits attention focuses on the explainability or interpretability of AI-related algorithms, often captured under the term **explainable AI**. To overcome the disadvantages and risks of the ‘black box’ of deep learning, namely a model that is hopefully accurate, but hard to understand, methods are being investigated to explain the behavior of such models. One way of doing so is by visualizing the trained neural networks. For example, the tool **CNN Explainer** shows a few sample images which are classified by a Convolutional Neural Network, including explanations of the separate layers together with their activations, based on the image.<sup>32</sup> An alternative approach is the explanation of the various features used by a model through **Shapley values**. The Python library SHAP features an example implementation which visualizes the relative importance of these features, see Figure 17.

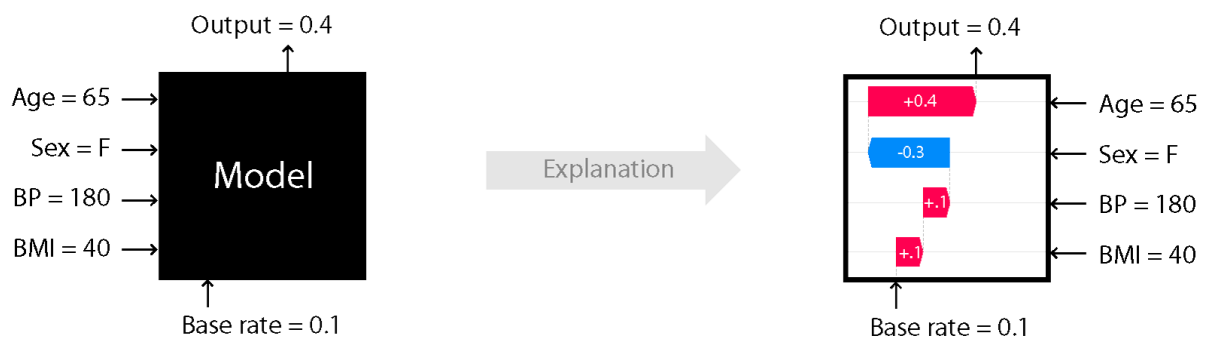


Figure 17: Schematic visualization of Shapley values.<sup>33</sup>

Additional methods for interpreting machine learning approaches are listed in the open access book **Interpretable Machine Learning**.<sup>34</sup>

## 7.6 ADDITIONAL RESOURCES

With the abundance of information available on deep learning and computer vision on the internet, several venues have organized collections, lists and taxonomies of resources on these topics. The following have been selected because of their size, recency, and qualitative descriptions.

<sup>28</sup> Available as open access through <https://www.automl.org/book/>.

<sup>29</sup> Available through <https://autokeras.com>.

<sup>30</sup> Available through <https://microsoft.github.io/FLAML/>.

<sup>31</sup> Accessible through <https://cloud.google.com/automl/>.

<sup>32</sup> Available at <https://poloclub.github.io/cnn-explainer/>.

<sup>33</sup> Available through <https://github.com/slundberg/shap>.

<sup>34</sup> Available at <https://christophm.github.io/interpretable-ml-book/>.

Name	Description	Topic	URL
Awesome Computer Vision	A curated list of computer vision resources, sorted by topic.	Computer vision	<a href="https://github.com/jbhuang0604/awesome-computer-vision">https://github.com/jbhuang0604/awesome-computer-vision</a>
Computer Vision Recipes	Examples and best practice guidelines for building computer vision systems, curated by Microsoft and based on PyTorch with Azure implementations.	Computer vision	<a href="https://github.com/microsoft/computervision-recipes">https://github.com/microsoft/computervision-recipes</a>
Machine Learning and Deep Learning in Computer Vision	A curated list of computer vision resources, sorted by topic, with recommendations for beginners.	Computer vision, machine learning, deep learning	<a href="https://github.com/mheriyanto/machine-learning-in-computer-vision">https://github.com/mheriyanto/machine-learning-in-computer-vision</a>
Awesome Deep Learning	A curated list of deep learning resources, sorted by topic.	Deep learning	<a href="https://github.com/ChristosChristofidis/awesome-deep-learning">https://github.com/ChristosChristofidis/awesome-deep-learning</a>
Data Augmentation Review	A list of useful data augmentation resources, sorted per data type.	Data augmentation	<a href="https://github.com/AgaMiko/data-augmentation-review">https://github.com/AgaMiko/data-augmentation-review</a>

## 7.7 CLOSING NOTES

Never forget that advanced techniques like deep learning or AI in general are means to an end. They are tools to help you achieve or build something. Keep in your mind that you should decide whether deep learning is the way to go for your specific use case. Perhaps ordinary computer vision would suffice? Or other forms of machine learning? Or even simple statistics?

One thing is certain: with the rise of deep learning, all its variations, possibilities, and challenges, you can always keep on learning. *Good luck and have fun with your next steps.*

## ACKNOWLEDGEMENTS

This manual has been written and published through the RAAK-mkb project [Focus op Vision](https://www.sia-projecten.nl/project/focus-op-vision) (<https://www.sia-projecten.nl/project/focus-op-vision>).

## REFERENCES

- Bagave, P., Linssen, J., Teeuw, W., Brinke, J. K., & Meratnia, N. (2019). Channel state information (CSI) analysis for predictive maintenance using convolutional neural network (CNN). In *Proceedings of the 2nd Workshop on Data Acquisition To Analysis* (pp. 51-56).
- Dijkstra, E. W. (1984). The threats to computing science. [Speech transcript]. <https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD898.html>.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
- Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, 61(4), 5-14.
- He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision* (pp. 2961-2969).
- Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges* (p. 219). Springer Nature.
- Ivakhnenko, A. G., & Lapa, V. G. (1967). *Cybernetics and forecasting techniques* (Vol. 8). American Elsevier Publishing Company.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 1097-1105.
- Landrieu, L., & Simonovsky, M. (2018). Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4558-4567).
- Linssen, J. M. (In press). The Why of AI: Managing expectations and limitations of AI-based technology in industry. In *2021 International Conference on Emerging Trends in Business and Management*.
- Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3431-3440).
- McCorduck, P. (2004). *Machines who think*. CRC Press.
- Nielsen, M. (2015). *Neural networks and deep learning*. Determination Press.
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).
- Sagiroglu, S., & Sinanc, D. (2013, May). Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)* (pp. 42-47). IEEE.
- Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61, 85-117.

- Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.
- Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*.
- Wang, T., Chen, Y., Qiao, M., & Snoussi, H. (2018). A fast and robust convolutional neural network-based defect detection model in product quality control. *The International Journal of Advanced Manufacturing Technology*, 94(9), 3465-3471.
- Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.
- Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.
- Zhao, H., Jiang, L., Jia, J., Torr, P. H., & Koltun, V. (2021). Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 16259-16268).