Michael Schoosleitner, BSc

Augmented Reality Construction System

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Dipl.-Math. Dr.techn. Torsten Ullrich

Institute of Computer Graphics and Knowledge Visualisation

Graz, November 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

Augmented reality extends the real-world environment with virtual information. In industrial applications, technicians are able to assemble complex real-world object models with the aid of augmented reality assisting systems where the assembly is extended with virtual information of instruction steps. This thesis takes up this topic by picking one of the existing methods, namely RotationNet, a multi-view convolutional neural network, in order to evaluate its practicability in real-world usage. To classify instruction steps, RotationNet is modified to fit to the conditions of the chosen 3D model and the used 3D model for evaluation is of high complexity so as to evaluate the abilities and limits of RotationNet.

The practical approach starts by classifying variations of 2D images of a partly-assembled model with RotationNet. The results of the classification are then evaluated and compared with the performance of the human cognitive system. For this, a survey has been conducted in which humans had to classify the instruction steps of 2D images in the same manner as RotationNet does. The survey evaluation results show that in a clean-room setting, RotationNet and humans are comparably alike and neither is significantly better. In a real-world scenario RotationNet does not reach the performance of a clean-room setting. The findings within this thesis show that the system needs further improvement, e.g, support of a higher image resolution which is closer to state-of-the-art cameras so that RotationNet can be used in real-world scenarios.

Parts of this thesis have been submitted to the *15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* taking place on 27 - 29 February 2020.

Kurzfassung

Augmented Reality erweitert die reale Umgebung mit virtuellen Informationen. In Industrieanwendungen unterstützt Augmented Reality Menschen beim Aufbau von komplexen realen Objekten mit virtuell angezeigten Informationen zu den einzelnen Aufbauschritten. Für diesen möglichen Einsatz wird in dieser Masterarbeit das Multi-View Convolutional Neural Network RotationNet für den realen Einsatz evaluiert. RotationNet wird an die Anforderungen des verwendeten komplexen 3D Models angepasst, um die Möglichkeiten und Herausforderungen von RotationNet festzustellen.

In der praktischen Umsetzung klassifiziert RotationNet verschiedene 2D Bilder eines teilweise fertig aufgebauten Modells. Die Klassifizierungsergebnisse werden evaluiert und mit denen des kognitiven visuellen Systems des Menschen verglichen. Dazu wurde eine Studie mit Probanden durchgeführt, die sich derselben Klassifizierungsaufgabe wie RotationNet stellten. Die Ergebnisse zeigen, dass in einer synthetischen Umgebung RotationNet und Probanden vergleichbar sind, sich jedoch nicht signifikant unterscheiden. In einer realen Umgebung erreicht RotationNet die Ergebnisse von einer synthetischen Umgebung nicht. Abschließend wird auf Optimierungen von RotationNet eingegangen, beispielsweise auf die mögliche Unterstützung höherer Auflösung der 2D Bilder, die modernen Anforderungen entspricht, damit diese Methode in Zukunft in einer realen Umgebung eingesetzt werden kann.

Contents

1. Introduction

Augmented reality for assistance systems is a promising research field in the future to assist technicians in manufacturing and repairing processes. In the future they will be able to use a computer-aided model organized into instruction steps to build up a model from the first step to the completely assembled machine, prototype or product. Technicians are assisted by a virtually augmented construction building system to display instruction steps and additional information of the actual status of the real world model. The field of application for such a system is very broad, from industrial production over customer services to the game industry. For example, in industrial production a new product is modeled by the constructor on a computer and then the model can be published to the assembly/production. Technicians start to assemble the real object with assisted virtual instruction steps of the product. The process is ideally dividable into different assembly steps, which can be performed by different technicians. For small production series or prototyping of the usage of such a system is able to increase productivity. Furthermore, existing model drawings can be transformed to the system to support technicians to repair products which are difficult to assemble due to their advanced and outdated age. In this case, specific knowledge of the product is needed. It occurs in many industries that outdated plans exist offline which are not covering all additional improvements or versions. This version differences are immediately displayable to the user with augmented reality systems. Due to the manifold possibilities of applications of augmented reality in the production process, more attention is drawn to the research of such systems. Researchers are trying to develop an ideal, general method which is able to answer all the demands with regard to all fields of application.

First this thesis gives an overview of the existing research and its development of the field. It will then pick out the most promising method from the ModelNet benchmark in the context of computer generated model classification in real images, which is RotationNet. For this thesis, a LEGO Technic^{TM} model of an “Airport Rescue Vehicle” (no. 42068) is used to

perform experiments to demonstrate the possibility of the application of the RotationNet method in real world usage. RotationNet will be explained in detail before reaching its practical implementation. The experiments are focused on image classification to predict the right instruction step in a virtual (only rendered models) and real environment (captured images and videos by a camera). The next step is to evaluate the results of the experiments. This is then compared with the results of a survey with humans. The comparison gives a more detailed view of the complexity for an image classification for both, computers and humans. The conclusion illustrates possible improvements and a future outlook.

2. Related Work and the Current State-of-the-Art

This Chapter gives an overview of the history and the existing methods in the research field of computer vision emphasizing the identification of categories of objects in an image or video which is called object recognition. This is of importance when searching for information within an image and classifying images according to these objects. Generally, images can be classified through their content, with or without the help of meta-data. The research with regard to image classification spans over global feature descriptors which represent the images as a whole and local descriptors which take the neighborhood of features into consideration to deep learning networks which use a learning approach. The development of research goes hand in hand with the increase of computational power of computer systems.

In the next sections, general terms concerning computational processing of images – from image representation to image classification – will be explained. The focus lies in the classification of instruction steps of a computer-aided design (CAD) model in the context of deep learning networks. A database of a CAD model with instruction steps is created for this thesis which will be explained and which refers to the image generation part. A benchmark for image classification methods exists and the data in this benchmark is freely accessible. A table shows the success rate of different approaches which is the basis for the decision-making which method for this thesis will be used. The chosen method is the most successful method by September 2019 in the ModelNet benchmark (Wu et al., 2015) and is called RotationNet (Kanezaki et al., 2018).

2.1. Image Semantics

The underlying fact of why image processing is needed is that computers are not able to classify image representations the same way as the human eye does. Humans perceive images with the eye and transfer the perceptual information to the brain which then processes this information. This is an extremely complex task:

> *Visual scenes tend to be very complex: a multitude of overlapping surfaces varying in shape, color, texture, and depth relative to the observer. [...] The cortical visual system processes information for objects first by coding visual features, then by linking features into units, and last by interpretation of units as objects that may be recognizable or otherwise relevant to the observer.* (Johnson, 2013)

A computational system does not have the same ability as the human eye-brain connection has. Therefore, computers have to be trained in order to get similar results in the identification of objects. Every task of the human visual system is modeled with the aim to bring the similar ability to the computational system. The processing works like a pipeline, similar to the human eye as stated in the quote above. It is separated into tasks, from low-level to high-level representation. For computational systems, low-level parts, e.g., coding of visual features, are easier to tackle than high-level parts, like the semantic connection of real objects with their representation.

An example of how this pipeline could look like is shown in Figure 1 with two images of different scenes with different objects of the same object class. These objects have a specific semantic meaning which can be further identified. For example, in the foreground of both images two different cats are shown, one sitting in a box, the other walking on a street. The semantics in this example can be a cat, a box and a street or a wooden shelf. The human eye is trained to recognize these objects in the image and so the computational system has to be trained in order to recognize the objects.

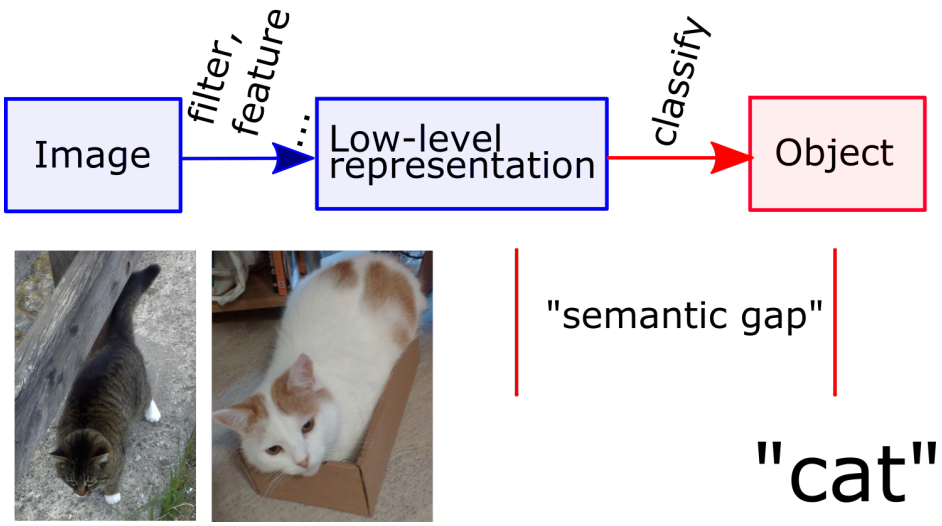


Figure 1.: Overall pipeline of the object recognition task. First an image is processed and low-level representations are used to close the semantic gap. The returned object category for the two example pictures is the category “cat”.

The identification of different objects and their relation to their meaning is the task of image classification in the research field of computer vision. An algorithm represents the image content and relates them with a semantic database. In the database, the semantic meaning of a special representation which depends on the used algorithm of objects is stored. Various algorithms exist to classify images or parts of images.

2.2. Image Representation

Before to start with image processing tasks, the way an image is specified has to be cleared up. An image is the digital version [1] of a scene with different properties, e.g., size, spatial resolution, color representation, sampling, quantization, intensity resolution and encoding.

The spatial resolution describes the size of dots per inch. In a 2D-image the x-axis refers to the width and the y-axis to the height and is called pixel resolution. The higher the resolution of the image, the more information which can be captured and stored is existing in this image.

[1] In this thesis only digital images are relevant.

In low resolution images, the accuracy and detail level is worse compared to higher resolution images. A limiting factor of high image resolutions and the generated higher size of data to be processed is the processing time of an algorithm from the moment of image capture to the classification output.

In the electromagnetic spectrum, light is the electromagnetic radiation in the range from $430nm$ to $790nm$. In this certain range the human visual system is able to see colors. The human eye is sensitive to the color format Red-Green-Blue (RGB) and this system is adopted by computers in order to adapt to the human eye. In an image every pixel has a color representation which is explained by a color system. The most used one in the context of display devices [2] is the RGB color format as stated above. Nowadays, the most common color channels $R, G, B \in [0, 255]$ uses 8-bit for the color depth. In certain cases, more bits per channel are used. Another color format is the Hue-Saturation-value (HSV) representation, which is closer to the human color system perception than the RGB color format. The Hue $H \in [0°, 360°)$ represents the perceived color, the saturation $S \in [0, 1]$ is the chroma and the value $V \in [0, 1]$ is the lightness. The transformation from RGB to HSV and vice versa can be studied further detailed in (Hanbury, 2003).

Image resolution and color information are details which are stored in digital 2D-images. These are encoded with lossy or lossless image compression methods. Different formats and methods exist with advantages and disadvantages with regard to the file size and loss of details during compression methods. In this thesis the trained data uses the PNG™ format published at libpng.org.

With this image representation no local information or relation between pixels in an image is given in order to get a low-level representation of global or local features, which are points of interest, e.g., edges or corners. To extract such features out of an image, the image gets filtered with a filter function h. The filtering is processed with the mathematical operation of convolution. The discrete version of a 2D circular convolution operation is used to filter the

[2] In this thesis only display devices are relevant. The Cyan-Magenta-Yellow-Key/Black (CMYK) color system is the mainly used in printing.

image (see Equation 2.1). $I(x,y)$ is a 2D matrix describing the original image data, which is convoluted with $h(x,y)$ by the convolution operator $*$. $h(x,y)$ is a 2D filter matrix and the result is the output image $g(x,y)$. x and y are the indices of the image matrix position and N,M are describing the size boundaries of the used filter.

$$g(x,y) = I(x,y) * h(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h(m,n) \cdot I(x-m, y-n) \tag{2.1}$$

There exist many different filters, which can be applied in this step. The filters can be used for edge detection, smoothing, or others. An example for an edge detection filter is the sobel filter which filters on the x- and y-axis of the image. This is a special hand-crafted filter, which is limited only to edge detection. An improvement is to use a filter collection with different filters to get a broader field of application. In deep learning approaches the filter collections are learned within the method depending on the input data. This is the transition from hand-crafted to learned filters, which is also the transition from rule-based and traditional machine learning methods to deep learning (Alom et al., 2019). *»A key difference between traditional ML [machine learning] and DL [deep learning] is in how features are extracted.«* (Alom et al., 2019). For a detailed overview of this transition see Table 1.

Table 1.: Comparison of different feature learning approaches and the transition from rule-based to deep learning methods. The further the transition goes the more learning steps have past. Table taken from (Alom et al., 2019, p. 4).

Approaches	Learning steps				
Rule-based	Input	Hand-design features	Output		
Traditional Machine Learning	Input	Hand-design features	Mapping from features	Output	
Representation Learning	Input	Features	Mapping from features	Output	
Deep Learning	Input	Simple features	Complex features	Mapping from features	Output

2.3. Image Classification

An image can be analyzed and classified accordingly. This is done in several steps from low-level-operations to higher-level-operations. The image is filtered by its geometric forms. In this step a histogram or other data can be generated depending on the algorithm. The goal of image classification is to interpret the filtered data in an image with the aim to output the related object description. As stated above, different steps are undertaken. Deep learning methods are the most modern methods within the research field and only those are relevant for this thesis. This is because the deep learning method RotationNet has been chosen for a real-world application. Here, the underlying deep learning method range of Convolutional Neural Networks is explained in order to build the foundation for understanding RotationNet as such.

2.3.1. Deep Learning

Convolutional Neural Networks (CNN) are neural networks which are able to work with 2D-image data and which use the advantages of convolution. The intention of neural networks is to model the biological visual system and perception of humans and animals. The biological ideal of neurons is used to model it into a computational representation.

2.3.1.1. Neuron

A neuron is a nerve cell in the human body (and in other animal's bodies with nerve systems) which builds together with a multitude of other neurons the nervous system in the body. The nervous system is densely entangled and transports important information throughout the body. This system is the exemplar for artificial neural networks because of its complexity in the transport of information. In Figure 2 a representation of an artificial neuron is shown. The model uses the input $\mathbf{x}$. The system parameters are a linear combination of weighted

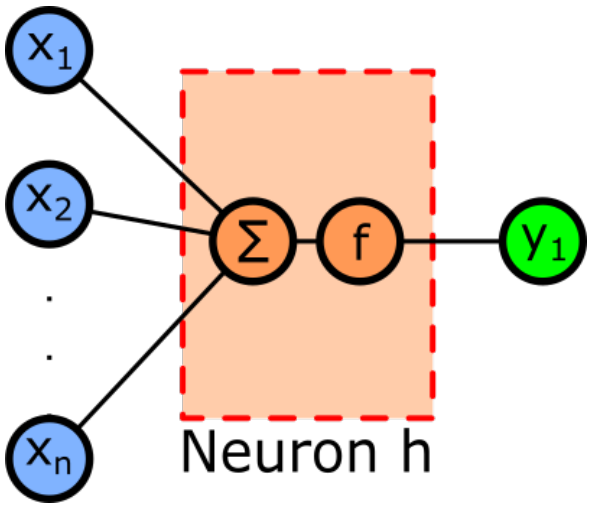


Figure 2.: An artificial neuron model exists of $[x_1, x_2, \ldots, x_n], n \in N$ inputs and one output y. Inside a neuron an activation function f transforms the summation of all inputs to the output value y_1.

inputs $\mathbf{w}$ and a bias b in Equation 2.2. This can be rewritten in a vector form as in the Equations 2.3- 2.5. Finally, an activation function f is used to get the output value y. The parameters which are not fixed are the weights $\mathbf{w}$, which are optimized during a training process.

$$z = \sum_{i=0}^{n} x_i \cdot w_i + b, \text{ where } n \in \mathbb{N} \tag{2.2}$$

$$x' = [x_1, x_2, \ldots, x_n, 1] \tag{2.3}$$

$$w' = [w_1, w_2, \ldots, w_n, w_{n+1}], \text{ where } w_{n+1} = b \tag{2.4}$$

$$z = w'^T \cdot x' \tag{2.5}$$

The neurons output z is activated or rated with a following activation function $f(z)$ in Equation 2.6. The activation function can be a binary, linear function or a non-linear function, e.g,

$$f(z) = max(0, z) \tag{2.6}$$

Historically, binary activation functions like a threshold are used to decide between two values, e.g., yes (+1) or no (-1) (McCulloch and Pitts, 1943). To model the biological system

much better non-linear activation functions are the state-of-the-art in deep learning networks. In the context of CNNs, e.g., a rectified linear unit (see in Equation 2.6) is used for the activation function (Fukushima, 1988). The activation functions' parameter have the ability to be learned (Bishop, 2006).

2.3.1.2. Neural Networks and their Layers

Several neurons can be combined and grouped in logical layers. The grouping of neurons is a linear combination of many neurons. One group is called layer in a neural network (Fukushima, 1988). In Figure 3 one layer has $\{x_1, x_2, \ldots, x_n\}, n \in N$ inputs connected with $\{h_1, h_2, \ldots, h_m\}, m \in N$ neurons. These are connected to $\{y_1, y_2, \ldots, y_k\}, k \in N$ outputs.

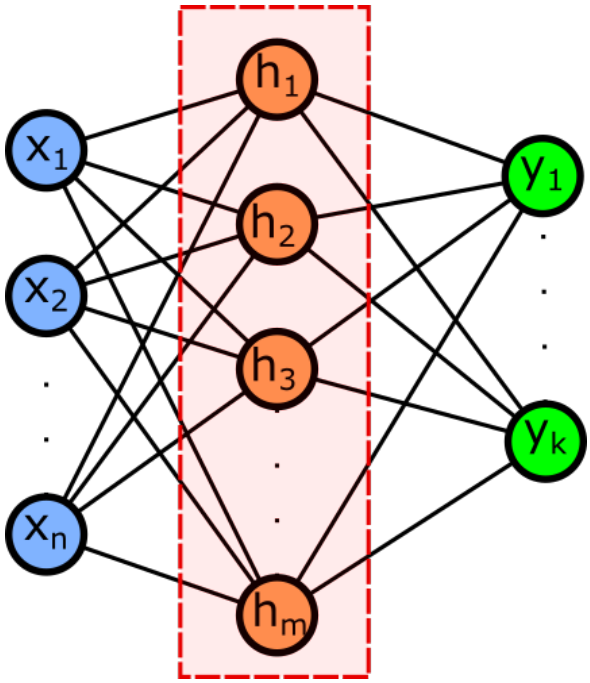


Figure 3.: A three layer neural network with input, middle and output layer. The left input layer has n inputs, m neurons are in the middle layer and the output has k neurons.

If every input is connected with all neurons and all neurons are connected to the output, it is called a fully-connected network. Layers between the input and output layer are called hidden layers. The dimension of every layer and the number of layers in a neural network

depends on the design of the system. The hidden layers can be seen as nontransparent boxes with known input and output, but unknown parameters (Haykin, 2007). These parameters can be trained (optimized) and are described by the weights **w** in Equation 2.2.

The using of simple non-linear activation functions $f(z)$ per layer, assumed many layers exists. This results in a composition of non-linear functions. In other words many simple non-linear functions are forming a complicated function (Fukushima, 1988).

2.3.1.3. Feed-Forward Network

A feed-forward network can be seen as a left to right or top down model with stacked layers. These layers merely have a connection to the successor but no backward or recurrent connection (Rumelhart et al., 1986). This is a base model used by deep networks which extends the number of layers in order to have *deeper* levels of layers. In Figure 4 an example of a neural network layer with 2 hidden layers is shown. Where $\mathbf{x} \in R^D$ is the input, $\mathbf{h_1} \in R^N$

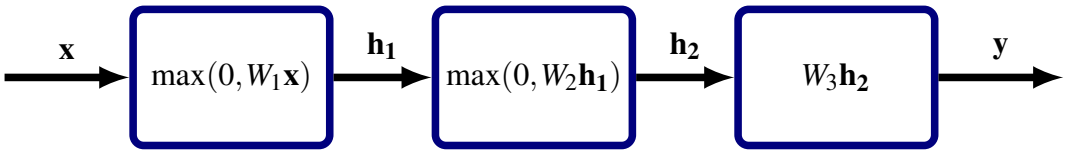


Figure 4.: Example of a neural network layer with 2 hidden layers. Where x is the input, h_1 and h_2 are the output of the hidden layers and y refers to the output.

is the first hidden layer output, $\mathbf{h_2} \in R^M$ is the second hidden layer output and $\mathbf{y} \in R^P$ refers to the output. The weights $W_1 \in R^{NxD}$ and $W_2 \in R^{MxN}$ have also biases $b_1 \in R^N$ and $b_2 \in R^M$ resp. which are not covered by the picture. h^1 is the input for the second hidden layer which indicates a "fully-connected" layer. As activation function a so-called Rectified Linear Unit (ReLU) (see Equation 2.6) is used in the first two layers.

2.3.2. Convolutional Neural Network

A CNN uses the advantages of the mathematical operation of convolution. The structure of CNN-based methods is that many layers are stacked together in hierarchical order. The dimensions between the layers can differ and only the output and the input layer are known. All other layers which are situated in between are hidden layers which are not visible. The features and the classifier are both learned in convolution neural networks. Compared to hand-crafted machine learning approaches this is much more flexible and can be used in a broader field of application (LeCun et al., 1998; Hinton et al., 2006). In Figure 5 the

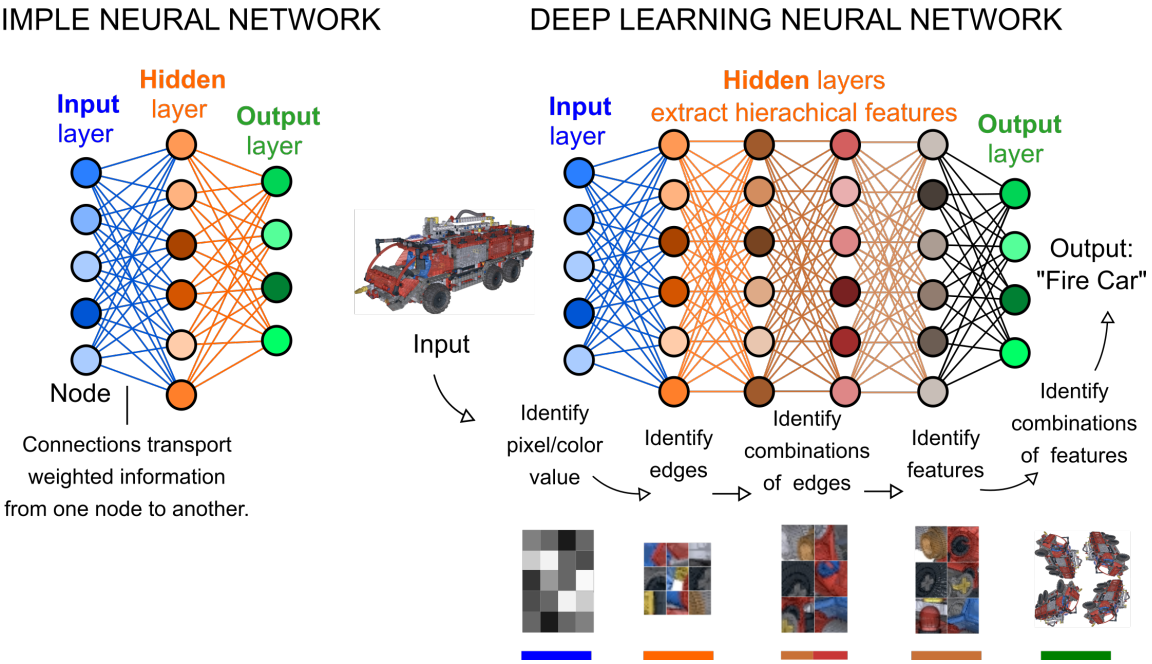


Figure 5.: Comparison of the simple (one hidden layer) neural network structure and the deep learning network structure. The main difference between these two is the level of layers. Deep learning networks process and link information from a low-level input image on a pixel basis to a higher level where edges are identified, to the next level where edges are combined. The next level is a higher level on which features are identified and the last layer combines the features to the output word. This is a schematic representation and not a specific method.

structural difference of a simple and a deep learning neural network is illustrated. A CNN has a 2D-image with a dimension of $dim = Height \cdot Width \cdot Depth$ as input, where the $Depth$ is equivalent to the color channel. The hidden layers which can have different tasks are

following after the input layer. These different tasks of the layers are, e.g.,

convolution: extracting features of an image

pooling: selecting a region and calculating, e.g., maximum or average of this region and outputs one single value. This can be seen as down sampling of the previous layer features

softmax: outputs a normalized probability distribution of the layer's neurons.

The convolution layers consist of multiple non-linear filters with a small kernel size, e.g., $kernel = 7 \times 7$, in contrast to the image size, e.g., $image = 256 \times 256$. This is an optimization to reduce the size of parameters, which is computed in every run and the computational power or memory needed.

This optimization can be illustrated with an example: A 2D image can be considered on the basis of pixels which makes the size of neurons in a hidden unit the same as the pixel size and every pixel has a connection to every neuron. The result is a huge amount of parameters to be stored and computed. However, spatial correlation is a local phenomenon. A 2D image can as well be seen on a patch basis, where a patch of, e.g., 7×7, has a connection to every neuron which has the size of a patch in the hidden unit. Based on the usage of patches, edge detection, where the edges are learned from the appearance in the input image, can be performed. This gives a higher-level representation than using raw pixels. The parameter size can be further reduced by reusing the same parameters on different locations in an image. Any edge can be located everywhere in an image, so it can be shared through the system. This results in fewer parameters and the convolution is calculated with learned kernels (LeCun et al., 1998). The content of an image can look very similar in different locations and dependencies are local. This statistical viewpoint and the fact that filtering is *equivariant* to translation brings the advantages of convolution into account. Translation *equivariance* results in fewer filters because no translation replicates have to be covered and only the orientation/frequency must be stored. This can be seen as a set of local parts (Lenc

and Vedaldi, 2019). If there are hierarchically stacked layers of non-linear filters, a system is called deep network.

The deep network is constructed by adding layers of a higher hierarchical order. The non-linearity of filters gives more detailed and adequate information about the data when it is applied to parameters with high hierarchical orders. The first layers have more basic filters with very local information. They can be compared with an edge detection filter engine. Each following layer has more detailed information – e.g., about the texture – in the analyzed region. Therefore, deeper levels have more combined information about the content representation at their disposal (Zeiler and Fergus, 2014).

A vast amount of data is needed to train a CNN. The images or training data, respectively, must be classified manually or can be taken from existing databases for some classes. It is of importance how granular the classes are defined and which and how many different classes are present in an image for a productive usage of the whole system. This manual step of preprocessing data has a strong influence on the output, thus needs to be very precise in terms of applicability.

With the possibility of parallelization and the rising computational power of graphics processing units (GPUs), the processing time of the vast amount of datasets decreases. The training or learning and classification is split into *offline* and *online* tasks. The CNN parameters are trained with a big dataset *offline* before they are used *online* to classify new and not yet trained data. When the training phase is finished, a trained network is present. In the classification process the learned data from the trained network is used to categorize the subsequent input data and a result is returned on the output data. This means that the classification process is a forward propagation of data which the system does not know prior to the process through the trained network.

Forward propagation is *»[...] starting at the bottom and working upwards until the states of the output units are determined«* (Rumelhart et al., 1986) (bottom refers to the input and upwards is the direction towards the output). It stands in contrast to the back-propagation,

which is used in the training process to learn parameters of a network (Rumelhart et al., 1986). This is the learning process with existing data. Rumelhart explains this as follows: *»The aim is to find a set of weights that ensure for each input vector the output vector produced by the network is the same as (or sufficiently close to) the desired output vector.«* (Rumelhart et al., 1986) *»The backward pass which propagates derivatives from the top layer to the bottom one [...]«* (Rumelhart et al., 1986) computes the gradient descent of the layers with the help of the chain-rule (top refers to the output and bottom is the previous layer). This is the learning process with existing data of the whole network. These deep learning network methods are the current state-of-the-art as they outperform the non-learning methods.

Therefore, the benchmark ImageNet publishes a database for image classification where images are manually classified according to the image content. Every year challenges are held in order to compare methods with each other on the basis of these databases in different categories, e.g., classification, localization or identification. The measurement used in these competitions is the error rate of the N best (maximal probability) classification results, which is called Top-N error rate, e.g., for $N = 5$ the Top-5 error rate includes the five best classification results. At the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012 (Russakovsky et al., 2015) the method AlexNet (Krizhevsky et al., 2012) was the winner with a Top-5 error rate of 16.4% on the supplied dataset (see in Figure 6). The second best implementation was not a deep learning network and had a rate of 26.2%.

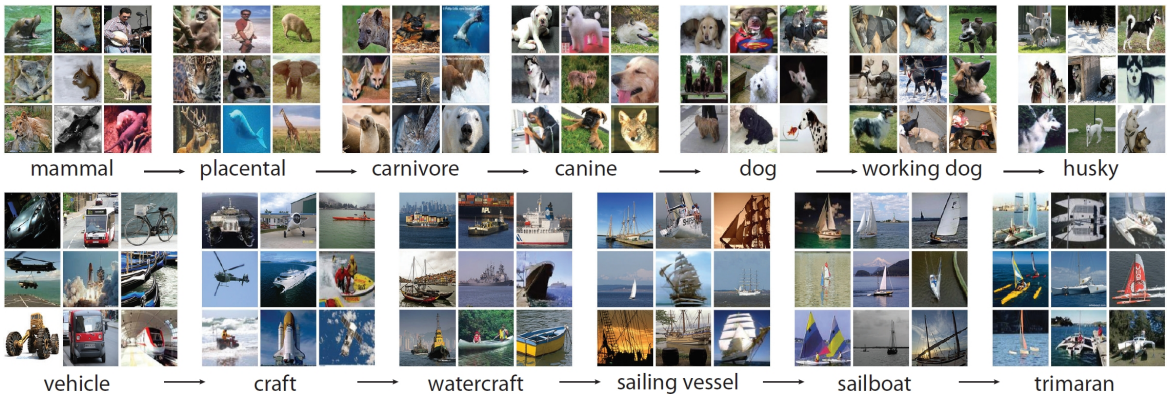


Figure 6.: A random selection of two root-to-leaf branches of the ImageNet benchmark. Image source: (Deng et al., 2009)

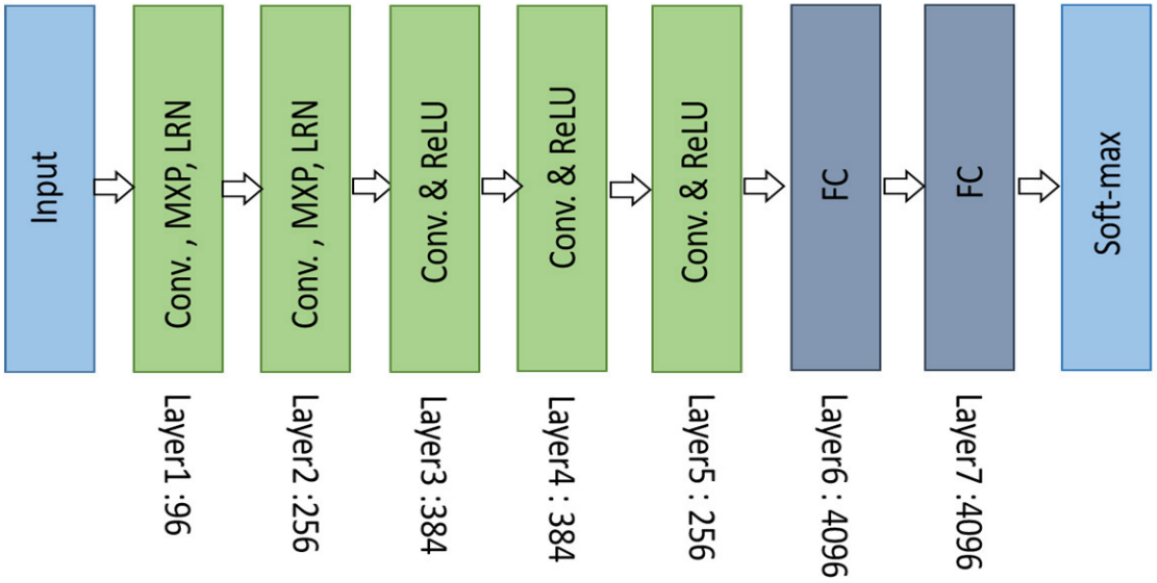


Figure 7.: The layer structure of AlexNet. The different layers have different functions. Layer1 and Layer2 are designed with convolution (Conv.), max-pooling (MXP) and local response normalization (LRN). Layer3, Layer4 and Layer5 are convolutions with a following rectified linear unit (ReLU). The Layer6 and Layer7 are fully connected (FC) layers and the last layer is a soft-max function, to categorize the classes with a probability value. Image source: (Deng et al., 2009)

Therefore, AlexNet had a significantly low error rate compared to others in this contest. AlexNet is designed with eight layers in total, five convolutional layers and two fully connected layers and at least a soft-max layer. In Figure 7 a graphical representation of the blocks of the structure of AlexNet is shown. The convolution layer is combined with a max-pooling, which is a form of down sampling a feature-map. The value in the resulting feature map is the maximum value of a region from the origin feature map, e.g., in a 2×2 region with the values $[1,4,2,2]$ the result is 4 in the resulting feature map, the dimension is reduced by $\mathbb{R}^{2\times 2}$ to $\mathbb{R}^{1}$. The local response normalization is more a *»brightness normalization«* (Krizhevsky et al., 2012). The fully connected layers are constructed like explained in Section 2.3.1.2 to classify the previous layer outputs. The soft-max function calculates the probability of every output class based on the last full connected layer.

Because AlexNet has produced such good results, nowadays many other methods are based

on AlexNet. Since then CNNs are used in other research fields and enhanced in performance and speed. Convolutional Neural Networks have a low error rate compared to hand-crafted methods as can be seen in the exemplar of AlexNet. A drawback is that an enormous amount of data has to be available before the method can be applied. AlexNet has overall 61 million unknown parameters, which are learned (optimized) in the training process (Alom et al., 2019). The training process is a time and computational intensive process and needs much computational power. Here, AlexNet will not be explained in further detail because the method used for this thesis is based on AlexNet with further improvements and enhancements. The used method RotationNet will be explained in further detail in Chapter 3.

The main goal of all methods is the identification of objects within images. For this thesis, a test of the RotationNet method on the assembly of a 3D model is made. The assembly is conducted in several instruction steps. RotationNet is used to predict the subsequent instruction step in order to serve as a virtual instruction manual for the person assembling the model.

2.4. Image Identification

The current status of the assembly by identifying the correct instruction step number is the main focus of attention of a method. In real-world applications, the model is a 3D object in front of a person assembling the model. However, the computer system needs to capture this 3D data in order to further process it. The problem can be approached using 3D techniques based on depth images and reconstruction algorithms (Häne et al., 2017), or by using 2D image-based methods. This distinction is blurred because 3D depth information can not only be extracted from several 2D images (Hartley and Zisserman, 2004), but can also be learned from a single 2D image (Saxena et al., 2006; Kuznietsov et al., 2017; Mahjourian et al., 2018). If necessary, machine learning approaches can implicitly learn the depth information and explicit handling of depth information is not required with image-based approaches on a machine-learning basis. For this thesis, the learning from a single 2D image

is used. However, the 3D object has to be captured in any case. Therefore, the data is imported as explained in the next Section.

2.5. Importing Data with the LDRAW Importer

The main function for the importer is to read a 3D model, which is stored in the LEGO model data format (LDRAW), and to output another 3D model, which is stored in Alias Wavefront OBJ data format (OBJ). The used LDRAW format specification is in version 1.2. The Alias Wavefront OBJ format consists of two parts: an OBJ and a material file MTL. For the output the OBJ format is used because it is widely supported by 3D rendering programs and other graphics software packages. The implementation is made in JAVA^{TM} as a library.

2.5.1. Format Specifications

The structure of LDRAW is similar to an offline LEGO^{TM} instruction manual with iterative instruction steps. It is possible, that the instruction steps are interrupted by sub-models which must be completed before resuming to the interrupted instruction step of the main model. The sub-modules can be seen as branches which are assembled separately of the main model where the instruction steps start from step one. After finishing the sub-model, it is integrated into the main model. This process is illustrated in Figure 8. In the resulting model, the sub-part (see in Figure 9) can be visible (attached to the existing model), partly-visible or occluded (inserted into the existing model). These sub-model steps are merged to the central theme and no branches are existent in the digital version. The digital version stores the necessary information for the steps in one file which can be viewed from different viewpoints with a rendering software. LDRAW is internally organized hierarchically, recursively and supports linking to primitive parts, groups of many parts or other LDRAW files. OBJ does not support this structure, but named groups are specified. The instruction steps from LDRAW are mapped one-to-one to the OBJ named groups. All other LDRAW specific

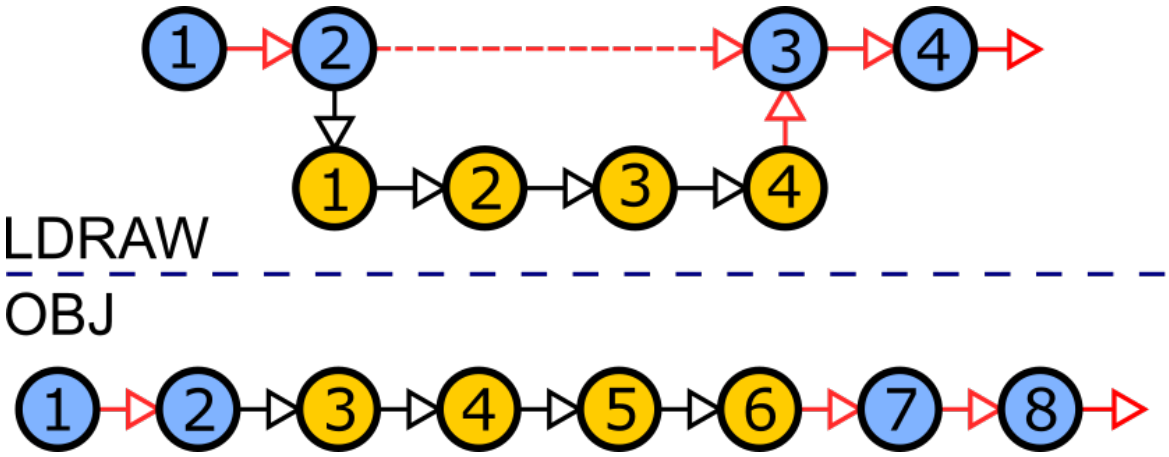


Figure 8.: Structure of the LDRAW model with the main model (blue) and sub-models (yellow). The sub-model branch interrupts the main model instruction step (blue;2) and resumes with step (blue;3). The sub-model instruction steps start from one. The OBJ merges the sub-model (yellow;3-6) instruction steps into the main model (blue;1-2,7-8) .

linking, hierarchies and recursions are resolved to fit to the OBJ specification, excluding unnecessary LDRAW specifications or extensions. The “Airport Rescue Vehicle” (no. 42068) in the LDRAW format consists of 72 instruction steps. In the OBJ format, this is extended to 137 instruction steps because of the sub-steps.

2.5.2. Implementation

The first step is to read the main LDRAW file. Only the information which is needed for an optimal output model is processed. Other information is ignored. The data is structured in instruction steps so that in every instruction step the model can be described with primitives, parts or other LDRAW files. There is, furthermore, the option to link to other parts from the official database or internally in the defined parts of the file. This is handled by a hash table to be able to resolve the links. Every primitive part or link comes with positioning, translation, scaling or rotation data of how it should be rendered. This is solved with a transformation matrix for all parts. All this information is converted to the internal data structure and exported to OBJ. The color information and materials are also converted. This

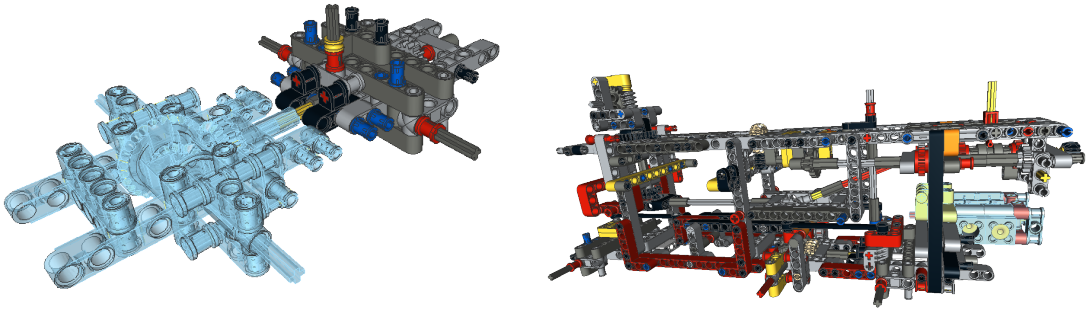


Figure 9.: Two different instruction steps with appended sub-models. The added sub-model is highlighted in light blue. The sub-model in the left picture is not occluded by the main part. In the right picture a partly occluded engine is appended to the main model. The pictures are generated with LDView© (LDVIEW.SOURCEFORGE.NET).

is stored in the extra MTL-file. The combination of the OBJ- and MTL-file can be used for further proceedings.

2.6. Conclusion

This chapter has shown that for object detection in images, the semantic description and the image representation must be connected. First, image classification is the method to get the semantic prediction. Historically, hand-crafted designs for image classification were used before the usage of CNNs became the standard. One key benefit of CNNs is a highly non-linear deep system with the ability to train parameters on the basis of the input training dataset. A CNN is then trained *offline* and the classification is taking place *online*. In CNNs, a supervised learning is used to train the network. This means that the training image dataset with the relation to the class must be known in advance. The image identification is performed with 2D images of a 3D model. Finally, the creation of the 2D image database for this thesis is accomplished with the LDRAW importer. The importer is the intermediate tool for converting 3D models to 2D images. This gives a thorough understanding of the theoretical foundation for the used method. In the next chapter, RotationNet and its modifications for this thesis will be discussed.

3. Theoretical Background of the Used Method

3.1. RotationNet

The method used for the practical implementation is called RotationNet (Kanezaki et al., 2018). It is inspired by the concepts of MVCNN (Su et al., 2015), a multi-view convolutional neural network to classify 3D objects from 2D images, and by the pose estimation technique of "Convolutional Models for Joint Object Categorization and Pose Estimation" (Elhoseiny et al., 2016). These two concepts are combined and extended by RotationNet: MVCNN uses different view points and camera positions distributed over a sphere around a 3D model to get 2D images. These images are trained with the network structure of AlexNet (Krizhevsky et al., 2012). The MVCNN approach demands for each class that all camera positions are available as images; i.e. for every pre-defined view point all 3D objects must be captured. This is, however, hard to realize in a real-world scenario, where view positions are often limited and not precise. Therefore, RotationNet removes this limitation of MVCNN by repositioning the pooling layer and combining it with the method of (Elhoseiny et al., 2016). They propose to use object prediction and pose estimation with one 2D image as input for the classification process. Elhoseiny et al. (2016) evaluate and analyze five different model configurations:

- unmodified AlexNet, as baseline model
- a parallel model, which splits the classification and pose estimation completely
- a cross-product model, which outputs the category and pose as cross-product
- a late branching model, which splits the category and pose at the last layer

- an early branching model, which branches the layers before all fully-connected layers and after the pool layer 5.

RotationNet finds a balance between these two methods and uses a partial set of multi-view images, which results in a set of images where at least one image is in a set of regional, alike viewpoints. With this combination, a partially captured 3D object can be categorized. This is an improvement of the other methods which classifies images for real-world use cases. The idea behind the proposal of RotationNet is to predict a class with some – not all – viewpoints rather than using all viewpoints. This leads to better results than MVCNN. As a consequence, RotationNet is an improvement of both approaches. As the success rate of the classification increases with a higher number of images in one set, RotationNet uses a compromise with a set size of 20. To estimate unknown poses of a 3D object during the training process, an unsupervised pose estimation is used, which is influenced by (Zhou et al., 2017). This is a "meta" task which is conducted in every training step. Another benefit is the feasibility to predict a class with a specific set of images captured from one limited view position region. This is important for estimating new unknown positions and simultaneously classifying objects. To get a low error rate, this step is of significant importance.

3.1.1. Viewpoints

As mentioned above, the camera positions used in RotationNet are highly influenced by MVCNN. Three approaches of where the view positions are located on a sphere enclosing the 3D object are proposed: First, camera positions rotating on a fixed axis around the model. Second, the cameras distributed on a dodecahedron. Third, camera positions rotating on a fixed axis around the model (same procedure as for the first proposal), repeated by using a second rotation axis. The resulting view positions are equivalent to a longitude and latitude description as, on the hemisphere. The first and third approaches have a fixed rotation direction and models have to be fixed in an upright position. The second approach rotates in three possible directions and models can be placed in any position. RotationNet uses the

second approach for its experiments on image benchmark data sets. The positioning of the camera viewpoints is of high importance. Size and distribution of virtual cameras influence the prediction accuracy. RotationNet uses a similar approach of MVCNN and extends it. Therefore, different approaches exist which differ in the number and positioning of virtual cameras. RotationNet discusses three different cases, which are similar to the MVCNN approach. These cases are:

- in upright position and with twelve views on a fixed axis (described in Section 3.1.1.1),
- not in upright position and with 20 views on a dodecahedron (described in Section 3.1.1.2),
- in upright position and with views spread in different circles over the sphere (described in Section 3.1.1.3).

3.1.1.1. Circle views

In the first case the upright positioned object is viewed by N_C camera positions. These positions are distributed ideally around the object in a circle around a single fixed axis. The distance between the camera positions is given by an angle α in 3.1.

$$N_C = \frac{2\pi}{\alpha}, N_C \in \mathbb{N} \tag{3.1}$$

Here, the object is seen from a limited perspective which results in a loss of information of the whole object. Top or bottom regions of a model are not captured. Therefore, some details, which are important might not be visible in the field of view of the camera or be occluded by other parts of the model. This information is lost for further training steps. Due to the fact that a highly representative training data set is aspired, this limitation is not desired and not adequate for the use case and evaluation in this thesis.

3.1.1.2. Dodecahedron

In the second case, 20 camera positions are equally spaced on a unit sphere which are equally distributed on a dodecahedron on this sphere. In Table 3.1.1.2 and in Equation 3.2 and 3.3 the coordinates and their calculation for every position is shown. This option is used for classification experiments by RotationNet.

$$p = \frac{1+\sqrt{5}}{2} \tag{3.2}$$

$$q = \frac{1}{p} = \frac{2}{1+\sqrt{5}} \tag{3.3}$$

Table 2.: Coordinates for all 20 view positions on a unit sphere distributed as a dodecahedron. The variables p and q are calculated in Equation 3.2 and Equation 3.3.

View Position	x	y	z	View Position	x	y	z
1	-1	-1	-1	11	-q	0	p
2	-1	-1	1	12	0	p	q
3	1	-1	1	13	p	-q	0
4	1	-1	-1	14	0	p	-q
5	-q	0	-p	15	p	q	0
6	-p	-q	0	16	q	0	p
7	0	-p	q	17	-1	1	1
8	-p	q	0	18	-1	1	-1
9	0	-p	-q	19	1	1	-1
10	q	0	-p	20	1	1	1

MVCNN uses a similar version to the dodecahedron. Camera positions are generated an icosahedra mesh, where 20 camera positions plus four rotations – with 0, 90, 180, 270 degrees per position – results in 80 views. This approach is adapted by the method of RotationNet. Here, the camera view direction is adjusted to the origin $c = (0,0,0)$, where the center of the 3D model is located. These camera locations are fixed for the rendering process and every camera has a unique number. Every 3D-model is rendered by these 20 camera positions to create 2D images. In Figure 10 a graphical interpretation is displayed.

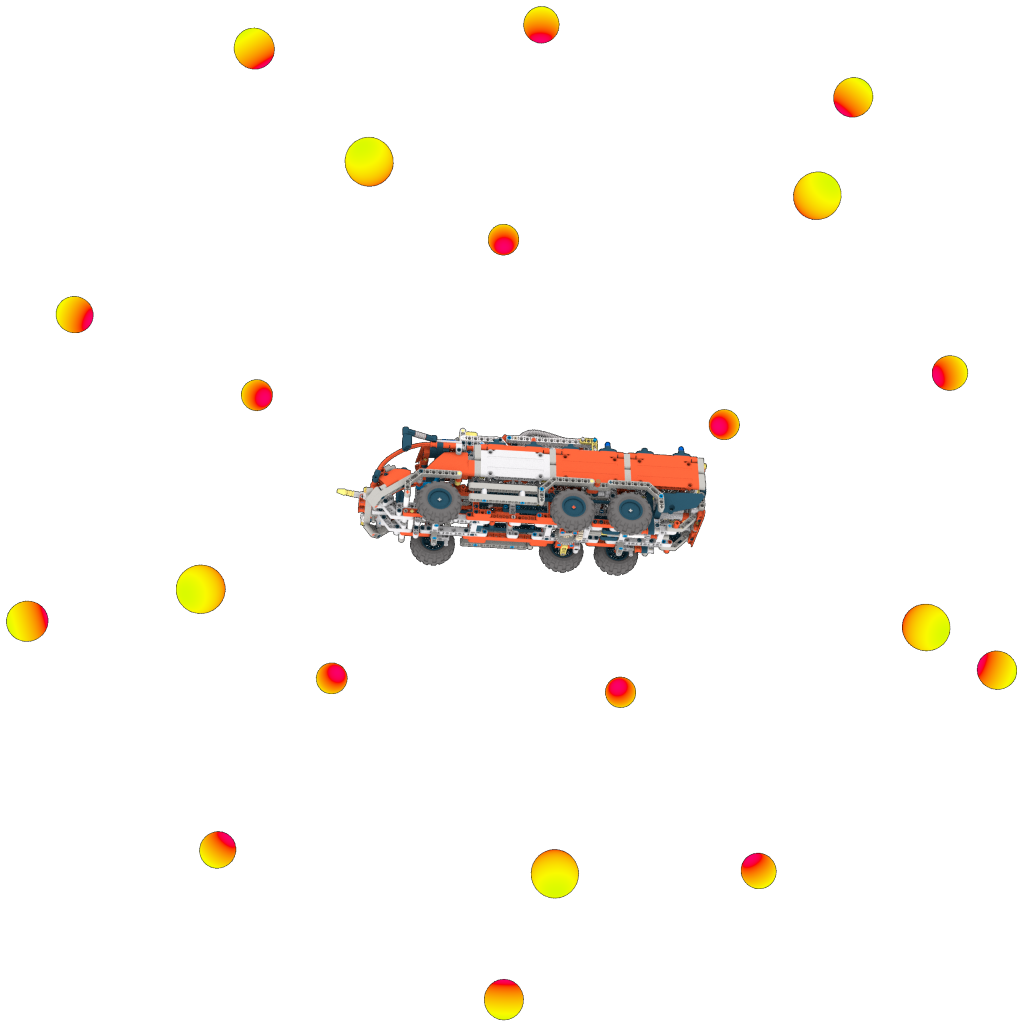


Figure 10.: The complete model in the center is surrounded by 20 cameras distributed on a dodecahedron. The dots are representing the positions of every camera around the model. The red color symbolizes the view direction of every camera towards the model.

3.1.1.3. N-Circle views

The case of N-circle views improves the entropy of the approach described in Section 3.1.1.1 to capture non-viewed regions and a larger camera viewpoint size. The model is positioned in an upright position. A second axis y is fixed to vary the camera positions on angle β in Equation 3.4 of the circle z, which is generated like in Section 3.1.1.1, in Equation 3.5 to get a spherical distribution of the viewing positions N in Equation 3.6) with a higher resolution.

$$N_y = \frac{\pi}{\beta} \text{ , where } N_y \in \mathbb{N} \tag{3.4}$$

$$N_z = \frac{2\pi}{\alpha} \text{ , where } N_z \in \mathbb{N} \tag{3.5}$$

$$N = N_y \cdot N_z \text{ , where } N \in \mathbb{N} \tag{3.6}$$

In Section 3.1.1.1 one axis is used as rotation axis. Here, the approach is repeated on every part of a second rotation axis y. The rotation is made on a half sphere in the range between $-\pi$ to π.

3.1.2. Pose estimation

A drawback for RotationNet is that object poses are not well aligned in existing object databases, like ModelNet. Therefore, to train such a network, several choices are possible. Two possible cases are either creating a database with aligned poses or to use pose estimation in an unsupervised manner. To resolve the issue viewpoints are estimated unsupervised in RotationNet, so-called latent parameters are established to receive the poses. After some training iterations the viewpoint variables are getting more precise, because they are optimized in the training process. The way RotationNet tackles the problem is that no pose estimation is needed beforehand, which is sensitive to noise and individual shape differences. The sensitivity comes from the image resolution, illumination and the resulting reduction of the parameters are erroneous (Kanezaki et al., 2018). This pose estimation is not needed

in this case, but the complete set of options is used, because no negative side-effects of the usage are expected.

3.1.3. Training

All pre-defined viewpoints N have a unique corresponding 2D image of a 3D model. Every instruction step is equal to a class in the network context. One class is trained with all images, which are referring to the given class, rendered from the pre-defined viewpoints. The class is known during the training process. However, the viewpoint is not given and is optimized during the training. The multi-layer network has a soft-max layer of the size of all viewpoints and classes in the final layer. This layer is used to compute the likelihood Px_i for the views in every class separately. RotationNet solves the optimization problem in Equation 3.7. The goal of this is to optimize (train) the network parameters Net in consideration of the viewpoints $\{v_i\}$. The training input images $\mathbf{x}_i$ in combination with the viewpoint position v_i from camera index $i \in [C_1, C_2, \ldots, C_m]$ with the approximated class $\hat{y}_i$ of the ground truth class y gives the likelihood probability for y.

$$\max_{Net, \{v_i\}_{i=1}^{N}} \prod_{i=1}^{N} P(\hat{y}_i = y | \mathbf{x}_i, v_i) \tag{3.7}$$

Otherwise, an existing database like the ImageNet (Deng et al., 2009) database can be used and fine-tuned with the new data. However, creating a new database is demanding due to the lack of enough training data in other scenarios.

3.1.4. Prediction

RotationNet uses a set of images for classification either, sequentially or simultaneously. The network output layer values are the probabilities for every class, including the *incorrect view* and all pose estimations per class. The best view position is chosen by taking the highest

probability value of a class without the *incorrect view*; i.e. inter-class probabilities are taken into account. This means not only one specific class is used to predict the pose, but more than one can be used for one prediction result.

The decision, which class matches best is calculated by the maximum value of the probability product of the views and classes as shown in Equation (3.8).

$$\{\hat{y}, \{\hat{v}_i\}_{i=1}^{M}\} = \arg \max_{y, \{v_i\}_{i=1}^{M}} \prod_{i=1}^{M} \frac{p_{v_{i,y}^i}}{p_{v_{i,N+1}^i}} \tag{3.8}$$

$p_{v_{i,y}}^i$ is the probability for all $M = 20$ viewpoints v_i in a class $y \in \{C_1, C_2, ..., C_N\}$ with $N = 137$. This is normalized to the *incorrect view* probability $p_{v_{i,N+1}}^i$. The product of all view probabilities gives the prediction for the class $\hat{y}$ and its viewpoint $\hat{v}_i$. Summarized, the prediction of the class and pose is a probability maximization over a set of images.

3.2. Modifications

Originally, RotationNet is trained on the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012 (Russakovsky et al., 2015) database and fine-tuned with the ModelNet10 and ModelNet40 (Wu et al., 2015) database. Therefore, RotationNet supports only 10 or 40 classes for the fine-tuning and prediction. This does not fit to new, created database using the LEGO Technic™ model mentioned before. In this thesis the proposed method does not use the ILSRVC database nor the ModelNet database, due to the fact that the databases have not much in common with the “Airport Rescue Vehicle”.

Before any pipeline workflow can be started a dataset of 2D images is needed. For this 2D images are rendered from a 3D model. The “Airport Rescue Vehicle” model from the LEGO Technic™ series is used as the reference model. The complexity of the model is comparable with a real CAD model. Due to the fact that the CAD model in the assembly assistance scenario is not similar to the existing dataset used by RotationNet at the ImageNet

Large Scale Visual Recognition Competition (ILSVRC) in 2012 (Russakovsky et al., 2015), a new database is created with the LEGO Technic^{TM} model mentioned before. The LDRAW format stores the model data as instruction steps which fit to the proposed use case.

3.3. Conclusion

RotationNet is a multi-view CNN with the ability of pose estimation. The method finds a balance between image classification and pose estimation. It mixes supervised learning for the classes and an unsupervised learning of the camera viewpoints. The optimization of the parameters for one specific class uses a maximization of the probability. The classification task returns the class with the highest probability for the input data. The camera viewpoint distribution on a dodecahedron is used for the generation of the training image dataset for the "Airport Rescue Vehicle" (no. 42068). RotationNet is modified to fit the size of classes to support this image dataset with 137 instruction steps (classes). The practical details of the modified version of RotationNet are illustrated in the next Chapter.

4. Practical Implementation of the Modified RotationNet Method

This Chapter handles the implementation details of the whole method. The sections of the chapter are arranged in the processing pipeline order of the whole process to learn and predict a 3D model with instructions steps. The processing parts can be split in an *offline* (see in Figure 11) and *online* (see in Figure 12) processing pipeline. The *offline* task consists of all training relevant steps, which has to be computed only once. This training process is time-consuming and takes several days on a high-performance computer (HPC) with 54 Intel® Xeon® CPU cores at 2.60GHz. This is the reason why the training process is made with HPCs. The process is a semi-automatic workflow, some sub-steps need manual configurations in advance in order to be able to perform further sub-steps. The prediction task is done in the *online* processing pipeline. For one prediction the processing time should ideally be in the time frame of seconds or milliseconds running on consumer devices with limited power. Compared to the *offline* task, processing time and computational power vary.

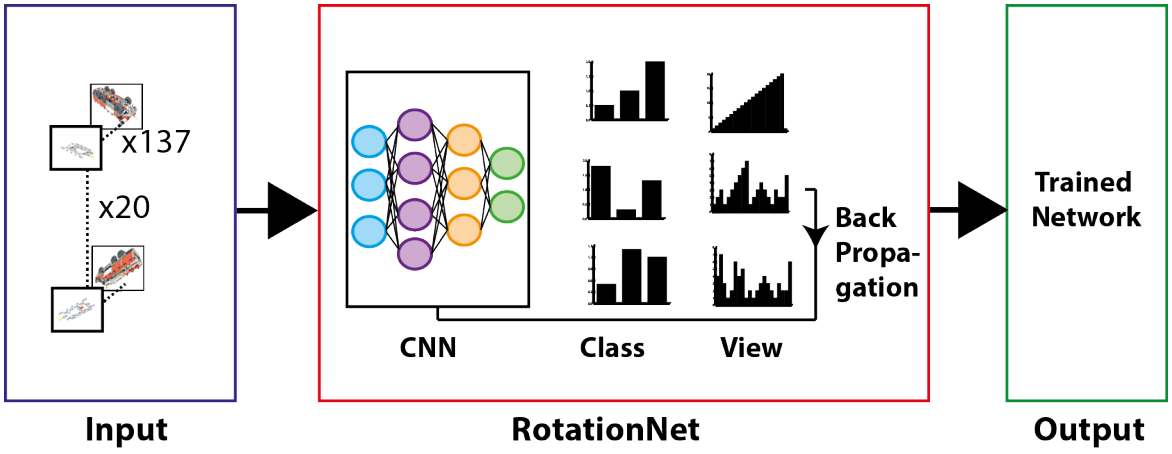


Figure 11.: Pipeline overview for the offline task: RotationNet is learned with the instruction steps on the basis of $n = 2740$ pre-rendered input images. Every 20 camera positions have 137 rendered instruction steps of the model.

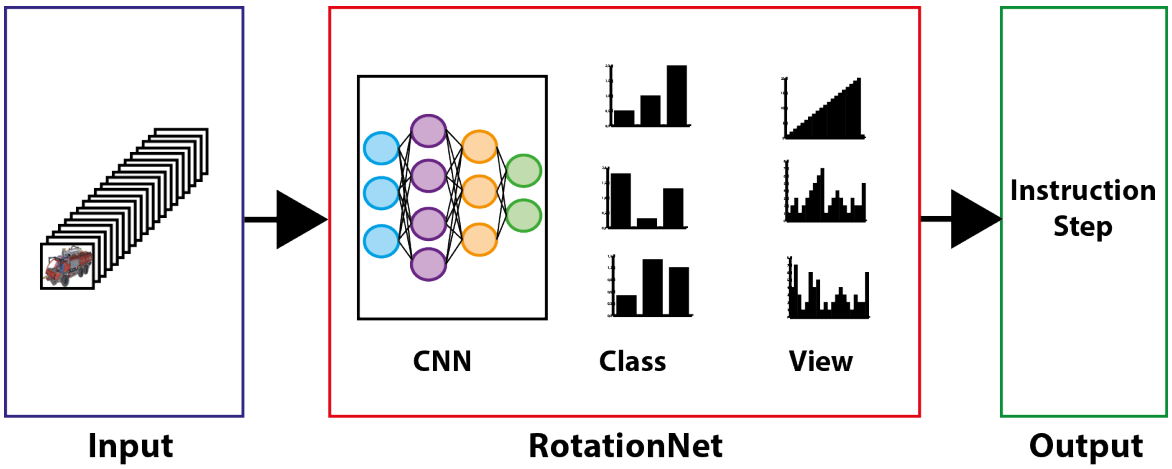


Figure 12.: Pipeline overview for the online task and test sets: RotationNet predicts the instruction step on the basis of pre-rendered trained input images.

The *offline* and *online* task run in CPU or GPU mode. This option is an advantage to get a wider support for many consumer devices, which are not able to run in GPU mode because of limitations of the hardware or software. In the *offline* task, the advantage is to be more independent to drivers and hardware, which are changing in very short time compared to CPUs. In other words, the life-time of the implementation without any changes is higher with the CPU mode than with the GPU mode. GPUs hardware or software is changing in a shorter time compared to CPUs and is more limited.

At the beginning of the *offline* task, the "Airport Rescue Vehicle" model has to be converted from LDRAW format to the "Alias Wavefront OBJ" format (OBJ). The rendering process uses the open source rendering software Blender™ to import the OBJ file and renders a 2D image for every instruction step and every camera view. After this process, the image database containing 2740 images is generated which is used to train RotationNet. When the training process is finished, the trained network is built and is ready to predict classes – instruction steps – with 2D images as input. This is the real-world application of the before trained system. In the next section, the creation of the image database is explained in detail.

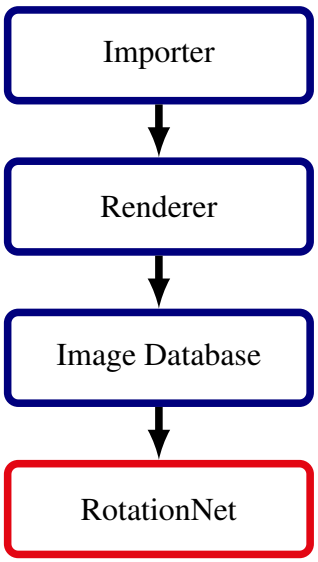


Figure 13.: Workflow of the import of the LDRAW model for the creation of the 2D images from different views. After the images are rendered, the resulting database is used as input to train RotationNet.

4.1. Framework Structure

The structure and organization of the framework is further explained. This is done to ease the search of configurations, rendered images or build files. In the GIT repository of this thesis, neither build files are included nor the CAFFE framework (Jia et al., 2014) as runnable configuration or an automated install script for all dependencies. In the following, the folder structure and the description of the content are listed:

build in the build folder, all files to build the system are stored. The build folder needs to be created on a new system. In the resources folder, a snapshot of RotationNet's *Caffe* CONFIGURATION is stored. The file name is CAFFE-ROTATIONNET2-MASTER.ZIP. This file has to be extracted to the build folder and then compiled. Previously installed all dependencies are mandatory (see in Section 4.3.1).

conf contains the configuration files for every training run of RotationNet. The files are training, validation and solver files in the PROTOTXT file format.

data contains all the generated data with a version connection: rendered 2D images are stored with the postfix *_full_test*, training output of RotationNet with the postfix *_train_data_white* and evaluation data for the prediction evaluation with the postfix *_npy_plots*.

framework contains all script files for various evaluations, helper scripts and deploy files. Mainly the scripts are used for the prediction part of the system (described in Section 4.3.3).

resources in this folder all resources which are needed to get a running RotationNet environment and the 3D model are stored. Furthermore, the script files for the rendering process to generate an image database (described in Section 4.2) are stored in the sub-folder *2018-04-16-batchrender*. The detailed organization is described in Section 4.2.3.

4.2. Image Database

The "Airport Rescue Vehicle" model of the LEGO Technic™ series is used as the reference model in this thesis. The complexity of the model is comparable with a complex model in industrial usage. Due to intellectual property protection regulations, the LEGO Technic™ digital model of comparable complexity is used instead of real CAD data. The real world model of the "Airport Rescue Vehicle" (no. 42068) consists of 1094 parts and measures 42 cm high, 45 cm long and 15 cm wide (see the fully assembled "Airport Rescue Vehicle" in Figure 14).

4.2.1. The Digital Model

The corresponding digital model of the "Airport Rescue Vehicle" (no. 42068) is generated by PHILIPPE HURBAIN. It is published under the license CCAL VERSION 2.0 at LDRAW.ORG.

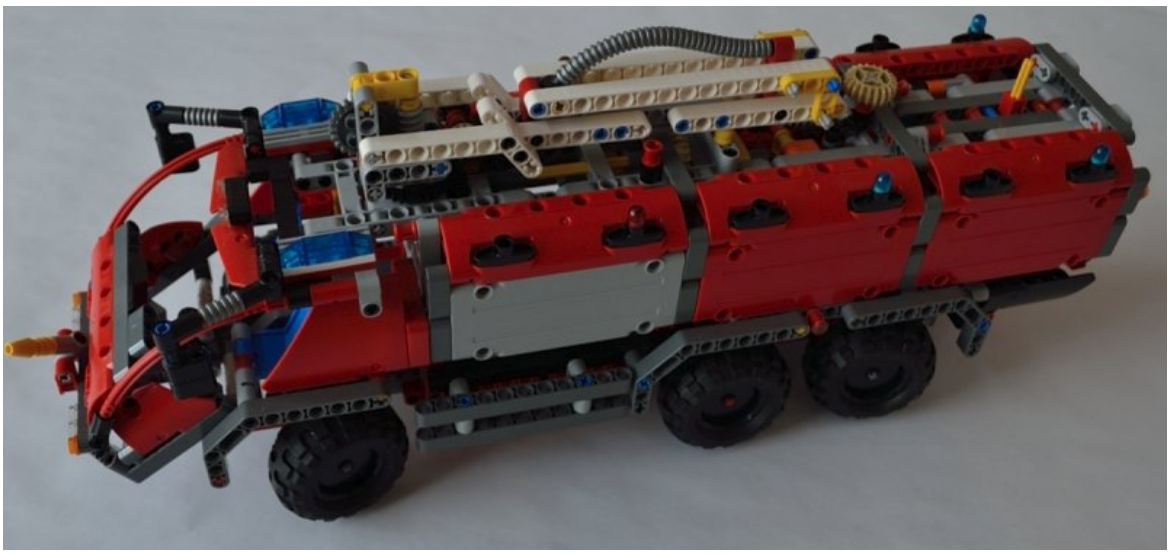


Figure 14.: The real world LEGO Technic™ “Airport Rescue Vehicle” (no. 42068) captured with a camera. This model is not available on the market anymore.

The model can be downloaded from OMR.LDRAW.ORG. Figure 15 shows a semi-transparent rendering of the model including its inner parts. According to the official publication website,

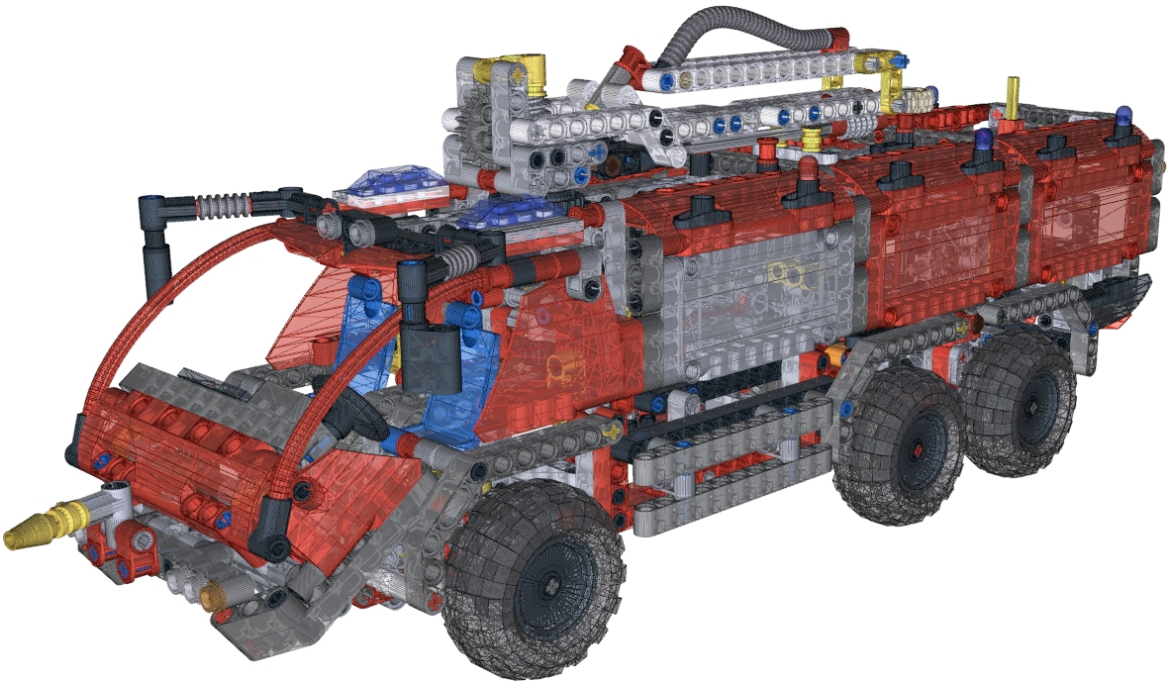


Figure 15.: The test set of the new assistance system is a LEGO Technic™ model consisting of 1094 parts that are assembled in 137 construction steps.

the model was first published on 28.12.2017 and contains all parts and patterns but does not contain any sticker drawings. The stickers of the real world model are omitted which does, however, have no influence on the method as such.

The digital model is stored in the LEGO model data format (LDRAW). The LDRAW data format stores metadata, the coordinate system, a unit factor, color information and line types. Within the line types, all the information of the bricks for rendering are stored. The line type contains a description of the geometric form, the coordinates and the color. One line type can be used as a reference for other parts in a database or for named parts within a file. This gives a throughout hierarchical and referenced linking data format organization. To correspond with the official offline instruction book of the LEGO model the LDRAW format supports the storage in an instruction step manner. Every instruction step includes uniquely necessary information of this specific step. The detailed and complete LDRAW format specification 1.02 can be found at LDRAW.ORG.

4.2.2. Importer

The LDRAW IMPORTER converts a LDRAW model into an "Alias Wavefront OBJ" model data. The detailed description of the LDRAW IMPORTER can be found in Section 2.5. In this section, the internal organization of the LDRAW format in instruction steps is explained in order to show the correspondence with the OBJ format named groups which are then used for further processing the instruction steps.

4.2.3. Renderer

Once an OBJ file of the "Airport Rescue Vehicle" model exists, it can be rendered with a rendering software. For this thesis, the free and open-source software Blender^{TM} is used in the version 2.79. The OBJ file of the "Airport Rescue Vehicle" model is imported into

Blender^{TM} and positioned at the origin. The named groups are imported as meshed grids and the camera positions for the rendering process are stored in a python script named *batchrender_blend.py*. This python script configures the camera position views, rendering options and the parallelization of the rendering itself.

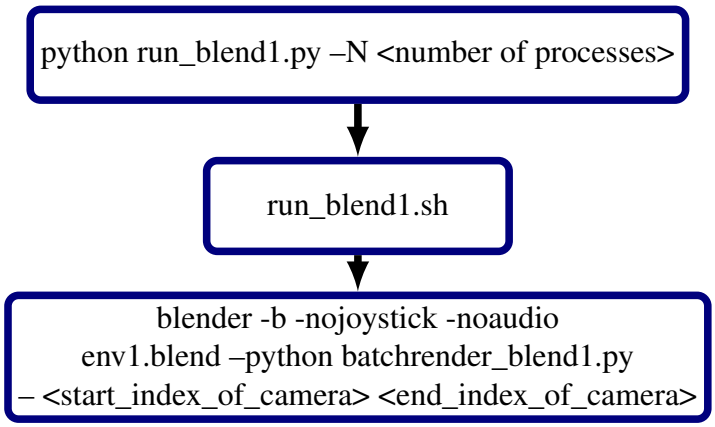


Figure 16.: Flowchart of the rendering process with Blender^{TM}. As input file, a previously converted blend file with the imported "Alias Wavefront OBJ" model is used.

In Figure 16 the flowchart with all included scripts, arguments and dependencies of the rendering process are shown. The named scripts are stored in the GIT repository in the directory *./resources/2018-04-16-batchrender/*.

4.2.3.1. Configurations

The python script *batchrender_blend1.py* stores the coordinates of the used camera positions for rendering all instruction steps. This is a list of the x, y, z-coordinates of the position of the cameras. Every x, y, z combination describes one camera. The view direction of every camera is automatically adapted to the origin. The size of the camera positions is 20 because it is distributed over a dodecahedron on a unit sphere, which is explained in detail in Section 3.1.1.2. To get an optimal zoom factor for the rendering, the unit circle generated coordinates are scaled with a fixed scaling factor of 2.5. With this scaling factor, the whole model is fully rendered without any parts missing in all instruction steps seen from all views.

Furthermore, the space of the image is ideally positioned within the limitations of the image size, to regard a high and confident information detail level.

4.2.3.2. Render Engine

Blender™ supports different render engines. One of them is the CYCLES renderer CYCLES-RENDERER.ORG which is developed by the Blender™ project. CYCLES is configured to use global illumination and GPU support for the rendering process. The output image is set to a resolution of 512×512 pixels with 50 samples and a clamping of 0.01. The clamping option influences the existence of fireflies – noise – in the rendered image. For Blender™, a suggestion for eliminating fireflies is given on QUORA.COM: the aspect ratio and the resolution is set to 1 and 100. This render process uses 8 threads and the background is set to white. No shadows or other local illumination is added. The RGB color format is used for the color representation.

4.2.3.3. File Name Specification

The file name of rendered 2D images follows the rules of placeholders, which are explained in detail in Table 3.

Table 3.: The file name specification has six placeholders. Every placeholder has a specific meaning (column two) and with this information a file can be immediately identified by its file name. The third column gives an example of a real image file. The placeholders are all concatenated with an underline ('_'), except for the last one which is represented by a dot ('.').

Placeholder	Description	Example
<filename>	file name string	render_FireCar_blend1
<step number>	number from 100001 or 'ldraw'	100001
<camera pose number>	number of camera position	000
<pre-steps used>	'A' (no pre-steps) or 'M' (pre-steps)	M
<color channel>	color channel as string	RGB
<file format>	file format as string	png

The specification (see in Table 3) of the file parts is combined to a complete string. The specification and an example of the complete, concatenated file name string is shown in Table 4.

Table 4.: Specification of the complete string of a 2D image file name. The first is the concatenated string with placeholders and the second is an example. This example is named *render_FireCar_blend1*, step 1, camera position 0, with pre-steps, RGB as color channel and the file format is png.

Specification:
<filename>_<step number>_<camera pose number>_<pre-steps used>_<color channel> .<file format>
Example:
render_FireCar_blend1_100001_000_M_RGB.png

As <file name>, the OBJ file name of the imported model is used. The <step number> is the instruction step number starting with 100001 or “ldraw” for the step 0. The <camera pose number> is the index of the camera position in the list of defined cameras in the render script. The <pre-steps used> symbolizes with the parameter *A* that no pre-steps are rendered and with the parameter *M* that previous steps exist in the image. Finally, the <color channel> string describes the used color channel, e.g., RGB or HSV. As file format, the png format is always used.

4.2.3.4. Sequence

After these configurations, all camera view positions are scaled with the scaling factor of 2.5. Then, all the views are iterated. In this iteration, the camera is selected and the direction of where it is looking is calculated. After that, every mesh group is hidden for rendering in order to start from the first step. Two iterative sub steps are then made. First, every mesh group is rendered without all pre-groups. This results in a 2D image which has only the actual instruction step information stored. Second, every mesh group is rendered with all pre-groups. This results in the iterative representation of the reassembled model. For this thesis, the images which are rendered for the second time are used.

4.2.3.5. Parallelization

For the rendering process some optimizations and modifications are made in the context of parallelization. The rendering is a time-consuming process, which profits from parallelization on a multi-CPU or HPC system. Two parallelization options are handled. The first is the parallelization of rendering one image. The second is the parallelization of rendering several images. This brings an advantage on big multi-CPU systems or clusters.

Blender^{TM} supports parallelization for the rendering of one image. In detail, the image is partitioned. The partitions are rendered separately each in one thread and combined at the end to get a complete image of the rendering. For this, eight threads/partitions are used.

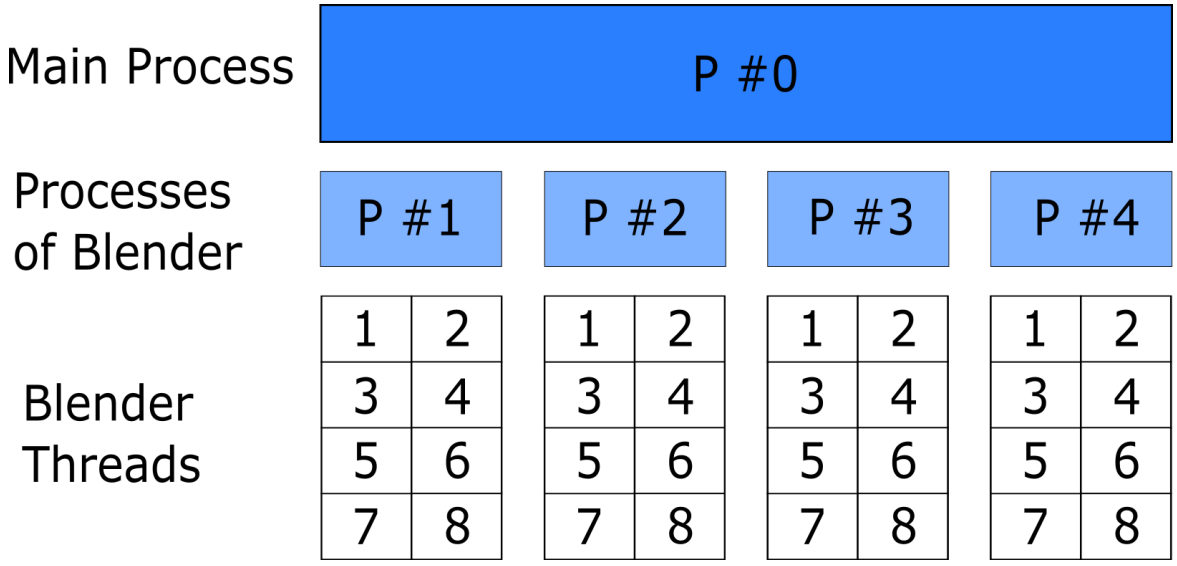


Figure 17.: Illustration of the parallelization of rendering with Blender^{TM}. The main process is started with the python call of run_blend1.py. $N = 4$ in this case, which is equivalent to 4 Blender^{TM} processes. Internally, Blender^{TM} renders one image with 8 threads. Summarized, 32 CPU cores are created to render the whole database. The main process waits until all child processes are finished.

For the second parallelization a workaround is implemented because Blender^{TM} has no option for this use-case. Blender^{TM} has the limitation that only one rendering per instance is possible. A first approach is to use the python parallelization framework to start more than one rendering process at a time, but this is not easily possible with Blender^{TM}. The

second option is to start separate Blender^{TM} instances simultaneously, where every instance renders a part of the camera positions. All instruction steps are sub-parts of camera positions. This is implemented with the python script *run_blend1.py* and the shell script *run_blend1.sh*. *run_blend1.py* uses the python parallelization framework and partitions the camera view list equally. For every equal part the shell script *run_blend1.sh* is called up with all mandatory arguments. Every instance runs on a specific part and in the end all 2D images are rendered successfully.

The python script takes as argument the number of process pools (–N), which have to be created. In every process pool one Blender^{TM} instance is started. The number of pools must be a factor of the size of camera positions which results in an integer value, e.g., if the camera position is $N_{view} = 20$, N must be an even number. If this requirement meets N pools, N instances of Blender^{TM}, are started with the help of the shell script *run_blend1.sh*. Three non-optional arguments are needed by the start script:

1. start index in the camera list,

2. end index in the camera list,

3. import an OBJ file or load a blend file.

4.2.3.6. Output

The first two arguments (1) and (2) as described above are representing the index of the camera view list. The argument (3) is a boolean value which controls how the model is stored. As the best practice, the usage of a blend file, which is the Blender^{TM} file format, is used for faster proceeding. The reason for this is that the blend file is ready much faster compared to an OBJ file. First the OBJ file must be imported to get the internal Blender^{TM} structure. In comparison, in a blend file the internal mesh structure is already available. For this thesis, the blend file was created manually.

When the rendering is finished, $N_{images} = 2740$ 2D images are rendered from $N_{steps} = 137$ instruction steps seen from $N_{views} = 20$ views. These 2D images represent the image database to train RotationNet. The HPC system, which is used for this thesis, runs two days to finish the training of the complete image database.

4.3. RotationNet

In this section, the implementation details of RotationNet are explained. The major two parts of RotationNet are the training process and the prediction process. RotationNet is important for instruction step prediction and is a tool for image classification. The prediction process is what is earlier explained as the *online* part of the system. Before the classification can be run on different devices, e.g., a Microsoft Hololens™, RotationNet needs to be trained. The training is the last step in the *offline* part of the system. The RotationNet implementation published by (Kanezaki et al., 2018) is found on GITHUB.COM. This implementation uses the CAFFE framework by (Jia et al., 2014), which is supported to run on CPU or GPU.

The clearly arranged folder structure is important for finding the right script for the specific use-case. Furthermore, the script references to data files or config files, which are referenced relatively to the scripts. For a working environment the structure should be left untouched. An overview of the folders can be found in Section 4.1.

4.3.1. Prerequisites

The most important prerequisite for a running RotationNet environment is the CAFFE framework. CAFFE comes with some basic and helpful scripts for converting the dataset and scripts for analyzing the training process log files. It is written in C++ and supports a PYTHON INTERFACE for the python version *2.7* and *3.3+* with numpy ($\geq$ *1.7*). Further prerequisites are one “Basic Linear Algebra Subprograms” (BLAS) out of the following three implemen-

tations: ATLAS, MKL BY INTEL® and OPENBLAS. For this thesis, the chosen library is OPENBLAS. Other prerequisites are *protobuf*, *glog*, *gflags*, *hdf5*, BOOST, OPENCV, *lmdb* and *leveldb*. For this thesis, no GPU support is needed and so it is neither installed nor configured. All the processings are running on CPU.

With these libraries installed, the CAFFE framework can be compiled. On GITHUB.COM the complete configuration of RotationNet, which works with the CAFFE framework, is published. Two parts have to be built: the basic framework (*make*) and the python interface (*make pycaffe*) one after the other.

4.3.2. Training

Originally, RotationNet uses the ImageNet Large Scale Visual Recognition Competition (ILSVRC) from 2012 (Russakovsky et al., 2015) dataset for the training and fine-tunes the network with the database of ModelNet10 or ModelNet40. These datasets have not much in common with the instruction step based assistance system used for this thesis. In Section 4.2, the database created for this thesis is described. Here, the configurations of RotationNet are modified to fit the created database.

For the configuration of training the CAFFE framework three different files are used:

1. a solver file, which refers to
2. a train file, and
3. a validation file.

All the files are text files with the file extension PROTOTXT. The solver file (1) is the main configuration file entry to train a network with CAFFE. This file refers to the train file (2) and the validation file (3). Furthermore, training, i.e the momentum, snapshot option, weight decay, and validation (test iteration), test interval, parameters and snapshot options

Listing 4.2: Configuration changes of a train file to fit to train the 2D images of the “Airport Rescue Vehicle” model.

```
1
2  layers {
3  ...
4  image_data_param {
5  source: "../../data/train.txt"
6  batch_size: 40
7  ...
8  }
9  }
10
11 layers {
12 bottom: "fc7"
13 ...
14 inner_product_param {
15 num_output: 2760
16 ...
17 }
18 }
19 layers {
20 ...
21 my_softmax_loss_param {
22 stride: 138
23 using_upright: false
24 }
25 }
```

are defined in this file. In Listing 4.1, a used solver file for the training task of RotationNet for the assistance system is listed.

Listing 4.1: An example configuration of a solver file. This configuration is used for the evaluation of the method.

```
1  train_net: "../../conf/rotationnet/Training/rotationnet_arcs_case2_train_061.prototxt"
2  #test_net: "../../build/rotationnet-master/Training/rotationnet_arcs_case2_val.prototxt"
3  #test_iter: 40
4  #test_interval: 1000
5  base_lr: 0.0005
6  lr_policy: "step"
7  gamma: 0.1
8  stepsize: 20000
9  display: 20
10 max_iter: 40000
11 momentum: 0.9
12 weight_decay: 0.0005
13 snapshot: 1000
14 snapshot_prefix: "rotationnet_arcs_case2_0_6_1"
15 solver_mode: CPU
```

All configuration parameters are the same as published by RotationNet with modifications of the train file and disabling of the validation file. The snapshot prefix has the version number appended, i.e. *_0_6_1*, to the base name *rotationnet_arcs_case2*. The string is not fixed and the specification is for better tracking and identification. Before the training can be started, the train file must be modified. These modifications are listed in Listing 4.2. First of all, the

last layer of the network, the output layer, is modified. The output parameters are changed to 2760 for the second last layer and the stride parameter to 138 for the last soft-max-layer. Detailed explanation of the values are described in Section 3.2. All other parameters are left untouched, except for the optional change of the batch size parameter of the input layer.

The batch size of the input layer influences the processing time on the running system. If the system runs on a CPU based system, the batch size should be lower than the size of CPU cores on a system. In other words, the CPU cores limit the batch. On a GPU the memory is limiting the batch size, due to the different structure compared to a CPU. With a batch size too high for a specific system, the processing time increases rapidly due to scheduling interruptions. The training process for the "Airport Rescue Vehicle" with the image database and a batch size of 40 takes about two days running in CPU mode using a HPC with 54 Intel® Xeon® CPU cores at 2.60GHz.

Another important parameter of the input layer is the source option. In Listing 4.2, the source file is *../../data/train.txt*. This file has specified a mapping between the training images file name and the class of the image. This is needed for the optimization of the parameters of the network in the training process. In Table 5, the mapping specification and an example is shown. It is important to use the *train.txt* and *val.txt* appropriately with the right class and

Table 5.: The internal structure of a *train.txt* file. There are two placeholders per line in the file, the image path (<path>) and the corresponding training class number (<class>). An example for one line is also given. This structure is the same as for a *val.txt* file.

Placeholder	Description	Example
<path>	file name string	render_FireCar_blend1_step_100001_000_M_RGB.png
<class>	number of class	0
Specification of one line: <path> <class>		
Example of one line: render_FireCar_blend1_step_100001_000_M_RGB.png 0		

paths over the whole input dataset. Any misconfiguration is difficult to find which results in a poorly and faulty trained network.

4.3.3. Prediction

The prediction depends on and corresponds to the configurations made in the training part. CAFFE uses another configuration file for this, the deploy file. The deploy file can be generated out of the training file or the file from RotationNet can be modified. The last two layers must be changed like in the training step to 2760 output parameters for the second last layer and 138 to the stride parameter of the last layer. With these modifications to the configuration of the CAFFE framework, predictions of images of the assistance system can be made.

Furthermore, the output data of such a prediction of the network is further processed. For this, RotationNet's framework and script files are modified. In its default configuration RotationNet uses the output data of the network and calculates the scores of all classes and views by performing a probability maximization (see Section 3.1.3). The outcome is that the prediction of the class and view is displayed as a string. The output of the framework is implemented in the python script file *save_scores.py*. The function depends on mandatory and optional arguments. The arguments are listed in Table 6. The source code is changed to support a *mean_file* in the BINARYPROTO file format, which is the default CAFFE format for mean files. The mean file stores the mean of all images in the training dataset. The BINARYPROTO file is converted to the NUMPY format, which is internally used by all python scripts. This conversion of BINARYPROTO to NUMPY is low high-performance because it is running on the same file on every script call. Therefore, the optimization is as follows: for the converting of the mean file the script file *convert_binaryproto_to_npy.py* is created, which takes as input the BINARYPROTO file and outputs the NUMPY file. For that, the optimization is as follows: the mean file is pre-converted with the script *convert_binaryproto_to_npy.py*, which takes as input a BINARYPROTO file and outputs a NUMPY file. With this pre-step, the NUMPY file is loaded directly in the right file format for further proceeding with python.

The output NUMPY file stores all output parameters of the prediction task of RotationNet. The data structure for one image is the probability of every instruction step and every view

Table 6.: Arguments in the usage of *save_scores.py*.

Argument	Description
input_file	Text file of images to predict
output_file	Prediction output as npy
mean_file	Mean of the training files
model_def	Deploy file of the network structure
pretrained_model	Trained model by RotationNet
gpu	GPU or CPU mode
center_only	Prediction of center crop only; not averaging prediction over crops

plus the indirect view per instruction step. If more than one image is predicted by the network, the data structure extends to a multi-dimensional array with the dimensions of $N_{images} \cdot N_{instructionsteps}$. In other words, for every input image a prediction for all instruction steps and views, including the incorrect view per instruction step, is stored. This multi-dimensional array is then used to calculate the probability maximization over the data.

RotationNet deploys the file *classify_npyfile_case2_all_views.py* for the probability maximization. The source is changed to use it directly in-line in the script *run_all_tests.py*. The functionality of the prediction is the same. First, the score NPY file and the class names are loaded. Then the scores are normalized to the *incorrect* view value and the *incorrect* view probability is set to zero for every instruction step. After that, the probability maximization is calculated over all images. Finally, the class with the highest probability is returned and the accuracy is calculated. The last point is extended to get more detailed statistics of the probability maximization. The instruction step with the highest probability is returned, furthermore the difference to the ground truth class and the Top-5 classes with the five highest probabilities (see in Section 5.2.4). This probability maximization is limited to the view position distribution on a dodecahedron (described in Section 3.1.1.2). For any other position distribution, the script file must be changed and extended.

5. Evaluation

In this Chapter the evaluation and its corresponding results are described. The complete pipeline (see Figure 13 in Chapter 4) of the training and classification is evaluated and the results are shown in this chapter. Every modification affects mostly the complete workflow, except the *Importer* step. As first step, the image rendering output is evaluated to get a good image database. The trained network is tested with three different cases of images captured from: training positions, unknown positions and real data. Finally, the accuracy rates are compared.

5.1. Image Rendering

The first step is to evaluate the rendered 2D images before a training process of the system makes sense to be performed. The main focus of the evaluation is on the parameters of the rendering process, the camera positions and the parallelization. These parameters are modified and optimized. The earliest test runs are made to optimize the scale factor and to remove the undesired noise – the occurrence of fireflies – in the 2D images. The camera positions are normalized to the unit sphere and the scale factor must be adjusted so that the complete model fits into the image for every instruction step (see Section 3.1.1.2). After the first complete run, for this thesis the processing time of rendering all 2D images was not satisfying and so the process needs further improvement.

The camera positions in the earliest renderings were not distributed on dodecahedron as explained in Section 3.1.1.2. Every position differ from the dodecahedron camera positions. However, the image database of this rendered 2D images were not discarded. They are used for the unknown positions evaluation in Section 5.2.2. In a further step, the background color was changed from black to white to adapt to a more ideal real-life connection.

5.1.1. Results of Rendering

The rendering process shows the occurrence of fireflies in all 2D images of a complete set of rendered images. In this context, a complete set includes all instruction steps and all camera views. These fireflies can be seen in Figure 18 in the center of the object. The background is black and the used color space is RGB. Another challenge was to tackle the optimal scaling

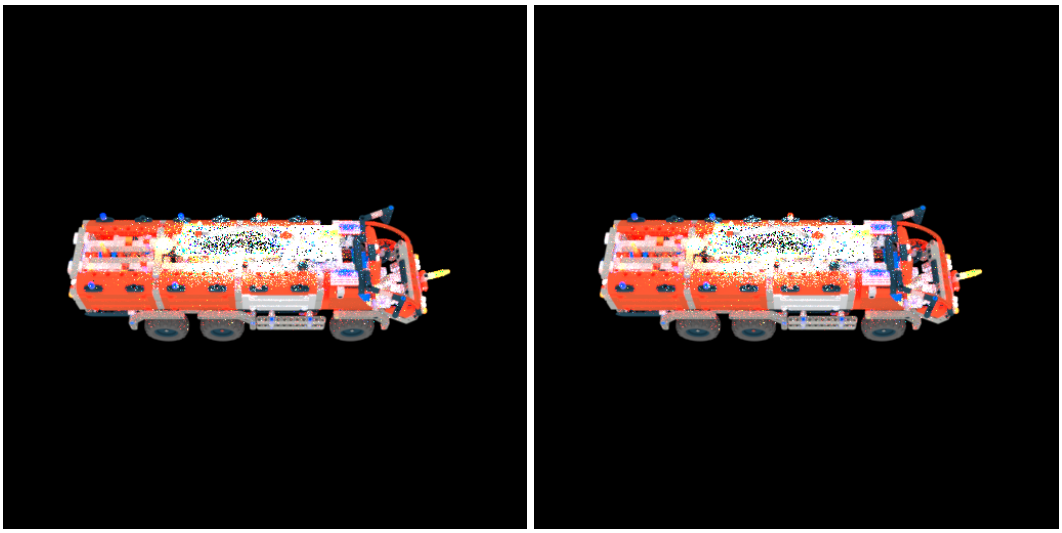


Figure 18.: Renderings with fireflies in the middle of the images and black background. The images are rendered to represent the instruction step 136 and 137 from the same view position.

factor to get the same amount of important information in one image. This is a maximization of the model to fit into the image bounds without any cropping over a complete dataset. In Figure 19, scaling was not adequate for all views. This means that the distance between camera and model is too short and the model is not fully rendered and subsequently cropped.

After tackling all the challenges for the *Renderer* part, a base database, used as training images for further evaluations with RotationNet, exists. A sub-set of rendered images of the database is shown in Figure 20.

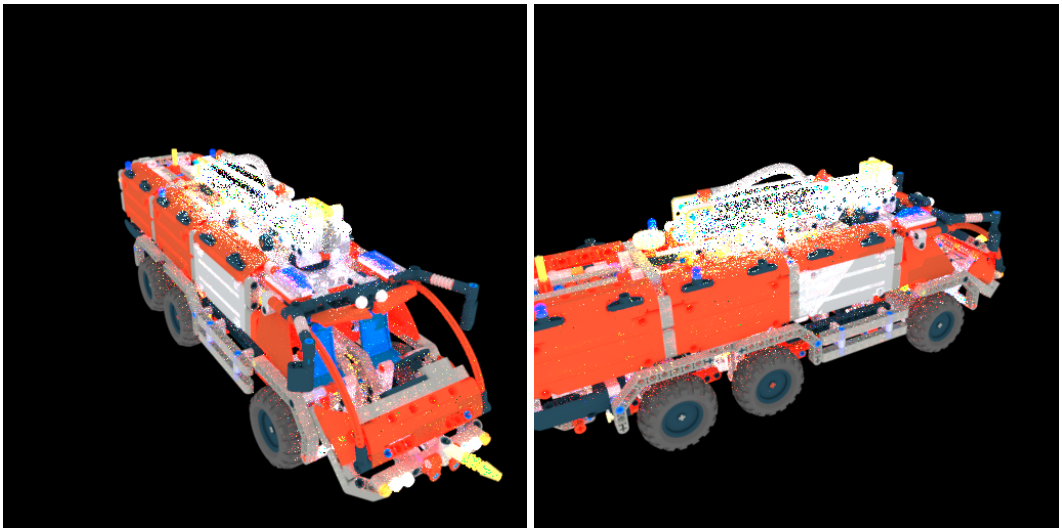


Figure 19.: The left image shows the model completely rendered into the image bounds. The right image is cropped at the left border which results in an information loss. The instruction step of the two images is 137 with different view positions.

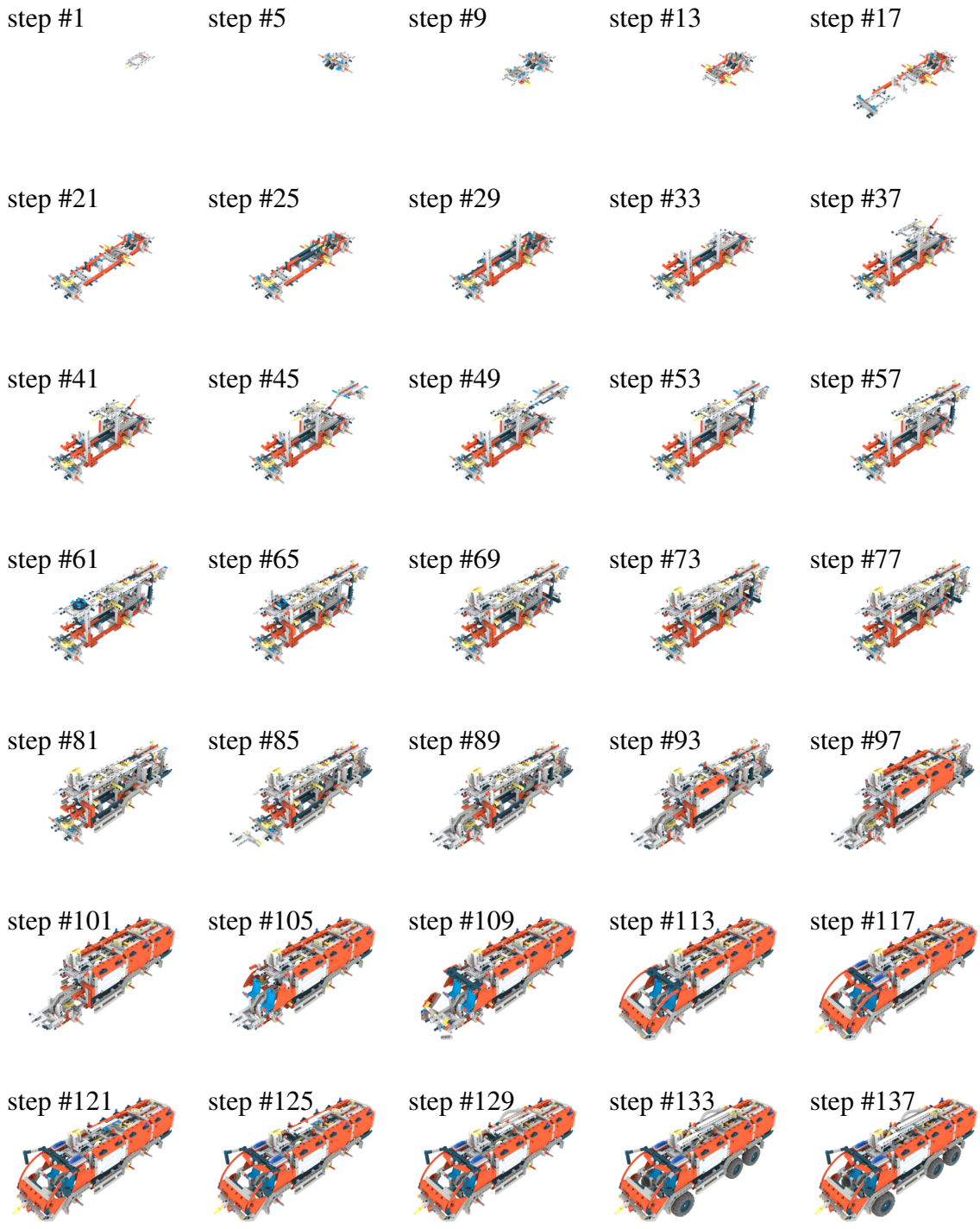


Figure 20.: The construction of the “Airport Rescue Vehicle” comprehends 137 instruction steps. This overview shows the result after every fourth instruction step. All views are rendered with the same camera perspective. Differences between individual steps are not always apparent from all views.

5.2. Training and Classification Results

Three different test series have been performed in order to evaluate the applicability of the modified version of RotationNet on the "Airport Rescue Vehicle" model. The task series is done to find out how well RotationNet performs in correctly identifying the construction steps for the "Airport Rescue Vehicle". The test series return a Top-1 as well as a Top-5 accuracy rate which is displayed in Table 7.

5.2.1. Training Positions

In the first test series, all the images that have already been used to train the system are reused to test the system:

$$Set_{Test} = Set_{Training} \tag{5.1}$$

In detail, for each class (137 in total) the 20 images of pre-defined camera positions distributed over a dodecahedron sphere (see Figure 12) are used, and the system returns the correct class (main objective) and the correct pose (secondary objective).

5.2.2. Unknown Positions

The second test series uses the same CAD model with new camera positions. A set of 24 view positions, which are not included in the training set, are distributed equally on a sphere as suggested in "Simple and Efficient Normal Encoding with Error Bounds" by (Schinko et al., 2011). In other words, the test set is disjunct to the training set:

$$Set_{Training} \cap Set_{Test} = \varnothing \tag{5.2}$$

5.2.3. Real Images

The final test series consists of real-life captured images. Using a video camera, a sequence of images is captured for one instruction step and the sequential frames are used as input for the trained network. Each image is converted to RGB color space, cropped to aspect ratio $1:1$, and re-sampled to 256×256 pixels to meet the requirements of the system for input images. Figure 21 shows an example frame, which has been extracted from a video at an early stage of the assembly.

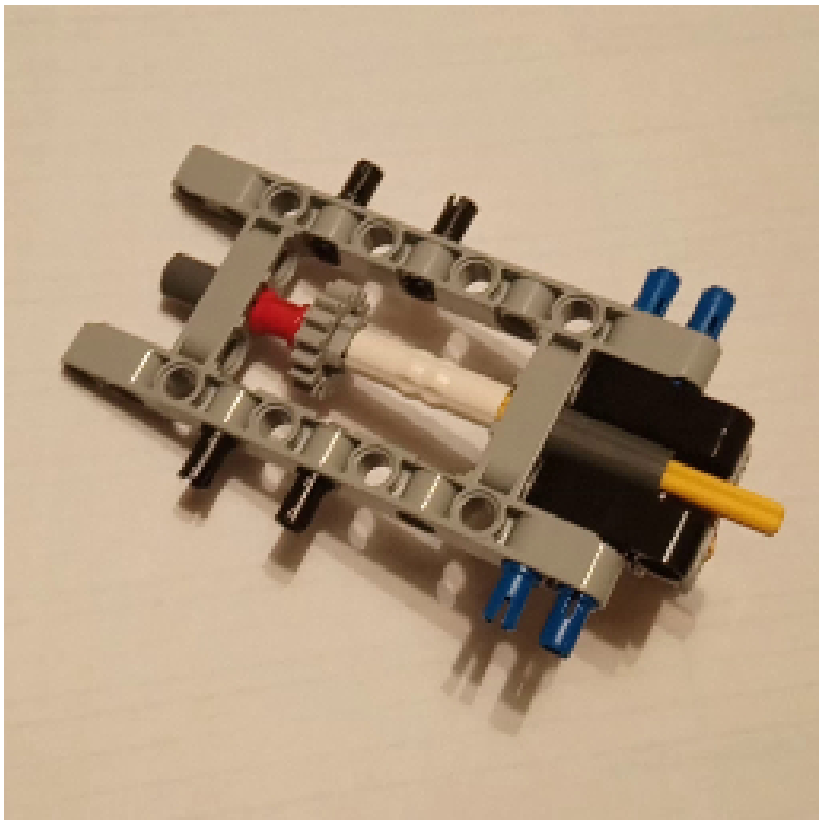


Figure 21.: The assembly of the "Airport Rescue Vehicle" has been recorded at different stages. The extracted frame (cropped and scaled to 256×256 pixels to meet the requirements for input images) shows the CAD model at an early stage.

5.2.4. Results

For every experiment, the success rates for the correctly identified construction step is referred to as Top-1; the correct construction step within the classification set with the five highest probabilities is referred to as Top-5. The results of the experiments are listed in Table 7.

Table 7.: Top-1 and Top-5 accuracy rates of the three test series (1) with rendered images already used for training, (2) with newly rendered images exclusively used for testing purposes, and (3) with real images captured using a video camera.

Test Set	Top-1 Accuracy	Top-5 Accuracy
Training Positions (see Section 5.2.1)	8.03%	27.74%
Unknown Positions (see Section 5.2.2)	4.38%	7.30%
Real Images (see Section 5.2.3)	0.73%	0.73%

The Top-1 success rate of the test series using images already used to train the system (described in Section 5.2.1) is 8.03%. The Top-5 success rate of this test series is 27.74%, which is a rise compared to the Top-1 rate by a factor of 3.45. In Figure 22 the divergence between the Top-5 error and the ground truth of the instruction steps. If the test series comprehends newly rendered images exclusively used for testing purposes with unknown positions (see Section 5.2.2) the success rates drop to 4.38% for Top-1 and to 7.30% for Top-5, respectively. In the real-world scenario, the success rates drop to 0.73% in both categories, Top-1 and Top-5. For these results, the image rendering optimizations are vital for the test sets of training positions (unknown positions). For real images, the results are more of a random process because one class out of 137 is predicted for different sequences. This means that modified RotationNet has a better performance when the images have already been trained by the system. It performs worse when the camera positions are unknown. Real images have comparatively the lowest accuracy, both in Top-1 and Top-5 accuracy rates. The overall result shows a fairly low accuracy rate. For this reason, a survey is additionally performed (see Chapter 6).

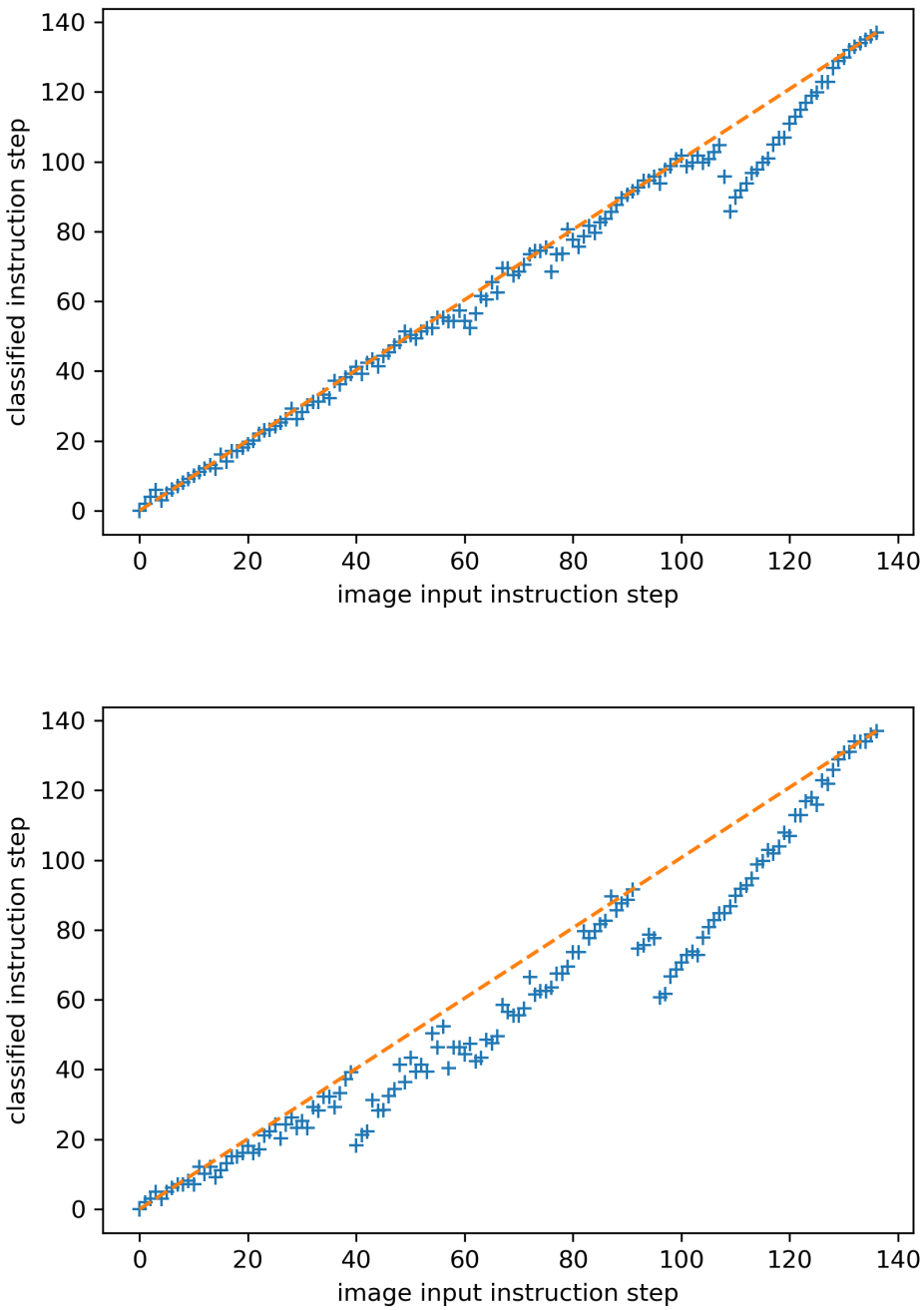


Figure 22.: Difference between the ideal line (orange) and the Top-5 error rate of the training positions (top plot) and the unknown positions (bottom plot). In some clusters the Top-5 error rate is bad and far away from the ground truth instruction step.

6. Survey

A survey is conducted on order to compare the results of RotationNet with the performance of humans. First, the questionnaire and the derivation of the hypotheses are described and suggested. After that, the methodology of the evaluation is clarified. Finally, the results of the questionnaire are statistically evaluated and discussed.

6.1. Questionnaire and Derivation of Hypothesis

The main target of this survey is to evaluate if the prediction probability of RotationNet differs from the cognitive performance of humans to predict the correct instruction steps. Since the success rate of RotationNet to predict the correct instruction steps is without a comparable reference, this survey has been added. The original plan was to only use RotationNet to predict the instruction steps with an acceptable success rate. Since this was not the case, further investigations on other influencing factors is required. It is essential to compare the cognitive performance of humans, which is highly flexible, with RotationNet, which is pre-trained and tested with the 3D model at hand.

6.1.1. Interpreting the Questionnaire and Hypothesis

The questionnaire deals with the question of how the performance of humans and RotationNet differ in the prediction of instruction steps. The special recognition and analysis of the models from different view points and small geometrical shape details is of mandatory significance.

In almost all the literature of the research field of CNNs, the comparison is made with different benchmarks, e.g., ImageNet or ModelNet, in a virtual environment and without a connection to a real world application. In the literature most real world applications

are special cases without a general evidence. Therefore, the classification success rate of humans and RotationNet is probed to get a connection to the real world usage. The following hypotheses are suggested:

Hypothesis 0 (H0): *The prediction probability of instruction steps of the "Airport Rescue Vehicle" (no. 42068) model does not differ between humans and RotationNet.*

Hypothesis 1 (H1): *The prediction probability of instruction steps of the "Airport Rescue Vehicle" (no. 42068) model differs between humans and RotationNet.*

6.2. Method of Evaluation

6.2.1. Sample Description

Altogether $N = 69$ attendees took part in the survey. Three test persons did not fill out the questionnaire completely, the questionnaire was not returned or the answered numbers were out of range of the possible answers. These participants were excluded. The adapted sample size is $N = 66$. As a consequence, the return rate is $r = 95.7\%$.

The original results of RotationNet are used as models for the survey. The survey consists of two parts. On the one hand, there are images which have been used in the training process. These images are all rendered in the same position. On the other hand, unknown images – images rendered in different positions than the training images – are used as the questionnaire.

The resulting data is evaluated with the help of python and the two python libraries numpy and scipy. The clearance of data is made after the end of the survey where not completed or not returned questionnaires are marked as "missing" and the values are deleted from the dataset.

6.2.2. Set-up

The questionnaire includes an oral explanation, one A4 answer sheet and two A4 reference sheets. Demographic data is not requested. The evidence of the attendance is evaluated, which has no central importance for calculating the results. The completed survey reference and answer sheets can be consulted in the Appendices A and B.

6.2.3. Instructions

All instructions are given orally at the beginning of the survey. The attendees gave their consent to the note of anonymity, confidentiality and voluntariness. Relevant information for filling out the answer sheet appropriately was given. A numerical classification of the 16 images on the answer sheet to the 40 images on the two reference sheets had to be made. The instructor was answering the attendees any questions or ambiguities about the questionnaire without giving any help for the numerical classification.

6.2.4. Questionnaire

All participants were asked to find the best correspondence between the 16 question images on the answer sheet to the 40 construction images on the reference sheets. The images were printed in high resolution and in color. The reference construction images were numbered and ordered increasingly by the sequence of instruction steps. Furthermore, the view position was fixed to one specific position for all reference images. The 16 images on the answer sheet were randomly ordered and displayed in different view positions. A single choice had to be made for each question, whereas the same answer could be given multiple times on different questions. There has been no time limit to answer the questionnaire. The main task was to find the best fit in the eyes of the participant in this survey.

6.2.5. Test design and material

This survey was executed as a paper and pencil test. It is a field study with a convenience sampling. The survey was completed by employees and members of the Institute of Computer Graphics and Knowledge Visualisationof Graz University of Technology (here referred to as *institute*) as well as by master students and friends.

The present survey is conducted in order to deal with the comparison of the cognitive performance of humans and of the results of the experiments of RotationNet. As a predictor, *human* and *machine* is used.

Random mapping of the attendee to the conditions was not possible due to the research question of *human-machine*. It is rather recorded what happens in reality in comparison to the computer system. Therefore, a causal theoretical statement is only possible under limitations. The ideal case is, however, empirical evidence to proof the causal hypothesis.

6.2.6. Survey execution

The survey took place from 05.06.2019 to 05.09.2019. The attendees were asked to participate anonymously in the survey.

The group of master students were asked to take part in the survey during their lecture hours. Alike, members of the "Institute of Computer Graphics and Knowledge Visualisation" and friends were asked to participate.

The answering of the questionnaire took approximately 25 minutes on average. No attendee had to ask any question. Due to this fact, it can be said that the instructions were clear and no ambiguities came up.

6.3. Statistical Evaluation

6.3.1. Descriptive Statistics

The sample size of the survey is $N = 66$, where the attendees are master students, members of the institute and friends. The frequency scale of the attendees is shown in Table 8.

Table 8.: Acquisition distribution of the test persons. The table shows the absolute value and the corresponding percentage value in relation to the total value.

Test person	absolute value	relative value [%]
Master students	29	43.9
Institute members	12	18.2
Friends	25	37.9
Total (N)	66	100.0

The sample consists of $N = 29$ (43.9%) master students, $N = 12$ (18.2%) members of the institute and $N = 25$ (37.9%) friends. The hypothesis of Section 6.1.1 is evaluated and calculated with the method of (Oliphant, 2006). The significance value is stated as follows: values $p \leq .01$ are highly significant and values $p \leq .05$ are significant.

6.3.2. Hypothesis Results

The hypothesis poses the question whether a significant difference between the cognitive performance of humans and the performance of RotationNet in the prediction of instruction steps exists. The result is that it shows no significant difference between the two. However, with the trends a result can be interpreted as follows: the results of the experiments and the survey are compared to each other. The results of the configurations described in Section 5.2.1 and in Section 5.2.2 are referred to as *Training Positions* and *Unknown Positions*, respectively; the survey results are referred to as *Survey*.

All results are listed in Table 9. In addition to Table 9, the results are also visualized in a Box-and-Whisker plot in Figure 23. Using real images, the modified RotationNet system

Table 9.: This overview lists the results of the modified RotationNet system compared to human persons as assessed in the survey. It shows the error distribution measured as deviation between the correct instruction step and the estimated instruction step by the system, resp. the guessed instruction step in the survey.

Test Set	Minimum	First Quartile	Median	Third Quartile	Maximum	Mean	Standard Deviation
Training Positions (test size $n = 2740$)	0	0	1	4	24	3.219	4.828
Unknown Positions (test size $n = 2740$)	0	3	8	16	36	10.175	8.875
Survey (test size $n = 1056$)	0	0	1	5	56	4.329	7.108

has an accuracy on the scale of a random process. Using rendered images, the accuracy improves significantly (see Table 7).

As a consequence, only tests with rendered images are further analyzed: testing the system with images already used during the training phase, the system achieves an error of 3.219 on average; i.e. the prediction of the instruction step is on average 3.219 steps off. In the case of images which are unknown to the trained system, the error rises to a difference of 10.175 steps on average. The average human error measured by the survey is 4.329 steps; i.e. the human performance is clearly better than the system with untrained images. With trained images, the system is slightly better. However, the improvement is not significant: since both data sets do not follow a normal distribution and no common distribution can be assumed either (according to Kolmogorov–Smirnov tests), the determination of the confidence intervals of the expected values is according to (Oliphant, 2006). The confidence intervals of the expected errors remain disjunct up to $p = 0.081$ – a value usually considered to be non-significant.

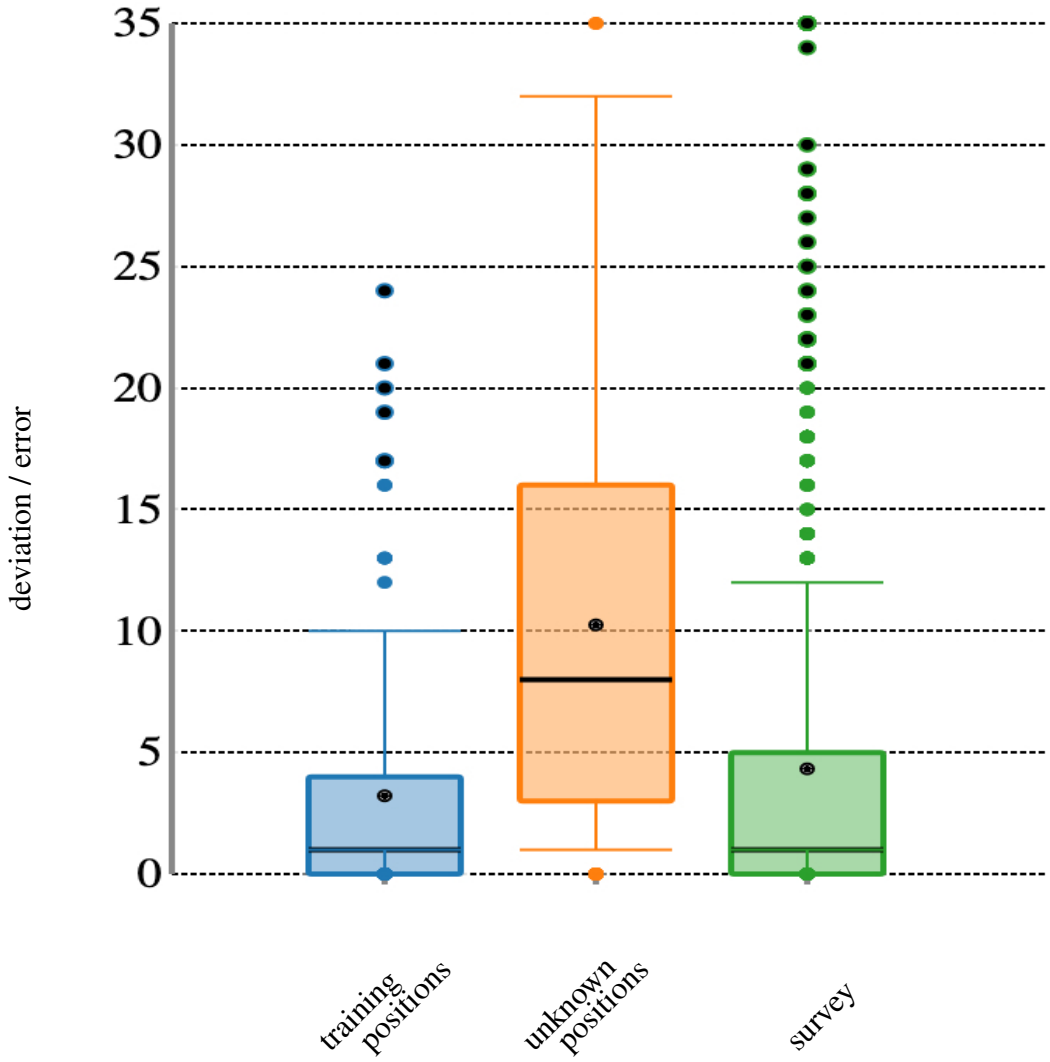


Figure 23.: The error distribution as listed in Table 9. The training, unknown positions and the survey data with its clustered outliers, median, and quartiles are shown. The survey has the most outliers compared to the others.

7. Conclusion and Outlook

The aim of this thesis is to test RotationNet in a practical application in order to find out its real-life usability. For this use-case, a database of an instruction step-based 3D CAD model is generated and used to train the instructions using RotationNet. The goal is to predict the current instruction step based on a simple image of the current assembly stage. As a second means for the test, the prediction success rate of humans is tested in a survey. The combined interpretation of machine learning results and survey results reveals limited applicability of RotationNet for real-life purposes.

The first Chapter handles the theoretical surroundings of the research field. The relevant information is explained which gives a deeper understanding of the whole research field of image classification. The semantics and representation of images is described before coming to the important part of image classification which is done in the context of deep learning. Convolutional Neural Networks (CNN) are the state-of-the-art methods of image classification. These are examined in detail because they are the basis of the method used for this thesis, which is RotationNet. In the second Chapter, RotationNet is analyzed in detail. The Chapter handles the positioning of the viewpoints, pose estimation, the training and the prediction phase in this method. It gives an outlook on the next Chapter which handles the modifications of this method. The next Chapter focuses on the implementation and modification of RotationNet. The way the modified method is implemented is discussed in this Chapter. The final two Chapters handle the evaluation of the modified method. Evaluations is done through testing the system itself on the one hand, and comparing the results with the human eye in a survey on the other hand. The next sections focus on the results and possible improvements of the RotationNet method. An outlook is given for possible future work.

7.1. Lessons Learned

Several challenges that RotationNet faces have been identified. A vast amount of difficulties have to do with the image resolution used. The input image resolution of RotationNet is limited to a size of 256×256 pixels. This leads to the fact that important details are hardly visible in many CAD renderings in this resolution. Furthermore, common cameras are featured with a much higher resolution and their captured images have to be scaled down for RotationNet. The down-scaling process affects the details in the target image negatively.

The viewpoints of the training data are equally spaced but the amount of the overall viewpoints is not very high. Considering the fact that arbitrary viewpoints of CAD models can be rendered with limited effort (compared to taking photographs), a higher resolution of viewpoints on a sphere is suggested. The expectation is that the training data per class rises and the number of unknown positions reduces. This will lead to a better recognition of minimal details of the object model.

Another issue is the invisibility of certain assembly steps. The result of neighboring instruction steps looks almost identical when the model is near to completion. This effect occurs when added bricks are occluded by others in the actual view. The ILSRVC image database which is used in the original RotationNet implementation contains 40 different classes with enough variance between the classes, e.g., car and dog. The differences between classes is high. The image database in this thesis has more than three times of input classes (instruction steps). This number is presumably too high for the network. Furthermore, there is not enough variance within different steps. Some neighboring classes have little difference, because the images for these instruction steps contain almost the identical information. This means the differences between classes is not very high and this weakens the class prediction.

Furthermore, when applying the new system to a real-life scenario, there is always a background behind the object model. The background information includes background noise and indirect model information such as local or global illumination and shading. The used

training images, however, do not contain any background information or noise at all. Currently, it is not clear how to train a network to handle background noise without having to manually capture many different backgrounds at unacceptably high costs.

Finally, the survey shows that it is difficult for the human eye to distinguish the instruction steps from each other as well and to assign them to the right image. The viewpoints have a strong influence on the detection of the right step and the low resolution makes it difficult to identify the right images.

7.2. Improvements and Future Work

Using a higher resolution for the images is the most important step when improving RotationNet in order to get more detailed information of the model per image. This improvement effects the granularity of details and the distinction between the instruction steps; this means that the differences between the single instruction steps raises. In a real-world scenario the images captured by a camera have a higher resolution and must be down-scaled. The down-scaling algorithm has negative influences on the preservation of details and may cause noise. To improve RotationNet, the input size and the parameters must be adapted to a higher resolution which needs, however, higher computational power for training and prediction task.

The resolution of the viewpoints on the sphere is equally distributed on a dodecahedron but the amount of viewpoints is not sufficient for real-world use. A higher resolution would lead to more training data which can be learned by the system which would result in a better prediction. This improvement needs higher computational power only during training and not in the prediction phase which is a benefit for the application of the system.

An overall challenge of all methods is the influence of the background on the recognition of the model. The recognition and reduction of background information may improve the

prediction. At the training it is hardly possible to know in advance how the background will look like in the productive application. The captured scene can be indoor or outdoor, with various illumination configurations. One option may be a pre-segmentation of an image and to pass only the extracted foreground to the prediction system. An advantage of this approach is to leave the network input image size untouched and crop the interesting image parts to this size. A downside is that it is difficult to find the right segment within the image.

7.3. Outlook

Augmented reality in combination with assisting systems is a promising research field which will be developed further in the future. The current state-of-the-art is only the beginning of the development of these systems to enhance the real-world with virtual information. It can be applied, e.g., in the field of industrial or consumer business. This will contribute to the industrial and consumer digitalization of processes and products. Once these systems are well-established, instruction manuals as we know them today will be redundant and boundaries concerning knowledge transfer can be overcome.

RotationNet is a thoroughly developed method which is working well under certain conditions. Future work will show if it is possible to develop a method which can be applied to more than one specific case or if its development will go into the direction of use-case-specific designs.

References

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Hasan, M., Van Essen, B. C., Awwal, A. A. S., and Asari, V. K. (2019). A State-of-the-art Survey on Deep Learning Theory and Architectures. *Electronics*, 8:292ff.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, Berlin, Heidelberg.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A Large-scale Hierarchical Image Database. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 12:248–255.

Elhoseiny, M., El-Gaaly, T., Bakry, A., and Elgammal, A. M. (2016). A Comparative Analysis and Study of Multiview CNN Models for Joint Object Categorization and Pose Estimation. *International Conference on Machine Learning (ICML)*, 33:888–897.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1:119–130.

Hanbury, A. (2003). A 3D-Polar Coordinate Colour Representation Well Adapted to Image Analysis. J. Bigun and T. Gustavsson (Eds.), *Image Analysis*, Springer Berlin Heidelberg, Berlin, Heidelberg, 804–811.

Häne, C., Zach, C., Cohen, A., and Pollefeys, M. (2017). Dense Semantic 3d Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1730–1743.

Hartley, R. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision (2nd Edition)*. Cambridge University Press, Cambridge, UK.

Haykin, S. (2007). *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computing*, 18(7):1527–1554.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, ACM, New York, NY, USA, 675–678.

Johnson, S. P. (2013). Object Perception. P. D. Zelazo (Ed.), *The Oxford Handbook of*

Developmental Psychology (Vol. 1: Body and Mind), Oxford University Press, New York, 371–379.

Kanezaki, A., Matsushita, Y., and Nishida, Y. (2018). RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 21:5010–5019.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *International Conference on Neural Information Processing Systems*, 25:1097–1105.

Kuznietsov, Y., Stückler, J., and Leibe, B. (2017). Semi-supervised Deep Learning for Monocular Depth Map Prediction. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 20:2215–2223.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lenc, K. and Vedaldi, A. (2019). Understanding image representations by measuring their equivariance and equivalence. *International Journal of Computer Vision*, 127(5):456–476.

Mahjourian, R., Wicke, M., and Angelova, A. (2018). Unsupervised Learning of Depth and Ego-motion from Monocular Video Using 3d Geometric Constraints. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 21:5667–5675.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Oliphant, T. E. (2006). A Bayesian perspective on estimating mean, variance, and standard-deviation from data. *Brigham Young University (BYU) Faculty Publications*, 1877-438.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115:211–252.

Saxena, A., Chung, S. H., and Ng, A. Y. (2006). Learning Depth from Single Monocular Images. *Advances in Neural Information Processing Systems*, 18:1161–1168.

Schinko, C., Ullrich, T., and Fellner, D. W. (2011). Simple and Efficient Normal Encoding with Error Bounds. *Theory and Practice of Computer Graphics*, 29:63–66.

Su, H., Maji, S., Kalogerakis, E., and Learned-Miller, E. (2015). Multi-view Convolutional Neural Networks for 3d Shape Recognition. *IEEE International Conference on Computer Vision (ICCV)*, 11:945–953.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d ShapeNets: A Deep Representation for Volumetric Shapes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 18:1912–1920.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014*, Springer International Publishing, Cham, 818–833.

Zhou, T., Brown, M., Snavely, N., and Lowe, D. G. (2017). Unsupervised Learning of Depth and Ego-Motion from Video. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 20:1851–1860.

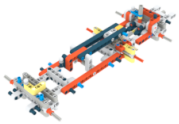
A. Appendix

Steps 1–20

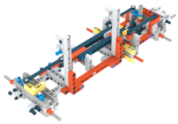
1

2

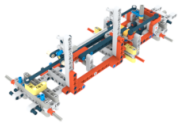
3



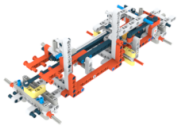
4



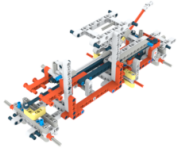
5



6

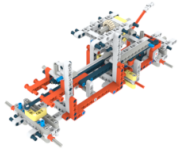


7



8

9



10

11

12

13

14

15

16

17

18

19

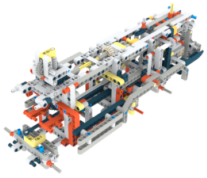
20

Steps 21–40

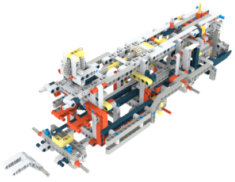
21

22

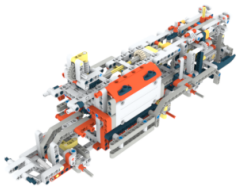
23



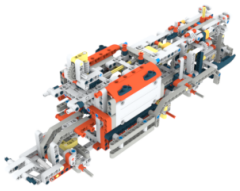
24



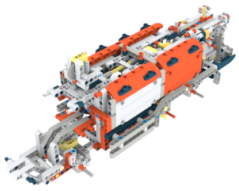
25



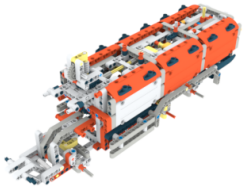
26



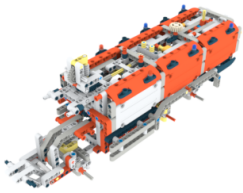
27



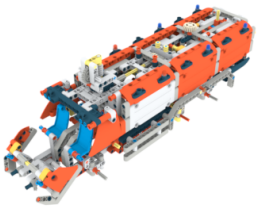
28



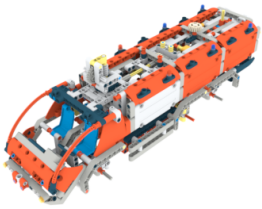
29



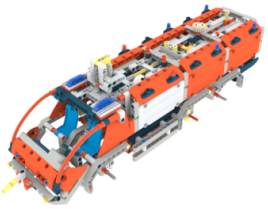
30



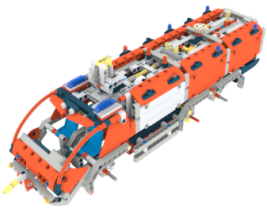
31



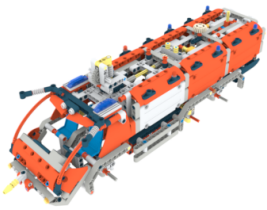
32



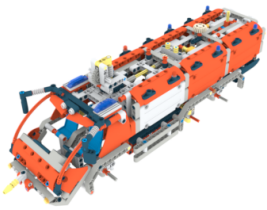
33



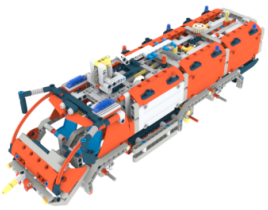
34



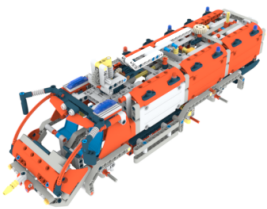
35



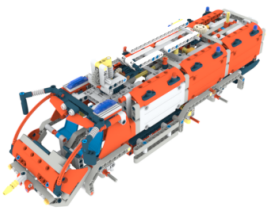
36



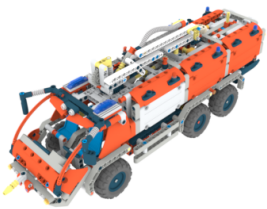
37



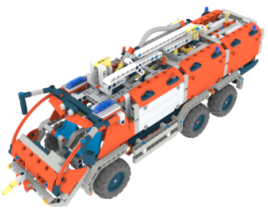
38



39



40



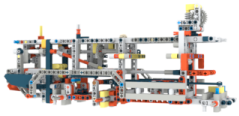
B. Appendix

Questionnaire

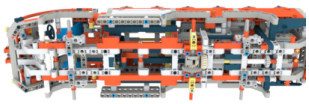
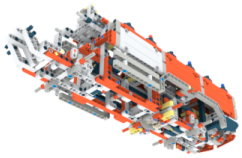
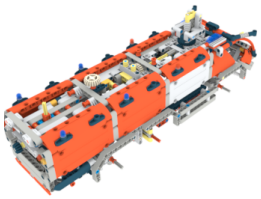
Please assign the correct construction step to the following images:

____ # ____ # ____ # ____

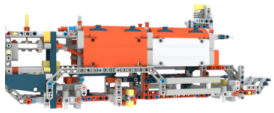
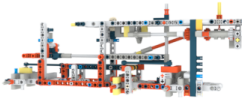
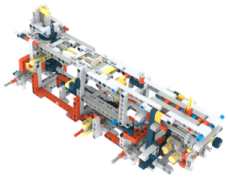
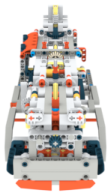





____ # ____ # ____ # ____

____ # ____ # ____ # ____

____ # ____ # ____ # ____