University of Groningen

Faculty of Science and Engineering

Industrial Engineering and Management

Design Project

Domain Knowledge Integration in Object Detection Tasks

Author:	R. S. Janssen (S2814188)
First supervisor:	M. Mohebbi, MSc
Second supervisor:	prof. dr. ir. B. Jayawardhana
Company supervisor:	Maarten C. Stol, MSc

January 21, 2021

Abstract

While rapid innovations have greatly improved the quality of image classification and object detection models, there still exists a large gap between the manner in which humans and machines learn. While state-of-the-art machine learning models are able to effectively find and learn patterns in data, they remain unable to reason in a human manner. BrainCreators, an artificial intelligence company specialized in object detection for visual quality and asset inspectors, wishes to lift knowledge transfer to a higher level, by integrating domain knowledge in the trainable machine learning pipeline. BrainCreators hopes that transferring knowledge to machine learning models in more than one way might improve predictive performance and decrease the amount of training data required to learn the patters in the data, thereby improving the knowledge transfer process from the client domain to the model. Domain knowledge can be represented by knowledge graphs, describing objects, attributes and relationships. In this project, an exploration of frameworks to integrate domain knowledge in the trainable machine learning pipeline was conducted, after which the Hybrid Knowledge Routed Modules (HRKM) framework was chosen.

A new implementation of the framework was created, leading to a modular software deliverable that could potentially be used in the BrainMatter platform.The framework was applied to a dataset of a Dutch telecom company used for cabinet inspection, in order to evaluate the framework on an industrial use case. Knowledge graphs describing the client domain were created and integrated into the object detection pipeline with the HKRM framework. The results show that the application of the framework on the client case did not yield an improvement in predictive performance, nor did the amount of training data required to achieve similar predictive performance decrease. While the new implementation of the framework made for this project was capable of replicating the results of the paper, unfortunately, the results did not persist on more challenging industrial data.

Contents

Abstract **I**

Contents **II**

List of Figures **IV**

List of Abbreviations **V**

1. Research Design **1**
1.1. Problem introduction 1
1.2. Stakeholder analysis 2
1.2.1. Requirements 3
1.3. System description 4
1.4. Research questions and problem statement 5

2. Theoretical Background **7**
2.1. Object detection 7
2.1.1. Training 7
2.1.2. Faster R-CNN 8
2.2. Knowledge graphs 15
2.3. Integrating domain knowledge 16

3. Implementation **18**
3.1. Model 18
3.1.1. Architecture 19
3.1.2. Explicit knowledge 19
3.1.3. Implicit knowledge 21
3.2. Dataset description 24
3.3. Testing 25
3.3.1. Influence of knowledge modules 25
3.3.2. Hyper-parameter λ 27
3.3.3. Ablation study 27
3.3.4. Hypotheses 28

4. Results **29**
4.1. Influence of knowledge modules 29
4.2. Influence of hyperparameter λ 32
4.3. Ablation study 35

5. Discussion **37**

6. Conclusion **40**

Bibliography **41**

A. Appendix **43**
A.1. Machine learning terminology 43
A.2. Dataset description 44
A.3. Knowledge Graph Creation 46
A.3.1. Attribute graph 46
A.3.2. Relational graph 48
A.4. Additional results 50
A.4.1. Ablation study 50

List of Figures

1.1. Power interest grid displaying the stakeholders in the research, where R denotes research, PD denotes product development, and C denotes clients. 2
1.2. Depiction of system describing the context of the project 4

2.1. Illustration of bounding boxes around objects in an image 7
2.2. Faster R-CNN architecture 8
2.3. Convolutional neural network (Saha, 2018) 9
2.4. Convolution operation with kernel 9
2.5. Max pooling operation 10
2.6. The residual block (He et al., 2016) 11
2.7. Region Proposal Network (Ren et al., 2015) 11
2.8. Undirected and directed graph (Fionda and Palopoli, 2011) 15

3.1. Model architecture of implementation based on Faster R-CNN with the addition of an explicit and implicit knowledge module 19
3.2. Depiction of the explicit knowledge module (where N is number of proposals, F is number of features per proposal, M is the size of the module) 20
3.3. Depiction of the implicit knowledge module (where N is number of proposals, F is number of features per proposal, M is the size of the module, T is the amount of graphs in the module) 22
3.4. Example image of cabinet inspection dataset with bounding boxes . 24
3.5. Depiction and equation describing IoU 26

4.1. Average precision for model several model configurations with and without knowledge modules 30
4.2. Average recall for model several model configurations with and without knowledge modules 31
4.3. Average precision for model configurations with various levels of λ in the loss function 33
4.4. Average recall for model configurations with various levels of λ in the loss function 34
4.5. Average precision for model configurations with and without modules, trained on various percentages of the original training data 35
4.6. Average recall for model configurations with and without modules, trained on various percentages of the original training data 36

A.1. Multi-layer perceptron 43
A.2. Step 1 of attribute graph creation 46
A.3. Step 2 of attribute graph creation 46
A.4. Step 3 of attribute graph creation 47
A.5. Step 1 of relational graph creation 48
A.6. Step 2 of relational graph creation 49

List of Abbreviations

AI	artificial intelligence
AP	average precision
AR	average recall
CNN	convolutional neural network
Faster R-CNN	faster regions with convolutional neural networks
GSNN	graph search neural network
HKRM	hybrid knowledge routed modules
IoU	intersecion-over-union
ML	machine learning
MLP	multi-layer perceptron
MSE	mean squared error
ResNet	residual neural network
RoI	region of interest
RPN	region proposal network
SGD	stochastic gradient descent

1.Research Design

1.1. Problem introduction

While rapid innovations in computer vision have greatly improved the quality of image classification and object detection models, there still exists a large gap between the manner in which humans and machines learn (Fang et al., 2017). Although current state-of-the-art machine learning (ML) approaches are able to effectively find and learn patterns in data, they remain unable to reason in a human manner. Humans are able to recognize objects after seeing it only a few times or from descriptions of the features of an object, reasoning with their knowledge of objects with similar features (Marino et al., 2017). For example, a human would be able to recognize a zebra, never having seen one before, if someone described it to them as a horse with black and white stripes. In an effort to provide machine learning models with similar reasoning capabilities within a specified domain, a major area of research in contemporary artificial intelligence (AI) has emerged, focused on combining explicit knowledge in the form of relational structure with the statistical methods of AI.

Generally, domain knowledge can be represented in various forms that best describe the relationships between variables, such as equations describing physical relations, logic rules, and probabilistic relations. In the case of reasoning within a domain, knowledge graphs have proven to be most useful (von Rueden et al., 2019).
Knowledge graphs are versatile modeling tools which can represent relationships between various objects and attributes in a structured manner, allowing a machine to conduct inference and reasoning over a graph to answer queries (Kejriwal, 2019). Without reasoning abilities, machines need to have access to large amounts of labeled training data to effectively learn patterns. In many situations, it is either simply not possible or very time consuming to generate a vast amount of labeled training data, evenly distributed among all identifiable classes or objects.

BrainCreators, an artificial intelligence company specialized in object detection in images, is currently experiencing this very limitation. With their software platform BrainMatter, BrainCreators aims to enable intelligent automation and enable the process engineer or owner of the process at their client (hereafter domain expert) to have access to AI without the help of a dedicated data team. An example application of the software is a quality control project done at TATA Steel. The quality control of the steel sheets was originally a manual process, which made the company heavily reliant on the availability and experience of their quality control employees. To automize this process, the original quality control employee now controls the process by teaching the software to recognize defects in the surface of the sheets, by manually labeling objects in images.

The main goal of BrainCreators is to transfer human domain knowledge to BrainMatter as efficiently as possible, which allows the back-end machine learning models to learn to recognize objects. The platform is currently limited in the manner of retrieving domain knowledge, which happens in two ways:

1. The domain expert manually labels objects in training images
2. Logic and conditional statements derived from consultation with the domain expert can be used for post-processing decisions, on top of the ML pipeline outcome

The first manner can be a strenuous process, especially if there are many possible objects to be detected. Additionally, some objects might not appear in training data often. The amount of training data that can be fed to the model without creating a data imbalance is then limited (von Rueden et al., 2019), since very imbalanced datasets will not teach the model to recognize all the patterns present in the data, rather causing it to focus on the highly present objects or classes. The second approach is effective, however, not favorable. The model is not trained on the knowledge captured by the logic rules, but rather corrected if necessary. Therefore, the quality of the prediction of the machine learning model will not improve. The maintenance of the logic rules requires a lot of effort, since small changes such as adding a new class of objects might change the set of rules.

BrainCreators wishes to lift knowledge transfer to a higher level and therefore explore new manners of knowledge transfer. As described above, knowledge graphs are a classic method to describe knowledge with the help of concepts and their relationships, and therefore BrainCreators aims to utilize knowledge graphs to further enable intelligent automation.

1.2. Stakeholder analysis

The stakeholders in the project are analyzed utilizing the power-interest grid introduced by Ackermann and Eden (2011), displayed in Figure 1.1.

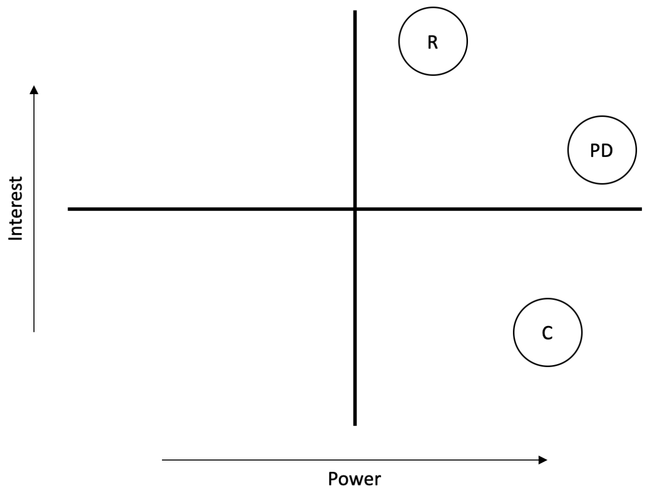


Figure 1.1.: Power interest grid displaying the stakeholders in the research, where R denotes research, PD denotes product development, and C denotes clients.

- **Research**
  The problem owner of the project is Maarten Stol, Principal Scientific Adviser at BrainCreators. In his daily work, he is responsible for the research and development of the company. His team finds new developments in literature and assesses whether these developments can be implemented in their software platform. Maarten has initiated this project due to the emergence of more hybrid machine learning frameworks, as also described by von Rueden et al. (2019). He thinks that integrating prior knowledge will be one of the major research themes of AI in the coming years, would like to know the possible opportunities for BrainCreators to adopt such frameworks. Therefore, his interest in the project is very high, as can be seen in Figure 1.1. Although in some ways his research department guides the future of the company, he does not have the final say in whether or not an idea will actually be adopted by product development, hence making his power in the project moderately high.
- **Product development**
  Tommaso Gritti is head of product development, responsible for both the back-end and front-end of the software. The artifact that will be designed during the project ultimately has to be approved by him, to make sure it is usable in the currentML pipeline. Many of the requirements of the artifact will be determined by him, therefore, giving him significant influence in the project. As head of product development, Tommaso is of course interested in new innovations in the product. However, since feasible new ideas are sought and provided by the research team, the overall interest in the project is moderate.
- **Clients**
  Glenn Brouwer is the Chief Revenue Officer (CRO) of the company and mostly in contact with potential and existing customers. In order to incorporate the client perspective and wishes in the project, Glenn was interviewed. Clients determine a large part of the context BrainCreators operates in, therefore imposing requirements on the project. The power of clients is high, as they are directly linked to the revenue stream, but their interest in the project is low.

1.2.1. Requirements

The artifact to be delivered during this project is a modular software deliverable that can be integrated in the platform and used in future client cases. The artifact will be tested on an academic benchmark dataset, which are used for many machine learning papers to make sure the performance of models can be compared accurately. Later, experiments are performed using data and domain knowledge of a client dataset. Next to that, an analysis added value of the method should be provided to support the company's business case.

The first weeks of literature research, research within the company, and stakeholder interviews have determined a preliminary list of requirements of the artifact that will be designed during this project. This list of requirements is used to evaluate the methods found during the literature review.

- **Modularity**
  The artifact should be a modular software deliverable that is able to run in BrainMatter on top of various object detection models.
- **Generalizability**
  The artifact should not only be usable in the specific client use case, but should be generalized to ensure applicability across domains.

1.3. System description

In order to describe the context the project is performed in, the system is depicted in Figure 1.2. The colored blocks refer to the domain and responsibilities of each player in the system. The blue block, 'Client domain', describes how the client's domain knowledge and data is used in the system. The client is responsible for delivering and capturing the data, as well as labeling objects in the training dataset. When using the platform, the client's new data is to be reviewed and classified by the software.

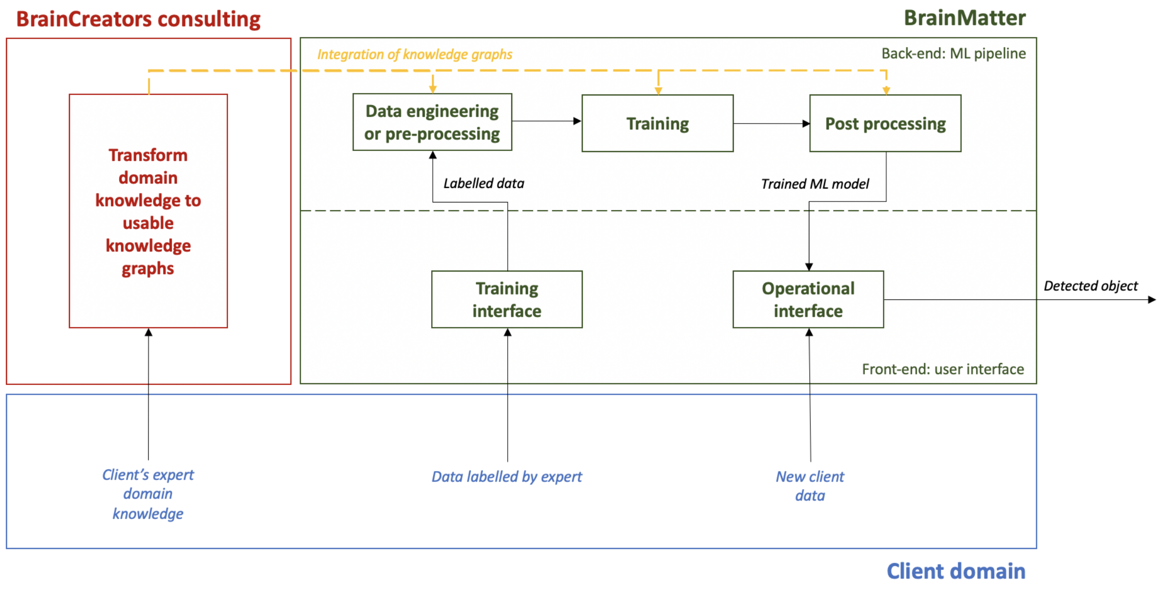


Figure 1.2.: Depiction of system describing the context of the project

The green block, 'BrainMatter', depicts the platform and distinguishes the back-end and front-end (user interface) of the software. The client is not aware of the back-end machine learning pipeline, but is only concerned with teaching the platform to recognize images through labeling and viewing detected results when using the platform.

The red block, 'BrainCreators consulting', refers to the process where domain knowledge is extracted from the client. BrainCreators is currently in the process of formalizing its knowledge transfer process to achieve uniform knowledge acquisition, which is in part why this project was initiated. The domain knowledge acquired in the knowledge transfer process should be transformed to a knowledge graph that describes the client domain and object classes, in manner that is consistent across projects.

The yellow arrow then indicates at which stages in the machine learning pipeline knowledge can potentially be integrated. The scope of this project is limited to red block and yellow arrows. During the project a decision will have to be made regarding the knowledge extraction and knowledge graph creation. The second decision to be made will decide where in the machine learning pipeline the domain knowledge will be integrated.

1.4. Research questions and problem statement

The research questions to be answered during the project are stated below. The literature research and rest of the project are structured according to the research questions.

RQ1 How can domain knowledge be described through knowledge graphs?
Knowledge graphs can describe any relationship between two concepts or between a concept and its attributes. Current client cases should be evaluated to see which types of domain knowledge are available, and how this knowledge can be captured by a graph. Knowledge types can include spatial relations, co-occurrence of objects in images, and shared attribute knowledge (Jiang et al., 2018). It should be decided whether all knowledge types can be captured in one graph or, for example, modular knowledge graphs can be utilized whenever required in the specific case, similar to the approach of Jiang et al. (2018).

RQ2 Where in the data pipeline can this knowledge be integrated?
There are various stages within the data pipeline where background knowledge can be integrated. von Rueden et al. (2019) created a survey of existing methods, which include:

a) Integration at the data engineering and pre-processing stage.

b) Integration during training of the machine learning models.

c) Integration of knowledge graphs in the design of the neural network itself.

d) Integration as a additional optimization step on the obtained results.

RQ3 What is the added value of adopting the framework?
Part of the research is aimed at quantifying the added value the technique could offer, in case the technique proves to be technically favorable.

Summarizing the information described above, the problem can be formulated as:

Currently, BrainCreators is only capable of transferring domain knowledge to BrainMatter through manually labeling images. This process of knowledge transfer relies on labeling many images of each object to be detected, which is not always possible due to a lack of data and the amount of man hours required to generate sufficient samples for a training dataset.

The goal of the project can be formulated as:

To create a general, modular software deliverable that integrates domain knowledge in the form of knowledge graphs in the machine learning pipeline in the BrainMatter platform, in order to become less dependent on training data and impose explicit knowledge on the learning process.

2. Theoretical Background

In this section, the relevant literature and theoretical background will be discussed to provide basis and grounds for the choices made during the design project.

2.1. Object detection

BrainCreators has chosen to specialize in processing visual image data to become the number one option for automated visual inspection. The most common branch of AI applied to images is image classification, which is the task of classifying the entire image, thereby describing the scene or context of the image. A more specific task is called object detection, which is aimed at locating and subsequently classifying objects in an image. The position of the object is described with a bounding box, which specifies the coordinates of the object in an image. The object inside the bounding box is then classified, to attach a label to it. This section will be dedicated to laying a foundation of knowledge regarding object detection, which the rest of the report can build on. A basic level of machine learning knowledge is assumed for the reader and common machine learning conventions are further explained in Appendix A.1.

2.1.1. Training

Machine learning models require vast amounts of data to learn patterns in the data, and to be able to predict numerical values from time-series data (regression) or attach a label (classification) based on the input data. For object detection, the data that the model learns from consists of input images in which bounding boxes have been manually drawn around objects, such as in Figure 2.1.

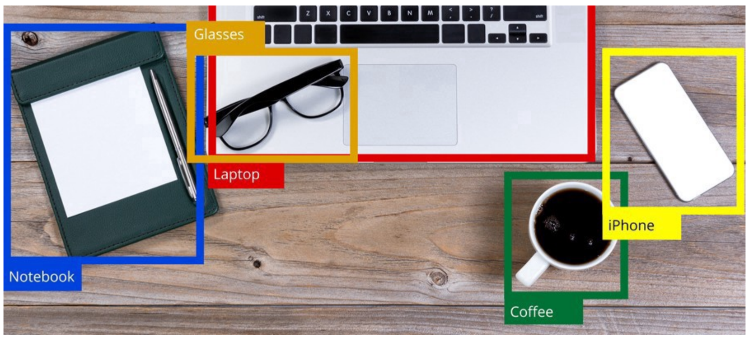


Figure 2.1.: Illustration of bounding boxes around objects in an image

The model learns the required behavior through computing the loss between the ground-truth, referring to the original label and bounding box in the training data, and the label and bounding box predicted by the model. While iterating over the dataset, the internal model weights of neurons and connections that compose the convolutional neural network, depicted in Figure 2.3, are then updated according to the computed loss, until the model has converged to the most optimal settings.

2.1.2. Faster R-CNN

While a multitude of object detection models are available, the majority of them rely on the same principles to be able to process the images and predict the bounding boxes. For this project, an object detection model called faster regions with convolutional neural networks (Faster R-CNN) is chosen as base architecture, since this model is often applied at BrainCreators and many methods build on the architecture of Faster R-CNN.

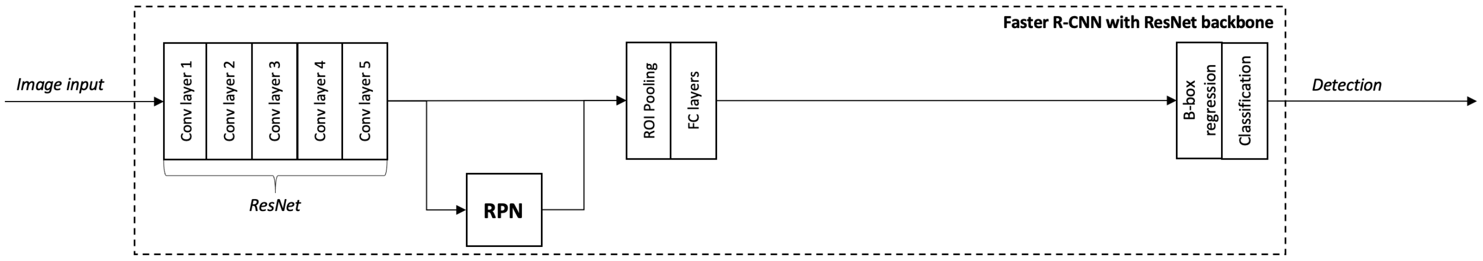


Figure 2.2.: Faster R-CNN architecture

Figure 2.2 depicts the main architecture of Faster R-CNN. The input of the model is a set of images, which will pass through the *residual neural network (ResNet)*. It operates as feature extractor, which are found through the application of filters and convolutional layers. These features are then shared with the *Region Proposal Network (RPN)*, which proposes regions of interest (RoI) which potentially hold an object. The RPN provides proposals with bounding box coordinates and an objectness score, which represents the likelihood of the region being an actual object. The proposed regions and corresponding features are then passed to an *RoI Pooling layer*, which crops the bounding boxes to fit the potential object. The cropped regions are then fed to two *fully connected linear layers.* The output of the box head is fed straight to the last two layers, which are the *bounding box regression and classification layers.* Below, the components of the model will be explained in more detail.

2.1.2.1. ResNet

The backbone network of Faster R-CNN are convolutional neural networks, which are purposed to extract features from the input images. Since processing images can be very computationally expensive, the purpose of convolutional neural networks (CNN) is to reduce the images to a smaller size and format that is easier to process, all the while maintaining the image's critical features.

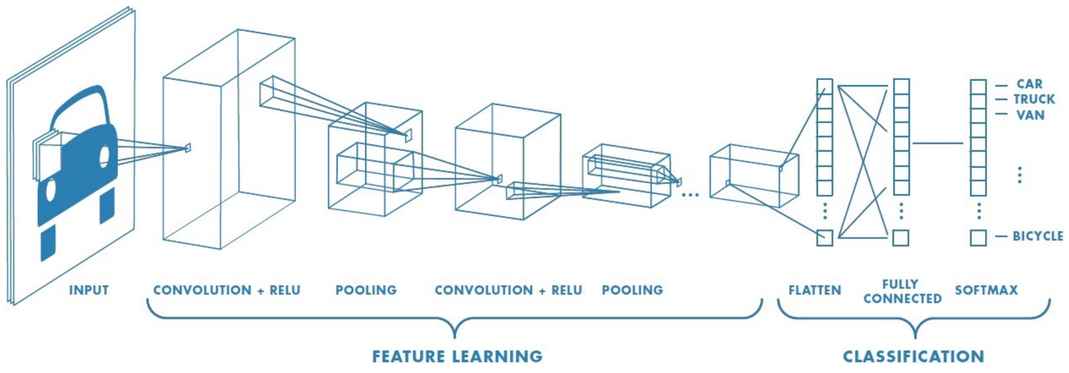
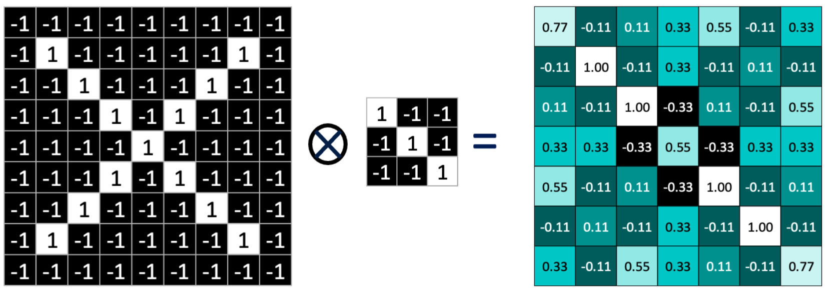


Figure 2.3.: Convolutional neural network (Saha, 2018)

Figure 2.3 outlines the architecture of a convolutional neural network. The image input is passed through several blocks comprised of *convolutional layers*, *activation functions*, and *pooling layers*, which will be explained below. These layers have the purpose of extracting image features and reducing the image's size to a feature map, which the classification layers will use to make a choice regarding the type of object the image contains.

Convolution
An input image will be separated into three channels based on its color planes: red, green, and blue (RGB). A filter, called the kernel, is applied to the different RGB channels of the image (Michelucci, 2019). The Kronecker product of the filter and the RGB layer of the image is averaged and placed at the center of the kernel in the filtered image, as can be seen in Figure 2.4. This kernel is dragged over the image to find patches of the image at which the pattern of the filter occurs.



$$\begin{bmatrix} -1&-1&-1&-1&-1&-1&-1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&-1&-1&1&-1&-1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&-1&-1&-1&-1&-1&-1&-1 \end{bmatrix} \otimes \begin{bmatrix} 1&-1&-1\\ -1&1&-1\\ -1&-1&1 \end{bmatrix} = \begin{bmatrix} 0.77&-0.11&0.11&0.33&0.55&-0.11&0.33\\ -0.11&1.00&-0.11&0.33&-0.11&0.11&-0.11\\ 0.11&-0.11&1.00&-0.33&0.11&-0.11&0.55\\ 0.33&0.33&-0.33&0.55&-0.33&0.33&0.33\\ 0.55&-0.11&0.11&-0.33&1.00&-0.11&0.11\\ -0.11&0.11&-0.11&0.33&-0.11&1.00&-0.11\\ 0.33&-0.11&0.55&0.33&0.11&-0.11&0.77 \end{bmatrix}$$

Figure 2.4.: Convolution operation with kernel

The kernels that a CNN uses to filter the images and gather valuable features are included in the trainable parameters of the network. While it is possible to use handcrafted kernels, the CNNs are capable of finding an optimal configuration.

Activation function

After convolution, the ReLU activation function is applied to the filtered image

$$f(x) = \begin{Bmatrix} 0, & \text{for } x < 0 \\ x, & \text{otherwise} \end{Bmatrix}. \tag{2.1}$$

It changes all negative values to zero, while all other values remain the same. Applying the activation function ensures that non-linearity is introduced, which allows for learning the complex mapping between input and output variables.

Max pooling

In order to reduce the size of the filtered image, a pooling operation is performed. In the example in Figure 2.5, a window with size 2 by 2 is selected. Out of this window, the maximum value is chosen and added to the feature map. The window is moved with a stride of 2 in the example, to cover every pixel of the filtered image.

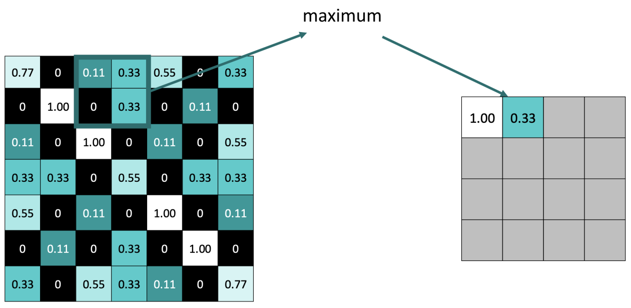
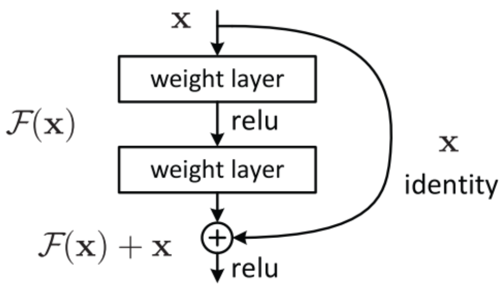


Figure 2.5.: Max pooling operation

Several configurations of convolutional neural networks (CNN) are available, however, this project focuses on ResNet. The intuition behind deep neural networks is that consecutive layers are capable of learning more complex features, figuratively zooming in on the image. However, as research on deep neural networks was progressing, a maximum depth threshold was found by He et al. (2016), for depth in traditional CNN models. The authors empirically showed that a 20-layer network had a lower training and test error than a 56-layer network (He et al., 2016). These results have been blamed on the optimization function, the vanishing gradient problem, or the network initialization. The authors have proposed a new neural network layer, the residual block depicted in Figure 2.6, to alleviate these problems. The framework presented is capable of adding significant depth to convolutional neural networks, without compromising on complexity.



x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.6.: The residual block (He et al., 2016)

The authors theorized that not all layers in a CNN have significant influence on the predictive performance, but some layers rather learn an identity function to copy the results of previous layers, with $F(x) = x$. Learning the identity function provides room for mistakes, since each kernel contains many trainable weights. Therefore, the authors theorized that adding the skip connection in the residual block allows for the network to learn transformation where it is deemed necessary and can skip the block otherwise. In case the block is skipped, the layers in the block can converge its output $F(x)$ to zero, since the final output of the block summed with the skip connection $H(x) = F(x) + x$ will ensure an accurate mapping of the identity function. ResNet networks can consist of up to 1001 layers, but common implementations consist of up to 50, 101 or 152 layers, consisting of multiple blocks of layers with similar kernel sizes.

2.1.2.2. Region Proposal Network

The RPN is purposed to propose regions of interest which potentially contain an object. It takes an image as input and consequently outputs a set of object proposals, with a corresponding objectness score (Ren et al., 2015). The objectness score represents the likelihood that the region contains an object.

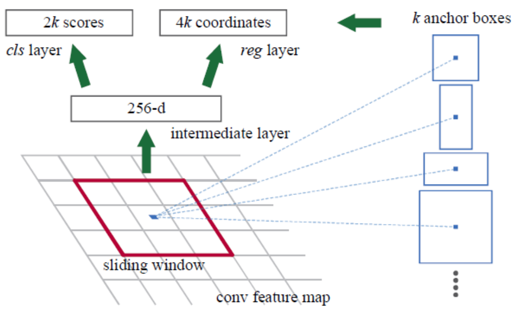


Figure 2.7.: Region Proposal Network (Ren et al., 2015)

The output feature maps of the backbone network are fed to a small fully convolutional network, which slides a window over the feature maps, as depicted in Figure 2.7. At each sliding window location, anchors of multiple scales and aspect ratios are applied to predict multiple region proposals. The anchors are fed to two fully connected layers, a box regression layer and a box classification layer. The box regression layer will refine the bounding box coordinates of the proposed region, whereas the box classification layer of two units (background or foreground) will calculate the objectness score.

This small network is trained with its own multi-task loss function (Ren et al., 2015). Each anchor is given a binary class label, stating whether or not it is an object. A positive label is assigned to the anchors with the highest Intersection-over-Union (IoU) overlap with a ground-truth box (original labeled box), or to anchors that have an IoU overlap with any ground-truth box that is higher than 0.7. A negative label is assigned to anchors if the IoU overlap is lower than 0.3 for all ground-truth boxes. Anchors that have IoU scores in between are not considered. The terms in the loss function will be explained in more detail in Section 2.1.2.4 and IoU is further explained in Section 3.3.1.

2.1.2.3. RoI pooling and fully connected layers

The object proposals from the RPN are fed to the RoI pooling layer, which extracts a fixed-length feature vector from the feature map (Girshick, 2015). Max pooling is used to convert the features inside the proposed RoI into a small feature map with a fixed spatial size of $H \times W$, which are hyper-parameters. The feature maps are pooled to a fixed size, because the fully connected layers that the RoIs will be fed to are of fixed size as well.

The pooled feature maps are then flattened and fed to two fully connected linear layers. The neurons in the fully connected layers are trained to determine which features are likely to belong to which object class, and will generate a set of votes which is used by the final classification layer.

2.1.2.4. Bounding box regression and classification layer

The proposed bounding box coordinates are passed through a bounding box regression layer, which uses a class specific regressor to refine the bounding box coordinates. Given the coordinates of the predicted bounding box $P = (P_x, P_y, P_w, P_h)$, with (P_x, P_y) referring to the box' center coordinates and (P_w, P_h) referring to the width and height of the box. Together with the coordinates of the ground-truth box $G = (G_x, G_y, G_w, G_h)$, the regressor is configured to learn a scale-invariant transformation between the centers of the boxes P and G, and a log-scale transformation between the boxes' width and height (Girshick et al., 2015). The transformations are parameterized by four functions $d_x(P), d_y(P), d_w(P)$, and $d_h(P)$. The proposed bounding box P is then transformed into a predicted ground-truth box $\hat{G}$ with

$$\begin{aligned}
\hat{G}_x &= P_w d_x(P) + P_x, \\
\hat{G}_y &= P_h d_y(P) + P_y, \\
\hat{G}_w &= P_w exp(d_w(P)), \\
\hat{G}_h &= P_h exp(d_h(P)).
\end{aligned} \tag{2.2}$$

Each function $d_*(P)$ (with $*$ one of x, y, w, h) is modeled as a linear function of the pooled features from the RoI pooling layer of proposal P, which is denoted by $\phi(P)$. This gives $d_*(P) = \mathbf{w}_*^T \phi(P)$, where $\mathbf{w}_*$ is a vector of trainable model parameters, which is optimized with a least squares objective

$$\mathbf{w}_* = \underset{\hat{\mathbf{w}}_*}{argmin} \sum_i^N (t_*^i - \hat{\mathbf{w}}_*^T \phi(P^i))^2 + \lambda \|\hat{\mathbf{w}}_*\|, \tag{2.3}$$

where λ represents the balancing factor for regularization and $\hat{w}_*$ represents the predicted model parameters. The regression targets t_* for training pair (P,G) are defined as

$$\begin{aligned} t_x &= (G_x - P_x)/P_w, \\ t_y &= (G_y - P_y)/P_h, \\ t_w &= log(G_w/P_w), \\ t_h &= log(G_h/P_h). \end{aligned} \tag{2.4}$$

The classification layer has a unit for each of the classes and passes the features obtained from the fully connected layers through a softmax activation function, therefore obtaining a vector of probabilities stating that the object belongs to a class.

Both the RPN and the R-CNN use an objective function aimed at minimizing a multi-task loss function, which combines a regression and classification loss term.

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*). \tag{2.5}$$

The difference in loss functions for the RPN and the R-CNN is in the amount of classes in the classification layer and therefore the classification loss term. For the RPN, it should predict either object or not an object, whereas for the R-CNN it should predict a vector of probabilities for each object class. Therefore, for the RPN, i represents the index of the anchor in a batch, p_i is the predicted probability that anchor i is an object or not an object, and p_i^* is the binary ground-truth label.

For the R-CNN, i refers to the index of the proposal in the batch, and p_i is a vector of probabilities that an object belongs to a class, and p_i^* is the ground truth object class label. For both the RPN and R-CNN, t_i is a vector representing the parameterized predicted bounding box coordinates and t_i^* represents coordinates of the ground-truth box. The classification loss L_{cls} is a log loss over the classes and the regression loss L_{reg} is a smooth L1 loss, which is given by:

$$L_{reg}(t, t^*) = \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i, t_i^*), \tag{2.6}$$

in which

$$smooth_{L1}(x) = \left\{ \begin{array}{ll} 0.5x^2, & \text{if}|x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{array} \right\}. \tag{2.7}$$

The terms are normalized by the batch size N_{cls} and the number of anchor locations or bounding boxes N_{reg}, and weighted by a balancing parameter λ. According to Ren et al. (2015), the value of $\lambda = 10$ provides the best results.

The final loss of the Faster R-CNN network is then a sum of the previously described loss terms,

$$L = L_{RPN,cls} + L_{RPN,reg} + L_{R-CNN,cls} + L_{R-CNN,reg}. \tag{2.8}$$

and is used to optimize all the trainable parameters in the model.

The architecture of the Faster R-CNN model was explained in the previous paragraphs and will serve as a basis for the extensions of the model explained in the rest of the report.

2.2. Knowledge graphs

Extensive research effort has been conducted in encoding semantic knowledge, describing real-world concepts and their relationships, in machine readable forms such as knowledge graphs (Paulheim, 2017). Knowledge graphs have been widely used in many data-driven applications, such as web search, social networks, and natural language processing (Dong et al., 2014). These applications use large-scale knowledge which have been constructed using automated extraction from online data or other relevant data sources. Recently, knowledge has also been deducted from other data sources, such as images, in for example the Visual Genome dataset (Krishna et al., 2017). While using such large-scale knowledge graphs has many advantages, including improved generalizability across multiple domains (Fang et al., 2017), they are not suitable for BrainCreators in most instances. Visual asset or quality inspection, the core task BrainCreators performs for clients, is usually not aimed at recognizing conventional objects, but rather domain specific objects and defects. Since it is BrainCreators' goal to lift knowledge transfer from domain expert to machine learning models to a higher level, creating a knowledge graph with the clients' domain expert could help in improving the knowledge transfer process. Therefore, it is favorable to select a method that allows for integration of custom knowledge graphs.

Although many variations of knowledge graphs exist, this project only focuses on the distinction between undirected (a) and directed (b) graphs, as displayed in Figure 2.8. Graphs, given by $\mathsf{G} = \{\mathsf{N}, \mathsf{V}\}$, are composed of nodes N, representing objects or attributes in this project, and are connected by edges V that represent a relationship. In undirected graphs the relationship described by the edge is valued similarly by the connecting nodes, while in directed graphs the relationship is only applicable in the direction of the arrow. To give an example: in "**man** *rides* **bicycle**", the relationship *rides* is not applicable in the other direction.

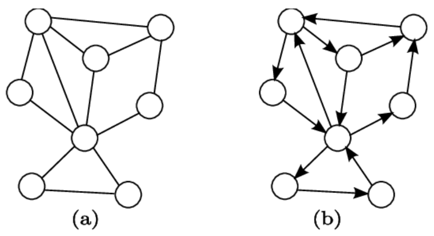


Figure 2.8.: Undirected and directed graph (Fionda and Palopoli, 2011)

Graph G can be transformed to an adjacency matrix of size (NxN), where the elements of the matrix describe whether or not a relationship exists between the nodes. The value of these matrix elements represent, in the case of knowledge graphs, either semantic consistency between concepts or a co-occurrence measure of concepts.

2.3. Integrating domain knowledge

A literature review has been conducted to find state of the art frameworks that integrate knowledge graphs in image classification or object detection networks. All frameworks were reviewed on a number of aspects, the most important being:

- The application; BrainCreators would prefer a framework that is aimed at object detection, since it relates to their core activities most strongly.
- Point of integration; the domain knowledge graph can be integrated at several points in the machine learning pipeline and the way the integration is intertwined with the main object detection pipeline can have consequences for future relevance of the framework. A framework that allows a modular application to be added to several existing object detection pipelines would be preferred.
- Type of knowledge graphs used; some frameworks allow for the integration of a custom knowledge graph, fitting exactly to the area of application, whereas others rely on large-scale knowledge graphs that are readily available.
- Availability of code; in case the paper's code can be found online, it would be considerably easier to implement the proposed framework in the amount of time available for the project.

One of the simplest ways to insert knowledge graphs in object detection models is to perform an additional optimization step on the proposed class labels of the model, as proposed by Fang et al. (2017). The proposed probability vector P of an object belonging to classes in the dataset is optimized by multiplying with the adjacency matrix of semantic consistency, generating optimized prediction $\hat{\mathsf{P}}$. The knowledge graph utilized is a large-scale knowledge graph called MIT ConceptNet by Liu and Singh (2004), out of which the semantic consistency is extracted with *random walks with restart* (Tong et al., 2006). Although code is not available for this method, it should be realizable to implement the framework. The drawback of the method is the fact that the knowledge graph is not integrated in the trainable machine learning pipeline, therefore the object detection model does not learn to generate better predictions based on the domain knowledge.

Contrarily, Marino et al. (2017) introduce a Graph Search Neural Network (GSNN) purposed for multi-label image classification which is part of the trainable pipeline. A detection network proposes detected objects, which are used as starting points for propagation in a knowledge graph. Neighboring nodes in the graph are then expanded based on a learned importance network in two iterations. After the final iteration, a set of predictions per graph node is concatenated with the features from the last fully connected layers. The final feature vector is then fed to the final classification layer. Although results are promising and the method is modular, the framework might not be suitable for the smaller graphs that BrainCreators would like to construct, as propagating through those graphs might not be possible. Additionally, the paper is focused on image classification and code for the paper is not available, making it hard to realize in the given time frame.

Another framework that incorporates the knowledge graph in the trainable pipeline called Hybrid Knowledge Routed Modules (HKRM) was introduced by Jiang et al. (2018). The authors make a distinction between knowledge forms; explicit and implicit knowledge. Explicit knowledge is defined as object-to-attribute and relational object-to-object knowledge, whereas implicit knowledge is defined as spatial knowledge that is usually not defined linguistically, such as: 'the sea is always below the ships'. Instead of propagating through the knowledge graphs, the edge values are extracted from annotations of large object detection datasets and knowledge graphs such as Visual Genome (Krishna et al., 2017). The visual features from the last fully connected layers is routed through both explicit and implicit knowledge modules, which are small neural networks themselves, to output knowledge enhanced features. These knowledge enhanced features are concatenated with the original feature set and send to the final classification step. HKRM is a modular method that can be applied to any state of the art object detection pipeline and can either be used with pre-existing large-scale knowledge graphs or smaller, custom knowledge graphs.

Combining the previously mentioned advantages and the fact that the authors have published the code of the paper, the choice has been made to create a custom implementation based on HKRM for BrainCreators.

3. Implementation

In this section, the several phases of implementation of the project are described. First, the model is created and adjusted to match BrainCreators' method of working. After that, a client dataset is chosen and a corresponding knowledge graph is created. The performance of the model is then evaluated, with the addition of various extents of domain knowledge.

3.1. Model

The code that accompanied the original paper by Jiang et al. (2018), found on `https://github.com/chanyn/HKRM`, turned out to be outdated in terms of used packages. Therefore, a new implementation was made using Facebook's Detectron2 library (Wu et al., 2019), which BrainCreators also uses in their BrainMatter platform. (Tests have been conducted to verify that the new implementation is able to replicate the same results as the original paper, compared on Microsoft COCO dataset (Lin et al., 2014), and is therefore an accurate implementation of the framework.) This section will be dedicated to explaining the general architecture used and how the two modules that are integrated work.

3.1.1. Architecture

Jiang et al. (2018) propose to build two modules that can be plugged into any object detection network. Their implementation in the paper is build with Faster R-CNN, of which the architecture is explained in Section 2.1.2. Detectron2 offers a base implementation of Faster R-CNN with ResNet, which can be modified as required.

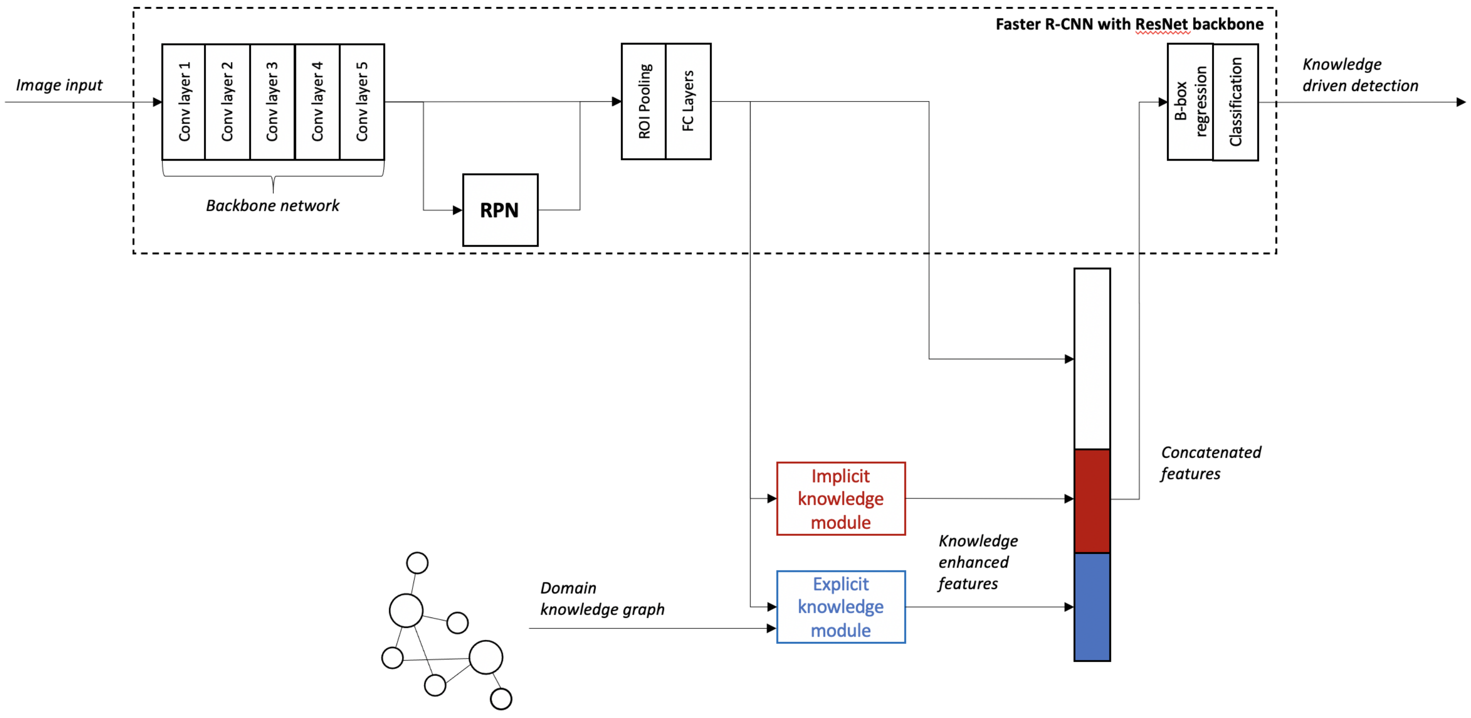


Figure 3.1.: Model architecture of implementation based on Faster R-CNN with the addition of an explicit and implicit knowledge module

In this implementation, instead of feeding the output features of the fully connected layers straight to the bounding box regression and classification layer, the features are also routed through the *implicit* and *explicit module*, as can be seen in Figure 3.1. Both modules hold a form of domain knowledge that will be used to output *knowledge enhanced features*, which will be concatenated with the original feature set. This way, final layers should be able to make a more informed decision regarding the object a proposal contains and make a *knowledge-driven prediction*.

3.1.2. Explicit knowledge

The explicit knowledge module is a supervised learning method that aims to use pre-defined information regarding the objects, their attributes, and relationships within the dataset. The knowledge is captured by a knowledge graph $Q =< C, V >$, where C represents object class nodes in the graph and $v_{i,j} \in V$ the edge weights. Graph Q is then transformed to an adjacency matrix, where each cell in the matrix describes either the similarity between object classes in the dataset in terms of attributes and properties, or the probability of a relationship between objects. The explicit knowledge module builds an adaptive region-to-region graph $\hat{G} =< N, E >$, where N represent the object class nodes and $e_{i,j} \in E$ represent the graph edges. Graph $\hat{G}$ is adaptive in the sense that it is adapted to the object classes that are represented in the batch of proposals.

The goal of the explicit knowledge module is to learn to encode the relationship between the features belonging to each object class, and the relationship of between features of different object classes. Learning to encode this knowledge assures that the knowledge can be applied in the testing phase.

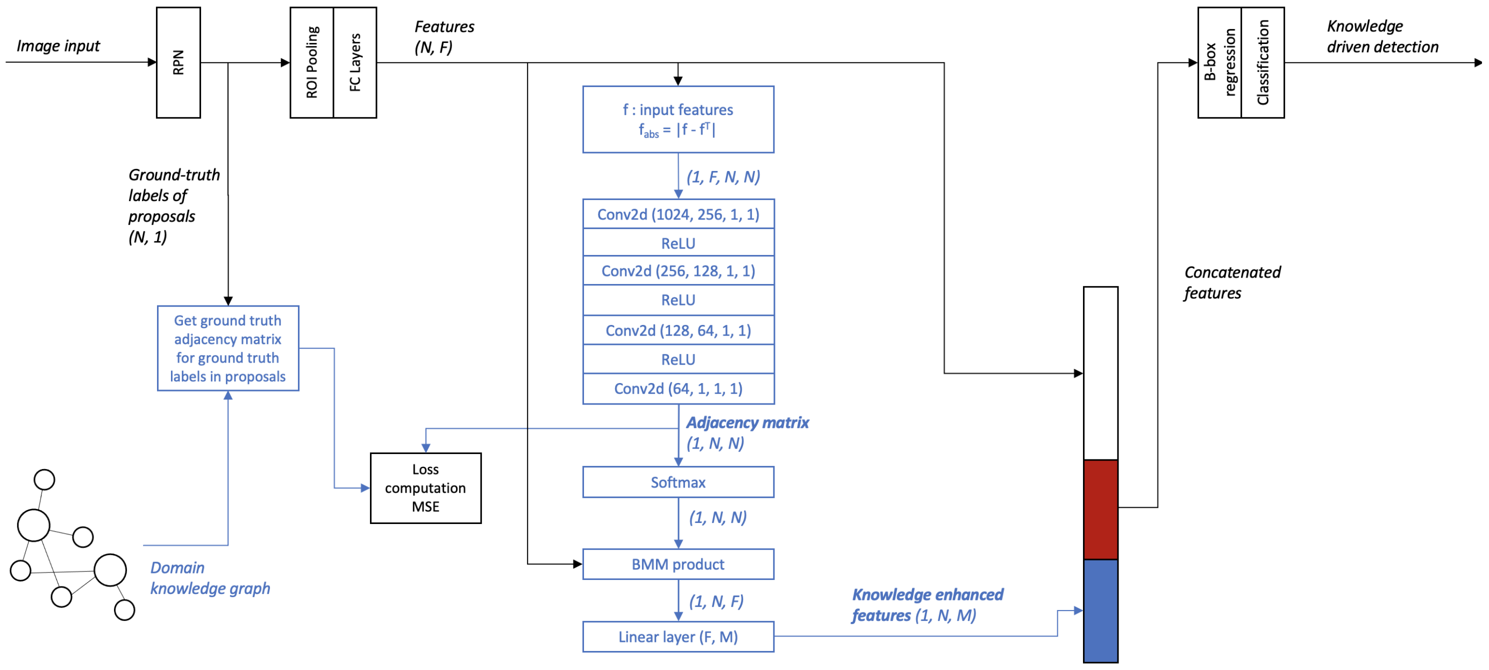


Figure 3.2.: Depiction of the explicit knowledge module (where N is number of proposals, F is number of features per proposal, M is the size of the module)

In Figure 3.2, the explicit module is depicted. The output of the fully connected layers is a list (NxF) of N proposed regions and F corresponding features per image. The goal of the explicit knowledge module is to learn the edge weights $e_{i,j}$ to approach the ground-truth edge weights $v_{i,j}$, which is done with a stacked multi-layer perceptron (MLP), composed of the convolutional layers of the module:

$$\hat{e}_{i,j} = \mathrm{MLP}_Q(\alpha(f_i, f_j)), \tag{3.1}$$

where α represents the pairwise L1 difference and (f_i, f_j) represents the features of a region pair. The list of proposed regions and features is fed to explicit module, denoted by f in Figure 3.2. The pairwise L1 difference between the features of each proposed region is calculated with

$$f_{abs} = |f - f^T|. \tag{3.2}$$

In the training phase the region proposals are accompanied by ground-truth labels out of the labeled training data, and these are used to create a ground-truth adjacency matrix $A_{GT}(N \times N)$, which contains all ground-truth edge weights $\nu_{i,j}$, for all region proposals in the batch. Four convolutional layers then apply filters to f_{abs} and reduce its size to a predicted adjacency matrix $A_P(N \times N)$ for the proposed regions, which contains all predicted edge weights $\hat{e}_{i,j}$. Through doing so, the model has learned to deduct the similarities or relationships between object classes from the features of each object.

Lastly, the knowledge enhanced features are computed by performing matrix multiplication, indicated with batched matrix multiplication () in the figure, with the predicted edge weights and the list of input features:

$$f_{enhanced} = \hat{E}f. \tag{3.3}$$

These enhanced features are concatenated to the base model features and fed to the bounding box regression and classification layers. The supervision of this module is in the comparison of the edge weights of the predicted adjacency A_P matrix and the ground-truth adjacency matrix A_{GT}, through computing the mean squared error (MSE)

$$L(f_{abs}, W_Q, Q) = \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{1}{2}(\hat{e}_{i,j} - \nu_{i,j})^2, \tag{3.4}$$

where W_Q represents the trainable model parameters of the MLP.

In the original paper and in the custom implementation, the knowledge graph is split into two graphs, where one describes the similarity between objects based on their attributes (attribute graph) and the other describes the likelihood of co-occurrence of objects (relational graph).

3.1.3. Implicit knowledge

The implicit knowledge module is an unsupervised learning method, which aims to capture spatial relationships between object classes in the dataset. The module is based on the concept of multi-head attention, which was first introduced by Vaswani et al. (2017). Attention mechanisms were first applied in networks dealing with natural language processing. Including an attention mechanism in neural networks should allow the model to learn the context of words in sentences and word sequences, and therefore improve the ability to answer queries. Multi-head attention combines multiple attention mechanisms in parallel, allowing each mechanism to focus on learning different aspects of the context. Applied to the implicit knowledge module in the network, multi-head attention is used to capture multiple spatial layouts of objects with respect to other objects.

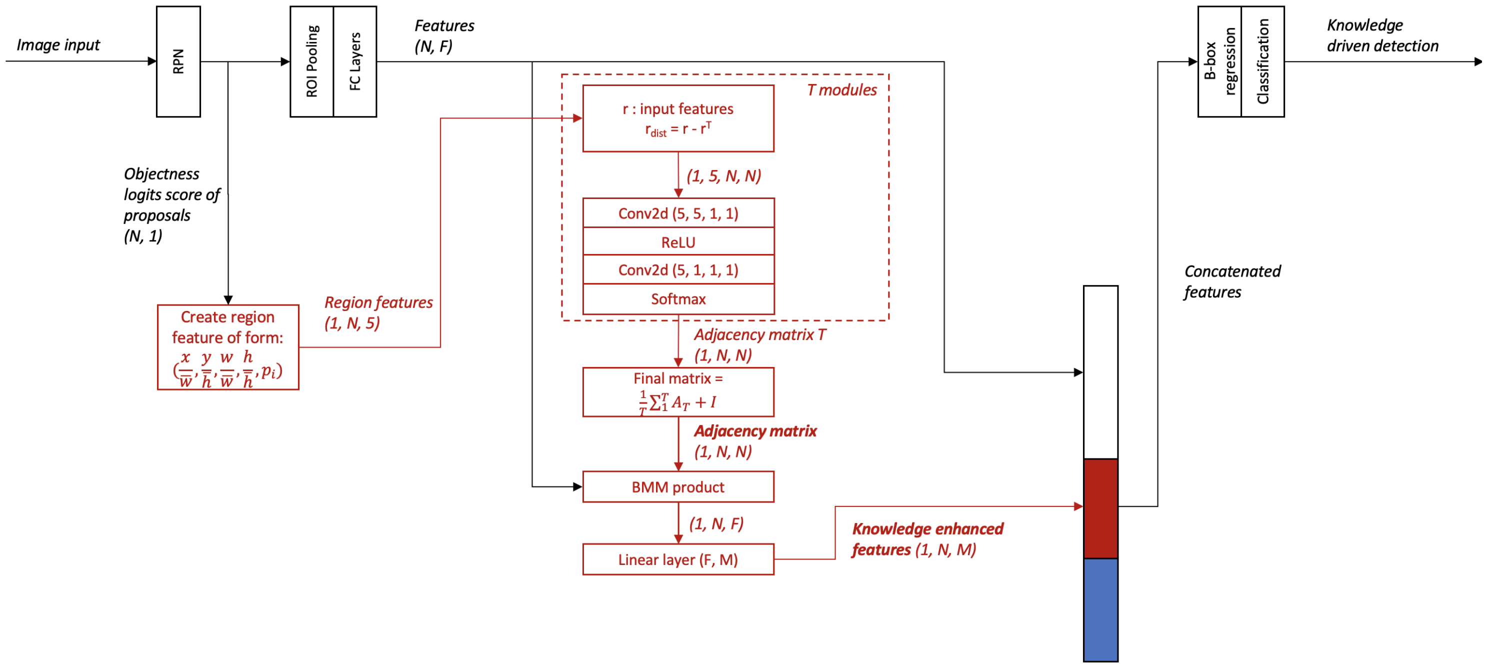


Image input
RPN
ROI Pooling
FC Layers
Features (N, F)
Objectness logits score of proposals (N, 1)
Create region feature of form: $(\frac{x}{\overline{w}}, \frac{y}{\overline{h}}, \frac{w}{\overline{w}}, \frac{h}{\overline{h}}, p_i)$
Region features (1, N, 5)
T modules
r : input features
$r_{dist} = r - r^T$
(1, 5, N, N)
Conv2d (5, 5, 1, 1)
ReLU
Conv2d (5, 1, 1, 1)
Softmax
Adjacency matrix T (1, N, N)
Final matrix = $\frac{1}{T}\sum_1^T A_T + I$
Adjacency matrix (1, N, N)
BMM product
(1, N, F)
Linear layer (F, M)
Knowledge enhanced features (1, N, M)
Concatenated features
B-box regression
Classification
Knowledge driven detection

Figure 3.3.: Depiction of the implicit knowledge module (where N is number of proposals, F is number of features per proposal, M is the size of the module, T is the amount of graphs in the module)

As input for the module, the proposed regions from the RPN are used to generate region coordinate features based on the bounding boxes coordinates of the proposed regions and their corresponding objectness score p_i. The bounding box coordinates are normalized with the image width $\overline{w}$ and height $\overline{h}$:

$$r = \left(\frac{x_1}{\overline{w}}, \frac{y_1}{\overline{h}}, \frac{w}{\overline{w}}, \frac{h}{\overline{h}}, p_i\right). \tag{3.5}$$

The objectness score represents a calculated probability that the proposed region is actually an object. The list of region features is then fed to T smaller neural networks, which are all composed of two convolutional layers. Based on the region features, each neural network builds a region-to-region undirected graph $\hat{G} :=< N, E >$, where N represent region proposal nodes and the edges $e_{i,j} \in E$ a form of spatial knowledge between two nodes. Each network predicts its own adjacency matrix for the input list of region features, representing the relationships between the regions.

The learned edges $\hat{e}^{(t)}_{i,j}$ of the individual graphs $\hat{G}_t, t = 1, ..., T$, are averaged and an identity matrix I is added to obtain the final edge connections $\hat{e}^I_{i,j} \in \hat{E}^I$:

$$\hat{e}^I_{i,j} = \frac{1}{T}\sum_{t=1}^{T} \hat{e}^{(t)}_{ij} + I. \tag{3.6}$$

The adjacency matrix is then multiplied with the list of input features, similarly as in the explicit knowledge module, to output knowledge enhanced features:

$$f_{enhanced} = \hat{E}f. \tag{3.7}$$

The weights of the T neural networks are updated according to the loss of the entire model, and are not supervised independently, since there is no ground-truth available for this module.

The loss of the entire model is an extension of the loss function of Faster R-CNN, described in Equation 2.8, where the MSE losses of the predicted attribute and relational graphs are added to the original loss function:

$$L = L_{RPN,cls} + L_{RPN,reg} + L_{R-CNN,cls} + L_{R-CNN,reg} + L_{Attr} + L_{Rela}. \tag{3.8}$$

While the HKRM framework aims to inform a machine learning model of explicit knowledge, it does not provide the model with reasoning capabilities as humans are capable of. Instead, it aims to guide the model in its quest for optimal settings by imposing an inductive bias on the learning process, both by regularization in the loss function, as well as the addition of extra features in the classification layer. Through regularization, model settings that lead to results that are inconsistent with the described domain will be penalized.

3.2. Dataset description

A dataset of one of BrainCreators' clients, a Dutch telecom provider, is used to test the performance of the implementation. The dataset contains images of cabinets, displayed in Figure 3.4, which were taken with the purpose of training a model which would be able to recognize the objects in the cabinets. An asset inspection employee can then take a picture of the cabinet, and the model will determine whether the items in the objects in the cabinet are present and in good condition. This dataset is chosen because many objects are to be detected in one image, which is what the modules were originally intended for. Additionally, the objects in cabinets are positioned in a highly structured manner, and are therefore very suitable to be described by a knowledge graph, as well as for capturing the spatial relations between objects in the implicit knowledge module. Since the annotations of the objects in the images were not complete, this project included initiating and coordinating a new annotation process, which led to a dataset of 500 fully annotated images with 12 object classes.

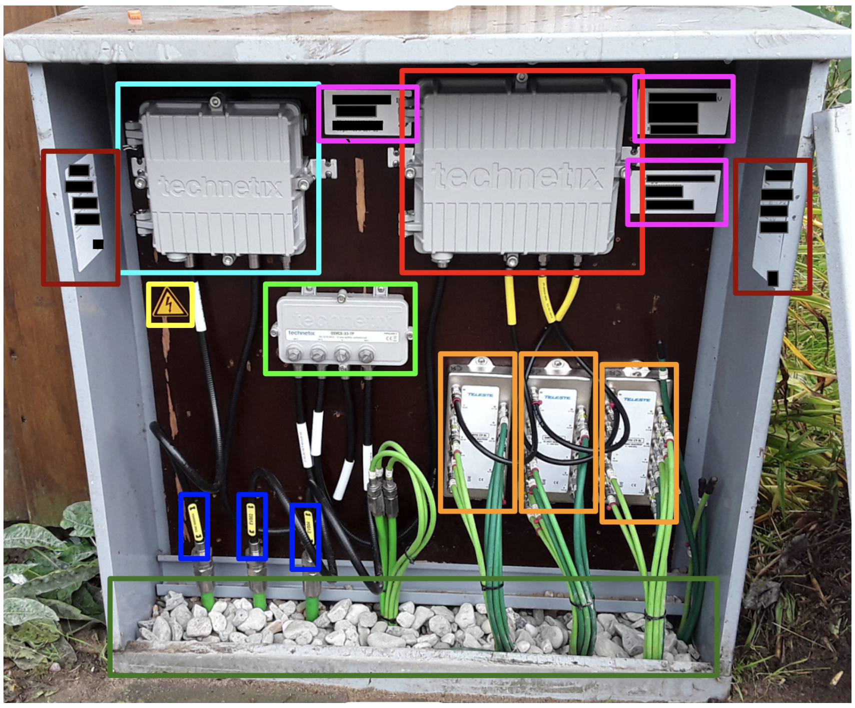


Figure 3.4.: Example image of cabinet inspection dataset with bounding boxes

Similarly to the paper, two knowledge graphs will be constructed, one with attribute knowledge and the other with relationship knowledge. The attribute knowledge graph will describe the objects by their color, shape, material, texture, and other noticeable attributes such as the presence of text or a barcode. The relationship knowledge graph will describe the co-occurrence of objects in the dataset. The edge weights can be deducted in two ways:

- Calculating the frequency of appearance of objects and their attributes and object-object relationships in the dataset. Frequency data is, however, highly reliant on the distribution among object classes in the dataset. If some objects are unrepresented in the dataset, this could lead to inaccurate knowledge representation in the knowledge graph.
- Consult with the domain expert to describe the likelihood of object-attribute and object-object relationships.

In this project, the attribute knowledge graph was constructed with the knowledge gathered from working with the dataset, operating as 'domain expert' in the context. The relational graph was deducted from the the frequency statistics of the dataset. From the two knowledge graphs, two corresponding adjacency matrices are constructed. A full description of the dataset and object attributes can be found in Appendix A.2, and the description of adjacency matrix creation can be found in Appendix A.3.

3.3. Testing

Various tests will be conducted to assess the overall influence of the knowledge modules on the *predictive performance*, the influence of *hyper-parameter* λ, and an *ablation study* will determine whether less training data is required when the knowledge modules are used. The setup of the individual tests is explained below.

3.3.1. Influence of knowledge modules

The performance of the model will be tested without modules, and with various combinations of the explicit (attribute, relational) and implicit (spatial) knowledge modules. The added value of each module and module combinations will be assessed.

Several metrics are used to evaluate the performance of an object detection model. The metrics used are adopted from the Microsoft COCO dataset (Lin et al., 2014).

- **Average precision (AP) and average recall (AR)**
  The *precision* metric measures how accurate the predictions are. It is given by the fraction of true positive predictions out of all the positive predictions a model did.

  The *recall* metric evaluates how many times the model was able to find an object out of all the labeled objects that were present in the data. The mathematical definitions are given by

$$\begin{aligned} \mathsf{Precision} &= \frac{\mathsf{TP}}{\mathsf{TP}+\mathsf{FP}}, \\ \mathsf{Recall} &= \frac{\mathsf{TP}}{\mathsf{TP}+\mathsf{FN}}, \end{aligned} \tag{3.9}$$

  where TP is true positive, TN is true negative, FP is false positive, and FN refers to the false negative samples. To obtain the average precision and recall, the scores are averaged over all object classes.

- **Intersection over Union (IoU)**
  IoU measures the overlap between the predicted bounding box and the ground-truth bounding box, as seen in Figure 3.5.

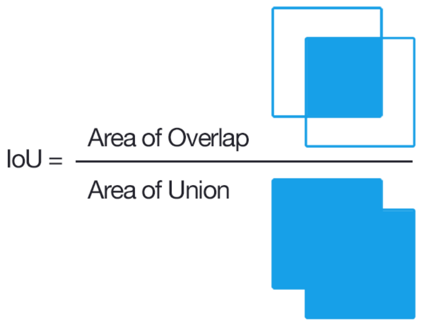


Figure 3.5.: Depiction and equation describing IoU

To further specify the results, the COCO metrics are split up into several categories, which specify results for different levels of IoU and different object areas as seen in Table 3.1. To further explain the notation with two examples: AP50 refers to the average precision calculated for objects with IoU of at least 50 %, whereas ARm refers to the average recall calculated for objects with medium size. In the client dataset used, there are no objects which have an area of $< 32^2$, therefore, measures APs and ARs will not be present in the results.

Table 3.1.: COCO metrics used to evaluate results

Metric	**IoU**	**Area (pixels)**
AP	0.50:0.95	All
AP50	0.5	All
AP75	0.75	All
APs	0.50:0.95	Small ($\mathtt{area} < 32^2$)
APm	0.50:0.95	Medium ($32^2 < \mathtt{area} < 96^2$)
APl	0.50:0.95	Large ($\mathtt{area} > 96^2$)
AR	0.50:0.95	All
ARs	0.50:0.95	Small ($\mathtt{area} < 32^2$)
ARm	0.50:0.95	Medium ($32^2 < \mathtt{area} < 96^2$)
ARl	0.50:0.95	Large ($\mathtt{area} > 96^2$)

The results for each metric will be tested without knowledge modules, with the addition of the attribute, relational, or spatial knowledge, and with all modules together. Each model configuration will be trained 10 times, to make sure the results are reliable, and average results will be reported.

3.3.1.1. Implementation details

As described above, the network used is a Faster R-CNN model with ResNet backbone, to which the knowledge modules have been added. The ResNet backbone network is composed of 50 layers. The model is trained on two GPU's with a batch size of 2. The sizes of the modules are all set to $M_a = M_r = 128$ for the attribute and relational modules, whereas it is set to $M_s = 256$ for the spatial module. The dimension of the original feature vector is $D = 1024$ before concatenating with the module features. Since the knowledge modules should be neither under- nor overrepresented in the final feature vector, the module sizes should be balanced. In the original paper, module sizes of 256 were chosen with an original feature vector size of 2048. The explicit knowledge modules sizes were therefore scaled to match the proportions, however, Jiang et al. (2018) argued that the implicit knowledge module requires a size of 256 to accurately capture spatial relations and has therefore retained its size.

$T = 10$ implicit knowledge modules are used, as the original paper indicated this configuration is best capable of capturing spatial relations. The model is trained for 25 epochs (5000 iterations), with stochastic gradient descent (SGD) as optimizer. A learning rate of 0.01 is used, which is reduced 8 times through multiplying the learning rate with 0.1. Experiments are repeated five times for each model configuration and training set size, all with fixed initialization but random batching, after which results are averaged.

3.3.2. Hyper-parameter λ

The output of the explicit knowledge module will be learned to comply with the values in the adjacency matrix that is used for supervision of the module. The values of the matrix representing the relationships between objects are either frequency statistics or estimations, and are therefore most likely not a completely accurate representation of the domain. Forcing the module towards the values in the adjacency matrix might therefore not be beneficial to the model prediction. Therefore, a hyperparameter λ is introduced in the loss function,

$$L = L_{RPN,cls} + L_{RPN,reg} + L_{R-CNN,cls} + L_{R-CNN,reg} + \lambda(L_{Attr} + L_{Rela}), \quad (3.10)$$

to limit the influence of the domain knowledge on the final model predictions. The influence of hyper-parameter λ is tested with values of 1, 0.1 and 0.01, and average precision and recall will be evaluated.

3.3.3. Ablation study

If the model is guided in its quest to find relationships in the data through the addition of domain knowledge, the model could potentially require less training samples to learn to recognize objects. This test should assess whether this hypothesis is actually correct and, potentially, the extent of the improvement. The training dataset was reduced in size gradually, while maintaining the same distribution of classes. The goal is to then find a break-even point.

Out of the original training set, subsets of 80, 50 and 20 percent of data were selected, while maintaining a similar distribution of object classes. The exact details of the training sets can be found in Appendix A.2. The learning rate schedule outlined in Section 3.3.1.1 was scaled to match the subset, and ensure that all datasets were trained for a similar number of epochs. Average precision and average recall of a model with all knowledge modules and model without knowledge modules is compared.

3.3.4. Hypotheses

Several hypotheses regarding the performance of the model will be formed, in order to evaluate the results.

HP1 The presence of the knowledge modules will improve the overall precision
Providing the model with reasoning capabilities should allow the model to take advantage of object similarity and different to detect objects more accurately.

HP2 The presence of the knowledge modules will improve the overall recall
Imposing inductive bias on the model and thereby guiding it with explicit knowledge should allow the model to take advantage of known co-occurrence and similarities of objects, and its learned spatial layouts from the implicit knowledge module, to detect more object instances. To give an example, some objects can only occur together with another object, since one is located on the other.

HP3 The presence of the knowledge modules will lower the amount of training data required to achieve similar average precision, compared to a model without knowledge modules trained on all of the training data
The framework should reduce the reliance of the model on training data to learn patterns, and therefore the number of examples required could decrease.

HP4 The quality of the knowledge graph and its edge weights will be of significant influence on the impact of the modules
The knowledge enhanced features comprise a large part of of the final feature vector that is fed to the classification layer, and therefore has a big influence on the final prediction. In case the edge weights of the knowledge graphs are not valued in the correct way, the model might be taught invalid patterns.

4. Results

4.1. Influence of knowledge modules

In order to evaluate the influence of the knowledge modules and individual knowledge graphs on the client dataset, multiple model configurations with the presence of the attribute (Attr.), relational (Rela.), and spatial (Spat.) knowledge modules were tested. Table 4.1 first outlines the baseline results on average precision, achieved with a plain Faster R-CNN network on the dataset. The model with all knowledge modules scores highest on AP, however, this is achieved by a 0.007% margin and is therefore far from significant. Figure 4.1 depicts the difference in AP for all model configurations, with corresponding standard deviation obtained from averaging the results of five training runs. The standard deviation of the model with all modules indicates that the difference of AP compared to the baseline could be attributed to deviation across runs.

The model with all knowledge modules also scores highest on AP50, whereas the baseline model achieves the highest scores on AP75 and APl, although all with small margins. The model with all knowledge modules achieved the highest APm score, indicating that smaller objects were detected more often, with a margin of 0.557%. The model has most likely had to give up precision in other areas to achieve this though, as the total AP does not reflect the same score. The individual presence of a knowledge module did not significantly improve the score on all of the AP measures.

Table 4.1.: Comparison of average precision with various model configurations, showing performance and the deviation from the baseline (all in %)

Modules	**Module size**			**λ**	**AP**	**AP50**	**AP75**	**APl**	**APm**
	Attr.	*Rela.*	*Spat.*		*(%)*	*(%)*	*(%)*	*(%)*	*(%)*
None					45.923	77.139	**50.205**	**46.303**	15.165
All	128	128	256	1	**45.930**	**77.312**	50.130	46.257	**15.722**
					(0.007)	*(0.172)*	*(-0.076)*	*(-0.046)*	*(0.557)*
Attr.	128				45.750	77.058	49.825	46.196	15.392
					(-0.174)	*(-0.082)*	*(-0.380)*	*(-0.107)*	*(0.227)*
Rela.		128			45.680	77.186	49.307	45.932	15.029
					(-0.243)	*(0.047)*	*(-0.899)*	*(-0.371)*	*(-0.136)*
Spat.			256		45.839	77.155	50.114	46.171	15.511
					(-0.084)	*(0.015)*	*(-0.091)*	*(-0.133)*	*(0.346)*

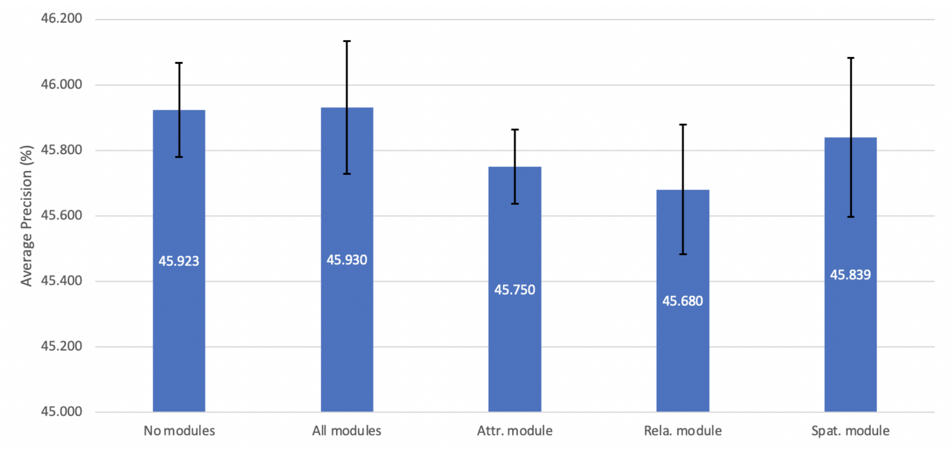


Figure 4.1.: Average precision for model several model configurations with and without knowledge modules

Table 4.2 shows the scores and deviation from the baseline for multiple model configurations, when evaluating the average recall. The baseline model performs best for AR and ARm, although again with tight margins. Comparing to the model with all knowledge modules, the difference cannot be attributed to standard deviation, as displayed in Figure 4.2. However, the difference with the model with attribute knowledge module can be connected to the standard deviation. The model containing solely the spatial knowledge module achieves the highest score on ARl, with a margin of 0.18%.

Table 4.2.: Comparison of average recall with various model configurations, showing performance and the deviation from the baseline (all in %)

Modules	**Module size**			λ	**AR**	**ARl**	**ARm**
	Attr.	*Rela.*	*Spat.*		*(%)*	*(%)*	*(%)*
None					**54.70**	27.02	**54.26**
All	128	128	256	1	53.90	26.62	53.30
					(-0.80)	*(-0.40)*	*(-0.96)*
Attr.	128			1	54.56	27.08	54.08
					(-0.14)	*(0.06)*	*(-0.18)*
Rela.		128		1	54.50	26.86	53.86
					(-0.20)	*(-0.16)*	*(-0.40)*
Spat.			256		54.46	**27.20**	53.86
					(-0.24)	*(0.18)*	*(-0.40)*

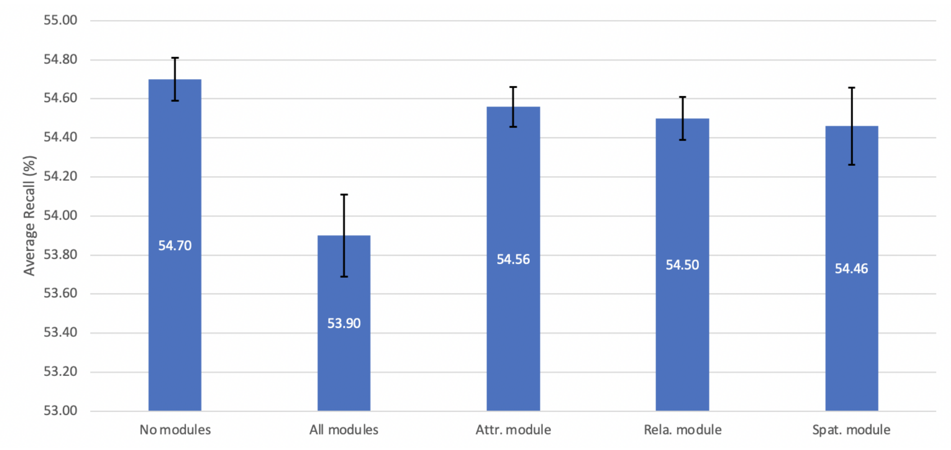


Figure 4.2.: Average recall for model several model configurations with and without knowledge modules

In conclusion, the influence of the knowledge modules on improving average precision and recall is, even if present in a few of the metrics, not significant enough to state that the knowledge modules have a positive effect on the model performance. The margins are so small that the differences might have simply been caused by the variance in training run results, which is always present since optimization does not necessarily lead to one optimal state.

4.2. Influence of hyperparameter λ

Hyperparameter λ was introduced to reduce the influence of the loss over the attribute and relational graph in the total loss of the model. Table 4.3 shows the AP scores achieved for models with various λ values. The first line of the table again shows the baseline results achieved with a plain Faster R-CNN model, and rows below show AP scores and the deviation from the baseline. The model with all knowledge modules modules and λ value of 0.1 is shown to obtain the highest score in AP, AP50, APl, and APm. On AP75, the model does not outperform the baseline model. The margins of improvement or deterioration are again minimal, with values of around 0.25%, except for the improvement on APm, which has a value of 0.641%.

Table 4.3.: Comparison of average precision for models with different λ values, showing performance and the deviation from the baseline (all in %)

Modules	**Module size**			**λ**	**AP**	**AP50**	**AP75**	**APl**	**APm**
	Attr.	*Rela.*	*Spat.*		*(%)*	*(%)*	*(%)*	*(%)*	*(%)*
None					45.923	77.139	50.205	46.303	15.165
All	128	128	256	1	45.930	77.312	50.130	46.257	15.722
					(0.007)	*(0.172)*	*(-0.076)*	*(-0.046)*	*(0.557)*
All	128	128	256	0.1	**46.167**	**77.482**	49.972	**46.553**	**15.806**
					(0.244)	*(0.342)*	*(-0.233)*	*(0.249)*	*(0.641)*
All	128	128	256	0.01	46.109	77.206	**50.328**	46.536	15.545
					(0.185)	*(0.066)*	*(0.123)*	*(0.233)*	*(0.380)*
All	64	64	64	1	45.871	77.393	49.473	46.334	15.344
					(-0.052)	*(0.254)*	*(-0.732)*	*(0.030)*	*(0.178)*

Evaluating the obtained results on AP with the standard deviation of training runs, as depicted in Figure 4.3, it is again evident that there is overlap in the ranges of the baseline model and the model with $\lambda = 0.1$. Therefore, the results are not conclusive.

In order to evaluate the influence of the module size on the performance of the model, a test run was also performed with a module size of 64 for all knowledge modules. As shown in Table 4.3, a smaller module size did not yield more favorable results than the larger module size.

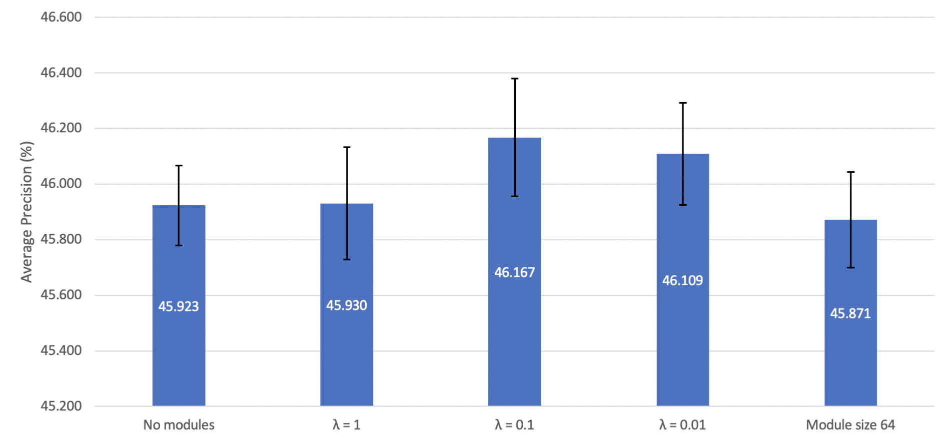


Figure 4.3.: Average precision for model configurations with various levels of λ in the loss function

In Table 4.4, the models with different λ values are compared on the average recall scores. The baseline model outperforms the knowledge enhanced modules with various levels of λ on AR and ARm, while the baseline model is outperformed by the model with module size 64 on ARl with a very tight margin. Figure 4.4 again shows that the results are inconclusive due to the standard deviation.

Table 4.4.: Comparison of average recall for models with different λ values, showing performance and the deviation from the baseline (all in %)

Modules	**Module size**			**λ**	**AR**	**ARl**	**ARm**
	Attr.	*Rela.*	*Spat.*		*(%)*	*(%)*	*(%)*
None					**54.70**	27.02	**54.26**
All	128	128	256	1	53.90	26.62	53.30
					(-0.80)	*(-0.40)*	*(-0.96)*
All	128	128	256	0.1	54.40	27.04	53.82
					(-0.30)	*(0.02)*	*(-0.44)*
All	128	128	256	0.01	54.20	26.94	53.56
					(-0.50)	*(-0.08)*	*(0.70)*
All	64	64	64	1	54.66	**27.24**	54.20
					(-0.04)	*(0.22)*	*(-0.06)*

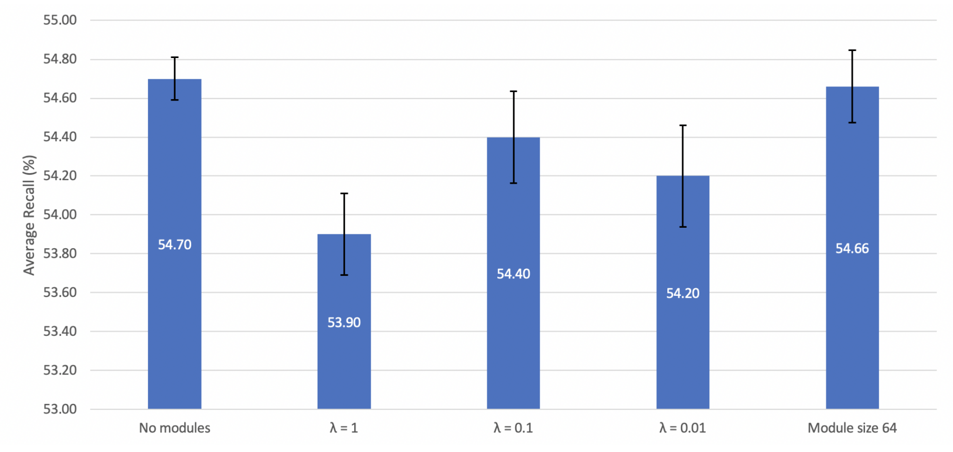


Figure 4.4.: Average recall for model configurations with various levels of λ in the loss function

Although the model with λ value of 0.1 was able to outperform the baseline model in terms of average precision on nearly all AP metrics, the margins are, again, too small to be of real significance. Since the results on AR do not reflect any significant improvement in performance either, it can be concluded that even with the presence of a balancing hyperparameter, the influence of the knowledge modules is not deemed positive in the application to the client dataset.

4.3. Ablation study

Several experiments were conducted to assess the influence of the knowledge modules on the amount of training data required to learn the correct patterns in the data. Figure 4.5 depicts a comparison of AP score with a plain Faster R-CNN model and a model with all the knowledge modules, trained with 100, 80, 50 and 20% of the training data. While the AP scores are nearly identical for 100 and 20% of the data, small differences in performance are visible for subsets of 80 and 50% of the data. However, these results contradict each other, therefore not allowing for a clear conclusion.

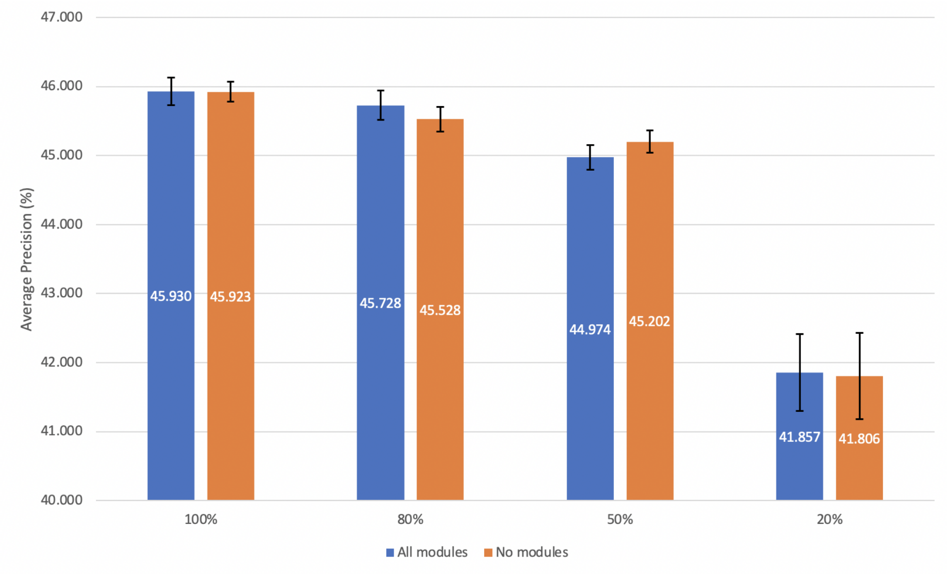


Figure 4.5.: Average precision for model configurations with and without modules, trained on various percentages of the original training data

Figure 4.6 depicts the AR score of the baseline model and the model with all modules, depicting the same contradicting results as where obtained for AP.

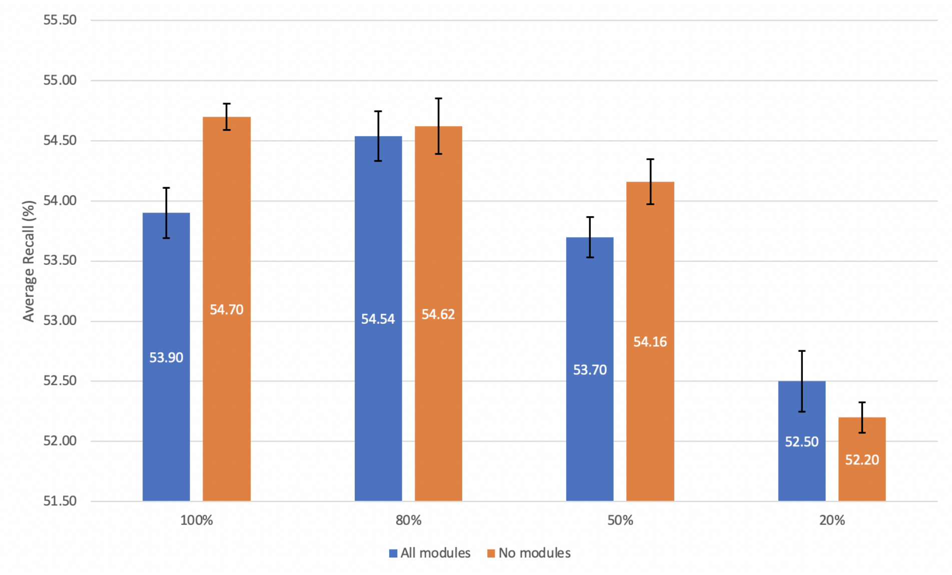


Figure 4.6.: Average recall for model configurations with and without modules, trained on various percentages of the original training data

The purpose of this test was to assess whether the same predictive performance could be upheld if training data was removed and knowledge was transferred through the knowledge modules. However, since the performance decreases with similar steps for both the baseline model and the model with all knowledge modules, it can be concluded that the model is not capable of transferring knowledge through the modules well enough to compensate for training data. Tables with results for all AP and AR metrics can be found in Appendix A.4.1.

5. Discussion

This project has aimed to explore available frameworks that integrate domain knowledge in object detection pipeline, and to design a modular software deliverable that can be used in future client cases. While designing the artifact, other design choices that had to be made where concerned with the client dataset annotation and evaluation, as well as the knowledge graph creation. Below, the results of Section 4 will be evaluated using the previously defined hypotheses.

The results in Section 4 have shown that the addition of knowledge modules to a Faster R-CNN object detection network did not yield favorable results on the telecom client dataset, thereby showing that hypotheses 1, 2, and 3 could not be validated. The hypotheses refer to, respectively, an increase in average precision and recall, and a decrease in the amount of training data required for similar accuracy. All of these expected results could not be obtained in the use case that was tested, as results for average precision and recall were not or not consistently higher than the results for a plain Faster R-CNN network. Therefore, no clear conclusion can be drawn, and the method does not offer enough improvement and generalization to apply to other use cases with confidence.

Hypothesis 4 relates to the influence of the quality of the knowledge graphs and edge weights on the impact of the knowledge modules on the final model accuracy. This hypothesis can neither be deemed fully valid or invalid, since it is only possible to speculate the reason for the results obtained regarding the influence of the modules, hyper-parameter λ, and the ablation study. Jiang et al. (2018) were able to obtain an improvement of 3.6% in average precision compared to Faster R-CNN results on benchmark dataset Microsoft COCO, and those results have been replicated with this implementation of the framework on the same dataset. However, there are a few other differences in the manner of implementation in the paper and the application of the client dataset in this project that could be a potential cause for the results obtained:

1. **Quality of the knowledge graph**
   Jiang et al. (2018) used a large-scale knowledge graph based on the annotations of the VisualGenome dataset, which describes many attributes of objects and relationships between objects. The edge values of the graphs were obtained by using the frequency statistics of appearance of object-object or object-attribute relationships in the images of the dataset. Since the client case application was based on a custom made graph, the edge values were obtained in a different manner, which could potentially lead to a different performance. For the attribute graph, the similarity of objects was still obtained by computing the Jensen-Shannon divergence over all attributes an object has, but the graph was not enriched by counting frequency of appearance of object-attribute relationships.

   The relational graph was created in an entirely different manner than in the paper, which was based on the same principle of counting the appearance of object-object relationships, described in the knowledge graph, in images. For the application on the client case, the graph was constructed through counting the frequency of co-occurrence of objects in images and defining a probability of occurrence of object A if B was present in the image.

These changes in graph creation could potentially lead to a representation of the domain that is not as accurate as the knowledge graph used in the paper, therefore lowering the quality of the knowledge inserted in the object detection pipeline.

2. **Quality of the dataset**
   The client dataset used in the application and tests was annotated for this project with the help of the BrainCreators team. The final dataset created consisted of 500 images, which is extremely small when comparing it to Microsoft COCO which consists of 118.000 images. It could potentially be the case that the quality of the dataset was therefore too low for the model and modules to effectively identify the patterns in the data and generalize well enough to be of real significance in towards the final model accuracy. Quality of annotations of objects in the dataset can also play a more significant role in small datasets, since the model could essentially be fed biased data.

3. **The framework**
   The authors of the HKRM framework have created it to alleviate problems faced with large-scale object detection tasks, while the client case application was focused on a limited set of 12 objects. Although there were always multiple objects present in each image, this cannot be considered a large-scale object detection dataset. It could potentially be the case that the knowledge graph and the use of knowledge modules, in the large-scale object detection dataset, helps the object detection network to identify sub-domains and differentiate between contexts within the dataset in which object classes appear, through taking into account object similarity and relationships between objects.
   Through learning these contexts, the modules can help the model to find objects that also often appear in this context. Contrarily, the client dataset used in the application consisted of objects that all appear in the same context, and therefore the knowledge modules might not be able to guide the model to differentiate contexts, limiting its influence.

BrainCreators' aim was to formalize and improve the knowledge transfer process, by exploring other methods to transfer knowledge from domain to the BrainMatter platform. The application on this client dataset is not capable of such knowledge transfer, therefore, there is currently no added value for BrainCreators in the application of the framework on this client case. The added value of this project lies, therefore, in the first steps that were taken in the exploration of hybrid machine learning methods for BrainCreators. In doing so, many lessons were learned with regard to the challenges faced in creating knowledge graphs for client domain applications, the creation of valuable datasets and corresponding process of annotating the data, and the implementation of the hybrid frameworks in the currently used object detection pipelines in BrainMatter.

BrainCreators can continue this line of research by further formalizing the knowledge transfer procedure from client domain to the BrainMatter platform, through further research into knowledge graph creation and hybrid machine learning frameworks. Potential areas of research include frameworks that perform graph propagation such as the work of Marino et al. (2017) or frameworks that impose logical constraints on the training process such as the work of Donadello et al. (2017).

6. Conclusion

The goal of this project was to find methods which could improve and expand the knowledge transfer process. The HKRM framework was chosen because of its modular character and since it allowed to incorporate a custom made knowledge graph, which the use cases require. Even though preliminary experiments showed promise, and the findings of the original paper on the Microsoft COCO dataset were reproduced with the new implementation of the framework, unfortunately, the results did not persist on more challenging industrial data. As there was not always improvement in performance visible with the addition of the knowledge modules and the results were not consistent, no conclusion can be drawn regarding the effectiveness of the framework. Therefore, the desired business outcomes, referring to an improved knowledge transfer process leading to potentially requiring less training data, were not attained. This project was, however, aimed at performing an exploration for BrainCreators, outlining methods available and testing one of those methods. While choosing the HKRM framework involved certain risks from the start, the relative ease of implementation of the framework in the knowledge transfer process was considered to be worth the risk. Further testing and more in depth research regarding hybrid machine learning frameworks should determine whether the desired business outcomes can be attained in the future.

Bibliography

F. Ackermann and C. Eden. Strategic management of stakeholders: Theory and practice. *Long range planning*, 44(3):179–196, 2011.

I. Donadello, L. Serafini, and A. D. Garcez. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968*, 2017.

X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

Y. Fang, K. Kuan, J. Lin, C. Tan, and V. Chandrasekhar. Object detection meets knowledge graphs. *IJCAI International Joint Conference on Artificial Intelligence*, 0:1661–1667, 2017. ISSN 10450823. doi: 10.24963/ijcai.2017/230.

V. Fionda and L. Palopoli. Biological network querying techniques: analysis and comparison. *Journal of Computational Biology*, 18(4):595–625, 2011.

R. Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 International Conference on Computer Vision, ICCV 2015: 1440–1448, 2015. ISSN 15505499. doi: 10.1109/ICCV.2015.169.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:770–778, 2016. ISSN 10636919. doi: 10.1109/CVPR.2016.90.

C. Jiang, X. Liang, H. Xu, and L. Lin. Hybrid Knowledge Routed Modules for Large-scale Object Detection. *Advances in Neural Information Processing Systems*, 2018-Decem(Nips):1552–1563, 2018. ISSN 10495258.

Kejriwal. *Domain-Specific Knowledge Graph.* 2019. ISBN 9783030123741.

R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision*, 123(1):32–73, 2017.

T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

H. Liu and P. Singh. Conceptnet—a practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226, 2004.

K. Marino, R. Salakhutdinov, and A. Gupta. The more you know: using knowledge graphs for image classification. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:20–28, 2017. doi: 10.1109/CVPR.2017.10.

U. Michelucci. *Advanced applied deep learning: Convolutional neural networks and object detection*. 2019. ISBN 9781484249765. doi: 10.1007/978-1-4842-4976-5.

H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.

S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

S. Saha. A comprehensive guide to convolutional neural networks-the eli5 way, Dec 2018. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-\way-3bd2b1164a53.

H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *Sixth international conference on data mining (ICDM'06)*, pages 613–622. IEEE, 2006.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, M. Walczak, J. Garcke, C. Bauckhage, and J. Schuecker. Informed Machine Learning – A Taxonomy and Survey of Integrating Knowledge into Learning Systems. pages 1–20, 2019. URL http://arxiv.org/abs/1903.12394.

Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

A. Appendix

A.1. Machine learning terminology

- **Features**
  The features referred to in this project in the context of object detection refer to the identifiers the object detection network has linked to the object classes in the dataset. Features could be described as the result of filter operations on the image, followed by multiple other operations such as the application of activation functions and pooling. The same fixed list of features is applied to every object proposal, and a numerical vector representing the relevance of each feature for the object proposal is called the feature vector, also referred to as features.

- **Multi-layer perceptron**
  The term multi-layer perceptron represents a class of feed-forward neural networks composed of only one or a few hidden layers, as depicted in Figure A.1.

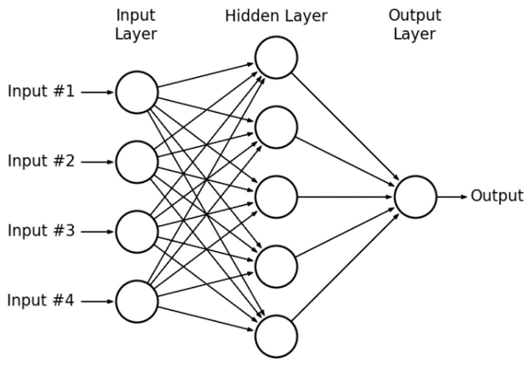


Figure A.1.: Multi-layer perceptron

- **Hyper-parameters**
  Hyper-parameters are parameters that are used to control the training process of a neural network. Much effort is put into finding an optimal set of hyper-parameters for a training process, as they can greatly influence the outcome of the results. An example of a hyper-parameter is the learning rate used in training. A learning rate that is too high will not allow the model to optimally find local or global loss minima, while a learning rate that is too low will not be effective at decreasing the model loss.

- **Model parameters**
  On the other hand, model parameters are derived from the training procedure. They are the optimal settings a model has found to converge to the final result, such the weights of the neural network's node weights.

A.2. Dataset description

Table A.1.: Description of the visual features of all objects in the dataset, as they were observed.

Objects	**Description**
Address label	White sticker which states the address of the cabinet, located against the backboard of the cabinet
Resistance cap	Resistance placed on top of 'Multi-tap' in case there are no cables connected to the plugs
Amplifier 1	Grey, square amplifier box, located somehwat in the center of the cabinet. Nearly identical to 'Amplifier 2'
Amplifier 2	Grey, rectangular amplifier box, located somewhat in the center of the cabinet. Nearly identical to 'Amplifier 1'
Rock filling	The presence of rocks at the bottom of the cabinet, to fill up the cabinet
Label power cable	Yellow labels around power cables, located at the bottom of the cables and cabinet
Cabinet label	White or yellow, rectangular label stating a code. Located against the backboard of the cabinet
Measurement sticker	White, rectangular sticker with information regarding the cabinet. Located on either the left or right sideboard of the cabinet
Multi-tap	Silver, rectangular box out of which the power is tapped with multiple cables. Located against the backboard
Power splitter 1	Grey, rectangular box with serves as a power splitter. Located against the backboard of the cabinet, below the amplifiers
Power sticker	Triangular, yellow sticker with black sign for power, located against the backboard of the cabinet
Power splitter 2	Silver splitter, which has a blue or white sticker on top. Located against the backboard of the cabinet

Table A.2.: Description of the count and area sizes of all objects in the dataset

Object name	**Count**	**Min area** (pixels^2)	**Max area** (pixels^2)	**Average area** (pixels^2)	**% of Data** (%)
Address label	505	138	576	310	6.36
Resistance cap	1585	24	176	72	19.97
Amplifier 1	660	327	1528	819	8.32
Amplifier 2	68	641	1723	972	0.86
Rock filling	432	252	1890	891	5.44
Label power cable	1391	37	378	128	17.53
Cabinet label	247	65	451	186	3.11
Measurement sticker	661	182	999	424	8.33
Multi-tap	1175	189	934	490	14.80
Power splitter 1	309	264	1049	456	3.89
Power sticker	689	73	410	205	8.68
Power splitter 2	215	101	560	285	2.71

Table A.3.: Description of the count and percentage of the data the objects occupy in the entire train set and its subsets, in order to show the equal distribution.

	100%		**80%**		**50%**		**20%**	
Object name	**Count**	**Data (%)**	**Count**	**Data (%)**	**Count**	**Data (%)**	**Count**	**Data (%)**
Address label	409	6.36%	331	6.41%	202	6.07%	79	5.75%
Resistance cap	1335	20.75%	1076	20.85%	744	22.34%	285	20.73%
Amplifier 1	529	8.22%	424	8.22%	273	8.20%	113	8.22%
Amplifier 2	56	0.87%	44	0.85%	30	0.90%	10	0.73%
Rock filling	351	5.46%	276	5.35%	177	5.32%	73	5.31%
Label power cable	1124	17.47%	909	17.62%	546	16.40%	241	17.53%
Cabinet label	192	2.98%	152	2.95%	107	3.21%	44	3.20%
Measurement sticker	531	8.25%	425	8.24%	267	8.02%	113	8.22%
Multi-tap	930	14.45%	739	14.32%	486	14.59%	206	14.98%
Power splitter 1	263	4.09%	214	4.15%	127	3.81%	57	4.15%
Power sticker	550	8.55%	440	8.53%	280	8.41%	113	8.22%
Power splitter 2	164	2.55%	130	2.52%	91	2.73%	41	2.98%

A.3. Knowledge Graph Creation

A.3.1. Attribute graph

In order to create the attribute graph, the attributes are described by their visual traits, such as color, material, and other specifications. A matrix is created, depicted in Figure A.2, where attributes are assigned to objects. If the object always appears with that attribute, a value of 1 will be assigned. In case an attribute of an object varies, such as the color of 'Cabinet label', which can be either white or yellow, both get assigned a value of 0.5. The next step is to perform row normalization

	Color							Material					Shape						Features		
	Grey	White	Black	Silver	Yellow	Blue	Brown	Stone	Metal	Plastic	Sticker	Rubber	Square	Round	Rectangular	Triangular	Sharp corners	Round corners	Ridges	Barcode	Text
Address label		1									1				1			1		1	1
Resistance cap				1										1							
Amplifier 1	1								1				1					1	1		
Amplifier 2	1								1						1			1	1		
Rock filling	1						1	1						1							
Label power cable			1		1					1					1						1
Cabinet label		0.5			0.5					1					1						1
Measurement sticker		1									1				1			1			1
Multi-tap		1		1					1						1		1				1
Power splitter 1	1								1						1			1			1
Power sticker			1		1						1					1		1			
Power splitter 2		0.5		1		0.5			1						1		1				1

Figure A.2.: Step 1 of attribute graph creation

over all assigned attributes. The third step is to compute the Jensen-Shannon

	Color							Material					Shape						Features		
	Grey	White	Black	Silver	Yellow	Blue	Brown	Stone	Metal	Plastic	Sticker	Rubber	Square	Round	Rectangular	Triangular	Sharp corners	Round corners	Ridges	Barcode	Text
Address label	0	0.17	0	0	0	0	0	0	0	0	0.17	0	0	0	0.17	0	0	0.17	0	0.17	0.17
Resistance cap	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0
Amplifier 1	0.2	0	0	0	0	0	0	0	0.2	0	0	0	0.2	0	0	0	0	0.2	0.2	0	0
Amplifier 2	0.2	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0.2	0	0	0.2	0.2	0	0
Rock filling	0.25	0	0	0	0	0	0.25	0.25	0	0	0	0	0	0.25	0	0	0	0	0	0	0
Label power cable	0	0	0.2	0	0.2	0	0	0	0	0.2	0	0	0	0	0.2	0	0	0	0	0	0.2
Cabinet label	0	0.13	0	0	0.13	0	0	0	0	0.25	0	0	0	0	0.25	0	0	0	0	0	0.25
Measurement sticker	0	0.2	0	0	0	0	0	0	0	0	0.2	0	0	0	0.2	0	0	0.2	0	0	0.2
Multi-tap	0	0.17	0	0.17	0	0	0	0	0.17	0	0	0	0	0	0.17	0	0.17	0	0	0	0.17
Power splitter 1	0.2	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0.2	0	0	0.2	0	0	0.2
Power sticker	0	0	0.2	0	0.2	0	0	0	0	0	0.2	0	0	0	0	0.2	0	0.2	0	0	0
Power splitter 2	0	0.08	0	0.17	0	0.08	0	0	0.17	0	0	0	0	0	0.17	0	0.17	0	0	0	0.17

Figure A.3.: Step 2 of attribute graph creation

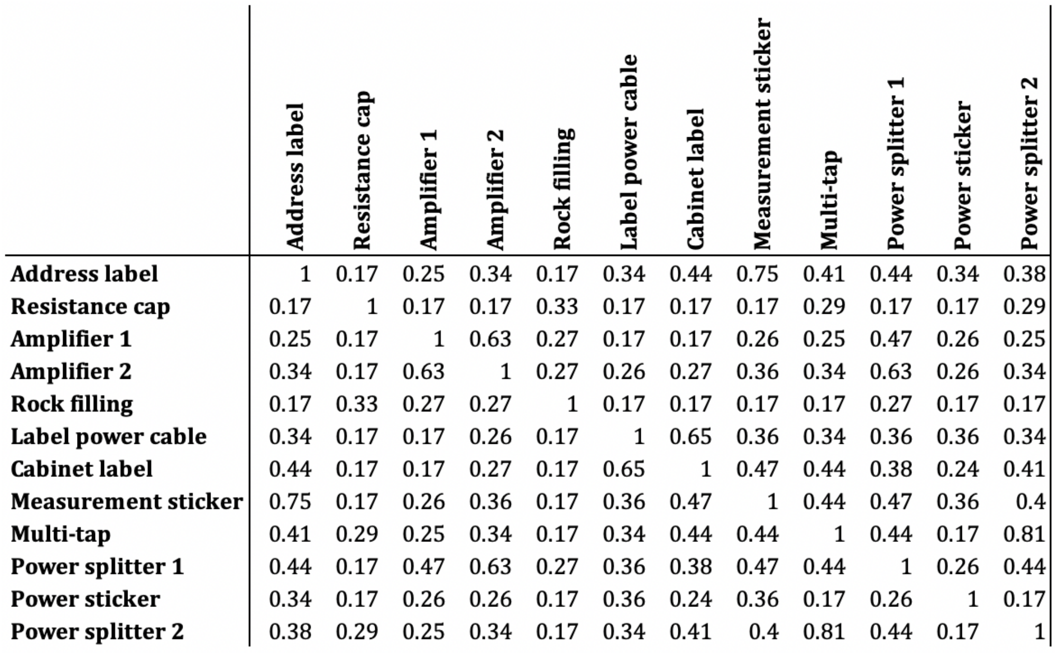
divergence, which calculates the difference between probability distributions, thereby obtaining the difference between objects described by their attributes. To compute the Jensen-Shannon divergence, first the Kullback-Leibler divergence should be computed, comparing probability distributions P and Q, given by

$$\mathsf{KL}(\mathsf{P}||\mathsf{Q}) = \sum_{x \in X} \mathsf{P}(x) \ln(\frac{\mathsf{P}(x)}{\mathsf{Q}(x)}). \tag{A.1}$$

The Jensen-Shannon divergence is computed by

$$\begin{gathered} \mathsf{JS}(\mathsf{P}||\mathsf{Q}) = \frac{1}{2}\mathsf{KL}(\mathsf{P}||\mathsf{M}) + \frac{1}{2}\mathsf{KL}(\mathsf{Q}||\mathsf{M}), \\ \text{where} \quad \mathsf{M} = \frac{1}{2}(\mathsf{P}+\mathsf{Q}). \end{gathered} \tag{A.2}$$

The final attribute graph is then obtained by subtracting the obtained result from a matrix of ones, resulting in the adjacency matrix depicted in Figure A.4.



	Address label	Resistance cap	Amplifier 1	Amplifier 2	Rock filling	Label power cable	Cabinet label	Measurement sticker	Multi-tap	Power splitter 1	Power sticker	Power splitter 2
Address label	1	0.17	0.25	0.34	0.17	0.34	0.44	0.75	0.41	0.44	0.34	0.38
Resistance cap	0.17	1	0.17	0.17	0.33	0.17	0.17	0.17	0.29	0.17	0.17	0.29
Amplifier 1	0.25	0.17	1	0.63	0.27	0.17	0.17	0.26	0.25	0.47	0.26	0.25
Amplifier 2	0.34	0.17	0.63	1	0.27	0.26	0.27	0.36	0.34	0.63	0.26	0.34
Rock filling	0.17	0.33	0.27	0.27	1	0.17	0.17	0.17	0.17	0.27	0.17	0.17
Label power cable	0.34	0.17	0.17	0.26	0.17	1	0.65	0.36	0.34	0.36	0.36	0.34
Cabinet label	0.44	0.17	0.17	0.27	0.17	0.65	1	0.47	0.44	0.38	0.24	0.41
Measurement sticker	0.75	0.17	0.26	0.36	0.17	0.36	0.47	1	0.44	0.47	0.36	0.4
Multi-tap	0.41	0.29	0.25	0.34	0.17	0.34	0.44	0.44	1	0.44	0.17	0.81
Power splitter 1	0.44	0.17	0.47	0.63	0.27	0.36	0.38	0.47	0.44	1	0.26	0.44
Power sticker	0.34	0.17	0.26	0.26	0.17	0.36	0.24	0.36	0.17	0.26	1	0.17
Power splitter 2	0.38	0.29	0.25	0.34	0.17	0.34	0.41	0.4	0.81	0.44	0.17	1

Figure A.4.: Step 3 of attribute graph creation

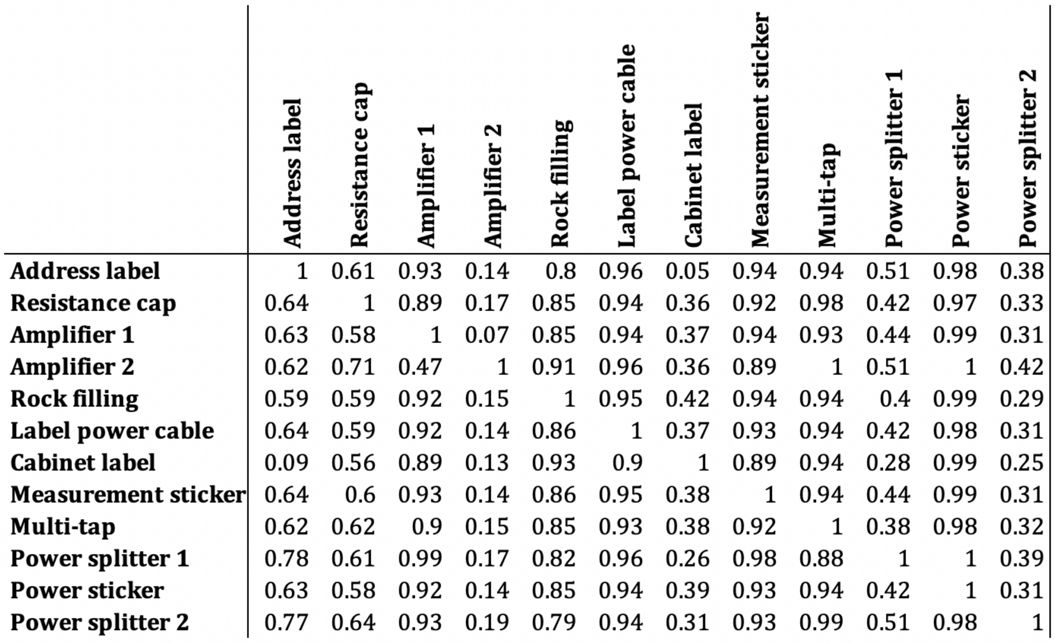
A.3.2. Relational graph

The relational graph is obtained through first counting the co-occurrence of objects in images. If the relationship of object A to object B exists in an images, a value of 1 will be added to the corresponding matrix field. This leads to the frequency table depicted in Figure A.5.

	Address label	**Resistance cap**	**Amplifier 1**	**Amplifier 2**	**Rock filling**	**Label power cable**	**Cabinet label**	**Measurement sticker**	**Multi-tap**	**Power splitter 1**	**Power sticker**	**Power splitter 2**
Address label	245	149	227	34	197	235	13	231	230	126	241	93
Resistance cap	149	233	208	39	197	218	85	215	229	98	225	77
Amplifier 1	227	208	358	26	304	337	134	335	334	159	354	112
Amplifier 2	34	39	26	55	50	53	20	49	55	28	55	23
Rock filling	197	197	304	50	332	317	140	311	313	132	329	95
Label power cable	235	218	337	53	317	368	136	341	345	155	362	114
Cabinet label	13	85	134	20	140	136	151	135	142	42	149	37
Measurement sticker	231	215	335	49	311	341	135	360	339	158	358	113
Multi-tap	230	229	334	55	313	345	142	339	370	142	363	120
Power splitter 1	126	98	159	28	132	155	42	158	142	161	161	62
Power sticker	241	225	354	55	329	362	149	358	363	161	385	119
Power splitter 2	93	77	112	23	95	114	37	113	120	62	119	121

Figure A.5.: Step 1 of relational graph creation

The relational knowledge graph is a directed graph, since a chance of co-occurrence does not necessarily have be valid in both ways. For example, some objects are located on top of other objects and can therefore not appear without them. Therefore, row normalization is performed on the obtained frequency table in step 1, which leads to the result depicted in Figure A.6. This way, looking from the perspective of the rows, if the object the row belongs to appears, the probabilities in the row indicate the chance that the object belonging to the column will appear as well.



	Address label	Resistance cap	Amplifier 1	Amplifier 2	Rock filling	Label power cable	Cabinet label	Measurement sticker	Multi-tap	Power splitter 1	Power sticker	Power splitter 2
Address label	1	0.61	0.93	0.14	0.8	0.96	0.05	0.94	0.94	0.51	0.98	0.38
Resistance cap	0.64	1	0.89	0.17	0.85	0.94	0.36	0.92	0.98	0.42	0.97	0.33
Amplifier 1	0.63	0.58	1	0.07	0.85	0.94	0.37	0.94	0.93	0.44	0.99	0.31
Amplifier 2	0.62	0.71	0.47	1	0.91	0.96	0.36	0.89	1	0.51	1	0.42
Rock filling	0.59	0.59	0.92	0.15	1	0.95	0.42	0.94	0.94	0.4	0.99	0.29
Label power cable	0.64	0.59	0.92	0.14	0.86	1	0.37	0.93	0.94	0.42	0.98	0.31
Cabinet label	0.09	0.56	0.89	0.13	0.93	0.9	1	0.89	0.94	0.28	0.99	0.25
Measurement sticker	0.64	0.6	0.93	0.14	0.86	0.95	0.38	1	0.94	0.44	0.99	0.31
Multi-tap	0.62	0.62	0.9	0.15	0.85	0.93	0.38	0.92	1	0.38	0.98	0.32
Power splitter 1	0.78	0.61	0.99	0.17	0.82	0.96	0.26	0.98	0.88	1	1	0.39
Power sticker	0.63	0.58	0.92	0.14	0.85	0.94	0.39	0.93	0.94	0.42	1	0.31
Power splitter 2	0.77	0.64	0.93	0.19	0.79	0.94	0.31	0.93	0.99	0.51	0.98	1

Figure A.6.: Step 2 of relational graph creation

A.4. Additional results

A.4.1. Ablation study

% of Data	Modules	AP (%)	AP50 (%)	AP75 (%)	APl (%)
100	All	**45.930**	**77.312**	50.130	46.257
100	None	45.923	77.139	**50.205**	**46.303**
80	All	**45.728**	**76.936**	**49.858**	**46.062**
80	None	45.528	76.879	49.143	45.890
50	All	44.974	**76.047**	48.677	45.266
50	None	**45.202**	75.901	**49.654**	**45.535**
20	All	**41.857**	**72.543**	43.242	**41.904**
20	None	41.806	72.327	**43.911**	41.763

Table A.4.: Results of average precision over various levels of IoU and object sizes

% of Data	Modules	AR (%)	ARl (%)	ARm (%)
100	All	53.90	26.62	53.30
100	None	**54.70**	**27.02**	**54.26**
80	All	54.54	**26.76**	54.32
80	None	**54.62**	26.56	**54.46**
50	All	**54.54**	**26.76**	**54.32**
50	None	54.16	25.78	53.78
20	All	**52.50**	22.52	**51.36**
20	None	52.20	**23.44**	50.98

Table A.5.: Results of average recall for various object sizes