UNIVERSITÄT
DUISBURG
ESSEN

Offen im Denken

Institute for Computer Science and Business Information Systems (ICB)
Department of Communication Networks and Systems
Prof. Dr.-Ing. Amr Rizk

COMPRESSION OF DEEP NEURAL NETWORKS FOR OBJECT DETECTION THROUGH RELATION-BASED KNOWLEDGE DISTILLATION

Master Thesis

submitted to the Faculty of Business Administration and Economics
of the University of Duisburg-Essen (Campus Essen) by

YANICK CHRISTIAN TCHENKO

Matriculation number: 3105372

February 14, 2023

Supervisor:	Michael Rudolph, M.Sc.
Examiner:	Prof. Dr.-Ing. Amr Rizk
Co-Examiner:	Prof. Dr. Klaus Pohl
Department:	Software and Network Engineering (M. Sc.)
Semester:	4

Abstract

Computer Vision is the subfield of computer science that deals with automatically extracting and exploiting features from optical data such as images and videos [Pri12; Sze22]. Recently, Machine Learning and especially Deep Learning, have proven to be a highly effective approach for solving tasks in this field, such as object classification, recognition, and image segmentation, with outstanding results [Sho+21]. However, the large size of Deep Learning models remains the biggest obstacle in applying this solution to real-world issues [MGD20]. In the industrial domain, trained models usually operate and are hosted on external high-performance GPUs to overcome this inconvenience. However, the related data transferring to external servers is legally risky, time-consuming, and expensive. This leads to high network traffic, latency, and privacy issues, making this solution inadequate. Hosting the models internally could help avoid this issue. In this work, we argue that compressing Deep learning models could provide more efficiency by optimizing their inference time and enabling their internal hosting. Therefore, we conduct an experiment with a well-known compression approach in Machine Learning: knowledge distillation (KD). The aim of the thesis is to apply this technique to object detection and evaluate its performance improvement. The results indicate that general, moderate, and adequate knowledge can be extracted from a pre-trained teacher model and transferred to a smaller student model, leading to similar or better accuracy. More specifically, we also explore four different knowledge distillation approaches: Response-based (RsKD), feature-based (RKD), relation-based (RKD), and multi-type (MKD) knowledge distillation. The results show that compressed models, and those compressed through relational knowledge distillation in particular, can improve the throughput and inference time of object detection and simultaneously reach better accuracy than non-compressed models.

Date of issue:	31.07.2022
Date of submission:	February 14, 2023
Author:	Yanick Christian Tchenko
Supervisor:	Michael Rudolph, M.Sc.
Examiner:	Prof. Dr.-Ing. Amr Rizk
Co-Examiner:	Prof. Dr. Klaus Pohl

I hereby declare that this thesis titled:

Compression of Deep Neural Networks for Object detection through relation-based knowledge distillation

is the product of my own independent work, and I have used no other sources and materials than those specified. The passages taken from other works, either verbatim or paraphrased in the spirit of the original quote, are identified in each individual case by indicating the source. I further declare that all my academic work has been written in line with the principles of proper academic research according to the official "Grundsätze für die Sicherung guter wissenschaftlicher Praxis an der Universität Duisburg-Essen Vom 13. *Juni* 2018" (University Statute for the Safeguarding of Proper Academic Practice).

Essen, February 14, 2023

Yanick Christian Tchenko

You don't have to be great
to start, but you have
to start to be great

— Zig Ziglar

ACKNOWLEDGMENTS

I would like to express my gratitude to Michael Rudolph, M.Sc. for his guidance and to Prof. Dr.-Ing. Amr Rizk for examining my master's thesis on this intriguing subject in the field of Computer Vision. Michael's supervision was instrumental in helping me maintain a rigorous scientific process throughout the thesis and achieve the results reflected in this work. I am deeply thankful for his support.

I also extend my appreciation to my fiancée Ruth Edimo, MBA, for her patience and support during my master's thesis. Her encouragement and backing helped me stay focused and on the right path toward completing the thesis project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AI Artificial Intelligence

ANNs Artificial Neural Networks

AP Average Precision

AR Average Recall

AUC Area Under the Curve

BoF Bag of Freebies

BoS Bag of Special

BR Bottleneck Residual

CIoU Complete Intersection over Union

CKD Convolutional/Convolution Knowledge Distillation

CNN Convilutional Neural Network

COCO Common Objects in Context

CPU Central Processing Unit

CV Computer Vision

DIoU Distance Intersection over Union

DL Deep Learning

DNN Deep Neural Networks

DNNs Deep Neural Networks

EDM Euclidean Distance Matrix

FC Fully Connected

FKD Feature-based Knowledge Distillation

FNN Feed-forward Neural Networks

FPS Frame Per Second

GIoU Generalized Intersection over Union

GPU Graphics Processing Unit

GTL Ground Truth Label

IKD Individual Knowledge Distillation

IoU Intersection over Union

KD Knowledge Distillation

KL Kullback–Leibler

mAP mean Average Precision

mAR mean Average Recall

MKD Multi-type Knowledge Distillation

ML Machine Learning

MLP Multi-Layer Perceptron

NMS Non-Maximum Suppression

OD Object Detection

R-CNN Region-based Convolutional Neural Networks

RDCD Remote Desktop Connection Devices

ReLU Linear Rectification Unit

RKD Relation-based Knowledge Distillation

RPN Region Proposals Networks

RsKD Response-based Knowledge Distillation

SL Supervised Learning

SPP Spatial Pyramidal Networks

SSL Semi-Supervised Learning

SVC Support Vector Classifier

SVM Super Vector Machine

SVR Support Vector Regressor

syncBN Synchronized Batch Normalization

USL Unsupervised Learning

1 INTRODUCTION

1.1 Motivation

In recent years, Machine Learning has emerged as a successful approach to Computer Vision (CV)[VKB]. CV enables the analysis of images captured by an optical recording device, such as a camera, and results in meaningful information about the images in question [VKB]. Deep Neural Networks (DNN) models are widely used for such tasks. Their effectiveness and efficiency are empowered by their design, the structure of the neural layers as well as activation functions included in each layer [XK19]. Despite their impressive performance, DNN-based approaches have certain limitations when it comes to model size, energy and power consumption, memory, and deployment requirements [Cap20]. Trained models are often too large to be deployed on Remote Desktop Connection Devices (RDCD), and optimizing them to merge speed and accuracy in the training process can be difficult, especially when running on resource-limited devices [Cap20]. Finding a balance between excellent performance, memory- and energy-efficient training process, and portability of DNN models becomes a challenge to which model compression comprises an optimal key. Through systematic modifications in the training routine, the compression operation shrinks the size of a trained deep neural network model, making it easier to use on devices with limited memory [Lee+18].

Many DNN model compression approaches exist nowadays. We can list weights pruning [Gho+19] and low-rank decomposition [TMK16] that perform particularly well for CV tasks such as Object Detection (OD). These approaches generally succeed in improving latency, although accuracy is still a critical point caused by model corruption (e.g., parameter rounding or increase in zero-valued elements) when the model is compressed. Thus, it becomes intriguing to investigate more precise compression techniques that can reduce the model size significantly while avoiding or minimizing this issue. A popular approach in this sense is Knowledge Distillation (KD), which aims to improve the accuracy of a small model by providing additional supervision by a larger model during its training process. The small model then inherits the knowledge of the larger one, allowing it to perform

the same task with quasi the same accuracy, acting as a compressed version of the larger model [Gou+21].

1.2 Problem Statement and Contribution

The common problem with DNN-based solutions discussed in the previous session is often seen in industries, particularly in OD tasks. The typical solving approach is to host the trained detection models externally on Graphics Processing Unit (GPU) servers, to which dependee devices can connect. This affords to transfer data (images or video frames) to be processed to the model and then to retrieve the output results afterward. In addition to the legal risks, time, and costs associated with transferring corporate data to external servers, there are concerns about high network traffic, latency, and data privacy. Furthermore, hosting models on internal devices that lack resources is not an ideal alternative option.

Having seen the benefits of compressing DNN models and the critical limitation of compression methods such as pruning and low-rank compression, the task of this master's thesis is to explore whether and how knowledge distillation can be used to address the presented challenges in the industries when solving their OD tasks. It is especially expected to find out which impact the compression process has on the training time and the accuracy of the original model. The Effects on the model inference should be further explored so that the trade-off between detection speed, compression degree, and accuracy can be worked out.

1.3 Thesis Structure

The thesis project is structured into six consecutive stages as shown in Figure 1.1. The first two stages involve gaining a better understanding of the issue at hand. They constitute the theoretical knowledge phases. (1) We begin by introducing the main areas of focus: KD), OD, and Deep Learning (DL). We explore use cases and application fields in each area. (2) We then examine the fundamentals and state-of-the-art technologies in each area, which allows us to gain a realistic understanding of the problem. This stage culminates in designing the individualized approach to solving the problem, thus concluding the theoretical duties. We describe these meticulously in Chapter 2, Chapter 3, and Chapter 4.

The following two stages outline the experimental phases. They are carried out according

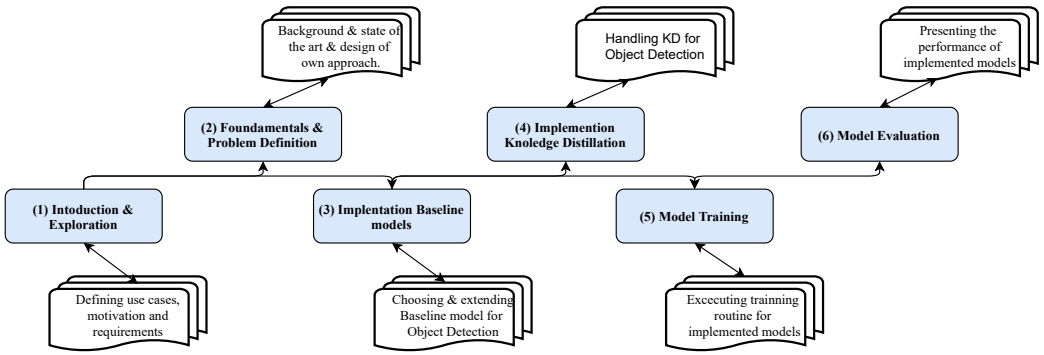


Figure 1.1: The guideline for the workflow in this work: The pipeline aims to obtain compressed models for object detection and compare their performance with existing uncompressed models for the same task.

to the own approach designed and presented in Chapter 5. Here, we primarily (3) select a base model for the OD task, which we extend by changing high-sized components to create a smaller detection model of the same category as the normal one. (4) The normal and minimized models are used to begin implementing the various KD variants for which we have made design decisions. The compression works from the normal (large) to the small model.

The last two stages, which are explained in Chapter 6, address the testing phases. (5) Considering the implemented distillation approaches, the training stage is started and results in a trained model, based on which (6) the different approaches implemented are evaluated. The evaluation process considers specific metrics for OD such as Average Precision (AP) and Average Recall (AR) to estimate the performance of the models in terms of throughput and accuracy. The compressed models are compared to the original ones based on these metrics. This allows Chapter 7 to summarize the effects of the KD compression method for DNN-based OD models, outlining evaluation results for relation-based KD and potential improvements.

2

BACKGROUND

This chapter provides a brief background of DL, which will serve as a fundamental for the implemented approaches. It begins with a conceptual description of Artificial Intelligence (AI) and Machine Learning (ML), the general context in which DL algorithms operate. Some learning methods are then described in terms of goals, application domains, operating algorithms, and evaluation metrics. Next, deep learning is presented, considering some machine learning algorithms, such as neural networks. Through the introductive presentation of some deep neural networks, we show the particularity of the convolution networks, which is the essential component of the architecture of the YOLO algorithms used to solve the issue of this thesis. Finally, we end this chapter by briefly summarizing the use of the presented approaches for the implemented solutions.

2.1 Artificial Intelligence & Machine Learning

Artificial intelligence, or AI, aims to enable computer systems to make decisions thoughtfully, like a human user. It follows a cognitive process that goes through different phases, from understanding the problem in a given context to communicating that problem, keeping data in context, thinking logically to find a solution, and adapting the knowledge acquired so far [Hun75].

2.1.1 *Processing Methods in Artificial Intelligence*

Depending on the self-learning ability, we thus distinguish two types of AI [Cop04]: Symbolic AI, referred to as weak AI, describes the process by which the machine receives data, learns to recognize and classify them and then specifies an appropriate common aspect depending on the type of data received. It is used in many fields, including Natural Language Processing and Gaming. According to Coppin [Cop04], a further variant of AI is neural or sub-symbolic or strong AI. It is developed as a continuous and autonomous learning neural system. Unlike symbolic AI, it is trained and encouraged to learn rather than recognize predefined symbolic features. Thus, AI systems learn autonomously over

the long term and can make predictions according to their intelligence. Neural AIis the driving force behind the incredible innovations realized in Artificial Intelligence nowadays. Based on Deep Learning (described in Section 2.2), an exclusionary sub-field of strong AI, computer systems can learn increasingly complex tasks such as autonomous driving, learning and speaking multiple languages, or recognizing handwriting [Sho+21].

2.1.2 *Processing Methods in Machine Learning*

In Machine Learning, the main processes of learning are supervised, unsupervised, semi-supervised, and reinforcement learning, as Figure 2.1 shows. The principal difference between them is that supervised learning is based on ground truth depicting available labeled training data [CCD08]. In the following sections, detailed aspects distinguishing these learning techniques are explored, neglecting reinforcement learning as unrelated to the present thesis project.

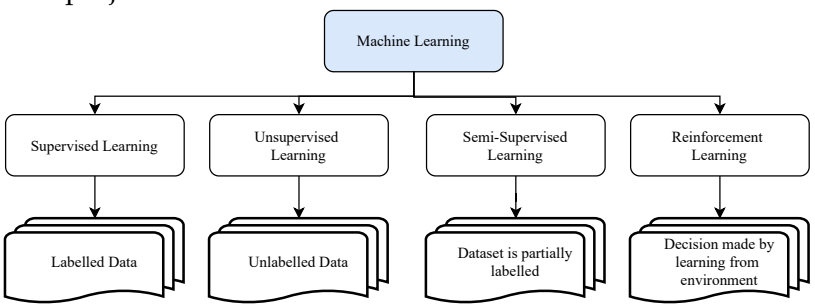


Figure 2.1: Machine Learning Sub-fields: The networks for object detection utilize the supervised learning method. (In accordance with [Sho+21])

Supervised Learning (SL) defines the process where an algorithm (f) learns to map input variables ($\mathcal{X}$) onto an output variable ($\mathcal{Y} = f(\mathcal{X})$). The goal is to understand the mapping function so well that, when new input data are formulated, we can predict the output variables ($\mathcal{Y}$) for these data with a relatively high probability of success [CCD08]. Since the data labeling refers to the knowledge of expected prediction answers (ground truth labels) for specific cases, the trained algorithm makes iterative predictions on a well-defined dataset, corrected by ground truth labels. A fine accuracy is reached when the algorithm's predictions are not diverging (too much) from these labels. SL is usually performed in the context of classification and regression [CNM06]. A regression problem occurs when it is aimed to build a predictive mapping function outputting a continuous variable [CNM06]. A classification problem arises when the output variable is a category or class membership

and mains a discrete variable [CNM06]. For example, supervised classification models are used in finance and banking to detect credit card fraud, while supervised regression algorithms predict real estate prices or stock prices. Common subjects related to classification and regression include forecasting and time series forecasting. Famous examples of these SL-solving approaches are [CNM06]: Decision Trees, K Nearest Neighbors, Linear Support Vector Classifier (SVC) (support vector classifier), Logistic regression, Naive Bayes, Neural networks, Linear regression, Support vector regression Support Vector Regressor (SVR), Regression trees.

Unsupervised learning involves having only input data ($\mathcal{X}$) and no given reference output variable [CA16]. The goal is to model the data's base structure or common distributions to learn more about the resulting structure that should be continuously adjusted. It is called unsupervised learning since, unlike supervised learning, there is no correct expected prediction to supervise the process. The algorithms are left to discover and present interesting patterns on a set of input data. They are categorized into two types: clustering and association algorithms [CA16]. Association is about discovering interesting relationships between variables in large databases [CA16]. For example, association AI algorithms could propose related setups for a given population buying new houses (new furniture, new Internet or electricity subscription, etc.). Association determines the probability of the co-occurrence of items in a given collection. Clustering, on the other hand, groups data points based on their similarities [CA16]. It uses probabilistic distribution approaches [EH20]. According to [CA16] and [EH20], some examples of Unsupervised Learning (USL) algorithms are: K-means clustering, Dimensionality Reduction, Neural networks, Principal Component Analysis, singular value decomposition, independent component analysis, distribution models and hierarchical clustering. These algorithms are used for several tasks, such as anomaly detection or ranking in medicine [EH20].

Besides the two types of learning presented above, we also distinguish Semi-Supervised Learning (SSL). It consists in training a model by incorporating the unlabeled data into a labeled training set [EH20]. In other words, not only is the training done on labeled data available but there are also successive modifications of the model using information extracted from unlabeled data. Semi-supervised learning problems are thus generally situated between supervised and unsupervised learning, as their solution calls upon the methods used in both approaches. This considerably increases the quality of the learning [EH20]. An example would be a set of images in which only some are labeled and others

are not. A given model f is trained based on labeled data. Assuming that for a defined set $\{x_1, x_2, x_3, x_4, x_5\}$ the items x_1 and x_2 are labeled, f could correctly classify objects that define the characteristics of the labeled training data. However, the accuracy and efficiency of f will be increased by considering information extracted by a non-supervised approach over the unlabeled data. The model could then classify a more significant number of object categories. Illustratively, having extracted information on the images for x_3, x_4, and x_5, f could get optimized to classify images approximating the features of the categories of these items.

Many real-world machine-learning applications fall into the category of semi-supervised learning issues. Indeed, labeling data can be costly or time-consuming as it may require access to domain experts [EH20]. Since unlabeled data are inexpensive and easier to collect and store, unsupervised learning techniques can be used to discover and learn the structure in the input variables and thus solve the data labeling problems. These techniques can also get employed to make optimal predictions for an unannotated dataset, transfer them to the supervised learning algorithm as training data, and hence increase the model to make predictions on new data unavailable during training [EH20]. Several approaches have been developed to enable semi-supervised learning. The different techniques developed in this direction can be broadly divided into the following four main categories [EH20]: (i) Self-training or pseudo-labeling, (ii) Generative models, (iii) Low-density separation, (iv) Graphs. Several performance measures are commonly used to evaluate the SSL algorithms. Some of them are loss squared, F1-score, and recall precision.

2.2 Deep Learning

As explained by Goodfellow, Bengio, and Courville [GBC16], a DL model is an algorithm that uses a Deep DNN architecture and is trained on a specific dataset to make predictions based on its experience with contained data. Decision-making training is accomplished using neural networks, which are composed of interconnected layers designed to learn patterns in an unstructured dataset. In daily life, there are numerous applications of DL, which are ranged from recommendation systems (e.g., online stores) over voice assistance systems (e.g., Siri) to autonomous driving. The tasks vary from data classification to object recognition and segmentation.

The process of Deep Learning is very similar to and closely related to machine learning.

It begins with the data retrieval phase [EAH20], where labeled training and test data are identified and structured in a format suitable for data processing algorithms[1] [EAH20]. The labeled data is then used for the modeling phase [EAH20], where deep learning models are created and trained. For this purpose, some machine learning algorithms operate over the data depending on the task to be solved. Then, they are evaluated using appropriate evaluation metrics ([EAH20]). This evaluation phase is recursively linked to the data preparation phase, where some validation data are provided. Finally, the model can be deployed in the last step of the pipeline [EAH20]. In general, implementing Deep Learning algorithms through the described pipeline is subject to essential components: Deep learning algorithms. The next section provides more details on DNNs, which represent the fundamental DL algorithms.

2.2.1 *Deep and Convolutional Neural Networks*

Deep Neural Networks are a specific variant of Artificial Neural Networks (ANNs) that have become standard for solving computer vision issues. They are employed to trigger the DL system's learning ability.

2.2.1.1 *General Architecture of Deep Neural Networks (DNNs)*

Simple ANNss are spawned from interconnected artificial neurons. Each neuron has some inputs that are weighted through parameterized weight matrices and extended with a defined bias before being integrated into an activation function to build an output provided to other neurons as input [Abb+07]. ANNs consist of Input-, Output-, and Hidden-layer. The input layer processes input data and converts them into feature maps. The hidden layer then uses these feature maps the evaluate the significance of the entered data to the output. It aggregates weighted input matrices with defined operations (addition, multiplication, or concatenation) and extends the aggregated expression by a bias before using the activation function [Abb+07].

Particular ANNs are convolutional neural networks (CNNs). A CNN is a Feed-forward Neural Networks (FNN), which describes acyclic artificial neural networks [Guo+16]. It consists of five main layer classes that sequentially process a given input volume, as can be seen in Figure 2.2. Before giving a detailed description of each layer in the next sections, we can, according to [Guo+16], list the CNN layers as follows: (1) convolution layers are pro-

1 https://www.expressanalytics.com/blog/what-is-data-wrangling-what-are-the-steps-in-data-wrangling/

cessing a limited portion of the image (called the "receptive field") through a convolution function.(2) Pooling layers are compressing and reducing the sizes of the input images to build a smaller intermediate output volume. (3) Activation layer defines the function used to hire the neurons in the considered layer. (4) Fully Connected (FC) layers are flattening the pooled output. (5) Loss layers determine the loss when training CNN-based models. A non-linear and weighted corrective treatment can be applied between layers to improve the result's relevance [CW08].

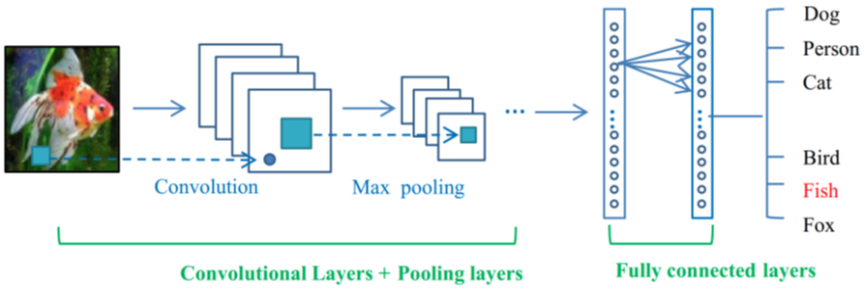
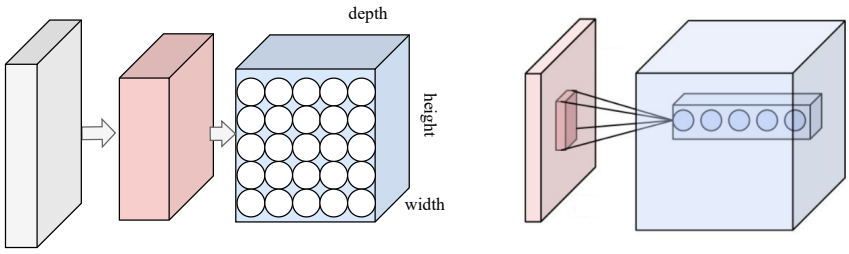


Figure 2.2: General Workflow for CNN-architectures [Guo+16]

In Figure 2.2, the input image is divided into small areas (patch) using kernels. Each patch is processed individually by an artificial neuron performing a filtering operation by associating a weight to each kernel pixel. The stratum of neurons with the same parameters is called a convolution kernel (see Figure 3.4). The pixels of a kernel are analyzed globally. In this case (color image), a pixel contains three inputs (red, green, and blue), which will be processed globally by each neuron. So the image can be considered as a volume and noted, for example, $320 \times 320 \times 3$, where the first 320 stays for the number of pixels for the image width, the second 320 describes the image height, and 3 represents the depth of the image corresponding to the three colors channels of the input volume.

2.2.1.2 *Properties of Convolutional Neural Networks*

CNNs limit the connections between a neuron and the neurons of adjacent layers, drastically reducing the number of parameters to be learned. Given an image of size $28 \times 28 \times 3$ (28 width, 28 height, 3 color channels), a single FC neuron in the first hidden layer of an FNN would have $28 \times 28 \times 3 = 2352$ input pixels. For a 300×300 RGB image, this would mean $270,000$ input pixels per neuron, which, multiplied by the number of neurons in the FC-layer associated, is relatively enormous and challenging to compute. This drawback is ameliorated by CNNs that aim to reduce the number of input pixels while preserving a



(a) A layer of the CNNin 3 dimensions: gray = input volume, red = receiver field volume, blue = CNN layer or output volume, circles = independent artificial neurons

(b) Set of neurons (circles) creating the depth of a convolution layer (blue cube). They are linked to the same receptive field (red). [Noaa]

Figure 2.3: 3*D* representation of Neuron inside a CNN

good local correlation of the original image. The three following properties enable CNN to have this unique ability [Guo+16], [Guo+16]:

- *Local connectivity:*
  thanks to the receptive field,CNNs ensure that the filters generate the strongest response to a spatially localized input pattern. This results in a sparse representation of the input that requires less memory. Since the set of parameters to determine is reduced, their statistical estimation is generally more robust than Multi-Layer Perceptron (MLP)s for a fixed data set.

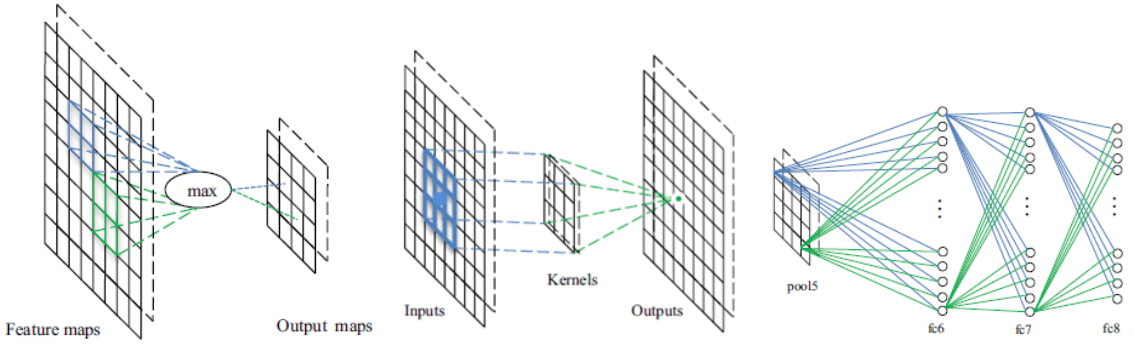
- *Shared weights:*
  In convolutional neural networks, the filter parameters of a neuron for a given receptive field are identical for all other neurons of the same nucleus or filter. This parameterization (weight vector and bias) is defined in a function map.

- *Translational invariance:*
  Since all neurons of the same network layer are identical, the pattern recognized by this layer is independent of the spatial location in the image.

In summary, the above properties allow convolutional neural networks to be robust enough in parameter estimation for Deep learning issues. The large amount of data per parameter is adjusted by weight sharing. Thus, the memory requirements for the network operation are reduced, allowing for learning the usage of more extensive networks.

2.2.1.3 *CNNs Layers Architecture*

As mentioned in Section 2.2.1.1, the architecture of convolutional neural networks is formed by a stack of five main processing layers classes [Guo+16]:
The baseline layer category for CNN is the convolution layer (seen Figure 2.4b). Its mode of



(a) Max pooling layer (b) Convolutional layer (c) Fully-connected layer

Figure 2.5: Operations at the various CNN-layers: These images describe the operations on the different conceptual layers of a CNN. Figure (a) deals with the max pooling operation. Figures (b) and (c) deal with the operations at convolution and fully connected layers. [Guo+16]

operation is particular, on the one hand, because of its parameterization, and on the other hand, because of its ability to share this parameterization. Convolution Layers (further noted Con2D or CONV) process data from a receiver field. Their respective output volume dimensioning is based on three parameters or hyperparameters: The layer's depth (1) represents the convolution kernels or neurons associated with a single receptive field. The stride (2) is a parameter of the overlap of the receptive fields. The smaller the stride, the more the receptive fields overlap and the larger the output volume. The (zero) padding (3) is the third hyperparameter. It allows controlling the spatial dimension of the output volume.

AnotherCNN layer class is The (max) pooling layer (see Figure 2.4a). It's often shortened POOL and allows compressing the information by reducing the size of the intermediate image. POOL layers define a form of subsampling of the image, where the input image is sliced into a series of non-overlapping rectangles of n pixels on each side [Noab]. Each rectangle can be seen as a kernel. The kernel output signal is a scalar determined according to the values taken by the different pixels of the kernel and the filter used. Pooling reduces the spatial size of an intermediate image, thus reducing the number of parameters and

computations in the network [Noab]. It is used to lower the risk of overfitting (described in Section 2.2.3) by periodically inserting a pooling layer between two successive convolutional layers of convolutional neural network architecture [ZF13]. The pooling operation also results in the form of translation invariance [WL22]. One can moreover use average pooling [ZF13], which formulates the scalar at the output as the average of the values of the input patch. This is usually referred to as $\mathcal{L}_2$-norm pooling. Max-pooling seems more efficient than average pooling because it significantly increases the importance of strong activations [ZF13]. In other circumstances, stochastic pooling or other regularization algorithms can be used [ZF13].

Improving mathematical operation in CNN also involves the tasks of the correction layer. The most used is Linear Rectification Unit (ReLU). It is usually applied in addition to pooling to improve processing efficiency. The ReLU is a link layer inserted between processing layers to apply a well-defined mathematical function to the pooled output signals. It refers to an operation that increases the nonlinear properties of the decision function and the general network without impacting the receptive fields of the convolutional layer [He+20]. ReLU generally applies to MLP, andCNNs [He+20]. Other activation functions include the hyperbolic tangent correction and the sigmoid function, widely employed for estimating predictions in recurrent neural networks (RNNs) [Coc+20]. The use of ReLU activation is often preferred as it results in faster training of neural networks, without significantly compromising the accuracy generalization.[Coc+20].

After several layers of convolution, max-pooling, and activation, higher-level processing in the neural network occurs via FC layers (see Figure 2.4c). Here, Neurons have connections to all outputs of the previous layer [ML17]. The loss layer is then applied, determining the deviation between the predicted outputs and the ground truth (label in Supervised Learning) and formulating a training penalization [Zha+18]. It represents the last layer of the neural network [Zha+18]. Depending on the task, different functions are used to determine the loss. The softmax loss predicts a single class from a set of mutually exclusive classes [Liu+17]. The sigmoidal cross-entropy loss is used to predict independent probability values, while Euclidean loss is used to regress on real values [Liu+17].

The most common form of a convolutional neural network architecture stacks a few $con2D - ReLU$ layers and follows them with Pool layers. It repeats this pattern until the input is reduced to a space of sufficiently small size. The last FC layer is connected to

the output [Bir+20]. Merging the different layer types to build ML algorithms requires appropriate parameterization. In the following, some techniques for balancing the network parameters are presented.

2.2.2 *Parametrization of DNNs*

CNN parameters or hyperparameters to be set include (i) the number of filters, (ii) their shape, and (iii) the shape of the max-pooling [Sto20].

Number of filters: Given that the size of the intermediate images (feature maps) decreases with depth, layers near the input have a tendency to have fewer filters, while layers closer to the output may have more filters [Sto20]. To keep the computations equal at each layer, one usually fixes the product of the number of pixels and features approximately constant at all layers [Sto20].

Shape of filters (Kernel-size): As described earlier, the convolution layer computes the convolution operation for extracting features from the input image and scans this image through defined filters. Each convolution layer has several filters. In literature, they are represented by numbers such as 32, 64, 128, 256, 512, 1024, etc., which correspond to the number of channels at the output of a convolution layer [Sto20]. The filters are slid across the input image, and at each position, dot products between the image and filter pixels are calculated [Sto20]. Each filter (also called kernel) is convolved with the input to compute an activation map.

Max-pooling: Typical values for the shape of max pooling are 2×2 matrices for small or middle and 4×4 matrices for huge input volumes. However, the choice of larger shapes will considerably reduce the size of the input image, which may result in the loss of too much information [Sto20].

The setting of the described parameter can be crucial to obtain a functioning CNN. Unfortunately, there is no guarantee that the set values are valid across all the training steps to meaningful output results. There could be an over- or under-fitting, which requires regularization when parameterizing the network.

2.2.3 *Overfitting and regularization of DNNs*

This expression *Overfitting*, which originated in statistics (opposite of underfitting), means an overinterpretation of data. It is a statistical analysis that fits a particular collection of a data set too precisely so that for additional data, no reliable predictions for future observations can be provided [Gav+18]. An overfitted model is thus a model that contains more parameters than the data can justify; it is also the direct result of incorrect parameterization in Machine or Deep Learning. The model then behaves like a table containing all the learning samples used and loses its predictive performance for new samples, rendering it completely ineffective for general prediction cases [Gav+18].

After finishing the configuration of the layers, regularization is applied to enhance the generalization abilities of the learning algorithm by reducing overfitting [Gav+18]. In ML, empirical and explicit methods have been developed to build strong regularizers [Gav+18]. Some empirical approaches often used for avoiding overfitting are: (i) *Dropout*, which consists of randomly disabling, with a given probability, the outputs of neurons during training [ITP15]. It is widely used in image recognition systems, speech recognition, and for computational biology problems [ITP15]. Another approach is (ii) *DropConnect* consisting of the random interruption of a connection during the training [ITP15]. Regarding speed and generalizability of learning, the results are similar to dropout but show a significant difference in the evolution of connection weights [ITP15]. Thus, a fully connected layer with DropConnect can be compared to a layer with sparse connections [ITP15]. (iii) *Stochastic Pooling* is based on the same principle as max-pooling. However, the difference is that the output is randomly selected according to a multinomial distribution defined following the activity of the area where pooling is applied [ITP15].

Explicit approaches against overfitting are: (i) *the Size of the network [Guo+16]:* As seen earlier with stochastic pooling, the size of the network dramatically improves the effect on overfitting. The idea here is to limit the number of layers in the network and release the network's free parameters (the connections). This approach directly reduces the efficiency and predictive potential of the network. One can also use (ii) the *Degradation of the weights*, which conceptually considers the vector of weights of a neuron, i.e., the list of weights associated with the incoming volumes, and adds to it an error vector proportional to the sum of the weights (regularization by norm 1) or the square of the weights (regularization by norm 2) [Kal12]. This error vector can then be multiplied by a proportionality coefficient,

increasing to penalize vectors with strong weights further. Regularization by norm 1 is characterized by decreasing the weights of random and weak inputs and increasing the weights of "important" inputs [Kal12].

2.3 Background summary

In summary, Deep Learning is an influential subfield of Machine Learning and Artificial Intelligence that contributes to the numerous innovations in these fields. It defines a specific class of weak AI and is conceptually based on DNNs. CNN-models can be trained under supervision, and optimizing their performance, i.e., their intelligent prediction efficiency, requires appropriate parameterization. Since imperfect parameterization can lead to overfitting the trained model, regularization methods have been advanced to solve this issue. Optimized CNN-models resulting from regularization are applied in different domains, depending on application areas considered in the implementation. In this master project, we have fundamentally involved CNNand its conceptual and architectural features to improve the autonomy of DL-Models for CV. We employed different techniques, such as stochastic pooling and dropout, against overfitting. Before going to the implementation of our methodologies in more detail in Chapter 5, we first present the CV field in the next section.

3

RELATED WORK

This chapter introduces concepts related to the compression approach implemented in the master thesis. Starting with a brief presentation of Computer Vision as an application area of Deep Learning, we give an overview of some incredible state-of-the-art approaches. Furthermore, we elaborate on the compression techniques studied in this thesis and explain their different variants. These descriptions are followed by a summary analyzing why existing approaches to computer vision tasks, such as object detection, still need to be investigated or improved.

3.1 Computer Vision: Object Detection

Computer Vision (CV) is an AI-based approach specialized in recognizing and exploring images or videos data, processing the related information to make predictions about the features contained within [Vou+18]. Its applications are diverse, ranging from education (smart boards, intelligent classrooms [See+21]) to medicine (stroke detection [Est+21]) to automobiles (autonomous cars [DZ87]) and astronomy (space robotics [GM20]). CV performs classification, segmentation, and object recognition tasks in these domains using methods based on the DNNs previously defined in Chapter 2. In the remainder of this thesis, we will present computer vision almost exclusively in terms of its operation in solving the object detection issue addressed in this thesis's requirements.

OD is one of the most widespread tasks in CV [Lu+20]. Detection models are reaching higher and higher accuracies, and the limits of progress mainly concern resource restrictions for their deployment and execution speed. Industrial use cases require models that are not only accurate but also computationally efficient so that they can be executed on remote devices with a single graphics card or even on embedded hardware, as in the case of autonomous cars, small drones, or surveillance cameras. Current object recognition techniques fall into two categories [Lu+20]: Single-stage and two-stage approaches. In one-stage detectors, a given image is viewed only once by a neural network, and the network returns all objects in the image as well as associated positions in the form of

bounding boxes [Lu+20]. These approaches are generally fast, as the network completes the task in one stage. It is precisely the approach introduced in 2015 by YOLO-v3, which differs from two-stage detectors, where images are first sent to a part of the network or a separate model, but not to the whole detection system simultaneously [Lu+20]. The main challenge hereby is to identify which areas of the image contain objects which should be passed to the next part classifying the current object in each area. Two-stage methods generally achieve better accuracy but are slower than single-stage detection. Some examples of two-stage detectors could be Faster Region-based Convolutional Neural Networks (R-CNN) and Mask R-CNN models [He+18].

3.1.1 *R-CNN*

The challenge mentioned in the last section arises from the fact that there are numerous regions to choose from, making it difficult to select regions of interest. To overcome this obstacle, Girshick [Gir15] proposed a technique in which only 2000 regions are extracted from a given image through targeted tracking. The authors called the method region proposals. This currently allows one to limit the number of regions to be considered to 2000. The region proposals are created using the following tracking or selective search algorithm [Gir15]: (1) an initial partial segmentation is performed, which generates many candidate regions, (2) a greedy algorithm is used to recursively group similar regions into larger ones, (3) the generated regions are used to create the final candidate region proposals. Based on the proposed regions, a CNN can then be used for feature extraction from the image; hence the name R-CNN, whose performance has been continuously improved.

3.1.1.1 *Simple R-CNN*

It is the primary approach of R-CNN. First, constellations such as edge box algorithms are used to extract or select region proposals. These regions are then cropped and resized from the input image. A CNN computes the features of the cropped and resized regions, e.g., AlexNet [MDRS19], one of the most popular and standard convolutional neural network architectures for image recognition [Alo+18]. Finally, the bounding boxes submitted by the CNN are binary classified by an Super Vector Machine (SVM) trained with the features of the regions extracted by the edge boxes. In this perspective, the simple R-CNN allows the proposal of tailored regions. The three processing units for extracting region suggestions (Edge Boxes), computing CNN features, and classifying the extracted region based on the

computed features are trained independently. This makes the training of such a system slow and even impractical for resource-poor devices [Gir15].

3.1.1.2 *Fast & Faster R-CNN*

It works similarly to simple R-CNN. This variant of R-CNN also uses algorithms such as edge boxes to generate region proposals. Therefore, instead of cropping and resizing region proposals from the input image, Fast R-CNN processes the entire image. Rather than classifying these regions through CNN-Features, it pools the features related to each region proposal [Gir15]. In this way, there is the particularity that overlapping regions are computed, and these computations are shared between a classifier and a regressor, of which a fast R-CNN algorithm consists. It makes classifying objects in an image and their bounding more accurate and the corresponding latency better. The customization of region proposals is guaranteed. However, the selective search of Edge Boxes remains slow [Gir15].

New expressions such as Region Proposals Networks (RPN) exist in faster R-CNN. It is a built-in network for generating region proposals without using - like in (fast) R-CNN - edge boxes, an algorithm external to the detection pipeline [Gir15]. This network uses anchor boxes (a set of predefined rectangular boxes of different sizes) for object detection. The result is that the computation and generation of region proposals are accelerated, and better region proposals are obtained for the objects in an image. This optimal run-time performance and more accurate classification and regression is the particular advantage of Faster R-CNN, while the custom of region proposals is not supported here [Gir15]. For this particular result, Faster R-CNN or, especially, RPN uses two shots, one for the generation of region proposals and the other to compute generated regions correspondingly to the metrics (scaling, coloration, etc.) of the image [Gir15].

3.1.1.3 *Mask R-CNN*

It is a state-of-the-art extension of CNN developed based on Faster R-CNN for the CV task image segmentation, especially for Instance segmentation, a combination of semantic segmentation[1] and object detection [He+18]. Hereby, an image is divided into multiple segments, enabling object recognition and localization. Mask R-CNN is very similar to Faster R-CNN, but both differ in some aspects, such as the number of outputs. It provides three outputs for each segmented object instead of two as in Faster R-CNN [He+18]: a class

1 Semantic segmentation: Process of classifying pixels of a given instance within an image for processing [Guo+18].

label, a bounding box offset, and an additional object mask. The object mask differs from the bounding box and class outputs in that it requires extraction of the finer spatial layout of a candidate object.

In general, the architecture in the Mask R-CNN approach could be summarized in the following steps [He+18]: (1) The basic model is defined by a typical convolutional neural network: a feature extractor. It would transform, for example, a $512 \times 512 \times 3$ image into a $16 \times 16 \times 2048$ feature map. (2) This map is fed back to the Region Proposal Network (RPN), which uses the regions defined with anchor boxes to scan each region and predict whether an object is present. A significant advantage here is that the RPN does not scan the image itself but the feature map, which makes it much faster. (3) The regions thus proposed are classified according to their interest and marked by bounding boxes. In this step, the convolution algorithm uses the regions of interest (meaningful eras proposed by the RPN) as inputs to classify (softmax) and regress bounding boxes. (4) Segmentation masks are then determined based on the detected objects. Indeed, during this final step, the algorithm takes the positive regions of interest as inputs and generates 28×28 pixel masks with floating values as outputs for the objects. During inference, these masks are scaled.

3.1.1.4 *Summary of R-CNN*

All in all, the variants of R-CNN, such as the simple, fast, or faster version, are used and show a growing and considerable efficiency for object detection tasks in Computer Vision. Mask R-CNN represents an extension of the Faster R-CNN in that it extends its activity in the resolution of much more complex CV-tasks, such as Instance Segmentation, which the object detection models (Faster R-CNN, YOLO-v3, YOLO-v4, etc.) cannot perform.

3.1.2 *YOLO-v3 and YOLO-v4*

YOLO-v3 and YOLO-v4 are the third and fourth versions of the YOLO approach, respectively, which stands for "You Only Look Once". This term represents a novel method for real-time object detection that was first introduced in 2015 by Redmon et al. [Red+16]. YOLO is considered to be a pioneer among one-stage detectors and is able to achieve faster inference times compared to two-stage models, while still maintaining reasonable accuracy.

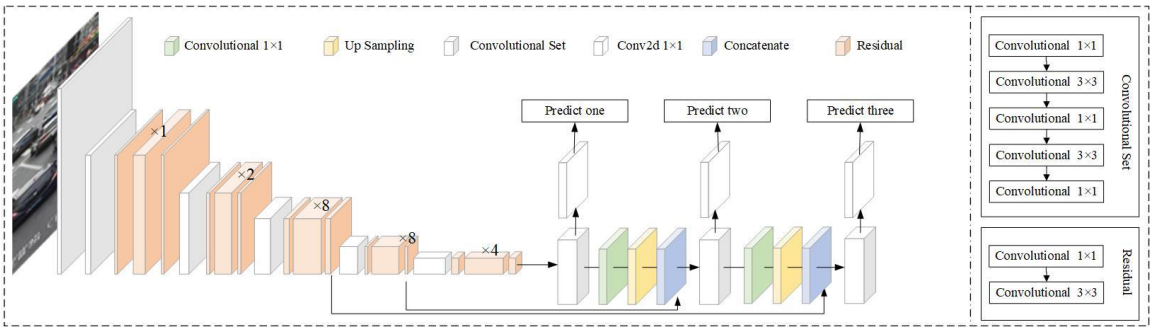


Figure 3.1: Structure of YOLO-v3: YOLO-v3 employs Darknet-53 as its backbone network for three-scale prediction. For each input mini-batch, predictions are generated from the final three convolution layers of the network. [Mao+19]

3.1.2.1 *YOLO-v3*

YOLO-v3 is the version of YOLO released in 2018 by Redmon and Farhadi [RF]. It is based on the Darknet-53 architecture (an improvement of the initially adopted Darknet-19) and independent logistic classifiers for the Softmax activation mechanism used in previous versions. In its architecture (see Figure 3.1), YOLO-v3 consists of a stacked sequence of convolutional and residual layers (described in Section 3.1.3.1). Binary cross entropy loss is used as the loss function Redmon and Farhadi [RF]. YOLO-v3 first divides an image into grid cells, where each cell predicts a certain number of bounding or anchor boxes around objects with high scores in the predefined classes. Each anchor box corresponds to a confidence value that defines the probability that it belongs to a class. Only one object is detected per anchor. This describes the ability distinguishing YOLO-v3 from R-CNN variants, such as simple and fast R-CNN: The final model is trained not only to propose bounding boxes but also to classify and regress them.

An important point justifying its preference for the described task could be formulated in terms of speed, precision, and specificity of classes, where YOLO-v3 shows a better performance than older versions. It is fast and accurate, looking at the achieved values for its performance metrics, such as mean Average Precision (mAP) and Intersection over Union (IoU) in the authors' evaluation score. With approximately the same performance, it runs significantly faster than other recognition methods such as RetinaNet-50, Faster R-CNN, and RetinaNet-101 [Mao+19].

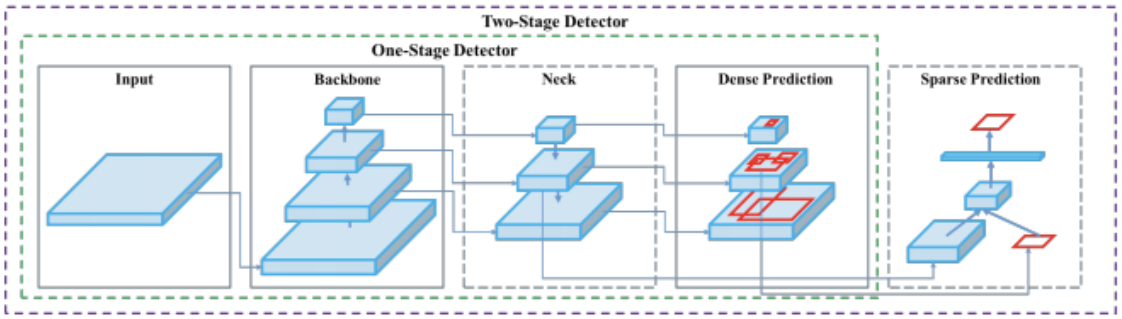


Figure 3.2: YOLO-v4 Architecture: YOLO-v4 consists of four main components that are connected in sequence to form a one-stage detector. If a compartment for sparse prediction is needed, then the two-stage detection process must be required. [BWL20]

3.1.2.2 *YOLO-v4*

YOLO-v4 model, launched in April 2020, represents a breakthrough in object detection [BWL20]. It promises state-of-the-art accuracy combined with real-time execution speed on a single GPU. This makes it possible to process images between 150 and 423 Frame Per Second (FPS) [BWL20].

YOLO-v4 works on the principle of its predecessors, optimized by two large packages of new technologies formed by re-implementing many advances, techniques, and architectures available in the scientific research of recent years and by the elimination of the expensive implementations in computing time. We thus distinguish according to [BWL20]:

The Bag of Freebies (BoF) is a set of improvements with no impact on the architecture of a network that can be used "for free", i.e., at no cost for modifications to an existing network [BWL20]. BoF are improvements applied during the learning process and accomplishes tasks such as data extension, and cross-mini-batch normalization [BWL20]. In contrast, the Bag of Special (BoS) is a set of improvements requiring specific network architecture changes [BWL20]. This new package contains recent advances in the scientific literature that improve the network's performance without affecting its speed. These include the Mish activation feature, Spatial Pyramidal Networks (SPP), and Path Aggregation Network [BWL20]. Combining the methods used for Bags of Specials and Freebies, the prediction accuracy is optimized from 33% of AP for YOLO-v3 to 44% of AP for YOLO-v4 by applying the models to the COCO object detection dataset [BWL20]. We also note a significant gain in execution speed (in FPS) of 12%. These new features define the architecture of YOLO-v4

[BWL20].

The architecture of YOLO-v4 is divided into three parts, which are functionally different (see Figure 3.1): Backbone, Neck, and Head (dense + sparse prediction). The backbone is the "body" of the network, defining a neural network that enables all decisions. In general, it represents an input image converter into a set of information that characterizes its content, the "features" (shapes, colors, textures, ...) that facilitate the recognition of objects. The basic framework consists of layers that extract feature maps, i.e., maps that indicate which features are present at which locations in the image. This network is usually trained separately on object recognition datasets such as Microsoft COCO and Pascal VOC. The diversity in the content of the datasets forces the backbone to learn various features according to the size, color, and shape of the elements it observes, making it more robust. Thus, it can extract useful features regardless of the image presented to it [BWL20]. However, the backbone has one major limitation: it provides too many features for a recognition task.

The second component of YOLO-v4 is the Neck, a sub-module of BoS. Its role is to address the challenge posed by the additional features introduced by the backbone by extracting relevant features from all the backbone layers. These extracted features are then combined to form features useful for recognition. The early layers generally have a higher spatial resolution and recognize simpler features such as lines and colors and smaller elements [BWL20]. On the other hand, the later layers have a lower resolution but provide more complex features, such as combining shapes and colors to recognize larger objects, such as a car tire that is a metal circle with a hole [BWL20]. In essence, the Neck enables the selection, integration, and combination of features of varying resolution and complexity to achieve fast and accurate detection of both large and small objects, as well as simple and complex ones [BWL20].

The final part of the YOLO-v4 architecture is the Head. It is responsible for the final decision of the network [BWL20]. Based on the information provided by the Neck, the Head determines the position of the elements to be recognized by drawing bounding boxes around them. It also decides which elements are involved by indicating the object type in each bounding box [BWL20]. This part is represented by YOLO-v3 and results in a model, a canvas of possible boxes. It learns to slightly adjust extracted features to fit the recognized objects (regression), resulting in the prediction of bounding boxes.

YOLO-v4 has a similar architecture to other object recognition models in the state-of-the-art, but it has new features that set it apart. These features mainly improve the backbone and head operations, making YOLO-v4 more robust and competitive compared to other high-accuracy models such as RetinaNet, EfficientDet, and Mask R-CNN. In the following sections, we will examine some of the backbones used by YOLO-v4 and explore how these features contribute to its improved performance [TPL20].

3.1.3 *Backbone Architectures for Object Detection*

Feature extraction is a crucial step in object detection, and the backbone component is responsible for this task. There are many variants of backbones available today. In the following sections, we will provide an overview of two backbones used in this thesis: Mobilenet and Darknet.

3.1.3.1 *Darknet*

Darknet is the feature extraction network used in the YOLO architecture. Until the second version, Darknet-19 was mainly used, consisting of 19 convolutional layers [BWL20]. It was developed by and was preferred for its efficiency and small size. Like GoogleNet and ResNet [SCY22], Darknet works based on notions such as network within the network, batch normalization, and inception [Yul+20]. It was combined with Residualnets [BP22] to create the hybrid version Darknet-53, a more extensive network consisting of a series of 53 consecutive 3×3 and 1×1 convolutional max-pooling layers with shortcut connections [BWL20]. 1×1 convolution kernels are used to reduce the parameter set, while the 3×3 kernels are privileged as filters for simple convolution. Darknet is the backbone of many object recognition methods, including YOLO-v2, YOLO-v3, YOLO-v4, and YOLO-v>4 [LCY14].

The Darknet architecture, as depicted in Figure 3.3, is comprised of four main sections according to Bochkovskiy, Wang, and Liao [BWL20]. The initial section is made up of two 3×3 convolution layers that reduce the input image's dimensions by half. Next, the output is processed by the Conv-Residual block1, which consists of a 1×1 convolution layer, a 3×3 convolution layer, and a residual layer to keep the input image's dimensions. The output then goes through a "shortcut connection" layer which increases the number of filters through a 3×3 convolution operation and reduces the image's dimensions for

further processing in Conv-Residual block2.

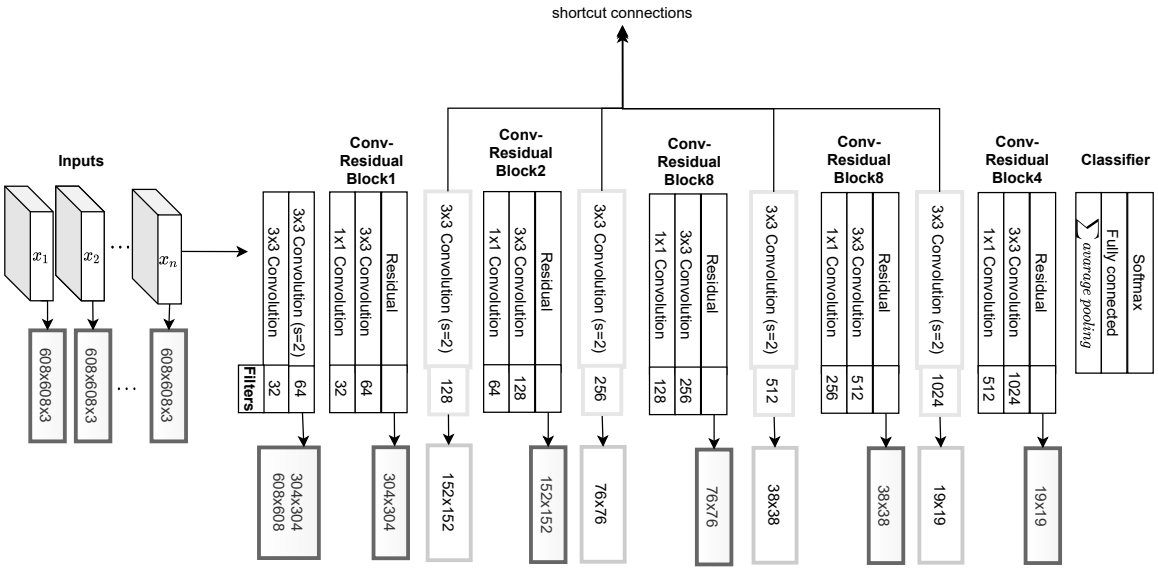


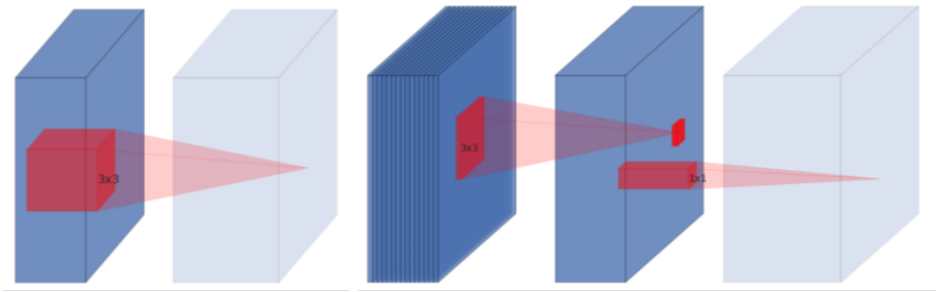
Figure 3.3: Darknet-53 layer structure: A nesting of layers using convolution to progress from a huge and quasi-informative volume at the input to a smaller volume at the output with an impactful information yield for determining the feature maps. (In accordance with Bochkovskiy, Wang, and Liao [BWL20])

This block is constructed similarly to Conv-Residual-Block1 but duplicates itself in the convolution operations. The shortcut layer applied afterward allows the input volume to be split. An image entered initially with the dimensions of 608×608 would reach a volume of 76×76, constituting the output volume of Conv-Residual-Block8. This block has the same structure as blocks Conv-Residual 1 and 2, with the peculiarity that it repeats the convolution operations 1×1 and 3×3 eight times in sequence alternately. Using a shortcut connection makes it possible to repeat the typical operations of this block with a smaller intermediate volume. The Conv-Residual-Block4 is then applied to obtain a smaller output volume with a more significant number of filters through its four repetitions. The resulting volume for the output image mentioned earlier would be $1024 \times 19 \times 19$. The classification block can use the features extracted in this way to estimate the probability of belonging to the predefined object classes.

3.1.3.2 *Mobilenet*

It is a CNN initiative, which should be easily deployable for mobile applications because of its simple architecture and small-sized model [San+19]. Mobilenet is one of the nets commonly used as backbone in the YOLO-v4 architecture, with only two convolution layers in a block in its native version. It follows the idea of replacing convolutional layers, which are essential to computer vision tasks but are expensive to compute, with depthwise separable convolutions [San+19].

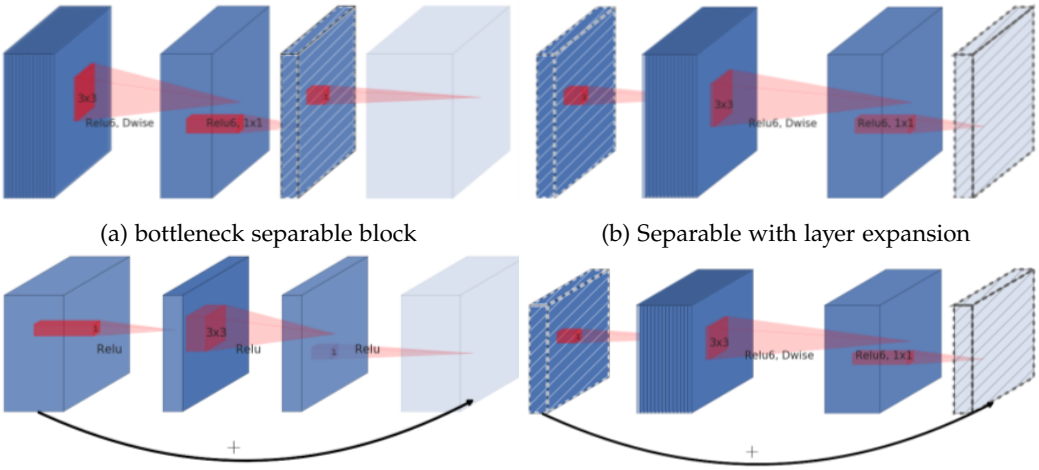
Separable convolutions define the CNN category where, depending on the depth, a convolution is applied to reduce the size of a classical convolution kernel into smaller kernels [San+19]. Thus, for a 3×3 kernel, depthwise convolution leads to pointwise convolution with a 1×1 kernel. Together, the depthwise convolution and the pointwise convolution form a depthwise separable convolution block that works similarly to conventional convolution but much faster. This separation is efficient because it greatly reduces the number of operations required to perform the convolution. The impossibility of separation is usually observed in the spatial dimension, so it is usually performed in the depth dimension (channel) [San+19].



(a) Regular (b) Separable

Figure 3.4: Mobilenet: Convolution blocks [San+19]

Version 2 of the Mobilenet series uses this convolution by introducing inverted residuals and linear bottlenecks to improve neural network performance. Unlike the first version, they consist of three convolution layers inside a block. Inverted residuals allow the network to compute activations more efficiently and obtain more information after activation [San+19]. For that, the last activation in the bottleneck block, as shown in Figure 3.5a and Figure 3.6, has a linear activation. In the last figure, the thicker blocks have more channels. The Mobilenet-v2 architecture comprises depthwise separable convolutional blocks and



(a) bottleneck separable block

(b) Separable with layer expansion

(c) bottleneck residual bock

(d) Inverted residual block

Figure 3.5: Design of separable convolution and residual blocks. The diagonally highlighted texture indicates layers that do not contain linear bottlenecks. The last brightly colored layer shows the layer that does not contain any linearity. Residual blocs connect the layers with many channels; inverted residuals connect the linear bottlenecks. Figures 3.5a & 3.5b represent equivalent blocks when stacked. [San+19]

Bottleneck Residual (BR) blocks (as depicted in Figure 3.6). A BR block comprises three main layers, arranged from left to right. The first and last layers are 1×1 convolutional layers combined with batch normalization layers, with the first layer also including an activation layer that processes the input volume for the second BR layer. This second layer is a 3×3 depthwise convolutional layer that filters the input volume and reduces the number of dimensions (channels) in the dataset. As the network goes deeper, the resolution decreases further and further, which is referred to as the bottleneck process aforementioned [San+19].

3.1.3.3 *Effects of BoF and BoS on Backbone and Head*

According to [BWL20], these effects can be summarized as follows: For Backbone, BoF provides new dependencies such as CutMix & Mosaic and DropBlock for data augmentation [2] and regularization tasks, respectively. These enable the improvement of examples for adjusted and adequate model fitting. Other positive effects of BoFs include class label smoothing and self-adversarial training, which uses the model's state to inform about vulnerabilities by transforming the input volume. BoS effects on the backbone lead to new options for neural activation and the connections of the convolutional and residual layers.

2 generating new training examples from existing training data ([SK19]).

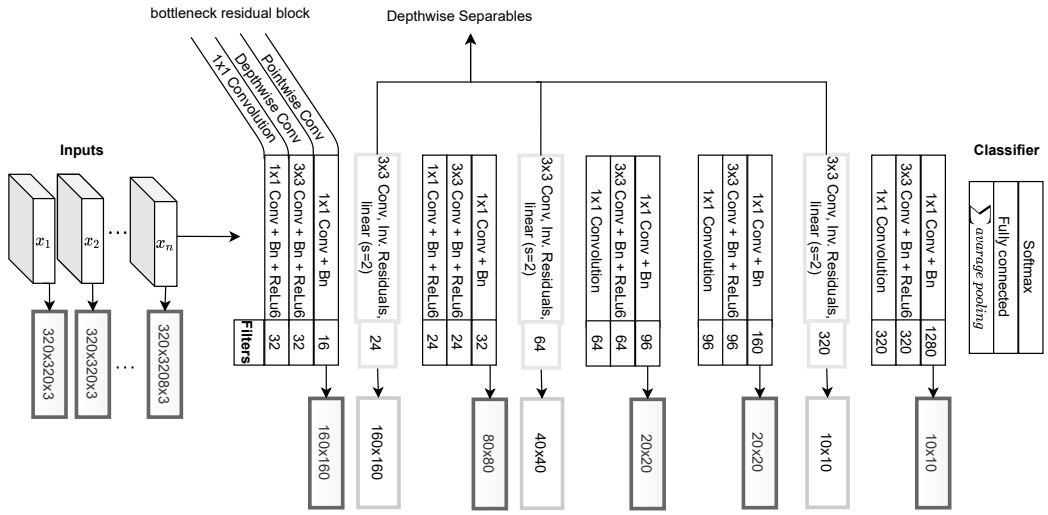


Figure 3.6: Mobilenet-v2 Layer structure: The network consists of bottleneck residual blocks transforming a large $320 \times 320 \times 3$ input volume to a $10 \times 10 \times 1280$ output volume. The classification is then made based on this resulting output. (In accordance with Sandler et al. [San+19])

We enumerate here, for example, mish activation, cross-stage partial connections, and multi-input weighted-residual connections.

BoF improves the Head operation and especially the detection. For this purpose, new functions for loss calculation and batch normalization or regularization are provided. Therefore, we distinguish between Complete Intersection over Union (CIoU), random training shapes, cross-mini batch normalization, drop block regularization, grating sensitivity elimination, and cosine annealing. On the other hand, BoS improves the detectors' operation with updated applications such as mish activation, SPP, multi-input weighted-residual connections, spatial attention module, Distance Intersection over Union (DIoU), Non-Maximum Suppression (NMS), and path aggregation network. With these modern features optimizing performance in feature extraction (Backbone), object detection, and sparse & dense prediction (Head), YOLO-v4 consequently achieves optimal speed and accuracy compared to older versions or state-of-the-art methods like EfficientDet, as seen in Figure 3.7. YOLO-v4 runs two times faster than EfficientDet, resulting in comparable performance.

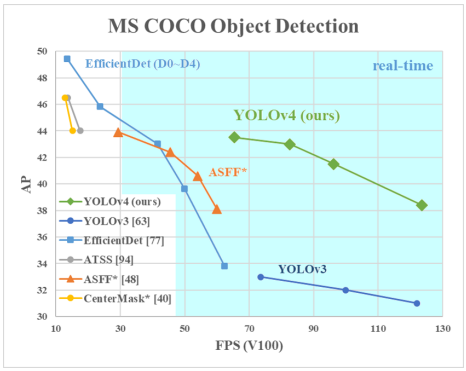


Figure 3.7: YOLO-v4 compared to other top-performing object detectors [BWL20]: With efficient inference, YOLO-v4 reaches similar levels of accuracy in terms AP, compared to EfficientDet, which currently has the highest precision level.

3.2 DNN-Models Compression: Knowledge Distillation

For compressing DNN-models, Various methods have been developed, such as Network Pruning, Sparse Representation, Bit Precision, Miscellaneous, and Knowledge Distillation, which are listed in [MGD20] and [GA22]. Due to the limited time frame of the thesis project and defined requirements, the investigation of compression techniques is limited to the Knowledge Distillation method.

3.2.1 *Teacher-Student Architecture*

The key aspect related to knowledge distillation is how to transfer qualitative knowledge. A popular approach in several papers such as [HVD15] is the teacher-student architecture, where a student model learns from a teacher model. For improved learning efficiency, the latest models should be selected, trained, and used to provide students with only the knowledge necessary for a specific task, leading to a distilled model. The challenges in this process can be summarized as follows: (i) how to design, structure, and select the right teacher for a given student? (ii) how to determine the relevance of the architecture for a suitable distillation quality?

The teacher-student architecture is usually implemented by choosing the student as a simplified version of the teacher network with fewer layers and neurons in each layer [Wan+18]. Another approach is to choose a version of the teacher network whose struc-

ture must be preserved to build the student network [SBS19]. Several transfer schemes for proper distillation have recently been developed for these different designs. A great example is adaptive teacher-student learning[3], which is used to optimize the matching between the student and teacher models [YW20].

A qualitative and well-structured teacher-student architecture constitutes the foundation of knowledge distillation [HVD15]. Here, the primary goal, after selecting the teacher and student models, is to determine what knowledge the teacher can transfer to the student so that the student can draw better conclusions and perform better. The second step of KD is to determine how to transfer the found knowledge from the teacher to the student. Current knowledge distillation techniques differ depending on the knowledge considered in a deep neural network, the distillation scheme, the transfer technique, the algorithm used, and the expected application of the compressed model. However, the main difference between the different performances is the type of knowledge used. In the following, we describe some existing knowledge types used by the current state-of-the-art KD frameworks. We briefly describe and review each type of knowledge (distillation) and its associated advantages and disadvantages.

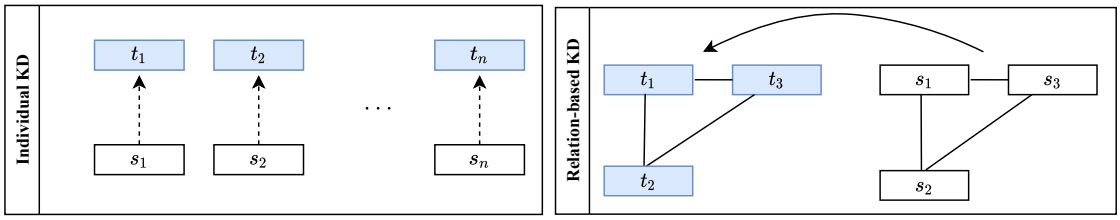
3.2.2 *Convolutional / Individual Knowledge Distillation*

Individual Knowledge Distillation (IKD) or Convolutional/Convolution Knowledge Distillation (CKD) is the class of KD algorithms for individual mapping between teacher and student outputs or feature maps [Gou+21], as can be seen in Figure 3.8a. Well-known variants of this class are RsKD and Feature-based Knowledge Distillation (FKD).

3.2.2.1 *Response-based Knowledge*

As mentioned earlier, finding a teacher's knowledge to use is the first step in knowledge distillation. This achievement also defines the task in the basic concept of RsKD, contrasting the predicted logit of the Teacher model and Student model, respectively (see Figure 3.11). Optimizing the student performance implies reducing the divergence loss from the teacher predictions operating as reference values [HVD15]. The divergence loss is determined by finding the divergence between the teacher's score z_t and the student's score z_s for a sample $(x_1, x_2, \ldots, x_n) = x \in \mathcal{X}$, where $\mathcal{X}$ represents the loaded, labeled training dataset

3 Adaptive Learning is a DL solution for real-time data collection, where the collected information is analyzed and adapted, and insights are provided almost instantaneously [YW20].



(a) Individual knowledge transfer (b) Relational knowledge transfer

Figure 3.8: Individual vs. relational knowledge distillation. In layered representations of the model, IKD seems to transfer the results of each layer directly from the teacher (t_i) to the student (s_i). In contrast, RKD does not match individually. It uses a relational function to extract the relationship between selected teacher layer outputs, which is adaptively transferred to the student model. The relationship between teacher layer outputs t_1, t_2, and t_3 is matched with the relationship between student layer outputs s_1, s_2, and s_3.

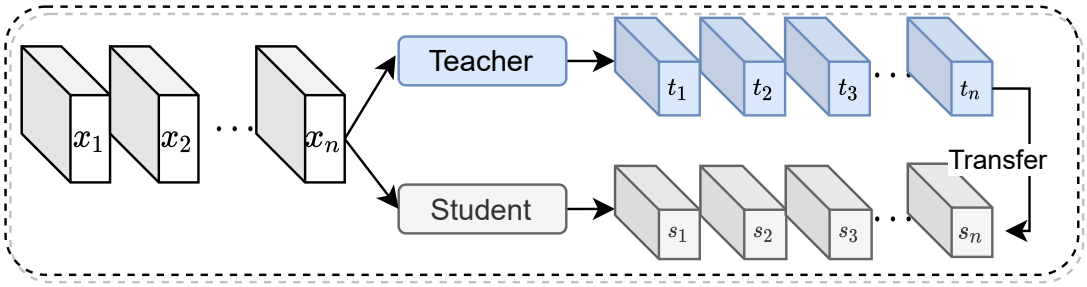


Figure 3.9: Response-based knowledge distillation: individual entities are considered and matched in any order. The leading matching position is the last layer of the teacher and the student, respectively. The resulting responses (output logits) are used to compute the transfer operation.

[HVD15]. Given a linear mapping f[4] that uniformly transforms the scores into probabilistic

4 Given vector spaces V and W and $f: \mathcal{V} \longrightarrow \mathcal{W}$, f is a linear map: if $\forall u, v \in \mathcal{V}, f(u \oplus v) = f(u) \oplus f(v)$

2D sets $p_s = f(z_s)$ and $p_t = f(z_t)$, the student training aims to optimize the following loss:

$$\begin{aligned}
\mathcal{L}_{RsKD}(z_s, z_t, x) &= (1-\mu)\mathcal{L}^x_{hard}(z_s, y_x) + \mu\mathcal{L}^x_{soft}(p_s, p_t) \\
&= (1-\mu)\mathcal{L}^x_{hard}(z_s, y_x) + \mu\mathcal{L}^x_{soft}(f(z_s), f(z_t)) \\
&\text{with} \\
\mathcal{L}^x_{soft}(p_s, p_t) &= -\sum_{i=1}^{\#x=n} f(z^i_t)\log f(z^i_s) \text{since } f \text{ is linear} \\
\mathcal{L}^x_{hard}(z_s, y_x) &= \begin{cases} \sum_{i=1}^n \mathcal{L}^i_2(\cdot) = \sum_{i=1}^n \frac{1}{2}\|z^i_s - y^i_x\|^2_2, \text{ if } |z^i_s - y^i_x| \leq 1 \\ \sum_{i=1}^n \mathcal{L}^i_1(\cdot) = \sum_{i=1}^n \left|z^i_s - y^i_x\right|, \text{ otherwise} \end{cases}
\end{aligned} \tag{3.1}$$

where $\mathcal{L}_{RsKD}(.)$ is the calculated total loss based on the responses. y_x is the ground truth label for x. $\mathcal{L}_{hard}$ or $\mathcal{L}_{soft}$ is the hard loss estimated using ground truth labels. The soft loss is determined using the teacher's prediction. $\mathcal{L}^i_a$ is the loss of type $\mathcal{L}_a$ summed for a sample $x_i \in x$ whose predictions and ground truth are further specified by z^i_s, z^i_s, and y^i_x. μ is the parameter used to balance the hard and soft losses. The function f normalizes or softens the outputs to form probabilities using a parameter $\mathcal{T}$ in a parameterizable softmax function [HVD15].

Response-based knowledge is relatively easy to implement and remarkably consistent with the expectation of dark knowledge[5]. However, RsKD seems to be detrimental to refined knowledge transfer (e.g., between models for tasks such as OD) due to the teacher's lack of supervision over the intermediate layers of the models considered.

3.2.2.2 *Feature-based Knowledge*

It is extracted during representation learning, which is usually a synonymous expression [Yan+22]. It is also designated attention-guided Knowledge by Romero et al. [Rom+15] and represents the extension of RsKD, where the knowledge arises in both the output and intermediate layers of the teacher network (seen Figure 3.10). The target entity in the output layer remains the same as in the response-based knowledge. Intermediate outputs or Hints for teacher and Guided for student [AM90], or feature maps are considered for the intermediate layers.

5 Dark Knowledge was introduced in 2014 by Hinton et al. in the context of knowledge extraction. It is the set of information obtained in computing new probabilities when the Softmax function or its temperature variable ($\mathcal{T}$) is modified [HVD14].

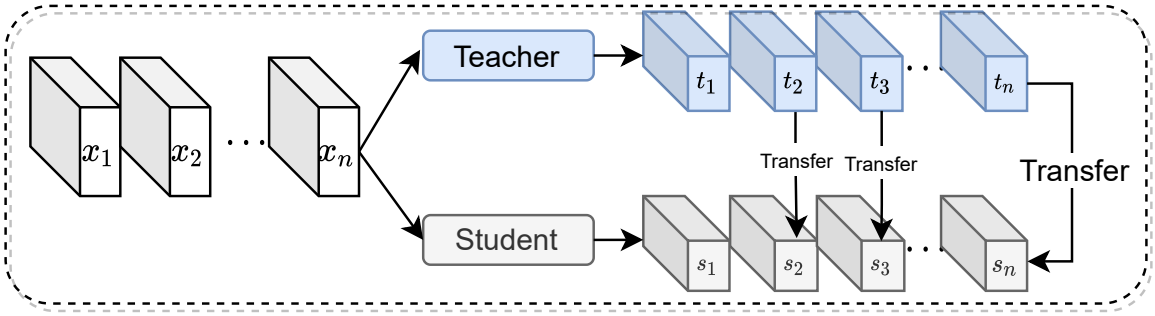


Figure 3.10: Feature-based knowledge distillation: Considered entities are matched with each other. Meaningful items here are logits and feature maps of student and teacher models. In this process, the feature maps to be matched must be carefully selected.

The overall loss is calculated by summarizing different types of feature-based knowledge from the perspective of the assumed source layer, feature types, and distillation loss. Given a labeled training set $\mathcal{X}$, the FKD loss of a given sample $(x_1, x_2, \ldots, x_n) = x \in \mathcal{X}$, corresponding to [Che+17] and [ZM20], is calculated as follows:

$$\begin{aligned}\mathcal{L}^x_{FKD}(z_s, z_t, f_s, f_t) &= \lambda_1 \mathcal{L}^x_{hint}(\cdot) + \lambda_{23} \mathcal{L}^x_{RsKD}(\cdot) \\ &= \lambda_1 \mathcal{L}^x_2(\psi(f_s(x)), \psi(f_t(x))) + \lambda_2 \mathcal{L}^x_{hard}(z_s, y_x) + \lambda_3 \mathcal{L}^x_{soft}(p_s, p_t) \\ &= \lambda_2 \mathcal{L}^x_{hard}(z_s, y_x) + \lambda_3 \mathcal{L}^x_{soft}(p_s, p_t) + \lambda_1 \sum_{i=1}^{n} \mathcal{L}^x_2(\psi(f_s(x_i)), \psi(f_t(x_i)))\end{aligned} \tag{3.2}$$

where $\mathcal{L}_{FKD}(\cdot)$ is the general feature-based loss, $\mathcal{L}_{hint}(.)$ is the hint loss function, λ_1, λ_2, λ_3 and λ_{23} are weighting hyperparameters. As in the previous section, z_s and z_t for x are the student and teacher prediction sets, respectively. p_s and p_t are the associated probabilistic 2D sets output by the linear map f defined for RsKD. f_s and f_t are functions that determine student and teacher feature maps, respectively, to be considered in the loss calculation. y_x is the ground truth associated with x. ψ reshapes the feature maps to the same size and results in a matrix. $\mathcal{L}_{RsKD}(\cdot)$, $\mathcal{L}_2(.)$, $\mathcal{L}_{hard}(.)$, and $\mathcal{L}_{soft}(.)$ denote, as in Section 3.2.2.1, respectively the RsKD loss, the $\mathcal{L}_2$ norm distance, the hard loss, and the soft loss. The divergence loss $\mathcal{L}_2$ for a entity (p, q), where $p = \psi(f_s(x))$ and $q = \psi(f_t(x))$ define equally sized student and teacher matrices $2D$ for x and p_i & q_i column vectors resp. of the matrices p & q at column i, the norm distance is calculated as [AM90]:

$$\mathcal{L}_2(p, q) = \sum_{i=0}^{\#q} \|q_i - p_i\|_2^2 \tag{3.3}$$

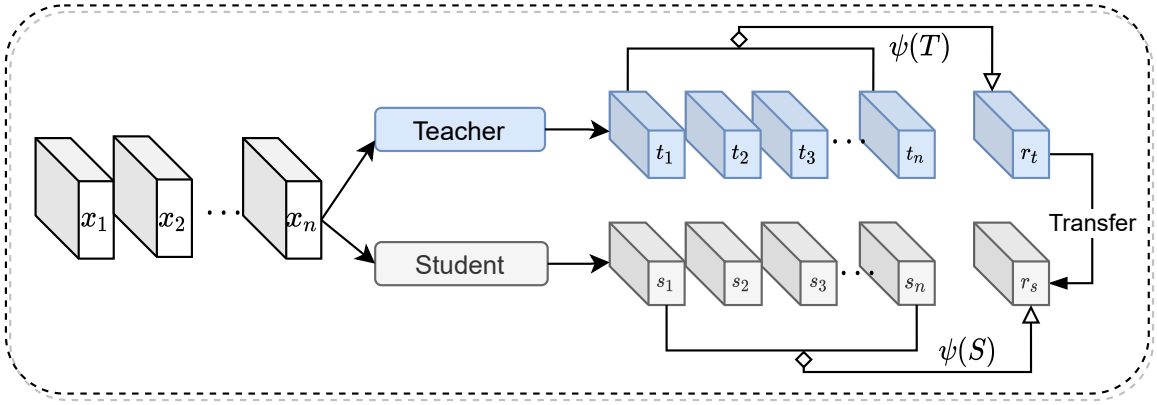


Figure 3.11: Relation-based Knowledge Distillation: T and S are sets of considered feature maps $(t_i)_{1\leq i\leq n}$ and $(s_i)_{1\leq i\leq n}$, respectively. ψ appraises and matches the relationships over T and S. The number of layers can be set arbitrarily between 1 and n, depending on how the relational function is designed.

Feature-based knowledge extends response-based knowledge by addressing feature maps. Therefore, FKD transfer is more accurate and provides favorable information for training. However, choosing, sizing, and matching hints and guided layers remain difficult to standardize [Che+21].

3.2.3 *Relation-based Knowledge Distillation*

In contrast to the IKD methods, where knowledge is transferred from point to point, relation-based knowledge distillation matches the structural relationship between chosen student and teacher (intermediate) outputs as shown in Figure 3.8b and Figure 3.11. In other words, RKD aims to compute structure-to-structure knowledge transfer between Teacher and student. Given the sample $(x_1, x_2, \ldots, x_n) = x \in \mathcal{X}$ from the precedent sections, where $\mathcal{X}$ is the labeled training samples set and f_s and f_t the linear functions

retrieving the future maps sets $S = \{f_s^1, f_s^2, \ldots, f_s^m\}$ and $T = \{f_t^1, f_t^2, \ldots, f_t^m\}$, $m \leq n$ for RKD, the model training aims to minimize the objective function [Liu+19], [Par+19]:

$$\begin{aligned} \mathcal{L}_{RKD}(T, S, x) &= \mathcal{L}(\phi(S, x), \phi(T, x)) \\ &= \sum_{i=0}^{n} \mathcal{L}(\phi(S, x_i), \phi(T, x_i)) \\ &= \sum_{i=0}^{n} \mathcal{L}(\underbrace{\phi(f_s^1(x_i), f_s^2(x_i), \ldots, f_s^m(x_i))}_{\text{Student-Relationship}}, \underbrace{\phi(f_t^1(x_i), f_t^2(x_i), \ldots, f_t^m(x_i))}_{\text{Teacher-Relationship}}) \end{aligned} \tag{3.4}$$

Where $\mathcal{L}$ is a divergence function (e.g., $\mathcal{L}_1$- or $\mathcal{L}_2$-norm), $(f_s^k(x_i))_{1 \leq k \leq m}$ and $(f_t^k(x_i))_{1 \leq k \leq m}$ are, for an input $x_i \in x$, the outputs in layers s_k and t_k, which could be the final outputs [Liu+19]. ϕ is the (potential) relational or predicate function that defines the relationship between the input variables. Setting $n = 1$ and $\phi = \mathbb{1}$[6], the formula yields the objective function for IKD. Therefore, RKD is a generalization of IKD. Optimizing effectiveness and efficiency by training RKD-student models depends on the involved predicate function [Liu+19]. Two popular predicate functions in the literature on RKD are:

Euclidean distances between selected feature maps [Liu+19]
Here, the distance-wise relational function, slightly changed to ϕ_D for the sake of distinction, aims to measure the Euclidean distance between pairs of selected layer output subsequently for student and teacher models. The training process is intended to minimize the divergence of Euclidean distances between the output couples considered. Given the last used mini-batch x, $k \neq l$ with $k, l \in \{1, 2, \ldots, n\}$, and chosen feature maps $S = (f_s^k, f_s^l)$ for the student and $T = (f_t^k, f_t^l)$ for the teacher model, following formulations are highlighted for ϕ_D [Liu+19]:

$$\phi_D(S) = \frac{1}{\mu} \left|\left| f_s^l - f_s^k \right|\right|_2^2, \; \phi_D(T) = \frac{1}{\mu} \left|\left| f_t^l - f_t^k \right|\right|_2^2, \tag{3.5}$$

Where μ is the normalization parameter defining, in this case, the average distance between pairs of samples considered. The resulting loss for x is then determined by the following modification of the aforementioned RKD objective Equation 5.8 [Liu+19]:

6 Identity function

$$\begin{aligned}\mathcal{L}_{RKD-D}(T,S,x) &= \mathcal{L}_\delta(\phi_D(S,x),\phi_D(T,x)) \\ &= \sum_{i=0}^{n} \mathcal{L}_\delta(\underbrace{\phi_D(f_s^k(x_i),f_s^l(x_i))}_{\text{Dist. relation student}},\underbrace{\phi_D(f_t^k(x_i),f_t^l(x_i))}_{\text{Dist. relation teacher}})\end{aligned} \tag{3.6}$$

where $\mathcal{L}_\delta$ is the Huber loss function for a δ-threshold defined by the following formula [Liu+19]:

$$\mathcal{L}_\delta(x,y) = \begin{cases} \frac{1}{2}(x-y)^2, \ for\ |x-y| \leq \delta \\ \delta|x-y| - \frac{1}{2}\delta^2, \ otherwise \end{cases} \tag{3.7}$$

The more significant the $\mathcal{L}_{RKD-D}$, the worse and more inappropriate the student's learning potential. $\mathcal{L}_{RKD-D}$ should converge to zero for good knowledge transfer [Liu+19].

Euclidean angles between selected feature maps [Liu+19]
Here, the angle-wise relational function ϕ_A aims to measure the angle between a couple or triplet of chosen layer outputs. Given again the last used mini-batch x, $k \neq l \neq m$ with $k,l,m \in \{1,2,\ldots,n\}$, and selected feature maps $S = (f_s^k, f_s^l, f_s^m)$ for the student and $T = (f_t^k, f_t^l, f_t^m)$ for the teacher model, following formulations are highlighted for ϕ_A [Liu+19]:

$$\phi_A(S) = cos\left(\angle(f_s^k, f_s^l, f_s^m)\right),\ \phi_A(T) \ = cos\left(\angle(f_t^k, f_t^l, f_t^m)\right) \tag{3.8}$$

The angle-wise loss can be obtained by considering angle-wise relational functions from both the student and the teacher as follows [Liu+19]:

$$\begin{aligned}\mathcal{L}_{RKD-A}(T,S,x) &= \mathcal{L}_\delta(\phi_A(S,x),\phi_A(T,x)) \\ &= \sum_{i=0}^{n} \mathcal{L}_\delta(\underbrace{\phi_A(f_s^k(x_i),f_s^l(x_i),f_s^m(x_i))}_{\text{Angle relation teacher}},\underbrace{\phi_A(f_t^k(x_i),f_t^l(x_i),f_t^m(x_i))}_{\text{Angle relation teacher}})\end{aligned} \tag{3.9}$$

$\mathcal{L}_\delta$ is the previously mentioned Huber loss function.

The magnitude of this angle-divergence is the basic information about the learning potential of the student model correspondingly to the angular structures to be transferred by

the teacher. Due to the higher order in terms of relational angular potential (considering three output patterns), $\mathcal{L}_{RKD-A}$ tends to transfer more knowledge from the teacher to the student than $\mathcal{L}_{RKD-D}$, which uses only pairs of output patterns for distance-wise potentials. Unlike IKD, these two approaches do not directly match the outputs of the teacher and the student, but the student model learns from the distance and angle structures of the hidden layers outputs of the teacher. The main challenge is yet to choose the appropriate distillation target layers for both the student and the teacher and find the right relationship to match.

3.2.4 *Online, Offline, and Self-distillation*

In a teacher-student-based approach to knowledge distillation, authors generally distinguish three main schemes that depend on the model architectures for teacher and student models and whether or not the teacher is trained simultaneously with the student. (1) Offline distillation is a method in which a pre-trained teacher model guides an associated student model [Gou+21]. In this type of distillation, the teacher model is first trained on a given training dataset, and its predictive ability then describes the distilled knowledge to train the student. As for recent Deep Learning model compression advances, offline distillation is the most widely used KD technique [Gou+21].

A pre-trained teacher model may not be available for several compression use cases, limiting offline distillation. To overcome this limitation, (2) online distillation has been developed [Gou+21]. It is a distillation scheme in which the teacher and student models are updated simultaneously in the same training process. Implementing online distillation requires many computational resources (e.g., parallel computing) [Gou+21]. Therefore, it gives outstanding results in terms of the efficiency of the trained model [Gou+21]. In (3) self-distillation, teacher and student models are defined with the same structure. It represents a particular case of online distillation, where the same model is trained and performs itself [Gou+21]. The model, acting as the teacher during training, builds knowledge in an ongoing epoch passed to the student in a later training epoch [Gou+21].

3.3 Related Work Summary: Analysis and Open Questions

All in all, well-configured (fine-tuned and regularized) deep learning models can achieve incredible performance in computer vision. Many CNN-based approaches have already been developed for tasks such as object detection. YOLO-v3 is currently the leading architecture used as the Head in YOLO-v4, which today represents a state-of-the-art for real-time detection. However, training or deploying YOLO-v4 to edge or RDCD is time-consuming and impractical for real-time applications such as mobile object detection. Compression employing knowledge distillation can circumvent this limitation, although the distillation scheme chosen can significantly affect the performance of the compression technique used. Nevertheless, considering the different variations of this compression technique and the distillation methods already presented, it remains to be investigated which methods might be suitable, especially in the case of offline learning, to achieve great compression results where the performance of the compressed model is still comparable to that of the original model. There is also the question of whether and how the presented variants of knowledge distillation, mostly used for classification, might be suitable for object detection models. Answering this question requires an experimental approach, which we realize in the next section.

4

DESIGN OF OWN APPROACH

4.1 Requirements and Assumptions

As stated in the introductory session, it is required to tackle the problem of compressing DNN models with KD. To do this, we are using YOLO-v4 to empirically investigate the different methods outlined in Section 3.2. A working implementation of YOLO-v4 must be considered as baseline model for the distillation. This model can be extended with a new backbone, resulting in an OD system that can operate with Daknet-53 or MobileNet-v2 as its backbone. The models are implemented using PyTorch. Training and evaluation will be conducted on the COCO dataset [Dat23]. The training and testing/evaluation processes are supervised, and the first time involves training a Teacher and a Student separately from scratch. The trained Student is essential for evaluating the effectiveness and performance of the implemented distillers (distilled models). The evaluation is done on the COCO evaluation set to measure the performance of the distilled models. This evaluation includes metrics such as inference time, precision, and recall [Lin+14].

4.2 System Overview

The distillers are incorporated as loss functions in the baseline model's backbone and detection components. The Darknet-based model serves as the Teacher and the Mobilenet-v2 variant as the Student, as seen in Figure 4.1. Losses penalize the transferred knowledge according to the Teacher-Student architecture. The three described types of knowledge in knowledge distillation are investigated: response-, feature-, and relation-based.

In the Teacher-Student architecture, the Student, who is untrained, is supervised by both the trained Teacher and the Ground Truth Label (GTL). This setup is used to explore the impact of the newly formulated losses on the Student's learning capacity. We can differentiate between three losses as illustrated in Figure 4.1. The Student loss is calculated using GTL, forming the hard loss discussed in the preceding section. The soft loss is the second loss, which is calculated using soft labels and soft predictions, as shown in Figure 4.1. Soft labels

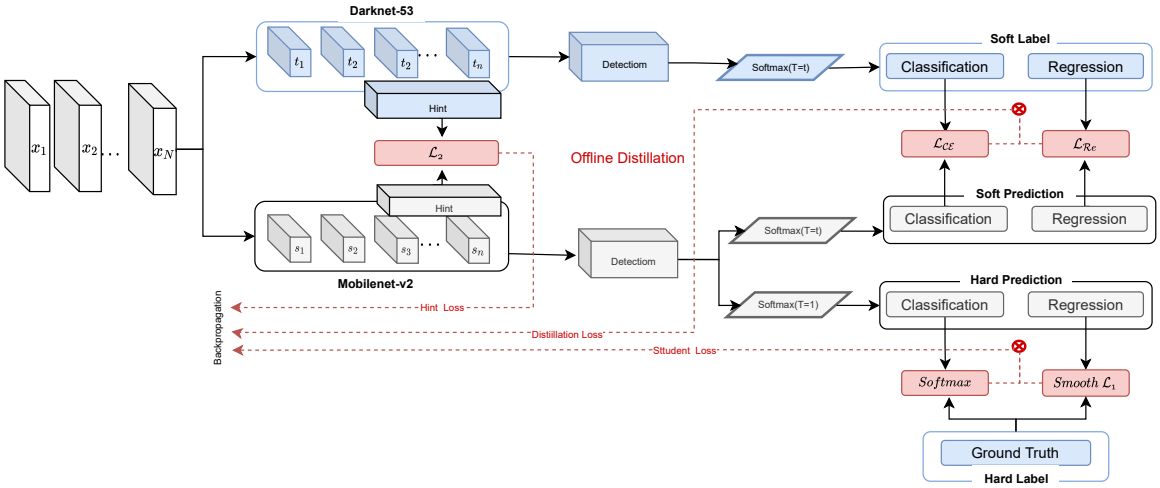


Figure 4.1: System overview: Distillation loss is computed using soft Teacher and Student predictions. Hint learning helps in matching intermediate knowledge between Teacher and Student intermediate layers.

describe the predicted classes and regressed bounding boxes; soft predictions are those normalized by a temperature hyperparameter $\mathcal{T} \neq 1$ that parameterizes the normalizing softmax function. Setting $\mathcal{T} = 1$ results in hard predictions. The soft loss illustrates the distillation loss. The hint loss is the third type of loss taken into account, which is related to the KD of the intermediate predictions. This is not applicable in the response-based variant, as only the logits, i.e., the final predictions, are necessary to calculate the distillation.

In FKD, the hint loss is determined by a pair (f_s, f_t) of intermediate feature maps from the Student and Teacher, respectively. In RKD, it refers to the loss by a pair (r_s, r_t) of the Student's and Teacher's formulated or computed relational structures. The total loss obtained by adding the hint loss to the two previous losses is then integrated into the backpropagation operation to update the parameter adjusted with the new knowledge.

The impact of this adaptation is estimated in the evaluation. We use COCO metrics such as AP and AR to measure the performance. Additionally, we will assess the model statistics (inference time) and training (training time).

5

IMPLEMENTATION

Having shown in the last part what design decisions are made of and what the overall implementation design looks like, we now go into detail about the implementation tasks. Firstly, we briefly discuss which technologies make up our development environment. After going to the dataset used, we then explain how we built the models on which the whole work is based. We present our own contributions by discussing the various KD variants we implemented. We will focus on how the loss functions used to transfer knowledge are enforced. Furthermore, we show how these losses ensure interactions with essential components of the baseline models via the training routine. The chapter ends finally with a summary of all implementations, wherein some related constraining aspects are reflected.

5.1 Technologies

As seen in Figure 5.1, the requirements are implemented in the Python programming language. PyTorch is the go-to library due to its extensive range of deep learning capabilities that are easy to use. Other Python libraries, such as Torchvision and TensorboardX, which are gaining traction for implementing CNN-based image processing algorithms, are also utilized. Data discussed here come, as mentioned earlier, from the COCO dataset (addressed in the next section) and are originally annotated in JSON format.

Due to the complex computations behind the implemented models, training and testing are performed in a separate environment from the implementation environment. Both computing units Central Processing Unit (CPU) and GPU are provided here. For training, provisioned cloud notebooks such as colab and JupyterLab are used. We additionally and mainly employ the Nvidia GPU server of the University Department of NCS, where the project is performed. The evaluation also takes place in colab.

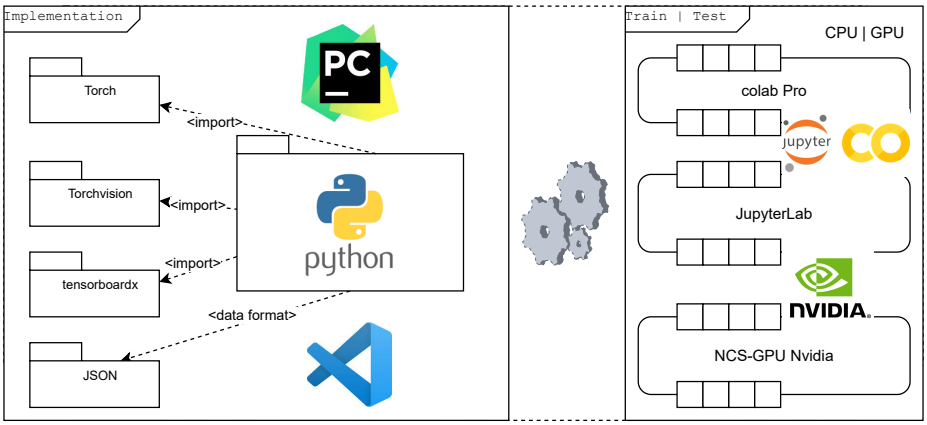


Figure 5.1: Implementation environment: Implementation and training take place in separate areas for optimal task performance.

5.2 Baseline Models: Implementation and Experiment

5.2.1 *COCO-Dataset and Data Retrieval*

5.2.1.1 *COCO-Dataset*

Common Objects in Context (COCO) is a large-scale dataset of images for object or key point detection, fabric or panoptic segmentation, and image annotation that contains about 1.5M object instances for 80 object categories [Lin+14; Dat23]. COCO images are annotated, and the annotations for OD are stored in a JSON file that contains full details of the bounding boxes for objects within a given image.

The building blocks (see listings 8.1 and 8.2) for the JSON annotation file contain (1) general descriptions, (2) licenses that specify the license for each image, (3) categories describing the object categories in the dataset, (4) unique image information without information about bounding box or segmentation, and (5) annotations that create a list of all individual object annotations for each image in the dataset. The annotations section in Listing 8.2 contains the object segmentation or bounding box output for object detection. If an image has four objects to be detected, annotations for all four objects would be built. In this manner, if the entire dataset consists, for example, of 300 images and has a total of 100 objects, then we will have 100 annotations.

image_id under annotations in Listing 8.2 describes the identifier field from the image dictionary. this value should be used to cross-reference the image with other dictionaries. It differs from the *id* field, which describes the unique identifier for annotation. It is only an annotation ID and doesn't point to a particular image in other dictionaries. The *category_id* describes the class of the objects corresponding to the annotation id field inside the section *categories*. The extension *bbox* stays for bounding box and describes the coordinates (top left x, top left y, width, height) of the rectangle around the object; it's beneficial for individual objects extracted from images.

5.2.1.2 *Data Retrieval*

The first step in training the baseline model involved retrieving the data in the right format. This entailed converting the COCO annotations into a format that the training routine could read. For this purpose, a Python conversion script has been written, providing an annotation schema of the form $< image\_path,\ x_1,\ y_1,\ x_2,\ y_2,\ id_1,\ x_1,\ y_1,\ x_2,\ y_2,\ id_2,\ ... >$ from information in the JSON file identifying an image. In this process, the annotations are converted to YOLO format consisting of the image path, positions of the objects inside the addressed image, and class-ids (*category_ids*) of these objects. The conversion is performed for each image, and the resulting converted annotations are collected in a text file serving as the principal annotation file. This file has been formulated in the current project according to the JSON-formatted Train/Val annotations of 2017 from the COCO website, describing images from the Train and Val COCO datasets from the year 2017. The annotation file obtained in YOLO format affords to start the training procedures of the models we have implemented, which we will discuss in the following sections.

5.2.2 *Baseline Model*

Tianxiaomo's GitHub repository was chosen as the baseline model. This PyTorch implementation of YOLO-v4 yields good performance compared to the C-implemented original YOLO-v4. Its comprehensive, clear documentation and good evaluation results make it stand out from other potential base models that have been explored. We name and use this implementation further as *YOLOteacher*.

5.2.2.1 *Model Architecture*

From an architectural perspective, the model includes all components listed in Section 3.1.2 for the original implementation by Bochkovskiy, Wang, and Liao [BWL20]. Darknet specifies the Backbone implemented in PyTorch, just like all other components of the basic two-stage YOLO architecture. The extracted features are passed to the Neck, which performs the upsampling and downsampling of the feature maps before they are available in the correct form to the Head. In addition, the SPP elements enable fine-tuned selection of feature candidates. Suitable and correctly shaped features are then sent to the Head to perform the task studied in Section 3.1.2 and output bounding box shapes. The last describe $4D$ tensors of the form $batch\ size \times channels\ number \times high \times width$. The batch size determines how many images from the training data are used to estimate the error gradient before updating the model weights. Throughout the implementation, the batch size is set to 64 with a mini-batch of 4.

5.2.2.2 *Training Routine*

The training routine is used to complete a trained model. All YOLO components are imported for this purpose. The $yolov4.cfg$ file supplied by the original YOLO-v4 authors provides the basic configuration. This file parameterizes the entire network and layer architecture, including layer classes (corresponding to Section 2.2.1.3), filter, stride, padding, activation functions, and more. Based on these parameters, the YOLO layers are defined. The Darknet model is initially used to process an incoming mini-batch during training. This performs feature extraction according to its own architecture, as outlined in Section 3.1.3.1. As a result, feature maps (intermediate output) are generated for each image in the mini-batch. Using the Neck, which inherits parameters from the YOLO configuration, the feature maps are then selected from the Backbone and converted to YOLO format. This process defines the general job of the so-called YOLO-Body (Backbone+Neck); it results in a list of three $4D$ tensors representing the outputs of the three last Neck layers. The YOLO-Body finally reaches the YOLO-Head, which uses the three $4D$ tensors as input. This Head, which in this case, as shown in the Related Work, describes YOLO-v3, also operates according to the defined YOLO-layers and -architecture. Ultimately, this yields a prediction set of three elements, also consisting of $4D$ tensors.

For example, an image of size 608×608 yields, at the Backbone, $4 \times 256 \times 76 \times 76$, $4 \times 512 \times 38 \times 38$, and $4 \times 1024 \times 19 \times 19$, which at the Head result prediction tensors of

the forms $4 \times 255 \times 76 \times 76$, $4 \times 255 \times 38 \times 38$, and $4 \times 255 \times 19 \times 19$, respectively. Thereby the kernel sizes are determined according to Figure 3.3. The predictions are achieved by applying 1×1 detection kernels to the three input feature maps [Mao+19]. Detection kernels in YOLO-V3 (the YOLO-Head) take the form of a $1 \times 1 \times (B \times (5 + C)$-tensor, where B defines the number of bounding boxes that a cell on the feature map can predict [BWL20]. The digit "5" refers to the four bounding box attributes (t_x, t_y, t_w, t_h) and the objectness score (p_0) [BWL20]. C represents the number of classes in the dataset. Since the training is done with COCO, we have $B = 3$ and $C = 80$, and the resulting kernel size is $1 \times 1 \times 255$. This justifies the predicted final size of the bounding boxes, which also describe the general results of the YOLO-Body.

Algorithm 1 Training routine for the baseline model: The model architecture is imported into the training script and called in the training loop. The result is a trained model continuously improved by backpropagation applied to the computed general divergence loss.

Input: *YOLOBody*, *YOLOloss*
Output: Trained Darknet-based YOLO-model
Function `train(·)`**:**
  for $e \in epochs$ **do**
    for $batch\_img, batcth\_bboxes \in data$ **do**
      $bboxes\_pred \longleftarrow YOLOBody(batch\_img)$
      $\mathcal{L}_{total}, \cdots \longleftarrow YOLOloss(bboxes\_pred, batcth\_bboxes)$
      $Backpropagation(\mathcal{L}_{total})$
      $e \longleftarrow e + 1$
    end
    save *trained_YOLO_student_e*
    save $metrics = eval(trained\_YOLO\_student\_e)$ `/* Evaluation metrics of YOLO-model resulting in the training epoch` e `are saved */`
  end
End Function

The training routine is performed for $\mathcal{E}$ epochs according to algorithm 1 and on the COCO Train 2017. A training epoch $e \in range(\mathcal{E})$ runs through iteration over the whole training dataset, which is a composition of the images annotated with the ground through bounding boxes. The processing of the dataset in e is done according to the defined mini-batch size (4 in our case). Hence, four images are considered and the YOLO-Body performs the necessary calculations described earlier to generate predicted bounding boxes. The YOLO-loss class helps to reduce the difference between predicted and ground truth

bounding boxes. It returns the total loss and the various auxiliary losses (IoU, regression, classification). The backpropagation in ML carries the loss from the current iteration to the next, thereby correcting the error gradient that accumulates during the forward propagation in the network [NLP20].

It takes approximately 6445 iterations to train a model on the 25779 images in the COCO 2017 training dataset in a single epoch. The training in an epoch is completed by passing the trained model into the evaluation process, where the evaluation metrics are determined. The evaluation routine is based on the COCO Eval 2017 dataset, which consists of 4952 labeled images. This function evaluates the performance of the trained model based on the average precision and recall scores, which will be discussed further in Section 6.1.2.

5.2.3 *Extended Baseline Model*

After the selected baseline mode was up and running, and all essential components regarding its architecture were clear, we are currently into the second milestone of the work, where a new module for the *YOLOstudent* has to be built. We name this new structure *extended/ext. baseline model* or *YOLOstudent*. The main task here is to use Mobilenet-v2 as a new Backbone in the YOLO architecture and to get the whole system working again with the new construct. To build this new module, the model architecture of the Baseline model is reused over almost entirely. Only some components must be rebuilt, namely the Neck and the Backbone.

The Backbone is rebuilt because Mobilenet-v2 is used here instead of Darknet-53, and, as explained in Section 3.1.3, both architectures operate and are structured differently. Mobilenet-v2 is an already established network whose architecture has become standardized. Accordingly, its complete PyTorch implementation is loaded from a reliable GitHub repository into the baseline system as *mnv*2. The new module is then built into the system as a new feature extractor, ensuring that the YOLO-Body is adapted accordingly and *mnv*2 is imported into it. Up- and downsampling convolution blocks are then adjusted according to their layer settings (input, output channels, activation function). The appropriate setting of the Neck ensures the selection of the appropriate feature maps. According to the Mobilenet-v2 architecture seen in Section 3.1.3.2, this uses a filters-list of the form $[32, 96, 1280]$ to parameterize the number of kernels in layers of the Neck network, where the selected outputs of the Backbone should be entered before upsampling. This filter is

the same as the one used in the feature extractor to set the output channel size at extraction points. Compatibility is created in this way between Backbone output and Neck input sizes. On the first Neck layer (matching the last Backbone layer), which outputs 1280 kernels, a SPP layer is applied, which generates or selects adequate feature maps according to defined pooling sizes to be forwarded to the Head. The pooling-based intermediate results are upsampled feature maps with 1024 channels. they are then downsampled to the YOLO format with where 256 input channels are required. The adjusted feature maps continue to operate as Head input, with the Head retaining the same structure as in the Darknet baseline model. The training routine of the extended baseline model does not significantly differ from that of the baseline. The only difference is that instead of the Darknet-based training and evaluation models, Mobilenet-based YOLO-Body is used.

5.3 Distilled Object Detectors

After the baseline models have been built, we now reach the distillation milestone. For this purpose, the three KD options addressed in the Related Work are implemented. Knowledge transfer occurs through different approaches, especially the previously presented loss calculation approaches. We thus go from RsKD, over feature- to relation-based KD. In addition, a new approach is tested, an extended RKD, where all three distillation types are used simultaneously during training. We call this new approach Multi-type Knowledge Distillation (MKD).

5.3.1 *Response-based Knowledge Distillation*

5.3.1.1 *Model Architecture*

As in Section 3.2.2.1, this is about the inheritance of the Teacher predictions from a Student model. Focus is put here thus only on the logits, which refer to the outputs of the detector (Head) in the used YOLO architectures. For this goal, the outputs of the regression and the classification assigned to the detected objects were considered. Hence, as shown in Figure 4.1, the total loss is calculated with respect to two main summands: the Student loss $\mathcal{L}_{Student}$ and the Distillation loss $\mathcal{L}_{distillation}$. For their estimation, they are first normalized to probabilities expressed by a softmax function. The happening softening process depends on the temperature parameter ($\mathcal{T}$) described in Section 3.2.2.1, which sets the intensity of the softening.

The Student Loss describes the value of the Loss function when training the Student model supervised, i.e., guided by well-defined ground truth labels for bounding boxes and object classes. The achieved loss values correspond to the loss obtained when training the model from scratch. In RsKD, the summands describe the hard loss properly to the hard predictions and hard label (ground truth) since an increased softening at the Student prediction is accomplished by setting $\mathcal{T} = 1$. The hard logits become smoothened by employing - for supervising through hard labels - a *Smooth* $\mathcal{L}_1$ and the usual temperature independently Softmax function in each case for the regression and the classification. However, the Student predictions are slightly softened when the distillation loss has to be estimated. For such an achievement, the softmax function with $\mathcal{T} > 1$ is required. The resulting loss is the loss that occurs during training supervised by the Teacher.

5.3.1.2 *Training Routine*

The training routine of the extended base model specifies the training routine for the RsKD. It works following algorithm 2, with additional inputs being: the network and a trained model of the *YOLOteacher*, as well as some functions to determine the distillation loss. A copy of the implementation of the *YOLOteacher* defines the training routine to be modified. It is expected that a trained Student model will be obtained that takes into account the Teacher's predictions and that its performance will be evaluated using some known evaluation metrics for ML models.

The training loop is implemented over a given epoch set $\mathcal{E}$. For each epoch, $e \in \mathcal{E}$, training and testing datasets are considered, on which the Teacher and Student prediction models are expected to operate. As seen earlier in algorithms for base models, the training dataset denoted $\mathcal{X}$ is divided into mini-batches x before being explored by a model. x is input to both *YOLOteacher* and *YOLOstudent* to compute predictions that are cached as *teacherpredictions* and *studentpredictions*, respectively. For a given mini-batch, predictions are computed in the same format, a list of three $4D$ bounding boxes tensors. By iterating over this equally long list, a transformation-function $trans(\cdot)$ optimizes the formats of the prediction tensors. For RsKD, *trans* defines, according to Section 3.2.2.1, the parameterized softmax-based linear function that transforms the predicted values into probabilities and modifies the output $4D$ tensors in $2D$ (matrices) so that they can be suitable for the divergence function.

Consider a mini-batch $(x_1, x_2, x_3, x_4) = x \in \mathcal{X}$ for the mini-batch size specified by 4, whose prediction sets by Student and Teacher models are $z_s = teacherprediction(x)$ and $z_t = studentprediction(x)$, respectively. For a given prediction subset $z_t^i[x] = (z_t^i[x_k])_{1 \leq k \leq 4}$ and $z_s^i[x] = (z_s^i[x_k])_{1 \leq k \leq 4}$ at index position i in resp. z_s and z_t, $F^x(i) = trans(z_s^i[x]) = (trans(z_s^i[x_k])_{1 \leq k \leq 4}$ and $G^x(i) = trans(z_t^i[x]) = (trans(z_t^i[x_k])_{1 \leq k \leq 4})$ in algorithm 2 are the probabilistic $2D$-expressions of the i_{th} outputs of Student and Teacher models since the model predictions are output at different levels in the model Head; the number of levels defines the cardinality of z_s and z_t. To calculate the distillation loss, they serve as inputs to the $logitloss(\cdot)$ function. The total loss is then computed according to the following modification of Equation 3.1:

$$\begin{aligned}\mathcal{L}_{RsKD}(z_s, z_t, x \in \mathcal{X}) &= \sum_{i=1}^{3} \mathcal{L}_{RsKD}(F^x(i), G^x(i)), \quad \text{since } \#z_s = \#z_t = 3 \\ &= \sum_{i=1}^{3} \mathcal{L}_{RsKD}(trans(z_s^i[x]), trans(z_t^i[x]))\end{aligned}$$

$$\Downarrow$$

$$\boxed{\begin{aligned}\mathcal{L}_{RsKD}(z_s, z_t, x \in \mathcal{X}) = \sum_{i=1}^{3} \Big(\alpha \underbrace{\sum_{k=1}^{4} \mathcal{L}_{distillation}(\overbrace{trans(z_s^i[x_k])}^{A}, \overbrace{trans(z_t^i[x_t])}^{B})}_{\mathcal{L}^x_{distillation}} + \\ (1-\alpha) \underbrace{\sum_{k=1}^{4} \mathcal{L}_{student}(trans(z_s^i[x_k]), y_{x_k})}_{\mathcal{L}^x_{student}} \Big)\end{aligned}} \tag{5.1}$$

with $\mathcal{L}_{distillation}(A, B) = KL(A, B) = A \cdot \log\left(\frac{A}{B}\right)$

where $\mathcal{L}_{RsKD}$ is the overall response-based loss represented in Section 3.2.2.1, $\mathcal{L}^x_{distillation}$ is the Teacher-based resp. smooth loss and $\mathcal{L}^x_{student}$ is the Student-based or hard loss for x. $\mathcal{L}^x_{student}$ is estimated based on the Kullback–Leibler (KL) function defining the Kullback-divergence operation or KL-divergence or KL-entropy. It is a function that denotes a measure of the difference between two probability distributions collected in the form of vectors of matrices [Joy11]. This justifies the preliminary conversion of logits into $2D$ probabilistic variables. Both $\mathcal{L}^x_{student}$ and $\mathcal{L}^x_{distillation}$ are respective summaries of classification and regression losses presented in Figure 4.1 and weighted by a hyperparameter . As with

the base models, the final step in the training loop of RsKD is to back-propagate the loss just estimated. Next comes saving and evaluating the trained model over the evaluation dataset.

Algorithm 2 Training routine for RsKD: The trained *YOLOteacher* model is incorporated into the *YOLOstudent* training script. It is called into the training loop, where its results are used to compute the divergence loss. The Teacher is not trained in this process; he only provides his results as auxiliary labels for the Student, through which his knowledge is thus transferred. The new loss is determined to fit the Student's training loss to the ground truth and is passed to backpropagation. The trained model is then evaluated, and the resulting metrics are stored.

Input: *trained_YOLO_teacher*, *untrained_YOLO_student*, *YOLOloss*, *logitloss*, *trans*, *eval*
Output: Trained Student model, Evaluation Metrics `/* Expected results */`
Function `trainrskd(·)`**:**
  $teachermodel \longleftarrow trained\_YOLO\_teacher$
  $studentmodel \longleftarrow untrained\_YOLO\_student$
  for $e \in \mathcal{E}$ **do**
    $\mathcal{L}_{distillation} \longleftarrow 0$
    for $x \in \mathcal{X}$ **do**
      $teacherpredictions \longleftarrow teachermodel(x)$
      $studentpredictions \longleftarrow studentmodel(x)$
      $\mathcal{L}_{student}, \cdots \longleftarrow YOLOloss(studentpredictions, y_x)$ `/* ` y_x`: Ground truth */`
      for $i \leq \#studentpredictons - 1,\ i \in \mathbb{N}$ **do**
        $F(i) \longleftarrow trans(studentpredictions[i])$
        $G(i) \longleftarrow trans(teacherpredictions[i])$
        $\mathcal{L}_{distillation} \longleftarrow \mathcal{L}_{distillation} + logitloss(F(i), G(i))$
      end
      $\mathcal{L}_{RsKD} \leftarrow \alpha * \mathcal{L}_{distillation} + (1 - \alpha) * \mathcal{L}_{student}$
      $\mathcal{L}_{RsKD}.Backpropagation$
    end
    save $metrics = eval(trained\_YOLO\_student\_e)$
    save $trained\_YOLO\_student\_e$
  end
End Function

5.3.2 *Feature-based Knowledge Distillation*

5.3.2.1 *Model Architecture*

The second approach used in our implementation is FKD. As mentioned in related work, this distillation type deals with an individualized matching of the prediction outputs, just like RsKD. However, not only the logits are considered here, but also the Backbone features maps. Implementing FKD requires three summands, firstly determined independently before getting merged to build the general component. These are the Student loss, the logit loss, and the hint loss. The method applied to compute these losses is the Attention- or hint-guided approach mentioned in Section 3.2.2.2. Hint or Attention denotes the subset of layers whose feature maps from the entire set of Teacher feature maps in the intermediate layers are considered for computing the feature-based loss. A similar subset for the Student is referred to as Guided. In addition to this consideration as layer sets, Hint and Guided indirectly describe the associated feature maps set. In order for the knowledge to get transferred from Hint to Guided adaptively, a function resizing identically Hint and Guided features maps at the same layer position is used as an adaptation unit. This function is essential because, without it, hint feature maps, which are determined over the Darknet intermediate layers, cannot match the guided ones computed in *YOLOstudent* by Mobilenet-v2.

For the sake of restriction, two feature maps are selected for the Student and two for the Teacher under the detection layers in the respective Backbone architectures for the formation of the hint and guides, namely the last and the penultimate ones. The sorted out *YOLOteacher* feature maps are respectively of shape $4 \times 256 \times 20 \times 20$ and $4 \times 512 \times 10 \times 10$ while the considered *YOLOstudent* feature maps are of shape $4 \times 1280 \times 10 \times 10$ and $4 \times 96 \times 20 \times 20$. This configuration results from the image resolution being set to 320×320 during the distillation process, which according to Figure 3.3 and Figure 3.6, corresponds to the estimated shapes at the mentioned detection positions. The formed feature maps entities $(fm_s^{last}, fm_t^{last})$ and $(fm_s^{2nd\_last}, fm_t^{2nd\_last})$ must be in the same format so that the matching function described in Section 3.2.2.1 can successfully be applied. This task is performed by the adaptation unit, which converts the Guided feature maps to the size of the hint using Neck settings of *YOLOstudent*. The new function *transhint* transforms the same-sized feature maps in both the Teacher and Student from $4D$ to $2D$ format without softening. The final predictions employ the *trans* function from the last section. These transforming functions ensure that the loss is computed since the divergence function

operating here only allows 2D format input. This calculated loss is added to the RsKD loss to represent the overall loss.

5.3.2.2 *Training Routine*

Algorithm 3 Training routine for FKD: The trained *YOLOteacher* model is integrated into the *YOLOstudent* training script. Network architectures called in the training loop are extended by being able to output intermediate feature maps. The prediction results are used to compute the divergence loss. It is expected to obtain a trained Student model based on the hint-guided constellation.

Input: *trained_YOLO_teacher*, *untrained_YOLO_student*, *YOLOloss*, *logitloss*, *trans*, *eval hint_loss*
Output: Trained Student model, Evaluation Metrics `/* Expected results */`
Function `trainfkd(·)`**:**

```
teachermodel ⟵ trained_YOLO_teacher
studentmodel ⟵ untrained_YOLO_student
for e ∈ ℰ do
    ℒ_distillation ⟵ 0
    ℒ_hint ⟵ 0
    for x ∈ 𝒳 do
        teacherpredictions ⟵ teachermodel(x)
        studentpredictions ⟵ studentmodel(x)
        ℒ_student, ··· ⟵ YOLOloss(studentpredictions, y_x)    /* y_x: Ground truth */
        for i ≤ #studentpredictons − 4, i ∈ ℕ do
            F(i) ⟵ trans(studentpredictions[i])
            G(i) ⟵ trans(teacherpredictions[i])
            ℒ_distillation ⟵ ℒ_distillation + logitloss(F(i), G(i))
        end
        for 3 ≤ j ≤ #studentpredictons − 1, j ∈ ℕ do
            H(j) ⟵ transhint(teacherpredictions[j])
            Gu(j) ⟵ tranhint(studentpredictions[j])
            ℒ_hint ⟵ ℒ_hint + hint_loss(H(j), Gu(j))
        end
        ℒ_FKD ← λ1 * ℒ_distillation + λ2 * ℒ_student + λ3 * ℒ_hint
        ℒ_FKD.Backpropagation
    end
    save metrics = eval(trained_YOLO_student_e)
    save trained_YOLO_student_e
end
```

End Function

The training routine in FKD runs according to algorithm 3. It also starts with an import of the trained Darknet-based Teacher model. The base implementation is still the extended baseline model, where no trained model is considered. The routine runs over a set of epochs $\mathcal{E}$, where data sets for training and evaluation are passed through precisely as in the last described training routines. A significant difference is a new function for collecting predictions in nets for *YOLOteacher* and *YOLOstudent*, where, as described before, feature maps are taken into consideration.

The sets $z_x = studentpredictions$ and $z_t = teacherpredictions$ are multiplied because, besides the three known predictions, the three related feature maps are entered into the lists. Therefore, the lists are sorted out to distinguish between final and intermediate outputs. This is done by respecting the order of the list elements when collecting the predictions and feature maps in networks for Student and Teacher. Thus, the first three list elements comprise the predictions at the last three prediction layers in the Head, while the last three list elements define those of intermediate outputs in the three selected detection layers of the Backbone. For the implemented approach, two loops are built at each iteration in the training loop over a mini-batch $x \in \mathcal{X}$. Let $i, j \in \mathbb{N}$ be two index variables designating positions in the outputs sets from z_s and z_t following algorithm 3. One loop allows, with respect to outputs $z_s^i[x]$ and $z_t^i[x]$, $1 \leq i \leq 3$, to determine the logit-loss according to algorithm 2. In the other loop, for each output $z_t^j[x]$ and $z_s^j[x]$, $4 \leq i \leq 5$ (only two feature maps), *transhint* operates. It is the earlier mentioned adaptation function mentioned adaptation, which results in the $2D$ representations $H^x(j)$ and $Gu^x(j)$ of the feature maps at Hint and guided layers for x. These feed into the *hintloss* function, whose output is incremented to $\mathcal{L}_{hint}$ updating the FKD-loss. The overall loss for x, $\mathcal{L}_{FKD}$, is then obtained computing the weighted summation of $\mathcal{L}_{hint}$, $\mathcal{L}_{distillation}$ and $\mathcal{L}_{student}$ as follows:

$$\begin{aligned}\mathcal{L}_{FKD}(z_s, z_t, x) =& \lambda_1 \sum_{i=0}^{2} \mathcal{L}_{distillation}(trans(z_s^i[x]), trans(z_t^i[x])) + \lambda_3 \overbrace{\sum_{j=3}^{4} \mathcal{L}_2(Gu^x(j), H^x(j))}^{\mathcal{L}_{hint}} + \\ & \lambda_2 \sum_{i=0}^{2} \mathcal{L}_{student}(z_s^i[x], y_x)\end{aligned} \tag{5.2}$$

where y_x is the subset of ground truth predictions for x. The refined iterations on predictions of the respective images in the mini-batch x made in Equation 5.1 are not explicitly

mentioned in Equation 5.2, but they are still performed under the *trans* function to calculate the loss for the whole x. The hyperparameters λ_1, λ_2 and λ_3 are carefully set as mentioned in Section 3.2.2.2 so that the new loss is appropriately calculated before backpropagation. The training routine for epoch e is rounded off by the evaluation of the trained model, where the evaluation metrics are determined and the trained distilled model is stored.

5.3.3 *(Extended) Relation-based Knowledge Distillation*

5.3.3.1 *Model Architecture*

The model architecture of RKD does not basically enormously differ from that of FKD. The outputs considered so far are maintained as well as the Teacher and Student constellations. However, as Figure 5.2 shows, the relations between them are appraised instead of individual matching out outputs, as mentioned in the Section 3.2.3. As in the last section, we perform here with the two selected intermediate outputs, namely the last and penultimate Backbone layers. In contrast to the Section 3.2.3, we consider here for the calculation of the general loss, in addition to the relational divergence, the Student loss further, which serves as a basic summand. The relational loss is determined using the two methods presented in Section 3.2.3: The angle-based and the distance-based relational distillation. These methods were presented for use cases with two to three $2D$ vectors and now need getting extended to tensors since our application domain is based on images. A well-known mathematical concept that facilitates the extension is the Euclidean Distance Matrix (EDM).

As outlined in [Lel93], the Euclidean distance matrix is defined as follows: Denote $\mathbb{E}$ a Euclidean vector space, i.e., a vector space equipped with the Euclidean norm $\|\cdot\|_2^2$ employed earlier as L2-Norm. Given $x_1, x_2, \ldots, x_m$, m k-dimensional vectors of $\mathbb{R}^k$, the

EDM denoted A is the $n \times n$ matrix representing the spacing of the set $\{x_1, x_2, \ldots, x_m\}$ in a Euclidean space. It applies that:

$$A = (a_{(i,j)})_{1 \leq i,j \leq m} = (d^2_{(i,j)})_{1 \leq i,j \leq m} = \begin{pmatrix} 0 & d^2_{(1,2)} & d^2_{(1,3)} & \ldots & d^2_{(1,m)} \\ d^2_{(2,1)} & 0 & d^2_{(2,3)} & \ldots & d^2_{(2,m)} \\ d^2_{(3,1)} & d^2_{(3,2)} & 0 & \ldots & d^2_{(3,m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d^2_{(m,1)} & d^2_{(m,2)} & d^2_{(m,3)} & \ldots & 0 \end{pmatrix} \tag{5.3}$$

where $d^2_{(i,j)} = \|x_i - x_j\|_2^2$ is the Euclidean distance between vectors x_i and x_j.

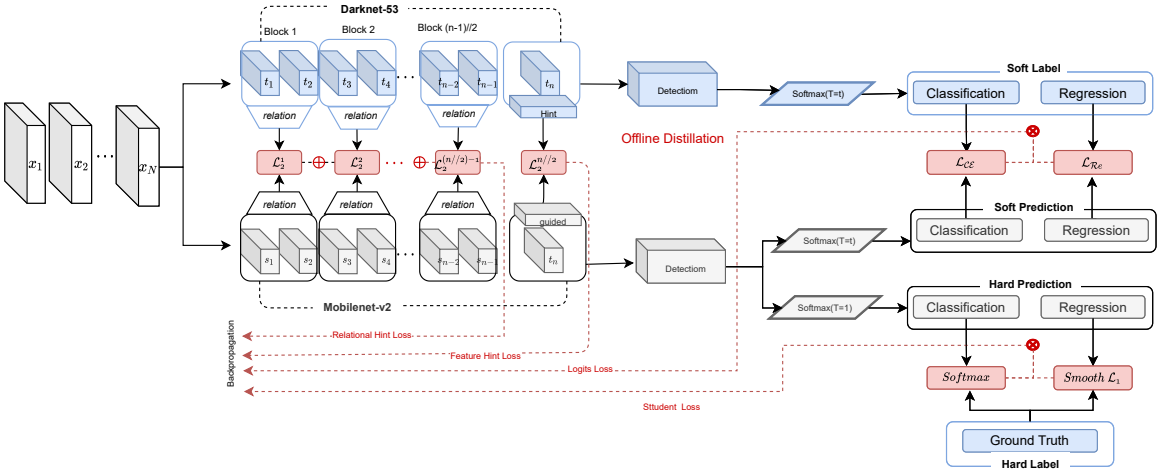


Figure 5.2: (Extended) Relationship-based knowledge distillation: instead of single entities, relationships in selected pairs of Student and Teacher intermediate outcomes are matched. The divergence between the calculated relationships helps to improve the training of the Student model.

5.3.3.2 *Training Routine*

The training routine is run according to algorithm 4. Like previous models, it builds on a trained darknet model, which is imported into the adopted extended baseline implementation. The aspiration is still to get a trained Student model using distillation, where angle and distance relations within the respective model feature maps set up the distillation. Just as in previous models, the training loop starts within an epoch by accumulating the predictions of the Teacher and Student models. Considering the same prediction sets z_s

and z_t as in Section 5.3.2 for a mini-bach x, the Student loss is calculated exactly as in previous training routines.

As already mentioned, the general loss in RKD takes this loss as a basis. Some additional summands are counted, like the already known logit loss, which we calculate in this approach exactly as in FKD. The new summand here is the relational loss. Given the feature maps $H_j = (z_s^j)_{4 \leq j \leq 5}$ and $G_j = (z_t^j)_{4 \leq j \leq 5}$ which result from the last section corresponding to the prediction sets z_s and z_t, the goal is to find the appropriate EDM for each tensor pair (H_4, H_5) and (G_4, G_5), respectively. Since H_4 and H_5 as well as G_4 and G_5 are respectively from different shapes within networks for Mobilenet and Darknet, they are $4D$ tensors but of different shapes, as we have seen so far. In FKD, in such a case, we individually matched the selected feature maps of the Student model to the appropriate Teacher feature maps by a slight rearrangement of the Neck net. Since no individual matching is expected here, we solve the problem by reshaping all feature maps into a common format. This gives us, on the one hand, the possibility to perform tensor operations respectively between selected Teacher or Student feature maps to determine the EDM. On the other hand, we can then get EDMs of the same size for Student and Teacher models, whose deviations from each other can then be determined.

Towards this end, we implement a new convolution-based transformation function *transrel*, which performs 1×1 convolution on input tensors to compute the expected format. For an image resolution of 320×320, for example, as seen earlier, the selected Backbone feature maps are shaped as $4 \times 256 \times 20 \times 20$ & $4 \times 512 \times 10 \times 10$ for *YOLOteacher* and $4 \times 1280 \times 10 \times 10$ & $4 \times 96 \times 20 \times 20$ for *YOLOstudent*. These are all converted to the following common shape format $4 \times 256 \times 20 \times 20$ by *trainsrel*, a new transformation function developed for RKD. The general conversion process is then performed by converting the identically formatted feature maps into sets of $2D$ vectors. The task is performed by the *trans* function known from the other approaches. Let us name (F_4^s,F_5^s) and (F_4^t,F_5^t) the respective resulting vector sets for the Student and Teacher. These sets have an equal number v of vectors after conversion, which depends on the mini-batch and image size. The number of $2D$ vectors available for a p input image in a mini-batch of size l is calculated as follows: $v = \frac{p \times l}{16}$. For our case, because $p = 320$ and $l = 4$, each vector set has 80 $2D$-vectors.

$\phi_D(S)$, the relational function studied in Section 3.2.3, is built here for *YOLOstudent* by the EDM between F_4^s and F_5^s. Then $\forall x_m \in F_4^s$ and $\forall x_n \in F_5^s$, $d^2_{(m,n)} = \|x_m - x_n\|_2^2$ holds.

This yields a Euclidean Distance Matrix $\phi_D(S) = (a_{(m,n)})_{1m,n\leq 80} = (d^2_{(m,n)})_{1m,n\leq 80}$. The same operation with (F_4^t and F_5^t) yields the EDM $\phi_D(T)$ for the Teacher model. Overall, we express ϕ_D as follows:

$$\phi_D = (a_{(i,k)})_{1\leq i,k\leq v} = (d^2_{(i,k)})_{1\leq i,k\leq v} = \begin{pmatrix} 0 & d^2_{(1,2)} & d^2_{(1,3)} & \dots & d^2_{(1,v)} \\ d^2_{(2,1)} & 0 & d^2_{(2,3)} & \dots & d^2_{(2,v)} \\ d^2_{(3,1)} & d^2_{(3,2)} & 0 & \dots & d^2_{(3,v} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d^2_{(v,1)} & d^2_{(v,2)} & d^2_{(v,3)} & \dots & 0 \end{pmatrix} \tag{5.4}$$

where, as mentioned earlier, $v = 80$, $d^2_{(i,k)} = \|x_i - x_k\|_2^2$ is the Euclidean distance between vectors x_i and x_k. $\phi_D = \phi_D(S)$ if $(x_i, x_k) \in F_4^s \times F_5^s$. $\phi_D = \phi_D(T)$ if $(x_i, x_k) \in F_4^t \times F_5^t$.

In addition, we also implement the angle-wise approach to determine the relational function. In doing so, we use the cosine-similarity matrix, which represents the similarity measure between two vectors of a scalar product by determining the angle-based relationship between them [NB11]. The calculation is done according to the following formula of Nguyen and Bai [NB11]:

$$cossim(x_i, x_k) := \cos(\theta_{(i,k)}) = \frac{\mathbf{x_i} \cdot \mathbf{x_k}}{\|\mathbf{x_i}\|\|\mathbf{x_k}\|} = \frac{\sum_{u=1}^{d} x_i^u x_k^u}{\sqrt{\sum_{u=1}^{d} (x_i^u)^2}\sqrt{\sum_{u=1}^{d} (x_i^u)^2}} \tag{5.5}$$

where, as mentioned earlier, $v = 80$, d is the dimension (dim) of the vector x_i. It is valid that $dim(x_i) = dim(x_k) = d$

$$\phi_A = (a_{(i,k)})_{1\leq i,k\leq v} = \begin{pmatrix} \cos(\theta_{(1,1)}) & \cos(\theta_{(1,2)}) & \cos(\theta_{(1,3)}) & \dots & \cos(\theta_{(1,v)}) \\ \cos(\theta_{(2,1)}) & \cos(\theta_{(2,2)}) & \cos(\theta_{(2,3)}) & \dots & \cos(\theta_{(2,v)}) \\ \cos(\theta_{(3,1)}) & \cos(\theta_{(3,2)}) & \cos(\theta_{(3,3)}) & \dots & \cos(\theta_{(3,v)}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \cos(\theta_{(v,1)}) & \cos(\theta_{(v,2)}) & \cos(\theta_{(v,3)}) & \dots & \cos(\theta_{(v,v)}) \end{pmatrix} \tag{5.6}$$

As for EDM, $\phi_A = \phi_A(S)$ if $(x_i, x_k) \in F_4^s \times F_5^s$. $\phi_A = \phi_A(T)$ if $(x_i, x_k) \in F_4^t \times F_5^t$.

Algorithm 4 Training routine for RKD: As in FKD, the trained *YOLOteacher* model is integrated into the *YOLOstudent* training script, and the network architectures invoked in the training loop are extended to output intermediate feature maps. Relational divergence is considered when fitting the loss function, and a trained Student model is expected.

Input: *trained_YOLO_teacher*, *untrained_YOLO_student*, *YOLOloss*, *logitloss*, *trans*, *eval*, *DistillerRKD*, *relational_loss*, *transrel*
Output: Trained Student model, Evaluation Metrics `/* Expected results */`
Function `trainrkd(·)`:

```
teachermodel ⟵ trained_YOLO_teacher
studentmodel ⟵ untrained_YOLO_student
for e ∈ ℰ do
    ℒ_distillation ⟵ 0
    ℒ_rel ⟵ 0
    for x ∈ 𝒳 do
        teacherpredictions ⟵ teachermodel(x)
        studentpredictions ⟵ studentmodel(x)
        ℒ_student, ··· ⟵ YOLOloss(studentpredictions, y_x)      /* y_x: Ground truth */
        for i ≤ #studentpredictons − 4, i ∈ ℕ do
            F(i) ⟵ trans(studentpredictions[i])
            G(i) ⟵ trans(teacherpredictions[i])
            ℒ_distillation ⟵ ℒ_distillation + logitloss(F(i), G(i))
        end
        fm_t ⟵ trans(transrel(teachertpredictions[3 : 5]))      /* fm_t = (F_4^t, F_5^t): 2D
        Teacher feature maps */
        fm_s ⟵ trans(transrel(studentpredictions[3 : 5]))       /* fm_s = (F_4^s, F_5^s): 2D
        Student feature maps */
        ℒ_rel ⟵ ℒ_rel + relational_loss(fm_s, fm_t)             /* φ_D or φ_A or (φ_D+φ_A)/2 */
        ℒ_RKD ← λ1 * ℒ_distillation + λ2 * ℒ_student + λ3 * ℒ_rel
        ℒ_RKD.Backpropagation
        save metrics = eval(trained_YOLO_student_e)
        save trained_YOLO_student_e
    end
end
End Function
```

Once the relational functions have been established, the relational loss is computed in the training loop. For this, the L2 norm is employed again. It determines the divergence between ϕ_S and ϕ_T, the respective relations in Student and Teacher models. We simultane-

ously adopt the angle- and distance-wise approaches to determine the general relational loss $\mathcal{L}_{rel}$. It is computed as the average of the losses $\mathcal{L}_D$ and $\mathcal{L}_A$ concerning distance and angle. Thus, one has:

$$\begin{aligned}\mathcal{L}_{rel}(z_s, z_t) &= \sum_{i=4}^{5} \mathcal{L}_{rel}(H_i = z_s^i, G_i = z_t^i) \\ &= \frac{1}{2} * \Big(\mathcal{L}_D(\phi_D(trans(H_4), trans(H_5)), \phi_D(trans(G_4), trans(G_5))) + \\ &\quad \mathcal{L}_A(\phi_A(trans(H_4), trans(H_5)), \phi_A(trans(G_4), trans(G_5)))\Big) \\ &= \frac{1}{2} * \Big(\mathcal{L}_2(\phi_D(F_4^s, F_5^s), \phi_D(F_4^t, F_5^t)) + \mathcal{L}_2(\phi_A(F_4^s, F_5^s), \phi_A(F_4^t, F_5^t))\Big)\end{aligned} \tag{5.7}$$

Accordingly, the overall loss over a training epoch changes as follows:

$$\boxed{\mathcal{L}_{RKD}(z_s, z_t) = \lambda_1 \mathcal{L}_{student}(zs, z_t) + \lambda_2 \mathcal{L}_{distillation}(zs, z_t) + \lambda_3 \mathcal{L}_{rel}(zs, z_t)} \tag{5.8}$$

where the logit loss remains conserved. As for the previous cases, the hyper-parameters λ_1, λ_2, and λ_3 have to be adequately get fixed for a reasonable adjustment of the loss function.

By extending the loss formula just listed and adding the hint loss applied only to the last intermediate output, as drawn in Figure 5.2, we have created the extended RKD, which we have named MKD. The new formula appears as follows:

$$\boxed{\begin{aligned}\mathcal{L}_{MKD}(z_s, z_t) &= \lambda_1 \mathcal{L}_{student}(z_s, z_t) + \lambda_2 \mathcal{L}_{distillation}(z_s, z_t) + \lambda_3 \mathcal{L}_{rel}(z_s, z_t) + \lambda_4 \mathcal{L}_{hint}(z_s^5, z_t^5) \\ &= \lambda_1 \mathcal{L}_{student}(z_s, z_t) + \lambda_2 \mathcal{L}_{distillation}(z_s, z_t) + \lambda_3 \mathcal{L}_{rel}(z_s, z_t) + \lambda_4 \mathcal{L} \in_{hint}(F_5^s, F_5^t)\end{aligned}} \tag{5.9}$$

where λ_4 denotes a hyperparameter set before the training starts. Also, in these cases, the final steps of the training routine are backpropagating the loss, evaluating the trained model, and saving the model and its evaluation metrics.

5.4 Summary

We began our investigation of Knowledge Distillation for the Object Detection Task in computer vision by taking a baseline PyTorch implementation of YOLO-v4. This preference stems from the fact that We chose to use Python and PyTorch-based development environ-

ments for our implementations. We improved the baseline implementation by extending it with a new Backbone and making the system operate using two different backbones. We enabled data retrieval by converting the data annotation from the COCO dataset from JSON to YOLO format, allowing us to train the models on this dataset. Based on the established baseline models, we then implemented the different KD variants we addressed in the Section 3.2. The main focus of the distillation was on the Backbone and YOLO Head, with several adaptive modifications made to the Neck. We accomplished the main work in the training routine for each KD approach. The general training loss was adapted on the basis of the newly formulated losses. Altogether, six models can be identified from the implementation stage: baseline and extended baseline models, as well as RsKD, FKD, RKD, and MKD models. All these models were completed according to well-known and confirmed mathematical and ML approaches and are subject to different hyperparameter sets. However, setting these parameters for adequate training is one of the major obstacles that can be confronted in model evaluation. An inadequate parameterization could cause the problem of the overfitting we presented in Section 2.2.3, or counter effects of our expectations. A general approach in ML against this constraint is to experiment, a task which we dedicate to the next chapter.

6

EXPERIMENT AND EVALUATION

To evaluate the implemented compression mechanisms RsKD, FKD and RKD, we simulate the training routines introduced in algorithm 1, algorithm 2, algorithm 3 and algorithm 4. The goal is to see if our compressed-form approaches can improve performance and accuracy compared to our baseline models. Therefore, we first show parameter settings that we adopt for training, focusing on the parameter definition and the motive of the usages made. We neglect the default parameterization with respect to the naive YOLO-v4 architecture, which is summarized in the general open source $yolov4.cfg$ discussed earlier. We then present the evaluation metrics we use for performance estimation. In addition, we move on to the second section of this chapter to discuss the results of our evaluation. Here, we focus on comparing the described metrics by summarizing the training outputs of Teacher and student models in comparison tables and graphics. We round this task by analyzing and interpreting the presented experiment results in Section 6.3.

6.1 Experimental Setup and Evaluation metrics

6.1.1 *Experimental Setup*

The first step for our experiment is to set the required parameters. General hyperparameters are defined and slightly adjusted to perform training and testing depending on the model being trained. As in the original *C*-based implementation, the set of hyperparameters for the classification task is up to $8M$ (Darknet) and $3.4M$ (Mobilenet-v2). The models are trained with batch and mini-batch 64 and 4, respectively, which indicates a subdivision fixed by 16. For the sake of time and limited resources, the number of training epochs is set to 205 so that at least five of the six models implemented could be trained. The overall learning rate is initialized to 0.001 and set up by a factor $\in \{0.01, 0.1, 1\}$ depending on the difference between the current train epoch count and a burn_in parameter of value 1000. The Momentum and Decay are respectively from 0.949 and 0.0005. The full parameterization is also valid for BoF and BoS components. BoF parameters such as CutMix and Blur are not activated because no data augmentation is desired. CutMix and MixUp

are activated, where the MixUp value is set to 3. This value changes depending on whether CutMix or Mosaic is set. Used activation functions are, among others, Mix, Leaky-Relu, and Sigmoid. Adam is used as the main optimizer. Among IoU, DIoU, CIoU, and Generalized Intersection over Union (GIoU) describing suitable regression methods for bounding boxes, the IoU is selected. Label smoothing is used as a regularization method. The experiments are elaborated on GPU only, namely NVIDIA GeForce RTX 3080 and Colab GPU, both from Cuda version 11.8. Applying techniques like Synchronized Batch Normalization (syncBN) optimizes the computation by allowing the simultaneous use of multiple GPUs.

Other hyperparameters are the weight parameters for the distilled losses. These are variable α, λ_1, λ_2, λ_3, λ_4, and $\mathcal{T}$, which are used to calculate the losses in Equation 5.1, Equation 5.2, Equation 5.8, and Equation 5.9. In addition, we assume a fixed maximum batch size for the dataset of 500.500, with increments of $[400.000, 450.000]$. We also set the image resolution to 320×320 or 608×608, depending on the model used, and assume a total number of training epochs of 205 for each model. This results in the following configuration:

Table 6.1: Overview of the parameter setting for training the models

Detector \ Parameter	$\mathcal{T}$	α	λ_1	λ_2	λ_3	λ_4	lr	Batch	Res.	Epoch
YOLO-v4	N/A	N/A	N/A	N/A	N/A	N/A	0.001	64	608^2	205
Ext. YOLO-v4	N/A	N/A	N/A	N/A	N/A	N/A	0.001	64	320^2	205
RsKD	10	0.1	N/A	N/A	N/A	N/A	0.001	64	320^2	205
FKD	10	N/A	0.05	0.3	0.07	N/A	0.001	64	320^2	205
RKD	10	N/A	0.05	0.05	0.3	N/A	0.001	64	320^2	205
MKD	10	N/A	0.05	0.05	0.2	0.1	0.001	64	320^2	205

where lr: Learning Rate, Res.: Image resolution, Batch: Batch size, Epoch: Epochs number

6.1.2 *Evaluation Metrics*

Some standard metrics for object detection are required to empirically evaluate our models. These are OD performance measurement units adopted by various popular competitions. We focus on their definition and how they should be combined and interpreted. Common competitions like PASCAL VOC and COCO Object Detection Challenges use both, mainly the mean average precision AP as evaluation metrics, but the implementations

differ slightly [PNS20]. COCO Object Detection Challenge, in particular, makes further use of average recall AR as a metric for detection. We, therefore, base our evaluation on the COCO metrics AP and AR.

To describe these metrics, some basic concepts have to be explained. (1) The confidence score [PNS20], which is usually determined in the classifier network, describes the probability that an anchor box contains an object. If the score is high enough, anchors (N) become bounding boxes (B), i.e., selected for the next processing step. The IoU estimates (2) the intersection of the bounding boxes and ground truth boxes (B_g) over their union [PNS20]. The definition of a confidence score threshold allows sorting out the detected anchor boxes, outputting a filtered list of bounding boxes, accordingly to Bodla et al. [Bod+17], by applying (3) NMS. The selection of B is tuned as follows[PNS20]: From N, the anchor B_{max} with the maximum confidence score is extracted and added to the empty list B. Iteratively over the remaining elements, the IoU with B_{max} is determined for each element following the formula [PNS20]:

$$IoU(x \in B, B_{max}) = \frac{erea(x \cap B_{max})}{erea(x \cup B_{max})} \tag{6.1}$$

If this exceeds the specified threshold, the affected element is removed from N and added to B, becoming the new reference element in the calculation of IoU for the next element, which is taken from B [PNS20]. In this way, one gets the list B of sorted-out bounding boxes, which are now evaluated with respect to B_g. We then have a False Negative evaluation (FN) if $\forall x \in B$, which should detect a certain ground truth, x confidence score falls below the threshold [PNS20]. If its confidence exceeds the threshold, we refer to it as a True Positive evaluation (TP). If x fails to detect anything but surpasses the threshold, it is referred to as a True Negative evaluation (TN) [PNS20]. However, this is not relevant for object detection tasks as objects, which are not labeled, shouldn't be detected within an image [PNS20]. An evaluation is considered False Positive (FP) when the model's precision falls below the threshold and fails to detect a ground truth bounding box that should have been detected [PNS20]. According to Padilla, Netto, and Silva [PNS20], precision and recall are defined by the following ratios:

$$precision = \frac{\#TP}{\#TP + \#FP} = \frac{\#TP}{all\ detections},\ recall = \frac{\#TP}{\#TP + \#FN} = \frac{\#TP}{all\ ground\ truths} \tag{6.2}$$

Subsequently, precision reflects the capability of the object detection model to make accurate predictions within a set of predictions, whereas recall indicates the ability of the model to correctly identify all objects that should be detected within a given image (ground truth bounding boxes) [PNS20]. The ideal detector should possess both high recall and high precision.

For different threshold values, there are different pairs (precision, recall). Thus one can construct the curve of the function $precision = f(recall)$, which represents the trade-off between precisions and recalls of different values. When this curve is interpolated, the area under the interpolated curve defines AP. In an interpolation in 11 points, this interpolation describes the mean sum of all maximal precisions (p_{imax}) in each recall $r \in \{0.1, 0.1, 0.3, ..., 1\}$ and thus man can write [PNS20]:

$$AP = \frac{1}{11} \sum_{r \in \{0.1, 0.2, 0.3, ..., 1\}} p_{imax}(r), \qquad mAP = \frac{1}{N} \sum_{c=1}^{N} AP_c \tag{6.3}$$

The left formula is valid only for evaluating single-class applications [PNS20]. The mean average precision formulated right in Equation 6.3 is defined for object detectors, which usually have several classes denoted by N in the formula [PNS20].

Another metric for OD is the AR or, more specifically, the mean Average Recall (mAR) [Hos+16]. It is the average of recall scores for detections with IoU values between 0.5 and 1 [Hos+16]. It is computed by taking the Area Under the Curve (AUC) of the recall function ($recall = f(\text{IoU})$). The following calculation formulas are provided, where, similarly to AP, the formula on the left is used for single-class detectors, as per COCO requirements, while the mAR formula on the right is used for multi-class detectors with N classes [Hos+16]:

$$AR = 2 \int_{0.5}^{1} recall(u)du, \qquad mAR = \frac{1}{N} \sum_{c=1}^{N} AR_c \tag{6.4}$$

General mAP variants for COCO challenges are distinguished in terms of IoU values and sizes of detected objects. They are mostly referenced as follows [PNS20]: (1) $AP_{.05:.005:.95} = AP$, which represents mAP for 10 IoU between 0.5 and 0.95. (2) AP_{50} and (3) AP_{75} describe mAP for $IoU = 0.5$ or 0.75. If the area covered by the object to detect is smaller than 32×32, then the metric is called (4) AP_{small}. If the area is larger than 32×32 but smaller than

96×96, then the evaluation precision metric is called (5) AP_{medium}. It is then called (6) AP_{large} if the detected area is larger than 96×96.

This constellation with AP also exists for AR values. Notations differ by the number of detections per image or, like AP, by the area of the detected object inside an image [Hos+16]. They are (1) AR_1, (2) AR_{10}, and (3) AR_{100} if the maximum of 1, 10, and 100 objects are detected per image [Hos+16]. Regarding area and according to the description for AP notations, (4) AR_{small}, (5) AR_{medium} and (6) AR_{large} are the conventional notations [Hos+16].

We use the quasi-totality of these AP and AR metrics in the evaluation of our models. We furthermore examine the model statistics by additional metrics such as the inference speed and the number of parameters. The training time, as well as the size of all models, are also considered for the evaluation of the training.

6.2 Experiment Results and Comparison of Detectors

Under the precedent-setting, a series of experiments have been carried out, the results of which are summarized in this section. These experiments were performed through various training processes. First, we trained both the baseline and extended baseline models and then focused on the two prioritized distillation techniques in the thesis, RsKD and RKD. This outlines the first experimental phase, which is succeeded by a second phase where we perform an ablation study of all the implemented approaches. Our goal is to identify the impact of the set of transferred knowledge, specifically the number and type of losses considered, on the learning process. We evaluate the performance of each process in terms of accuracy and training time.

6.2.1 *Experiment Evaluation*

In the fundamental experiment, the results of the RKD and the RsKD are compared based on those of the baseline models. We operate in three steps to report the performance of the models: (1) We resume the statistics of models, (2) measure their precision, and (3) estimate the corresponding recall scores.

6.2.1.1 *Models statistics*

Initially, the Teacher model is trained using input images with a resolution of 608×608. One complete training epoch takes roughly 90 minutes on a GPU to produce a trained and evaluated model. This model has over 64 million trainable parameters and a fixed size of 737 MB. On the other hand, the extended baseline model realizes a shorter training time and employs input images with a resolution of 320×320, as using higher resolutions for training a compact backbone like Mobilenet-v2 would require more computing resources than currently available. It takes about half an hour to complete one training epoch using the same processing machine as the Teacher model. It also results in a smaller model size of 372 MB with approximately 50 million trainable parameters. All aforementioned results are summarized in Table 6.2, where the experimental results of distilled models can also be observed.

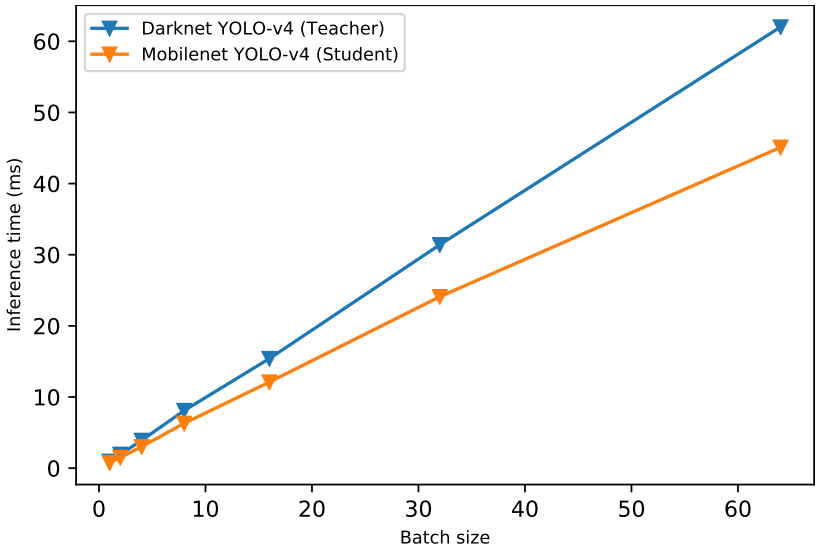


Figure 6.1: Inference time of Teacher and Student Models: Depending on the batch size, the durations required by the teacher and student models to perform detection may differ. The YOLO-V4 student model exhibits a better inference time than the YOLO-v4 teacher model.

From a statistical point of view, the distilled models have some similar characteristics to the extended base model (original model), mainly in terms of the number of parameters that can be trained and the model sizes. However, there are some differences with respect to the training time and inference. Distilled models require a longer training time than the original model. Here, the degree of distillation depends on the number of feature maps

Table 6.2: Evaluation of model statistics for distilled and baseline models. Distilled Models get compressed. However, the training processes using distilling are longer than normal training.

Metrics	Distilled YOLO-v4				YOLO-v4	
	RsKD	FKD	RKD	MKD	Mobilenet-v2^{Torch}	Darknet-53^{Torch}
Model size (MB)	372	372	372	372	372	737
FPS	77.3	77.3	77.3	77.3	77.3	62.5
Training (Min/Epoch)	55	59	69	75	35	95
Trainable Params	50M	50M	50M	50M	50M	64M

considered and the operations between them. Thus, in ascending distillation order, we write RsKD $<$ FKD $<$ RKD $<$ MKD. The more distillation we apply, the greater the training time. For instance, RsKD takes 55 minutes to complete a training epoch, while FKD, RKD, and MKD take 59, 69, and 75 minutes, respectively. Additionally, as depicted in Figure Figure 6.1, the student model demonstrates superior latency compared to the Teacher. This results in a faster inference time for both the Student and the compressed models. For instance, with a batch size of 1, the student requires approximately 0.76 milliseconds for detection, whereas the Teacher takes about 0.96 milliseconds. This difference in performance implies the FPS divergence between the Teacher model and student and distilled models, as can be seen in Table 6.2.

6.2.1.2 *Average Precision*

In the second experiment stage, the performance of the models is evaluated using the AP metrics. This evaluation occurs in the evaluation loop of each epoch, as previously discussed in the last chapter. After 205 training epochs, the precision metrics seen in Section 6.1.2 result for baseline, original, RsKD, and RKD models in illustrations in Figure 6.2. Up until the 10th epoch, all models have a similar mAP, close to zero. After that, the baseline model grows the slowest and reaches a performance of about 1% mAP for the first time only after about 80 epochs. After 205 epochs, its mAP is close to 6.3%. Conversely, the other models grow faster in terms of mAP. For instance, the extended baseline reaches, the extended baseline already reaches a mean AP of 1% after 35 epochs. After 205 epochs, it has an mAP of 6.8%, which is relatively high compared to the RsKD.

The illustration demonstrates that when using RsKD, the accuracy of the Student model

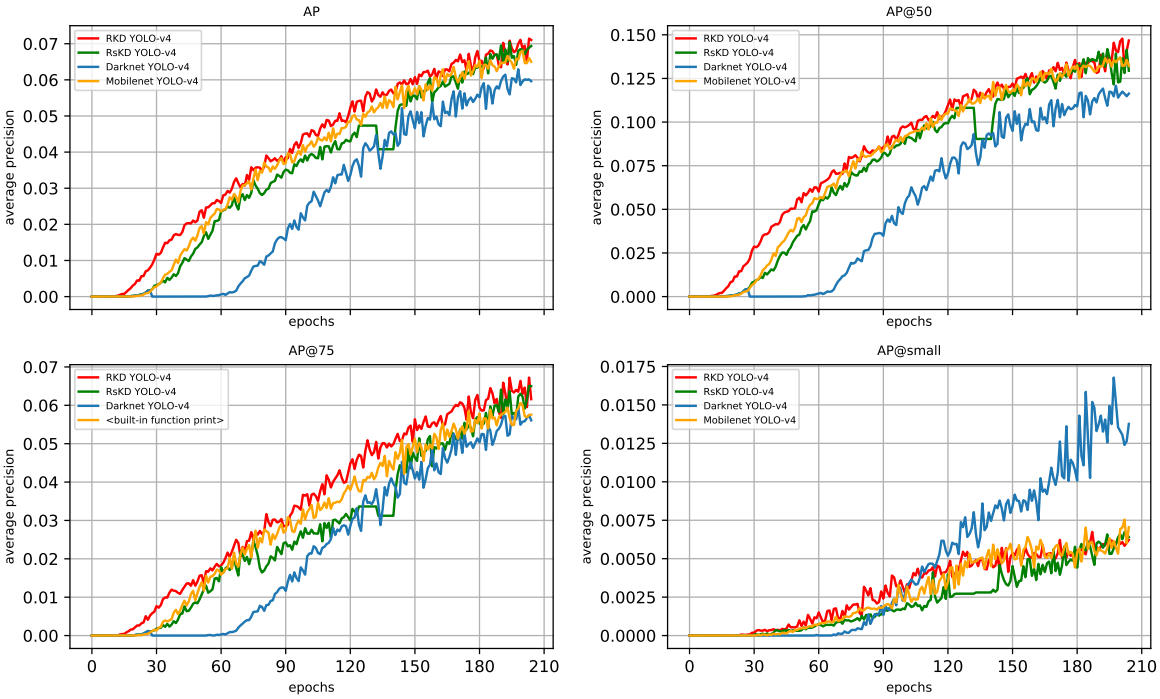


Figure 6.2: Precision comparison between RsKD-, RKD-YOLO-v4 and baseline models: The plots show that RKD improves the training process by reaching better performance in earlier training steps than other approaches. However, the RKD-model, in contrast to the baseline model, struggles to detect small and medium objects in images.

decreases under our settings. It takes around 40 epochs for the mAP to reach 1%, and its values continue to align with those of the original model, even if they appear to be superior to the baseline models. On the other hand, when using RKD, there is a significant improvement as the model learns faster than the other models. After only 30 training epochs, the mAP reaches 1%, and the progress continues quickly, reaching approximately 7.1% mAP after 205 epochs, as Table 6.3 .

Regarding precision metrics like AP_{50} and AP_{75}, RKD performs better than RsKD, the original model, and the baseline models. After 205 epochs, the AP_{50} for RKD reaches 13.9%, while the baseline models only reach 12.1%. For RsKD and the original model, it is 13.8% and 11.8%, respectively. If the IoU threshold is set to 75%, after 205 epochs, the average precision is 5.9% for the Teacher model, 6.1% for the baseline model, 6.2% for RsKD, and 6.6% for RKD. Regarding object size, the training processes give different outputs. Concerning AP_{small}, up to the 90th to the 95th epoch, the same behavior occurs as for IoU-based AP: the RKD performs better than the other models. After this phase, however, the AP of

Table 6.3: Performance evaluation for distilled models and baseline models 2: AP and AR are appreciated for both model categories. The values show that distillation improved the training process. The number of training epochs is fixed by 205.

Metrics	Distilled YOLO-v4		YOLO-v4	
	RsKD	RKD	Mobilenet-v2^{Torch}	Darknet-53^{Torch}
mAP	0.067	0.071	0.068	0.063
AP_{50}	0.136	0.139	0.138	0.121
AP_{75}	0.060	0.066	0.061	0.059
AP_{small}	0.006	0.006	0.007	0.015
AP_{medium}	0.053	0.055	0.052	0.071
AP_{large}	0.116	0.123	0.120	0.093
AR_1	0.122	0.127	0.122	0.135
AR_{10}	0.220	0.224	0.219	0.245
AR_{100}	0.273	0.281	0.274	0.320
AR_{small}	0.034	0.037	0.035	0.101
Epochs	205	205	205	205

the distilled models converges to the original model, which does not evolve very fast in detecting small objects. Hence, the maximum value of these models stagnates at 1% on average after the 205 epoch. On the contrary, the baseline model develops uninterruptedly with small objects. After 205 runs of the training routine, as shown in Table 6.3, this model already reaches 1.5% AP and thus performs significantly better for small objects than all other models. This behavior can also be observed for medium models. However, the convergence point for the original, Teacher and RsKD models is higher here and fixed at approximately 5.3%. The Teacher model still delivers better performance with 2% more AP_{medium} after the same number of epochs. Nevertheless, it drops back down to the lowest performance for wide objects, almost repeating the results of the IoU-based performances with higher accuracy values. The distilled model reaches an AP_{large} value of 12.3% after 205, whereas the original and the Teacher model perform 12% and 9.3%, respectively.

6.2.1.3 *Average Recall*

Regarding AR, baseline models, and distilled ones also show different evaluation results, as seen from the graphs in Figure 6.3. For a maximum detection number of 1, 10, or 100 objects per image, RKD shows better training results than all other models considered. Compared to the original model, RsKD does show lower performance, while the slope of the performance of the Teacher model becomes particularly fast when more objects have to be detected. For AR_{100}, it is especially noticeable that the Teacher starts to achieve good precision later than the RKD model, but after about 60 epochs, its precision ability increases much faster than that of all other models. After about 150 epochs, the RKD YOLO-V4 remains better than the original and RsKD models, but its ability to detect multiple objects in an image falls progressively below that of the Teacher, which becomes more effective and efficient. The dominance of the Teacher's abilities is also pronounced in the detection of small images. It achieves an AR score of 5%, while the most effective compressed model, the RKD, has only half of that (See Table 6.3).

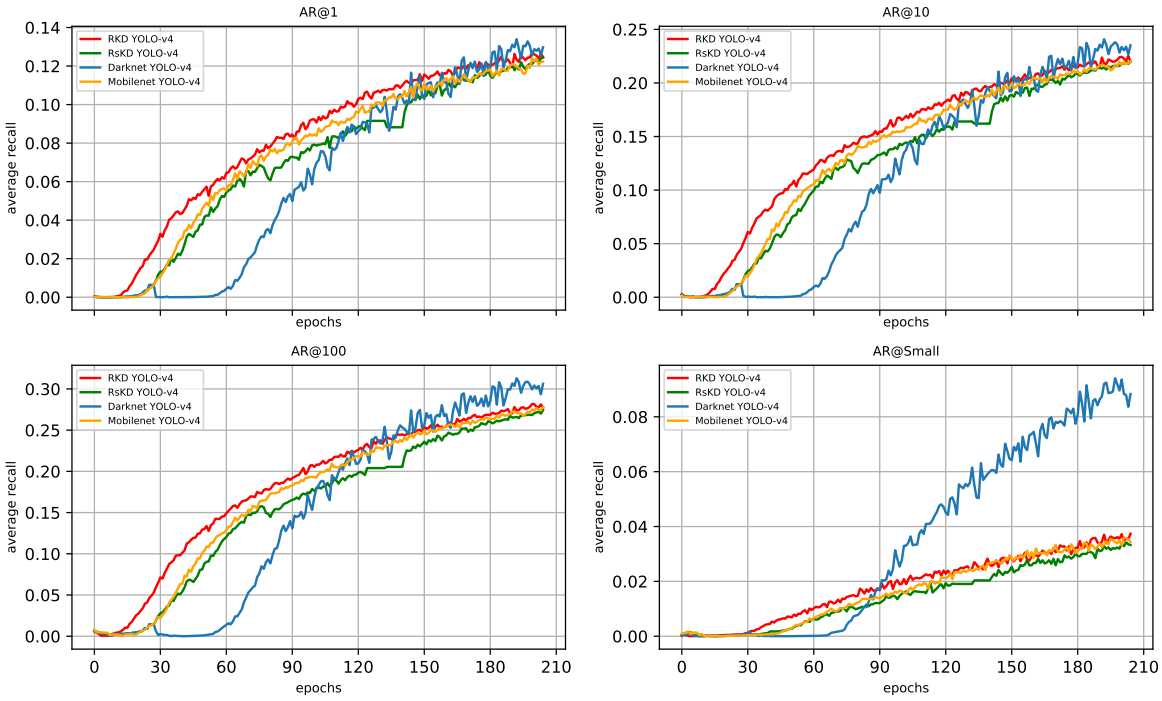


Figure 6.3: Average recalls comparison between distilled and baseline models

6.2.1.4 *Evaluation Summary*

In short, RKD YOLO-v4 demonstrates better object detection performance than the Teacher model, the original model, and the RsKD model across all IoU thresholds and for large object sizes. This superiority extends to the detection of small objects in an image up until the 95th epoch. However, it then shows a decline in precision compared to the Teacher model, although it still outperforms the original and RsKD models. These benefits are also reflected in the RKD's AR scores. Therefore, it can be concluded that, in general, RKD improves the accuracy of the original Teacher model for large object detection, but it may not have a positive impact on the performance of the Teacher model, especially after a long training period (100-150 epochs), particularly for small objects and large detection sets.

6.2.2 *Ablation Study about Training Losses*

In our extensive investigation of RKD, we also evaluated other KD approaches, FKD and MKD, which differ from the first two evaluated compressed models in that they consider different or multiple losses in the distillation process. We have fixed the number of training epochs at 50 for all models, except for the Teacher, which was trained for 205 epochs to provide a better supervisor for the compressed models.

The accuracy, which is still measured using the AP and AR metrics, is summarized in Table 6.4. Compared to RsKD and original models, FKD and MKD generally perform better but are still less accurate than RKD. In particular, FKD, which performs better than MKD, has a 1.5% mAP after 50 epochs, while RKD is able to predict with 2.4% precision. These loss influences can be seen in Figure 6.4, which briefly reveals the reasons for the inefficiency of RsKD and MKD compared to FKD and RKD. By the 50th training epoch, we can see that RsKD's loss increases instead of decreasing, while FKD and MKD tend to lower their losses. However, the rate of decrease in FKD's loss is faster than in MKD's loss, which yields the FKD model better accurate than the MKD model. So, in general, the correlation between the rate of change in losses and the improvement in accuracy values can be observed.

6.3 Analysis of Results

From the results of the experiment and evaluation, the following analytical conclusions can be drawn: the AP, AR, Inference, and Model size all demonstrate that all compression

Table 6.4: Performance evaluation for distilled models and baseline models 1: AP and AR are appreciated for both model categories. The values show that distillation improved the training process. The number of training epochs is fixed by 50.

Metrics	Distilled YOLO-v4				YOLO-v4	
	RsKD	FKD	RKD	MKD	Mobilenet-v2^{Torch}	Darknet-53^{Torch}
mAP	0.014	0.022	0.024	0.015	0.019	0.063
AP_{50}	0.034	0.052	0.056	0.035	0.045	0.016
AP_{75}	0.001	0.015	0.016	0.011	0.013	0.059
AP_{small}	0.000	0.001	0.001	0.000	0.001	0.015
AP_{medium}	0.003	0.009	0.010	0.005	0.006	0.071
AP_{large}	0.024	0.037	0.040	0.025	0.031	0.093
AR_1	0.004	0.056	0.056	0.041	0.048	0.135
AR_{10}	0.075	0.104	0.105	0.074	0.086	0.245
AR_{100}	0.088	0.127	0.130	0.089	0.104	0.320
AR_{small}	0.003	0.007	0.007	0.002	0.003	0.101
Epochs	50	50	50	50	50	205

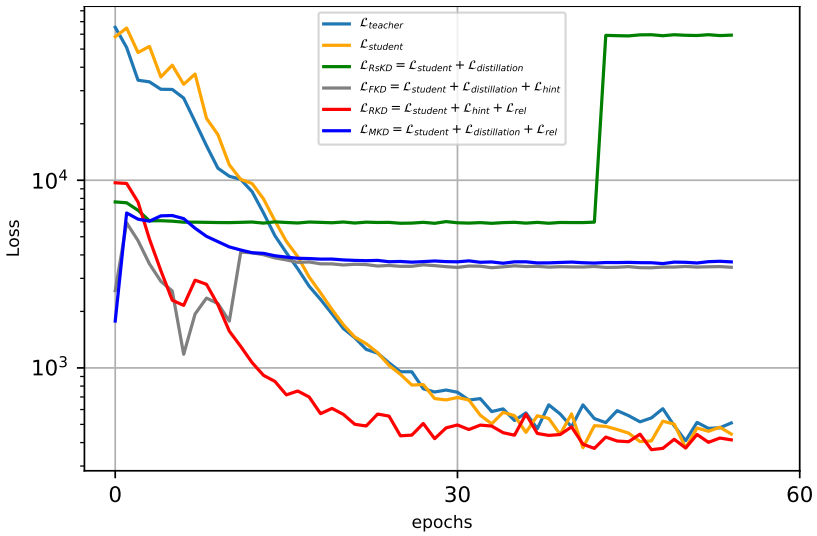


Figure 6.4: Ablation study on components of total loss function: expressed loss formulas represent the loss variation for the models used, neglecting the parameterization. All different distillation approaches are based on student loss. Adjusting the total loss by adding new specific losses ($\mathcal{L}_{distillation}$, $\mathcal{L}_{hint}$, $\mathcal{L}_{rel}$) has a significant impact on its evolution, thus on the accuracy of the model. $\mathcal{L}_{teacher}$ and $\mathcal{L}_{student}$ are resp. the teacher and student losses without distillation.

methods implemented appear to effectively compress the Teacher model. The YOLO-V4 student model confirms this with its reduced size and ability to learn from the Teacher, whose Inference has been improved thanks to the Student's better execution speed. This improvement in inference is also due to the fact that no impact is made on the student network during KD operations, and no parameters are suppressed or modified, unlike in other compression techniques such as pruning and Lecun's Optimal Brain Damage [LDS89]. All parameters are saved, and the focus is mainly on the training process. However, compression techniques have different effects on precision and recall ratios. RsKD, FKD, and MKD generally perform worse than RKD. On the one hand, this is because RKD, as previously discussed in the introduction, transfers more knowledge from the Teacher to the Student compared to FKD and RsKD. Unlike these latter techniques, which focus on individual-level outputs from teacher-student pairs of layers, RKD takes into consideration not only the individual relationship but also the relational properties established. This method allows the teacher and student models to transfer more accurate and meaningful knowledge.

On the other hand, proper parameterization plays a crucial role in the effectiveness

of the chosen distillation approach. The impact of this argument can be seen in Figure 6.4, where more knowledge is transferred using MKD (which takes into account multiple losses). Despite this, MKD performs worse than the original model, which received no additional knowledge. We succeeded in tuning the parameters for RKD after many attempts by using the values listed in the Table 6.1. This has contributed to the best results for this distillation type, besides the loss calculation. Although RKD detects more accurately and effectively than the original model and all other compressed models, it still has deficits in AR, AP_{small}, and AP_{medium} compared to the Teacher model, as seen in previous sections. This failure can be attributed to the fact that the original and compressed models were trained on an image resolution of 320×320 due to limited computational resources. In contrast, the Teacher was trained on a larger image resolution of 608×608, making it better suited for detecting smaller and medium-sized, and thus more objects in an image. The Teacher and Student models were fine-tuned by first training the Teacher model using the configuration from the GitHub repository that was used for the original implementation. The second training round for the Student failed for the first time since more computing resources were needed for training with the same image resolution as the Teacher. Only after reducing the resolution, the training could proceed. Given the long duration of a training epoch and the approaching project deadline, the Teacher model couldn't be retrained for the Student's specified resolution. If both the Teacher and Student models are trained at the same resolution, this issue would be resolved.

In conclusion, the evaluation results show that compressing YOLO-v4 models using knowledge distillation can give good results. This can also be seen in some examples of detection tasks with compressed models, as shown in Figure 6.5. Using this method, the YOLO-v4 model can be scaled down without greatly affecting accuracy or inference. The performance of the model depends on the parameterization and the KD approach, with appropriate parameterization and implementation, such as in our case with RKD, leading to improved accuracy, especially in terms of precision. Our model, therefore, makes it possible to save up to 30 of the necessary training epochs for the original and Teacher model by learning more and faster during compression. However, an epoch lasts longer than in the normal training process. The duration of the epoch depends on how much knowledge is required. As can be seen in Figure 6.5, our model also struggles to detect small and medium-sized objects, making it slightly less effective than the YOLO-v4 teacher model.

Figure 6.5: YOLO-v4 vs. RKD-YOLO-v4: The recognition results of the teacher are the four images at the top, where we see the good precision as well as a good ability to recognize multiple objects. The recognition results of the student model (four images in the lower part of the figure) also show good precision. However, as we can see in the image with the train, the student model has difficulty recognizing small objects like the cars that the teacher model recognized within the same image. This is also the case when looking at the unrecognized broccoli in the third image to the left of the giraffe image.

7

CONCLUSIONS

7.1 General summary

This thesis pursued the goal of finding out whether and how Knowledge Distillation, a state-of-the-art DNN compression technique implemented in principle for classification tasks, can be used in the field of computer vision for processing to achieve object detection tasks. We wanted to contribute to improving the internal hosting of DNN models in industrial applications by compressing the models using KD. In doing so, we chose to experiment with the KD compression effects on DNN model performance. Therefore, we wanted, to find a trade-off between inference, compression degree as well as accuracy. Starting from an investigation of the current state of research on DNN model compression and on computer vision, we decided to use a standard object detector. YOLO-v4 has been used, which then had to act as a teacher. We subsequently build the student YOLO-v4 by changing the network backbone of the detector. We replace the backbone from Darknet-53 to Mobilnet-v2, with appropriate adjustments in the Neck to allow the Head of the detector to make suitable predictions. Based on the implemented Student model, we then implemented and examined the different KD variants. The implementations corresponded to some well-known approaches that can be found in the literature. Among them, we can list the KL-divergence, the $\mathcal{L}_1$ and $\mathcal{L}_2$-Norm-Losses, as well as the Euclidean distance and angle matrices. Emerging distillers were named RsKD, FKD, RKD, and MKD, all of which were used on YOLO-v4. After several training epochs and evaluation of the models, it has been found that even though the training time may sometimes be higher than that of the Teacher model, the compressed models show the following relatively good results: (1) Trained YOLO-v4 models to weigh from about 50% less. (2) By inheriting the inference time of the Student detector, the compressed models improve the detection speed up. (3) Moreover, distilled models, and more precisely RKD, improve the accuracy by making it possible to detect more precisely (high AP) and faster. It is also important to mention that the use of RKD can have a positive influence on the training process, which could be gained up to 30 under 205 normal training epochs.

7.2 Contributions of the Thesis and Future Work

Considering the technical research status, this thesis has made use of a new application for calculating the relationship between feature maps. The simultaneous application of two techniques (Distance- and Angle-wise) to express the relationship between feature maps is a new approach that has not been found in any literature. The good results of the resulting RKD YOLO-v4 confirm that our contribution comprises a beneficial approach, especially for the domain of compression of DNN models for OD. This KD variant, which is the best among all the implemented and studied approaches, accordingly could provide a suitable solution to the problem of local hosting of DNN models in the industrial field. It fulfills the formulated requirement of finding a compromise between model reduction, speeds up, and accuracy conservation and/or improvement.

Even if the implemented approach shows good results, there is still room for improvement, especially when looking at parameterization. The models were not trained with many different parameters due to time constraints. It is completely possible that better results occur under other parameterizations. In addition, many other applications are used to calculate the relationship between feature maps. For example, one could use a knowledge graph [Par+19] to simultaneously relate more than two feature maps (as in our approach) and determine the edge and vertex divergence. The Gramian matrix [SA94] is another approach that could be used for RKD as a relational potential. Other subjects of improvement are also the parameterizations for the insets RsKD, MKD, and FKD. It is arguable that under an appropriate parameterization, these insets could represent suitable alternatives to RKD. Especially MKD could most likely provide good results through improvement since it includes more knowledge than all other implemented models. Here longer training time could also bring a gain of knowledge.

8 APPENDIX

Listing 8.1: General structure of JSON COCO-annotation

```
{
    "info: "info,
    "licenses: " [
        license],
    "categories: " [
        category],
    "images: "[image],
    "annotations: " [
        annotation]
}
```

Listing 8.2: Example of JSON COCO-annotation

```
{
    "info": { "year": "2021", "version": "1.0", "
        description": "Exported from FiftyOne", "
        contributor": "Voxel51", "url": "https://fiftyone
        .ai", "date_created": "2021-01-19T09:48:27"
    },
    "licenses": [{
          "url": "./licenses/by-nc-sa/2.0/", "id": 1, "
              name": "Attribution License"
        },...
    ],
    "categories": [{
            "id": 2, "name": "cat", "supercategory": "
                animal"
        },...
    ],
    "images": [{
            "id": 0, "license": 1, "file_name": "<
                filename0>.<ext>",
            "height": 480, "width": 640, "date_captured":
                 null
        },...
    ],
    "annotations": [{
            "id": 0, "image_id": 0, "category_id": 2, "
                bbox": [260, 177, 231, 199], "
                segmentation": [...], "area": 45969,
            "iscrowd": 0
        },...
    ]
}
```

BIBLIOGRAPHY

[Abb+07] Maysam F. Abbod, James W. F. Catto, Derek A. Linkens, and Freddie C. Hamdy. "Application of artificial intelligence to the management of urological cancer." eng. In: *The Journal of Urology* 178.4 Pt 1 (Oct. 2007), pp. 1150–1156. ISSN: 0022-5347. DOI: 10.1016/j.juro.2007.05.122.

[AM90] Yaser S Abu-Mostafa. "Learning from hints in neural networks." en. In: *Journal of Complexity* 6.2 (June 1990), pp. 192–198. ISSN: 0885-064X. DOI: 10.1016/0885-064X(90)90006-Y. URL: https://www.sciencedirect.com/science/article/pii/0885064X9090006Y (visited on 12/18/2022).

[Alo+18] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. "The history began from alexnet: A comprehensive survey on deep learning approaches." In: *arXiv preprint arXiv:1803.01164* (2018).

[BP22] Faraz Bagwan and Nitin Pise. "Efficient Diagnosis of Covid19 by Employing Deep Transfer Learning on Pretrained VGG and ResidualNet Architectures." In: *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*. Vol. 1. 2022, pp. 667–672. DOI: 10.1109/ICACCS54159.2022.9785320.

[Bir+20] Philip Birch, Navid Rahimi, Peter Overburry, Rupert Young, and Chris Chatwin. "Implementations and optimisations of optical Conv2D networks designs." In: *Semiconductor Lasers and Laser Dynamics IX*. Vol. 11356. SPIE, Apr. 2020, pp. 178–185. DOI: 10.1117/12.2554021. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11356/113561C/Implementations-and-optimisations-of-optical-Conv2D-networks-designs/10.1117/12.2554021.full (visited on 12/20/2022).

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934 [cs, eess]. Apr. 2020. DOI: 10.48550/arXiv.2004.10934. URL: http://arxiv.org/abs/2004.10934 (visited on 12/11/2022).

[Bod+17] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. "Soft-NMS – Improving Object Detection With One Line of Code." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[Cap20] Jarred Capellman. *Hands-On Machine Learning with ML. NET: Getting started with Microsoft ML. NET to implement popular machine learning algorithms in C*. Packt Publishing Ltd, 2020.

[CNM06] Rich Caruana and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms." In: *Proceedings of the 23rd international conference on Machine learning*. ICML '06. New York, NY, USA: Association for Computing Machinery, June 2006, pp. 161–168. ISBN: 9781595933836. DOI: 10.1145/1143844.1143865. URL: https://doi.org/10.1145/1143844.1143865 (visited on 10/31/2022).

[CA16] M. Emre Celebi and Kemal Aydin, eds. *Unsupervised Learning Algorithms*. en. Cham: Springer International Publishing, 2016. ISBN: 9783319242095 9783319242118. DOI: 10.1007/978-3-319-24211-8. URL: http://link.springer.com/10.1007/978-3-319-24211-8 (visited on 11/01/2022).

[Che+21] Defang Chen, Jian-Ping Mei, Yuan Zhang, Can Wang, Zhe Wang, Yan Feng, and Chun Chen. "Cross-layer distillation with semantic calibration." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8. 2021, pp. 7028–7036.

[Che+17] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. "Learning efficient object detection models with knowledge distillation." In: *Advances in neural information processing systems* 30 (2017).

[Coc+20] Marco Cococcioni, Federico Rossi, Emanuele Ruffaldi, and Sergio Saponara. "A Fast Approximation of the Hyperbolic Tangent When Using Posit Numbers and Its Application to Deep Neural Networks." en. In: *Applications in Electronics Pervading Industry, Environment and Society*. Ed. by Sergio Saponara and Alessandro De Gloria. Lecture Notes in Electrical Engineering. Cham: Springer International Publishing, 2020, pp. 213–221. ISBN: 9783030372774. DOI: 10.1007/978-3-030-37277-4_25.

[CW08] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: deep neural networks with multitask learning." In: *Proceedings of the 25th international conference on Machine learning*. ICML '08. New York, NY, USA: Association for Computing Machinery, July 2008, pp. 160–167.

ISBN: 9781605582054. DOI: 10.1145/1390156.1390177. URL: https://doi.org/10.1145/1390156.1390177 (visited on 11/05/2022).

[Cop04] Ben Coppin. *Artificial Intelligence Illuminated*. en. Google-Books-ID: LcOLqodW28EC. Jones & Bartlett Learning, 2004. ISBN: 9780763732301.

[CCD08] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. "Supervised Learning." en. In: *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Ed. by Matthieu Cord and Pádraig Cunningham. Cognitive Technologies. Berlin, Heidelberg: Springer, 2008, pp. 21–49. ISBN: 9783540751717. DOI: 10.1007/978-3-540-75171-7_2. URL: https://doi.org/10.1007/978-3-540-75171-7_2 (visited on 11/01/2022).

[Dat23] COCO Dataset. *Common Objects in Context - Dataset*. en. 2023. URL: https://cocodataset.org (visited on 01/31/2023).

[DZ87] E. D. Dickmanns and A. Zapp. "Autonomous High Speed Road Vehicle Guidance by Computer Vision1." en. In: *IFAC Proceedings Volumes*. 10th Triennial IFAC Congress on Automatic Control - 1987 Volume IV, Munich, Germany, 27-31 July 20.5, Part 4 (July 1987), pp. 221–226. ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)55320-3. URL: https://www.sciencedirect.com/science/article/pii/S1474667017553203 (visited on 12/16/2022).

[EAH20] Hisham El-Amir and Mahmoud Hamdy. *Deep learning pipeline*. 2020.

[EH20] Jesper E. van Engelen and Holger H. Hoos. "A survey on semi-supervised learning." en. In: *Machine Learning* 109.2 (Feb. 2020), pp. 373–440. ISSN: 1573-0565. DOI: 10.1007/s10994-019-05855-6. URL: https://doi.org/10.1007/s10994-019-05855-6 (visited on 11/01/2022).

[Est+21] Andre Esteva, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean, and Richard Socher. "Deep learning-enabled medical computer vision." en. In: *npj Digital Medicine* 4.1 (Jan. 2021), pp. 1–9. ISSN: 2398-6352. DOI: 10.1038/s41746-020-00376-2. URL: https://www.nature.com/articles/s41746-020-00376-2 (visited on 12/16/2022).

[Gav+18] Andrei Dmitri Gavrilov, Alex Jordache, Maya Vasdani, and Jack Deng. "Preventing Model Overfitting and Underfitting in Convolutional Neural Networks." en. In: *International Journal of Software Science and Computational Intelligence (IJSSCI)* 10.4 (Oct. 2018), pp. 19–28. ISSN: 1942-9045. DOI: 10.4018/IJSSCI.2018100102. URL: https://www.igi-global.com/article/preventing-model-overfitting-and-underfitting-in-convolutional-

neural-networks/www.igi-global.com/article/preventing-model-overfitting-and-underfitting-in-convolutional-neural-networks/223492 (visited on 12/20/2022).

[Gho+19] Sanjukta Ghosh, Shashi K K Srinivasa, Peter Amon, Andreas Hutter, and André Kaup. "Deep Network Pruning for Object Detection." In: *2019 IEEE International Conference on Image Processing (ICIP)*. 2019, pp. 3915–3919. DOI: 10.1109/ICIP.2019.8803505.

[Gir15] Ross Girshick. "Fast r-cnn." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[GM20] Roberto E. González and Roberto P. Muñoz. "AstroCV - A computer Vision Library for Astronomy." In: 522 (Apr. 2020). ADS Bibcode: 2020ASPC..522..425G, p. 425. URL: https://ui.adsabs.harvard.edu/abs/2020ASPC..522..425G (visited on 12/16/2022).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. ISBN: 9780262035613.

[Gou+21] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. "Knowledge distillation: A survey." In: *International Journal of Computer Vision* 129.6 (2021), pp. 1789–1819.

[Guo+18] Yanming Guo, Yu Liu, Theodoros Georgiou, and Michael S. Lew. "A review of semantic segmentation using deep neural networks." en. In: *International Journal of Multimedia Information Retrieval* 7.2 (June 2018), pp. 87–93. ISSN: 2192-662X. DOI: 10.1007/s13735-017-0141-z. URL: https://doi.org/10.1007/s13735-017-0141-z (visited on 12/13/2022).

[Guo+16] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S. Lew. "Deep learning for visual understanding: A review." en. In: *Neurocomputing*. Recent Developments on Deep Big Vision 187 (Apr. 2016), pp. 27–48. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.09.116. URL: https://www.sciencedirect.com/science/article/pii/S0925231215017634 (visited on 11/05/2022).

[GA22] Manish Gupta and Puneet Agrawal. "Compression of Deep Learning Models for Text: A Survey." In: *ACM Transactions on Knowledge Discovery from Data* 16.4 (Jan. 2022), 61:1–61:55. ISSN: 1556-4681. DOI: 10.1145/3487045. URL: https://doi.org/10.1145/3487045 (visited on 12/18/2022).

[Noaa] *Hardware Accelerators for Machine Learning (CS 217) by cs217*. URL: https://cs217.stanford.edu/weightlayers (visited on 11/13/2022).

[He+20] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. "ReLU Deep Neural Networks and Linear Finite Elements." In: *Journal of Computational Mathematics* 38.3 (June 2020). arXiv:1807.03973 [math], pp. 502–527. ISSN: 0254-9409, 1991-7139. DOI: 10.4208/jcm.1901-m2018-0160. URL: http://arxiv.org/abs/1807.03973 (visited on 11/17/2022).

[He+18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. *Mask R-CNN*. arXiv:1703.06870 [cs]. Jan. 2018. DOI: 10.48550/arXiv.1703.06870. URL: http://arxiv.org/abs/1703.06870 (visited on 12/13/2022).

[Noab] *Hierarchical Convolutional Deep Learning in Computer Vision - ProQuest*. de. URL: https://www.proquest.com/openview/62c046242f67ce115a76b9224e66a69c/1?pq-origsite=gscholar&cbl=18750 (visited on 11/16/2022).

[HVD14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Dark knowledge." In: *Presented as the keynote in BayLearn* 2.2 (2014).

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531 [cs, stat]. Mar. 2015. DOI: 10.48550/arXiv.1503.02531. URL: http://arxiv.org/abs/1503.02531 (visited on 12/18/2022).

[Hos+16] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. "What Makes for Effective Detection Proposals?" In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.4 (2016), pp. 814–830. DOI: 10.1109/TPAMI.2015.2465908.

[Hun75] B. Hunt. *Artificial Intelligence*. Academic Press series in cognition and perception. Academic Press, 1975. ISBN: 9780123623409. URL: https://books.google.de/books?id=j76EAAAAIAAJ.

[ITP15] Alexandros Iosifidis, Anastasios Tefas, and Ioannis Pitas. "DropELM: Fast neural network regularization with Dropout and DropConnect." en. In: *Neurocomputing* 162 (Aug. 2015), pp. 57–66. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.04.006. URL: https://www.sciencedirect.com/science/article/pii/S0925231215004130 (visited on 12/20/2022).

[Joy11] James M Joyce. "Kullback-leibler divergence." In: *International encyclopedia of statistical science*. Springer, 2011, pp. 720–722.

[Kal12] John H. Kalivas. "Overview of two-norm (L2) and one-norm (L1) Tikhonov regularization variants for full wavelength or sparse spectral multivariate calibration models or maintenance: Sparse multivariate calibration & maintenance." en. In: *Journal of Chemometrics* 26.6 (June 2012), pp. 218–230. ISSN: 08869383. DOI: 10.1002/cem.2429. URL: https://onlinelibrary.wiley.com/doi/10.1002/cem.2429 (visited on 12/20/2022).

[LDS89] Yann LeCun, John Denker, and Sara Solla. "Optimal Brain Damage." In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[Lee+18] Jun Haeng Lee, Sangwon Ha, Saerom Choi, Won-Jo Lee, and Seungwon Lee. "Quantization for rapid deployment of deep neural networks." In: *arXiv preprint arXiv:1810.05488* (2018).

[Lel93] Subhash Lele. "Euclidean distance matrix analysis (EDMA): estimation of mean form and mean form difference." In: *Mathematical Geology* 25 (1993), pp. 573–602.

[LCY14] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. arXiv:1312.4400 [cs]. Mar. 2014. URL: http://arxiv.org/abs/1312.4400 (visited on 12/18/2022).

[Lin+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[Liu+17] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. *Large-Margin Softmax Loss for Convolutional Neural Networks*. arXiv:1612.02295 [cs, stat]. Nov. 2017. DOI: 10.48550/arXiv.1612.02295. URL: http://arxiv.org/abs/1612.02295 (visited on 12/20/2022).

[Liu+19] Yufan Liu, Jiajiong Cao, Bing Li, Chunfeng Yuan, Weiming Hu, Yangxi Li, and Yunqiang Duan. "Knowledge Distillation via Instance Relationship Graph." In: 2019, pp. 7096–7104. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Liu_Knowledge_Distillation_via_Instance_Relationship_Graph_CVPR_2019_paper.html (visited on 09/03/2022).

[Lu+20] Xin Lu, Quanquan Li, Buyu Li, and Junjie Yan. "MimicDet: Bridging the Gap Between One-Stage and Two-Stage Object Detection." en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 541–557. ISBN: 9783030585686. DOI: 10.1007/978-3-030-58568-6_32.

[ML17] Wei Ma and Jun Lu. *An Equivalence of Fully Connected Layer and Convolutional Layer*. arXiv:1712.01252 [cs, stat]. Dec. 2017. DOI: 10.48550/arXiv.1712.01252. URL: http://arxiv.org/abs/1712.01252 (visited on 12/20/2022).

[Mao+19] Qi-Chao Mao, Hong-Mei Sun, Yan-Bo Liu, and Rui-Sheng Jia. "Mini-YOLOv3: Real-Time Object Detector for Embedded Applications." In: *IEEE Access* 7 (2019), pp. 133529–133538. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2941547.

[MDRS19] Fazla Rabbi Mashrur, Amit Dutta Roy, and Dabasish Kumar Saha. "Automatic Identification of Arrhythmia from ECG Using AlexNet Convolutional Neural Network." In: *2019 4th International Conference on Electrical Information and Communication Technology (EICT)*. 2019, pp. 1–5. DOI: 10.1109/EICT48899.2019.9068806.

[MGD20] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. *A Survey on Deep Neural Network Compression: Challenges, Overview, and Solutions*. arXiv:2010.03954 [cs, eess]. Oct. 2020. DOI: 10.48550/arXiv.2010.03954. URL: http://arxiv.org/abs/2010.03954 (visited on 12/16/2022).

[NB11] Hieu V Nguyen and Li Bai. "Cosine similarity metric learning for face verification." In: *Computer Vision–ACCV 2010: 10th Asian Conference on Computer Vision, Queenstown, New Zealand, November 8-12, 2010, Revised Selected Papers, Part II 10*. Springer. 2011, pp. 709–720.

[NLP20] Thuy-Anh Nguyen, Hai-Bang Ly, and Binh Thai Pham. "Backpropagation neural network-based machine learning model for prediction of soil friction angle." In: *Mathematical Problems in Engineering* 2020 (2020), pp. 1–11.

[PNS20] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. "A Survey on Performance Metrics for Object-Detection Algorithms." In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[Par+19] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. "Relational Knowledge Distillation." In: *CVPR*. 2019. URL: https://cvlab.postech.ac.kr/research/RKD/.

[Pri12] S.J.D. Prince. *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012, pp. 1–5. URL: http://www.computervisionmodels.com/.

[Red+16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: 2016, pp. 779–788. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html (visited on 12/13/2022).

[RF] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement." In: *arXiv preprint arXiv:1804.02767* ().

[Rom+15] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. *FitNets: Hints for Thin Deep Nets*. arXiv:1412.6550 [cs]. Mar. 2015. DOI: 10.48550/arXiv.1412.6550. URL: http://arxiv.org/abs/1412.6550 (visited on 12/18/2022).

[SCY22] G. V. Sai Charan and R. Yuvaraj. "Efficient Tumor Classification using GoogleNet Approach to Increase Accuracy in Comparison with ResNet." In: *2022 3rd International Conference on Intelligent Engineering and Management (ICIEM)*. 2022, pp. 490–494. DOI: 10.1109/ICIEM54221.2022.9853203.

[San+19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. arXiv:1801.04381 [cs]. Mar. 2019. DOI: 10.48550/arXiv.1801.04381. URL: http://arxiv.org/abs/1801.04381 (visited on 11/26/2022).

[See+21] Vivek Seelam, Akhil kumar Penugonda, B. Pavan Kalyan, M. Bindu Priya, and M. Durga Prakash. "Smart attendance using deep learning and computer vision." en. In: *Materials Today: Proceedings*. International Conference on Materials, Manufacturing and Mechanical Engineering for Sustainable Developments-2020 (ICMSD 2020) 46 (Jan. 2021), pp. 4091–4094. ISSN: 2214-7853. DOI: 10.1016/j.matpr.2021.02.625. URL: https://www.sciencedirect.com/science/article/pii/S2214785321017764 (visited on 12/16/2022).

[SBS19] Sungho Shin, Yoonho Boo, and Wonyong Sung. "Empirical analysis of knowledge distillation technique for optimization of quantized deep neural networks." In: *arXiv preprint arXiv:1909.01688* (2019).

[SK19] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning." In: *Journal of Big Data* 6.1 (July 2019), p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0 (visited on 12/14/2022).

[Sho+21] Roni Shouval, Joshua A. Fein, Bipin Savani, Mohamad Mohty, and Arnon Nagler. "Machine learning and artificial intelligence in haematology." en. In: *British Journal of Haematology* 192.2 (Jan. 2021), pp. 239–250. ISSN: 0007-1048, 1365-2141. DOI: 10.1111/bjh.16915. URL: https://onlinelibrary.wiley.com/doi/10.1111/bjh.16915 (visited on 11/01/2022).

[SA94] Victor Sreeram and P Agathoklis. "On the properties of Gram matrix." In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 41.3 (1994), pp. 234–237.

[Sto20] Ruxandra Stoean. "Analysis on the potential of an EA–surrogate modelling tandem for deep learning parametrization: an example for cancer classification from medical images." en. In: *Neural Computing and Applications* 32.2 (Jan. 2020), pp. 313–322. ISSN: 1433-3058. DOI: 10.1007/s00521-018-3709-5. URL: https://doi.org/10.1007/s00521-018-3709-5 (visited on 12/20/2022).

[Sze22] Richard Szeliski. *Computer Vision: Algorithms and Applications*. en. Google-Books-ID: QptXEAAAQBAJ. Springer Nature, Jan. 2022. ISBN: 9783030343729.

[TPL20] Mingxing Tan, Ruoming Pang, and Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection." In: 2020, pp. 10781–10790. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Tan_EfficientDet_Scalable_and_Efficient_Object_Detection_CVPR_2020_paper.html (visited on 12/14/2022).

[TMK16] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. "BranchyNet: Fast inference via early exiting from deep neural networks." In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2016, pp. 2464–2469. DOI: 10.1109/ICPR.2016.7900006.

[VKB] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. "Deploying deep neural networks in the embedded space." In: *arXiv preprint arXiv:1806.08616* ().

[Vou+18] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. "Deep Learning for Computer Vision: A Brief Review." en. In: *Computational Intelligence and Neuroscience* 2018 (Feb. 2018), e7068349. ISSN: 1687-5265. DOI: 10.1155/2018/7068349. URL: https://www.hindawi.com/journals/cin/2018/7068349/ (visited on 12/16/2022).

[Wan+18] Hui Wang, Hanbin Zhao, Xi Li, and Xu Tan. "Progressive Blockwise Knowledge Distillation for Neural Network Acceleration." In: *IJCAI*. 2018, pp. 2769–2775.

[WL22] Travis Williams and Robert Li. "Wavelet Pooling for Convolutional Neural Networks." en. In: Feb. 2022. URL: https://openreview.net/forum?id=rkhlb8lCZ (visited on 12/20/2022).

[XK19] Xiufeng Xie and Kyu-Han Kim. "Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision." In: New York, NY, USA: Association for Computing Machinery, 2019. ISBN: 9781450361699. URL: https://doi.org/10.1145/3300061.3345448.

[Yan+22] Jing Yang, Brais Martinez, Adrian Bulat, and Georgios Tzimiropoulos. "Knowledge distillation via softmax regression representation learning." en. In: Feb. 2022. URL: https://openreview.net/forum?id=ZzwDy_wiWv (visited on 12/18/2022).

[YW20] Jun Yang and Fei Wang. "Auto-ensemble: An adaptive learning rate scheduling based deep learning model ensembling." In: *IEEE Access* 8 (2020), pp. 217499–217509.

[Yul+20] Tang Yulin, Shaohua Jin, Gang Bian, and Yonghou Zhang. "Shipwreck Target Recognition in Side-Scan Sonar Images by Improved YOLOv3 Model Based on Transfer Learning." In: *IEEE Access* 8 (2020), pp. 173450–173460. DOI: 10.1109/ACCESS.2020.3024813.

[ZF13] Matthew D. Zeiler and Rob Fergus. *Stochastic Pooling for Regularization of Deep Convolutional Neural Networks*. arXiv:1301.3557 [cs, stat]. Jan. 2013. DOI: 10.48550/arXiv.1301.3557. URL: http://arxiv.org/abs/1301.3557 (visited on 12/20/2022).

[ZM20] Linfeng Zhang and Kaisheng Ma. "Improve object detection with feature-based knowledge distillation: Towards accurate and efficient detectors." In: *International Conference on Learning Representations*. 2020.

[Zha+18] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. *Loss Functions for Neural Networks for Image Processing*. arXiv:1511.08861 [cs]. Apr. 2018. DOI: 10.48550/arXiv.1511.08861. URL: http://arxiv.org/abs/1511.08861 (visited on 12/20/2022).