UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Smart Mobility Data Science and Analytics

Wei Zheng

Impact of Input Dataset Size and Fine-tuning on Faster R-CNN with Transfer Learning

Master's Thesis (30 ECTS)

Supervisors: Tomas Björklund, PhD
Victor Henrique Cabral Pinheiro, PhD

Tartu 2023

Impact of Input Dataset Size and Fine-tuning on Faster R-CNN with Transfer Learning

Abstract:
Deep learning models are widely used for machine learning tasks such as object detection. The lack of available data to train these models is a common hindrance in many industrial applications, where data gathering/annotation and insufficient computational resources often impose a barrier to the financial feasibility of deep learning implementations. Transfer learning is a possible answer to this challenge by exploiting the information learned by a model from data in a different domain than that of the target dataset. This technique has been typically applied on the backbone network of a two-stage object detection pipeline. In this work, we investigate the association between the input dataset size and the proportion of trainable layers in the backbone. In particular, we show some interesting findings on Faster R-CNN ResNet-50 FPN, a state-of-the-art object detection model, and MS COCO, a benchmarking dataset. The outcomes of our experiments indicate that, although a model generally performs better when trained with more layers fine-tuned to the training data, such an advantage reduces as the input dataset becomes smaller, as unfreezing too many layers can even lead to a severe overfitting problem. Choosing the right number of layers to freeze when applying transfer learning not only allows the model to reach its best possible performance but also saves computational resources and training time. Additionally, we explore the association between the effect of learning rate decay and the input dataset size, and also discuss the advantage of using pre-trained weights when compared to training a network from scratch.

Keywords:
deep learning, transfer learning, fine-tuning, resnet50, convolutional neural network, faster r-cnn

CERCS: P170 - Computer science, systems, control

Sisendandmete suuruse ja peenhäälestamise mõju Faster R-CNN-ile ülekandeõppega

Lühikokkuvõte:
Süvaõppemudeleid kasutatakse laialdaselt masinõppe ülesannete, näiteks objektide tuvastamise jaoks. Olemasolevate andmete puudumine nende mudelite treenimise on paljudes tööstusrakendustes ühine takistus, kus andmete kogumine/märgendamine ja ebapiisavad arvutusressursid takistavad sageli süvaõpperakenduste rahalist teostatavust. Ülekandeõpe on võimalik vastus sellele väljakutsele, kasutades mudeli abil saadud teavet sihtandmekogumist erineva valdkonna andmetest. Seda tehnikat on tavaliselt rakendatud kaheetapilise objekti tuvastamise töövoogu selgroovõrgustikus. Käesolevas töös uurime sisendand-

mestiku suuruse seost treenitavate kihtide osakaaluga selgroos. Eelkõige näitame mõned huvitavad leiud Faster R-CNN ResNet-50 FPN, tipptasemel objektide tuvastamise mudel ja MS COCO, võrdlusandmete kogum. Meie katsete tulemused näitavad, et kuigi mudel toimib üldiselt paremini, kui seda treenitakse rohkemate treeningandmetele peenhäälestatud kihtidega, väheneb selline eelis, kui sisendandmekogum muutub väiksemaks, kuna liiga paljude kihtide külmutamine võib põhjustada isegi tõsise ülesobitamise probleem. Õige arvu kihtide valimine külmutamiseks ülekandeõppe rakendamisel mitte ainult ei võimalda mudelil saavutada oma parimat võimalikku jõudlust, vaid säästab ka arvutusressursse ja treenimisaega. Lisaks uurime seost õppekiiruse lagunemise mõju ja sisendmestiku suuruse vahel. Arutleme ka eeliste üle, mida annab eelnevalt treenitud kaalude kasutamine võrreldes närvivõrgu nullist treenimisega.

Võtmesõnad:
sügavõpe, ülekandeõpe, peenhäälestus, resnet50, konvolutsiooniline närvivõrk, faster r-cnn

CERCS: P170 - Arvutiteadus, süsteemid, kontroll

Contents

Acknowledgement

I would like to express my sincere gratitude to, first and foremost, my supervisors, Tomas and Victor, for their invaluable guidance and constructive feedback throughout my research. Their insightful comments have been instrumental in shaping this work, and their expertise and dedication have enriched my academic experience. I would also like to thank my colleagues for their friendship, encouragement, and helpful discussions during my studies. Finally, I extend my thanks to my friends and family, who have been standing by my side and showing unconditional support whenever I need it.

1 Introduction

Machine learning is a field of study that has progressed dramatically over the past few decades. It addresses the question of how a computer system can be constructed so that it automatically learns through experience [18]. The growth of published information online has been so rapid that its massive amount contributes greatly to the fast development of machine learning algorithms [18].

Among many different subfields in machine learning, deep learning serves the task of recognising patterns from a training dataset and making decisions with the use of artificial neural networks [1]. Deep learning's exceptional performance in learning from data has made it highly popular [1]. However, because a deep-learning model is designed to learn from the data by extracting high-level complex abstractions, the amount of input data is very often required to be very large, so that the trained model will be able to generalise rather than fit the given data too closely [36].

Therefore, collecting data is essential for training deep learning models. Traditional machine learning models are designed to train and test on data in the same domain, which effectively means that the training data should serve the target task. However, in real-world cases, despite the enormous amount of data online, suitable training data is not always available or accessible, in which case it has to be gathered. The process of obtaining such data can be expensive or difficult at times. In fact, lack of data is a common hindrance in many industrial applications, where data gathering/annotation and computational resources for model training are often the main barriers to the financial feasibility of deep learning implementations. A dataset of brain tumour MR images, for example, is usually small because it requires expert radiologists to annotate an image [48]. In addition to that, training a model in a way that is more complex than necessary is a waste of computational resources, for example, adapting a large model to a small dataset or training a model so extensively that it is not learning in the final stage.

In opposition to the aforementioned traditional approach, transfer learning refers to the technique of pre-training a machine learning model on a dataset in a different domain than the target one and then exploiting such obtained information. Empirical studies have shown that transfer learning, which is currently widely used in various fields, such as computer vision and object detection, is an effective solution to the lack of data [7, 37, 48]. It pre-trains a model on a typically large dataset that is already in existence, such as ImageNet, or whose data is easy to obtain, so that the model can benefit from the information gathered from the pre-training when trained on a target dataset. This approach, however, poses several questions: How much of an impact does the size of a target dataset have on the effectiveness of transfer learning? How much should the pre-trained model fine-tune to adapt to the input data of different sizes?

This study aims to address these questions through experiments on a state-of-the-art architecture, Faster R-CNN ResNet-50 FPN [12], and a benchmarking dataset widely used for object detection and image segmentation tasks, COCO [6]. We discuss how

the size of an input dataset and how different ratios of trainable weights to the entire backbone network (i.e. the network that extracts features from an input image) affect the performance of the model and the computational resources the training job consumes. The impact on the effectiveness of learning rate decay is also discussed. We also compare the models with and without pre-trained weights to investigate the benefit of pre-training.

Based on the aforementioned reasons, the main goal of this practical study is to arrive at empirical insights that may ultimately help save computational resources and costs with data gathering in industrial applications.

The remainder of this paper is organised as follows. Chapter 2 presents both the theoretical framework and related work. Next, Chapter 3 provides information on the methodology covering both datasets and models involved in this paper. Sequentially, Chapter 4 explains how the experiments were conducted. Afterwards, Chapter 5 details the results of the experiments and discusses the arguments drawn from these outcomes. Finally, Chapter 6 summarises our study and gives suggestions on potential future research work.

2 Background

In this chapter, the theoretical frameworks relevant to our study and the related works are presented. We first describe the database involved in our work, followed by an outline of some fundaments of deep neural networks applied to object detection. Then, we give an overview of transfer learning, a main component of this study as well as an evaluation metric used throughout this study.

2.1 MS COCO

Lin et al. [26] introduced Common Objects in Context (COCO) dataset in 2014. In our study, we use an updated version of the COCO dataset, namely COCO Detection 2017 [27], which was introduced for COCO 2017 Object Detection Task Challenge [28]. This dataset is composed of images of objects across a diverse range of 80 classes. These images are split into a training set (118K) and a validation set (5K). Figure 1 shows a few examples of these images with the bounding box annotations. As seen in Figure 2, which illustrates the instance distribution of the 7 most frequent classes in the training set, the number of instances varies from class to class. Such instance statistics were preserved throughout the experiments in this study, as will be discussed in Section 4.2.

The COCO dataset presents the objects in their natural context in everyday scenes [26] and has been extensively used as a benchmark for object detection tasks [2, 42]. Several state-of-the-art object detection models, such as YOLOv7 [54], YOLO-LITE [16], Faster R-CNN [43], Efficientdet [51], were trained on the COCO dataset. Although the images in the dataset are publicly available, since some images contain sensitive information, e.g. personal identifiable information, we keep viewing individual images to a minimum. After all, we are only interested in the overall performance as a benchmark; therefore, viewing individual images to evaluate and improve performance has not been necessary.

2.2 Deep Learning for Object Detection

Deep learning methods have been seen to drastically improve computational models that execute tasks such as speech recognition and visual object recognition by allowing these models to learn data representations [21]. More recent deep neural network algorithms and architectures have further accelerated such progress. Introduced by LeCun et al. [22] in 1989, **deep convolutional neural network** (CNN) is a specific type of deep neural network that has shown excellent performance on multiple tasks involving computer vision and image processing in recent years [20]. Based on CNN, there mainly exist two types of modern objection detection pipelines - one-stage and two-stage architectures [3]. Figure 3 outlines the typical architecture of a two-stage detector that is composed of a backbone, a neck and a head module.

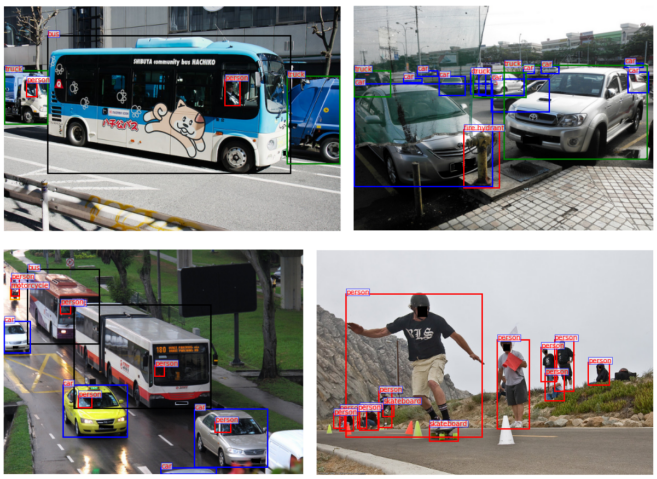


Figure 1. Example images in the COCO dataset with bounding box annotations [45]. Sensitive information is masked due to privacy regulations.

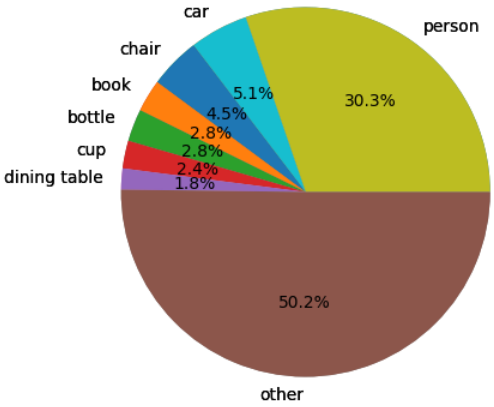


Figure 2. Instance distribution of the most frequent classes in COCO training set.

2.2.1 Backbone

Li et al. [24] gives an overview of different types of object detection methods. Older object detectors rely on hand-engineered object component descriptors, of which a few examples include HOG [5] and Selective Search [52]. More recent object detectors, on the other hand, leverage their learning ability made possible by deep convolutional neural networks (CNN). The CNN that extracts features of an input image is typically

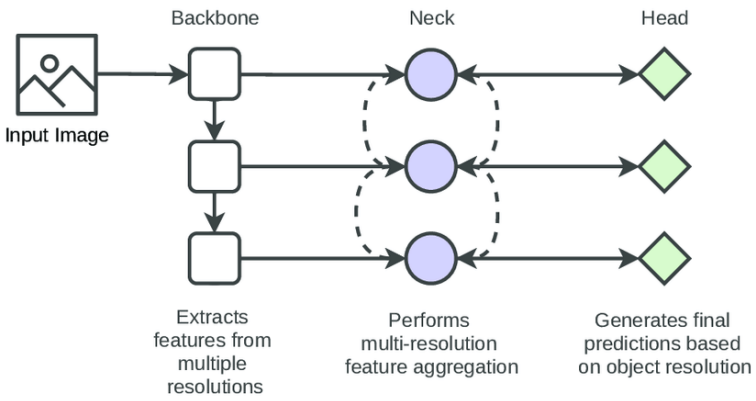


Figure 3. Typical architecture of an two-stage backbone-neck-head object detection pipeline [19].

referred to as the **backbone**. Among these CNN-based detectors, there are two different architectures:

1. one-stage design, where the backbone is also used for object instance prediction;
2. two-stage design, where an additional classifier (also referred to as the head) is included for proposal classification and regression.

Since we focus on a two-stage object detector (Faster R-CNN), as mentioned in this section, **backbone** in our study refers to the CNN that extracts features from an input image.

Figure 4 illustrates the overall structure of **ResNet-50**, a CNN introduced by He et al. [12], among other CNNs, such as VGG-19 [47], Inception V3 [49], and DenseNet201 [14]. ResNet-50 is commonly used as a backbone network and has achieved outstanding outcomes in image recognition tasks, for example, medical face mask detection [31] and malicious software classification [44]. It incorporates 1 stem and 4 residual stages in addition to some other layers. The stem mainly consists of a convolutional layer and a max pooling layer, whereas each residual stage is made of 1 conv block and several identity blocks. Each conv or identity block includes 3 or 4 convolutional layers, respectively. The shortcut connections presented in this start-of-the-art CNN were found helpful in solving the problem of vanishing/exploding gradients in a very deep neural network [12]. In our study, following the practice of the PyTorch library introduced in Section 2.3, we refer to the stem and all 4 residual stages as **stages**, which leads to 5 stages in total.

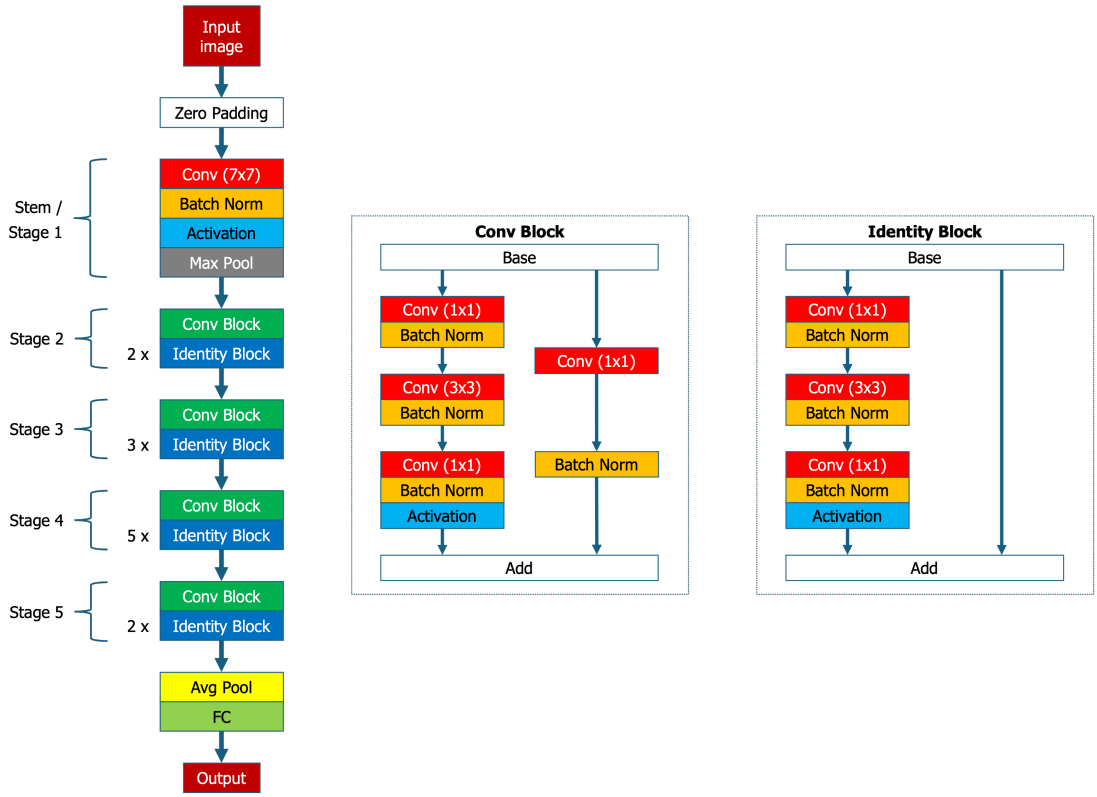


Figure 4. Architecture of ResNet-50. Layers involved are convolutional (Conv), batch normalisation (Batch Norm), max pooling (Max Pool), average pooling (Avg Pool), and fully connected (FC).

2.2.2 Neck

The **neck** is another module that a two-stage object detector may contain. Situated between the backbone and the classifier, this module takes features extracted by the backbone at different scales as input and then fuses these features before passing the feature maps to the classifier [19]. Feature Pyramid Network (FPN) [25], Path-augmented Network (PAN) [29], and Bi-directional FPN (BiFPN) [51] are some examples of networks used as a neck module. Although the advantages of implementing a neck module are not directly addressed in these studies, significant improvements in the accuracy of several tasks, such as object detection and instance segmentation, are seen over the baselines that do not bear a neck module [25, 29].

Lin et al. [25] introduced **Feature Pyramid Network (FPN)** as a generic feature extractor. The top-down pathway and lateral connections that an FPN bears make object detection at vastly different scales of an image possible, which consequently produces a

multi-scale feature representation with semantically strong features at all levels. Such an advantage eventually leads to model performance improvement at a marginal extra cost. For example, a basic Faster R-CNN detector with an FPN implemented outperformed the challenging COCO detection benchmark.

2.2.3 Faster R-CNN

Ren et al. [43] proposed a Faster Region-based Convolutional Neural Network (Faster R-CNN), which is a state-of-the-art object detection method containing three main modules: a backbone network, a Regional Proposal Network (RPN), and a classifier - the Fast R-CNN [10] detector. Figure 5 presents the overall architecture of Faster R-CNN. An RPN is a fully convolutional network that is trained to produce high-quality regional proposals at nearly no extra cost as it shares full-image convolutional features with the detection network so that the entire integrated network concentrates on these proposed regions. A Faster R-CNN model accepts an RGB image as input and then scales it so that the longer side is capped at 1000 pixels and the shorter side at 600 pixels. In the original work [43], the results on MS COCO were reported, with VGG-16 [47] being the backbone and FPN being the neck. Faster R-CNN models are applied in a wide range of fields, such as textile fabric defect detection [11], surgical instrument recognition [23], and traffic sign detection [59].

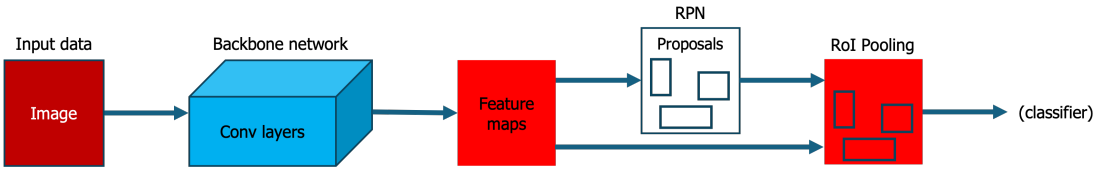


Figure 5. Typical architecture of Faster R-CNN that consists of a backbone network, an RPN and an RoI pooling layer. Adapted from Ren et al. [43].

Faster R-CNN ResNet-50 FPN is an implementation of Faster R-CNN, where the original backbone (VGG-16) is replaced with ResNet-50 and FPN is added into the pipeline as the neck so that this object detection method benefits from the aforementioned advantages of using ResNet-50 and FPN.

2.3 Transfer Learning

Transfer learning is a process of improving the model performance on the target dataset and its corresponding task (target learner) by exploiting the information gained from the source dataset and its associated task (source learner), where source and target datasets are not in the same domain [56]. Specifically in the field of computer vision and object

detection, transfer learning involves using a pre-trained model on a related task to learn to extract features or to classify objects so that a target learner can benefit from these pre-trained parameters rather than starting from completely random weights [56]. Although traditional machine learning methods were assumed to be trained on the data in the same domain as the target data, transfer learning can benefit real-world cases where training data is expensive or difficult to obtain, for example, annotated cancer/tumour images [7, 41, 48] and animal facial expressions [37].

Pre-training has been a prevailing technique applied in computer vision due to the connection between many vision tasks [61]. As the backbones of object detection and instance segmentation models, convolutional neural networks pre-trained on a large dataset such as ImageNet [6] have seen remarkable success, which has now also become a common practice [4, 61]. After pre-training, a target learner is typically trained while freezing the early layers that extract features (i.e. the backbone network in an object detector) and **fine-tuning** the last few layers so that the specific information related to the target dataset as well as the target task is captured [13, 50, 53]. This corresponds to the explanation that the early layers, due to their smaller field of view, learn smaller details such as object textures, which are often reappearing in different objects, whereas the late layers with a larger field of view capture more complex contextual features that distinguish objects of a class from another and are more sensitive to the specific task of a network [8, 9].

PyTorch [34] is an open-source library. Two implementations of the Faster R-CNN ResNet-50 FPN model for the task of object detection were provided in the PyTorch library, both of which had been pre-trained on ImageNet-1K. The pre-training was conducted on a plain ResNet-50 model as introduced in Section 2.2.1. The weights transferred from this pre-trained model to those two implementations only involved the parameters in the 5 residual stages, leaving the FPN, RPN and FC layers initialised with random weights.

2.4 Evaluation Metric

An evaluation metric is used to measure the performance of a machine-learning algorithm. **Mean average precision** (mAP) is one of these metrics, which has been adopted as an assessment of the performance of many object detection modules [40]. It is defined as:

$$\mathrm{mAP} = \frac{1}{N} \sum_{i=1}^{N} AP_i \ , \tag{1}$$

where N denotes the number of classes found in an input dataset and AP denotes the average precision of a class. Zhu [60] derives the trade-off between precision and recall and concludes that both should be taken into account when evaluating and comparing object detection models. Hence, AP, which considers false positives and false negatives

per definition and embodies both precision and recall, is an actual metric for object detection [43]. In practice, AP finds the area under the precision-recall curve and is defined as:

$$\mathrm{AP} = \sum_{i=1}^{n} p(i)\Delta r(i), \tag{2}$$

where n denotes the number of instances retrieved, $p(i)$ denotes precision as a function of recall for a class, and $\Delta r(i)$ denotes the change in the recall from $i-1$ to i [60]. Precision, which is the fraction of correctly classified instances among all retrieved instances, and recall, which is the fraction of correctly classified instances retrieved, are defined as:

$$\mathrm{Precision} = \frac{tp}{tp+fp}\,, \tag{3}$$

$$\mathrm{Recall} = \frac{tp}{tp+fn}\,, \tag{4}$$

where tp denotes true positives (TP), fp denotes false positives (FP), and fn denotes false negatives (FN) [38]. TP is the number of detections whose intersection-over-union (IoU, also referred to as Jaccard index [17]) is greater than or equal to a given threshold, while FP is the number of detections with an IoU smaller than that threshold or of a nonexistent object [40]. FN is the number of ground-truth bounding boxes that are not detected [40]. IoU is defined as:

$$\mathrm{IoU} = \frac{\mathrm{area\ of\ intersection}}{\mathrm{area\ of\ union}}\,, \tag{5}$$

where $\mathrm{area\ of\ intersection}$ is the area of the intersection of the ground-truth bounding box and the predicted bounding box of an instance, and $\mathrm{area\ of\ union}$ is the area of the union of these two bounding boxes.

It is worth noting that true negative, which refers to not detecting objects that do not exist, is not considered for the task of object detection. This is because whenever no predicted bounding boxes are placed outside of the ground-truth bounding boxes in an image, such nonexistent instances are constructed, which are practically uncountable.

3 Methodology

In this chapter, the methods and approaches of our study are detailed, including the dataset, architecture, evaluation, and training of models.

3.1 Dataset

Typically, a model is first pre-trained on a large dataset and then fine-tuned on the target input dataset as explained in Section 2.3. In principle, any kind of data can be adopted as long as it is suitable for the model(s) selected. Since we focus our study on the fine-tuning phase rather than the pre-training, we choose a network that is pre-trained on a dataset deemed sufficient. As mentioned in 3.2, because the proposed network in our study is a Faster R-CNN ResNet-50 FPN model with the task of object detection in RGB images, a dataset of RGB images is adequate. These images should be annotated in a way such that a bounding box, as well as a corresponding class, are assigned to an object found in the images.

Considering that the purpose of our study is to examine the effect of the input data amount on transfer learning fine-tuning, several *data amount scenarios* are created. Hence, an input dataset of large enough size befits the goal for the reason that it can be split into smaller subsets. These fractions of the full input dataset should maintain the characteristics of the full set in order to allow for fair comparisons among the results obtained from all fractions and the full set. Hence in our study, the distributions of instances in all classes will be kept the same in subsets as in the full dataset.

3.2 Model Architecture

Considering the component of transfer learning in our study, a source learner model on which experiments are conducted ought to bear layers that can be pre-trained, and the weights of the layers of interest can be transferred to a target learner model, as explained in Section 2.3. We concentrate our study on the fine-tuning phase (i.e. target learner) of transfer learning instead of the pre-training phase (i.e. source learner).

The network Faster R-CNN ResNet-50 FPN is chosen to be the base target learner model in our study for the reason that, as mentioned in Section 2.2, it is a state-of-the-art object detection pipeline benefitting from the advantages of using ResNet-50, RPN, and FPN. As illustrated in Figure 4, the backbone of this network, ResNet-50, comprises 5 stages, each of which can be in either of the following two states:

1. **trainable** state, where the weights in the residual stage are updated during back-propagation and consequently adapt to the input dataset and consume more computational resources such as memory and time;

2. **frozen** state, where the weights will be locked and remain constant throughout the training of the model.

Given this context, the action of unfreezing a stage refers to changing the state of a stage from frozen to trainable. This unfreezing process is done sequentially from the last stage (closer to the output) towards the first, as shown in Figure 6. Therefore, 6 *trainable stage scenarios* can be formed when having different stages frozen as follows:

1. all 5 stages are trainable;
2. the last stage is frozen, while the first 4 are trainable;
3. the last 2 stages are frozen, while the first 3 are trainable;
4. the last 3 stages are frozen, while the first 2 are trainable;
5. the last 4 stages are frozen, while the first 1 is trainable;
6. all 5 stages are frozen.

3.3 Model Evaluation

The selected model is trained in several scenarios, between which the trainable stages in the backbone network, as well as the dataset size, vary. To evaluate and compare the trained models in our study, mean average precision (mAP) and training time are compared.

3.3.1 Mean Average Precision

As mentioned in Section 2.4, mAP is a common evaluation metric for object detection models, including Faster R-CNN models [43]. In our study, a model is evaluated on its average mAP over different IoU thresholds from 0.5 to 0.95, step 0.05. This metric is COCO's standard evaluation metric [15, 43]. We denote it by mAP or mAP@[0.50:0.95].

3.3.2 Training Time

Training time is defined as the total time consumed in training a model, excluding loading data and the model and validating the model, i.e.:

$$\text{Training time} = \sum_{i=1}^{m} E(i), \tag{6}$$

where m denotes the total number of epochs in a model training job and $E(i)$ denotes the total time elapsed in each epoch of the training job. All the servers where the model

training jobs are executed bear identical specifications, including GPU models, numbers of GPUs and CPUs, and system memory, such that the comparisons made between the models are fair and meaningful.

3.4 Model Training

In our study, the model training processes are grouped into 3 parts - the investigations into statistical significance, the variation of subsets, and the comparison of models.

3.4.1 Statistical Significance

There are several factors that contribute to the randomness of an individual execution of model training jobs. For example, some layers in a model are initialised with random weights if they are not transferred from a pre-trained model. In addition to that, during the feed-forward and backpropagation processes, the training set is randomly split into two subsets for training and testing. To establish whether particular results occur due to such randomness or due to the factors of interest, which in our study are mainly the freezing of convolutional layers and the dataset size, executions of model training jobs are repeated multiple times with an identical setup and input dataset. A comparison is then made between the resulting performance of these models, i.e. mAPs and training times, for inspection of possible inconsistency. Such repetitions are kept to a minimum to save training time.

3.4.2 Variation of Subsets

Subsets of the full target input dataset are created to construct different *data amount scenarios*, as mentioned in Section 3.1, for the purpose of understanding the impact of the size of an input dataset. In order to determine whether certain outcomes result from a particular fraction of the full set, models are trained on several variations of subsets with an identical setup. The proportion of these subsets in the full dataset remains constant while they incorporate various fractions of the full dataset. The performance of the models is then compared for observation of the potential discrepancy. These attempts are kept to a minimum to save training time.

3.4.3 Comparison of Models

As mentioned in Sections 3.1 and 3.2, the study is conducted in multiple scenarios where the two variables of interest, namely the size of a dataset and the number of trainable stages, differ. As a way to ensure a fair comparison between the outcomes, the models are trained following a fixed setup. I.e., the hyperparameters used in all training executions,

e.g. the number of iterations of batches with backpropagation (hereinafter ‘iterations’) and the learning rates, remain constant.

4 Experiments

In this chapter, the approach to the experiments conducted in our study is detailed. Figure 6 gives an overview of the experiment design proposal. We applied the ResNet-50 weights, which were pre-trained on ImageNet [6], on a Faster R-CNN Reset-50 FPN network. Both the ResNet-50 weights and the network were provided in the PyTorch repository [34].

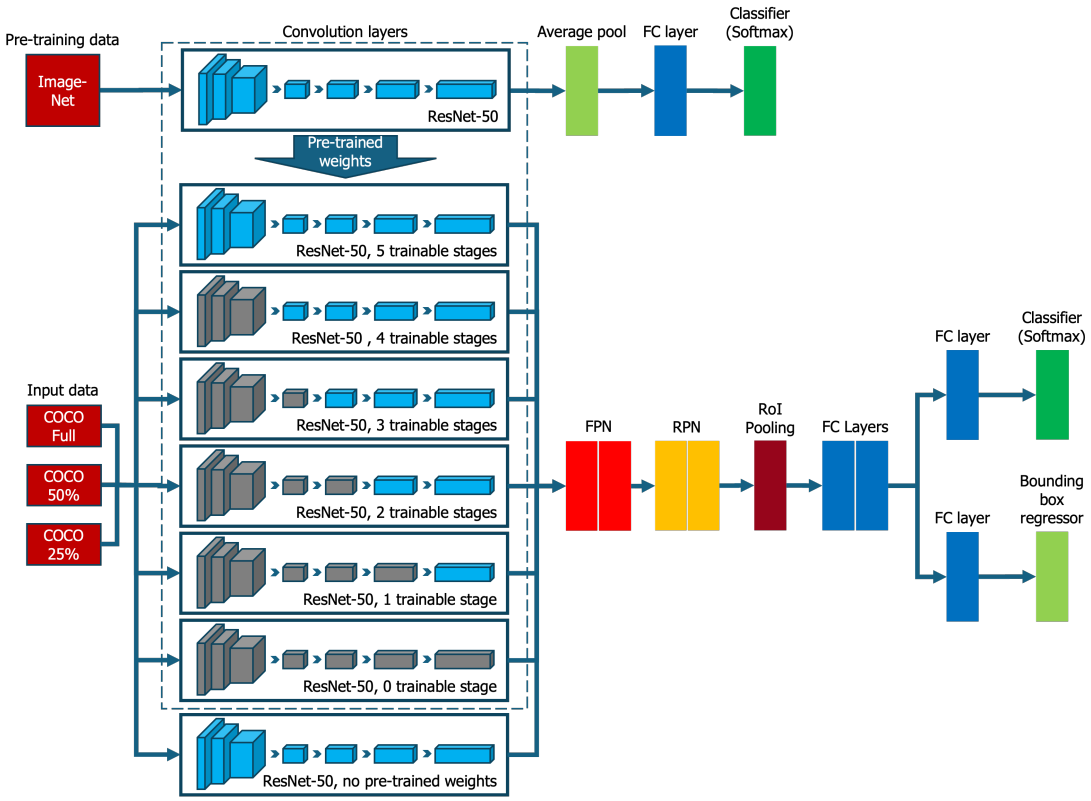


Figure 6. Overall design of the experiments.

4.1 Setup

All the experimental work was performed on 6 cloud-based virtual Linux servers (Ubuntu 20.04), each equipped with 4 NVIDIA T4 Tensor Core GPUs.

4.2 Dataset

In this subsection, we will describe how the training and validation datasets were set up in the experiments. The 2017 version of the MS COCO dataset, described in Section 2.1, was the only dataset involved in our study.

4.2.1 Training Set

The training set of COCO (hereinafter 'full dataset' or 'full set') was adopted as the training set in our work. To construct the proposed *data amount scenarios* mentioned in Section 4.3, several subsets were created with COCO minitrain [46], which followed a stratified sampling method. For the purpose of ensuring the generalisability of the outcomes from one subset, three 25% subsets were created, each of which was made of 25% of the full dataset while corresponding to a different fraction of the population. Additionally, one 50% subset containing 50% of the full dataset was generated. Table 1 summarises the numbers of images and annotations contained in each subset of the full dataset.

The characteristics of the full dataset should remain unchanged to avoid introducing undesired impacts. Thus, in each of the curated subsets, the ratio of the annotation count of each class to the entire dataset was essentially kept the same as that of the full dataset. Figure 7 shows the distribution of annotations per class normalised by total annotation count in each *data amount scenario*. Due to round-off errors, the numbers of instances in each class across these subsets are not exactly the same. However, the most significant difference found in the proportion of a class between a subset and the full dataset was 0.005 pp.

Table 1. Image and annotation counts in each *data amount scenario*.

Dataset	**Alias**	**Image count**	**Annotation count**
Full set	COCO100	118,287	861,002 (100%)
50% subset	COCO50	59,144	434,964 (50.5%)
25% subset 1	COCO25-1	29,572	216,127 (25.1%)
25% subset 2	COCO25-2	29,572	216,887 (25.2%)
25% subset 3	COCO25-3	29,572	217,750 (25.3%)

4.2.2 Validation Set

The validation set of COCO was adopted as the validation set in our work. No subset of this validation set was created because, as explained in Section 4.3.3, the hyperparameters of a training job were not optimised based on the evaluation of a model with a validation set, meaning the validation set did not contribute to the improvement of model

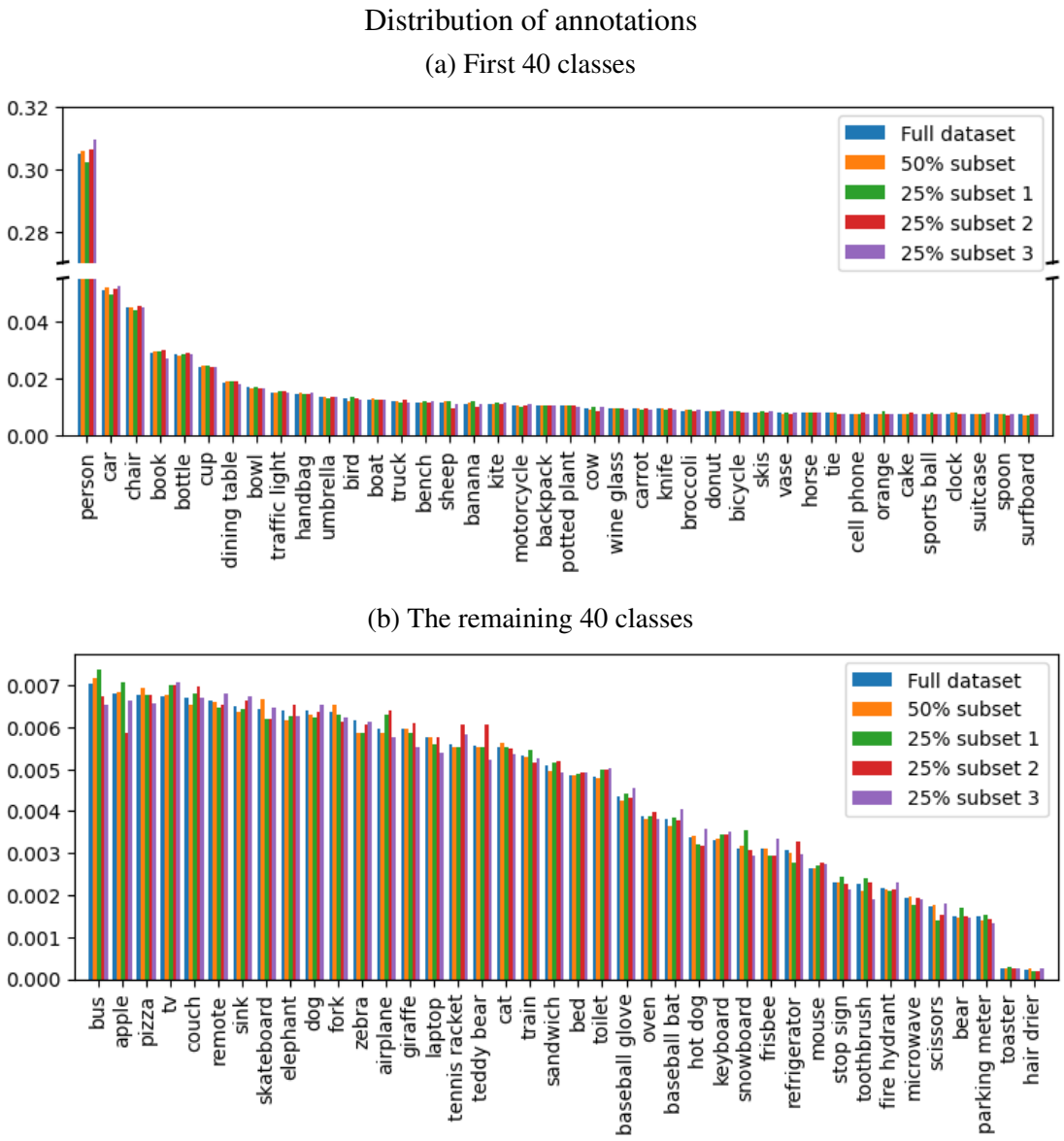


Figure 7. Number of annotations per class normalised by total annotation count in each subset compared to the full dataset.

performance. Additionally, a consistent validation set ensured fair comparisons between the performance of models.

4.3 Training

As mentioned in Section 2.3, there were two implementations of the Faster R-CNN ResNet-50 FPN model in the PyTorch library - V1 and V2. It was not necessary for us to pre-train these two models ourselves because both instances were pre-trained on ImageNet, a dataset that was deemed adequate for initialising a deep network [35]. Although the V2 implementation (mAP 46.7%) outperformed V1 (mAP 37%), we selected V1 because V2's training recipe showed that it was trained in a workload manager called Slurm, to which at the time of writing we had no access, and we were able to reproduce the outcome of V1 to ensure consistency with the training script. The architecture of the V1 instance was left in its default setting. A noticeable difference between this implementation and the original work [43] was that this model instance scaled an input image to cap the longer side at 800 pixels as opposed to 1000 pixels. However, by keeping this constant, we do not expect this difference to have an impact on our experiments.

4.3.1 Statistical Significance

We first conducted the experiments of repeating the model training jobs with all 6 *trainable stage scenarios* on one selected dataset, specifically dataset COCO25-1 defined in Section 4.2, for 3 times, with the plan to run more extensive experiments on the other datasets in case a strong sign of randomness in the outcomes would be observed. However, as shown later in Section 5.1, we could not motivate spending more computational efforts on the matter and kept this experiment to the minimal.

4.3.2 Variation of Subsets

Similarly to the approach in Section 4.3.1, for the purpose of verifying whether certain outcomes were specific to a particular fraction of the full dataset, we conducted experiments on one *data amount scenario*, namely the 25% subset variations (COCO25-1, COCO25-2, and COCO25-3 defined in Section 4.2), in all 6 *trainable stage scenarios*. More extensive experiments on the 50% *data amount scenario* were intended if noticeable variances would be recognised in the outcomes. Nevertheless, as explained later in Section 5.2, such additional experiments were not deemed necessary.

4.3.3 Comparison of Models

All experiments in our study were based on the benchmark from the PyTorch library, whose training recipe (the training recipe) [33] was followed. The majority of the hyperparameters, e.g. momentum, weight decay, and RPN score threshold, remained in default as defined in the training recipe. In order to ensure fair comparisons between the outcomes, a model evaluation during the process of training did not contribute to

optimising these hyperparameters 4.2.2, which could be verified from the training script. All evaluations were performed on the validation set (except when stated otherwise, e.g. in Sections 5.3.1 and 5.3.3).

Training with pre-trained weights. We kept the number of GPUs constant at 4 throughout our experiments, which was different from that for the benchmark. To compensate for the different batch size, which was dependent on the number of GPUs, the base learning rate was adjusted from 0.02 to 0.01, as suggested by the authors of the training recipe.

Additionally, since the number of backpropagations, and thereby the opportunities for a model to learn, would be different if we kept the number of epochs constant, considering the size of an epoch varied between different *data amount scenarios*, we instead kept the number of iterations constant. Specifically, according to the training recipe, the model was trained for 26 epochs, each with 14,658 iterations, resulting in 381,108 iterations in total. Consequently, to preserve the total number of iterations, as the dataset size reduced to 25% and 50% of the full dataset, the model was trained for 52 and 104 epochs, respectively.

Furthermore, the training recipe also included a multi-step learning rate scheduler, which made the learning rate decay after Epochs 15 and 21. To adapt to the adjusted total number of epochs for a smaller input dataset, the learning rate was scheduled to decay after Epochs 31 and 43 for a 50% subset and at Epochs 63 and 87 for a 25% subset so that the proportion of iterations trained at a certain learning rate among all iterations was maintained in all training jobs.

The selected network, i.e. Faster R-CNN ResNet-50, was trained in all 3 *data amount scenarios* combined with all 6 *trainable stage scenarios*, resulting in 18 different models for comparison.

Training from scratch. Moreover, we trained the network on the full dataset without applying the pre-trained weights (using random initialisation of weights). In this execution, all stages were set to trainable. In addition to that, since the other models had received a significant amount of training during pre-training, it would be unfair to follow the preset step scheduler in the training recipe when the network was completely untrained. Hence, the learning rate did not decay following the step scheduler but was adjusted manually. This was due to allowing the model to stabilise at its peak performance with a given learning rate before changing it, which we defined as no more than a 3% increase in mAP over at least 5 epochs, a similar approach used in training the benchmark as shown in Figure 11a. To achieve this, the training script in the PyTorch library [34] was modified to ignore the optimiser, of which the multi-step learning rate scheduler was a part, and to enable resuming the training job at a newly appointed learning rate after being interrupted. As a result, we maintained a learning rate equal to the base

learning rate in the training recipe, i.e. 0.01, until Epoch 50 when it decayed to 0.001, and subsequently to 0.0001 – the same ending learning rate as in the training recipe – after another 8 epochs.

Variables to compare. As stated in Section 3.3, mean average precision and training time are two variables used to compare model performance. Additionally, in our experiments, significant discrepancies between the highest mAP (best mAP) and the mAP of the last epoch (last mAP) within a training execution were found when the dataset size was lowered. Hence, both the best and last mAP metrics were recorded and discussed.

5 Results and Discussions

This chapter describes the outcomes of the experiments we conducted and the discussions we put forward. The experiments on statistical significance and variations of subsets are first examined, then the model performance and training time, and finally, the effects of pre-trained weights as well as learning rate decays.

As defined in Section 3.2, 6 *trainable stage scenarios* were constructed with the backbone network (ResNet-50) with 0-5 residual stages being trainable and the rest being frozen. As defined in Section 4.2, 25% and 50% of the COCO training dataset were created as subsets, which, together with the full set, established 3 *data amount scenarios* (COCO25-1, COCO50, and COCO100).

5.1 Statistical Significance

Figure 8a shows the best and last mAPs of the pre-trained Faster R-CNN ResNet-50 FPN network trained on dataset COCO25-1 for 3 times with an identical setup, and Figure 8b shows the training times. Between these 3 executions, the most significant difference in best mAP was found to be 0.4pp (1.6%), in last mAP 0.3pp (1.3%), and in training time 2.9 hours (4.9%). Therefore, we argue with such small differences that due to the minor discrepancy observed, under the experiment settings in our study, outcomes are not specific to one individual execution of the model training job but are consistent across the repetitions, and hence the repetition of such executions is not necessary.

5.2 Variations of Subsets

Figure 9a shows the best and last mAPs of the pre-trained Faster R-CNN ResNet-50 FPN network trained on datasets COCO25-1, COCO25-2, and COCO25-3, and Figure 9b shows the training times. Between these 3 executions, the greatest difference in best mAP was found to be 0.5pp (2.0%), in last mAP 0.7pp (2.9%), and in training time 3.4 hours (5.1%). Accordingly, with such small variances, we argue that a particular outcome is not due to a certain fraction of the full dataset under the experiment settings in our study, and hence it is sufficient to consider only one subset for each *data amount scenario*. Consequently, such experiments were not repeated for the 50% subset scenario.

5.3 Model Performance

As mentioned in Section 4.3.3, we kept all hyperparameters constant to enable fair comparisons between models, except for the number of epochs, which was adjusted to ensure the same number of iterations. The learning rate scheduler was also adjusted to adapt to the number of epochs.

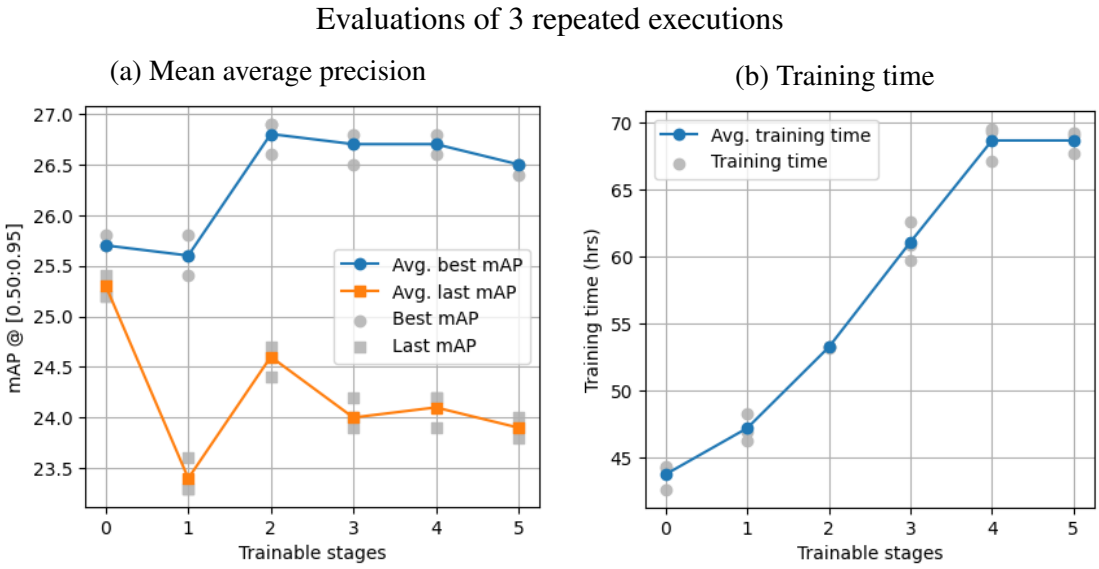


Figure 8. Best (grey circle marker) and last mAPs (grey square marker) of each run as well as average best (blue curve) and last mAPs (orange curve) (a) and training times of each run (grey circle marker) and average training times (blue curve) (b) of 3 repeated executions of training models on COCO25-1 with respect to the number of trainable stages.

Figure 10 illustrates the performance (mAP) of models trained with 6 *trainable stage scenarios* (i.e. 0-5 trainable stages in the backbone network as defined in Section 3.2) on 3 *data amount scenarios* (i.e. datasets COCO25-1, COCO50, and COCO100 as defined in Section 4.2).

5.3.1 Overfitting Problem

As seen in Figure 10, the largest gaps in all 3 *data amount scenarios* were 0.3pp for the full set, 2.3pp for the 50% subset, and 2.7pp for the 25% subset. Seemingly, the smaller the size of an input dataset was, the larger the gap was between the last and best mAPs, indicating more worsening of the model performance from the best-performing model until the last model. This phenomenon was conceivably due to a severer overfitting problem when an input dataset was smaller.

To verify this potential cause, we evaluated the models with 3 trainable stages on the training sets COCO25-1, COCO50, and COCO100, as well as the validation set (w.r.t. the ground-truth annotations). The results of these evaluations are demonstrated in Figure 11. For the benchmark as shown in Figure 11a, although the validation mAP was diverging from the gradually increasing training mAP, the performance on the validation set was only stagnating, not decreasing, which was not necessarily a sign of overfitting

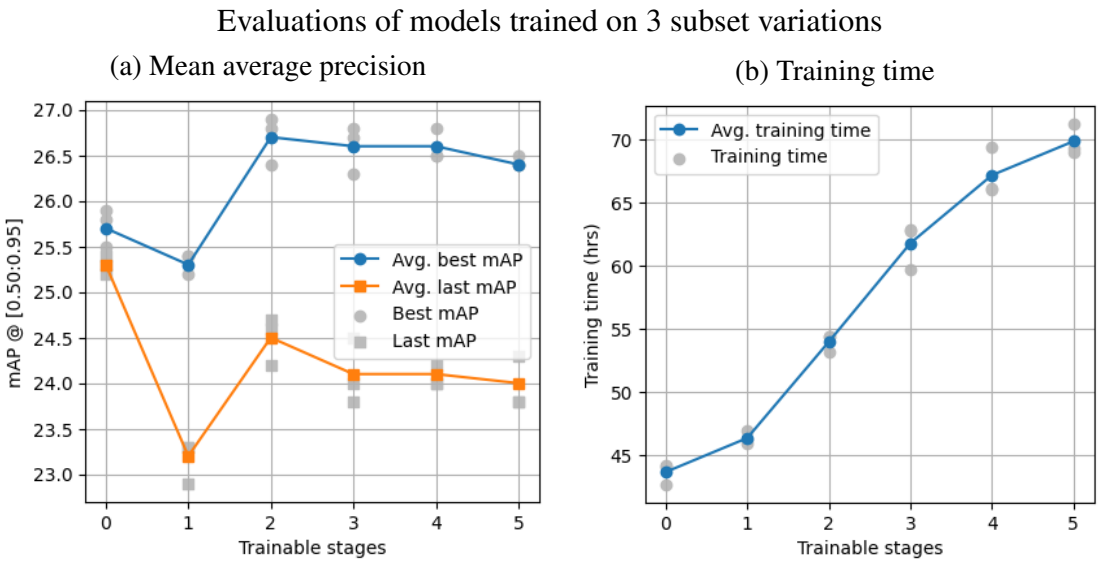


Figure 9. Best (grey circle marker) and last mAPs (grey square marker) of each run as well as average best (blue curve) and last mAPs (orange curve) (a) and training times of each run (grey circle marker) and average training times (blue curve) (b) of models trained on 3 variations of 25% subsets (COCO25-1, COCO25-2, and COCO25-3) with respect to the number of trainable stages.

but over-training. Such over-training was not harmful since the model performance did not worsen, though it was wasteful given no improvement in the model mAP. However, as the size of the input dataset reduced to 50% and 25 %, this was no longer the case. As seen in Figures 11b and 11c, as the training and validation mAPs diverged, the validation set performances were clearly decreasing even though the training set performances continued to increase, which was a strong sign of overfitting. Overall in these 3 scenarios, despite the initial synchronous increases in mAP at the first change of learning rate, the validation set mAPs descended while the training set mAPs continued to ascend. Hence, we find that the smaller the dataset is, the severer the overfitting problem is.

Similar findings can be found in other studies. For example, Zhao [58] pointed out that the massive parameters in a CNN model required a large dataset because otherwise, the model would fit the training data too closely and would have a lower generalisation ability, which would lead to overfitting. Liu and Deng [30] also argued that simpler models should be used for smaller datasets to avoid overfitting and improve generalisation.

As initially suspected, we can indeed see that there is an association between the input dataset size and the worsening of the model performance. Even with the use of pre-trained weights, overfitting will still occur if the input dataset is too small. And more specifically, overfitting increases as the size of an input dataset decreases.

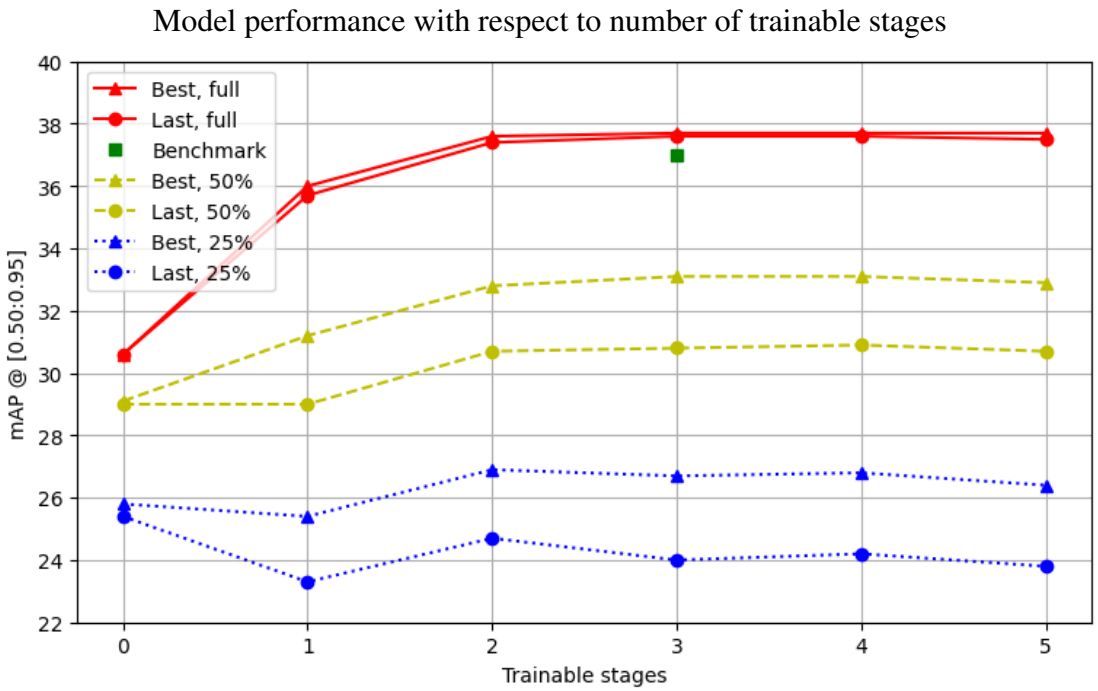


Figure 10. Best (triangle marker) and last (circle marker) mAPs in the three data amount scenarios, 100% (red solid line), 50% (yellow dashed line), and 25% (blue dotted line) in the six scenarios of 0-5 trainable stages, as well as the benchmark with 3 trainable stages and 100% data amount from PyTorch (green square marker).

5.3.2 Advantage of Unfreezing

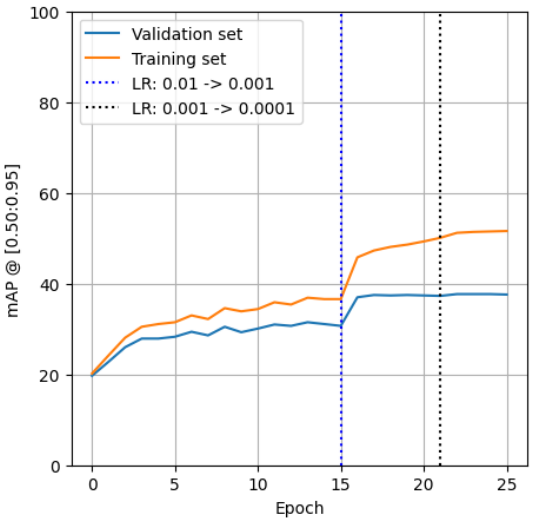
As seen in Figure 10, the change in mAP across different trainable stages was found to be milder on a smaller dataset than on a larger one. In terms of the best mAP, the greatest differences were 7.1pp on the full set, 4.0pp on the 50% subset, and 1.5pp on the 25% subset.

Zhao [58] put forward that when fitting a small dataset on a CNN model with massive parameters, an effective approach to solving an overfitting problem was to reduce the number of parameters. Liu and Deng [30] concluded that strong dropout settings, among other methods, should be in place for adopting a deep model to a small dataset. In our experiments, similar to dropout, freezing a stage can be seen as a form of excluding parameters from fitting the training data.

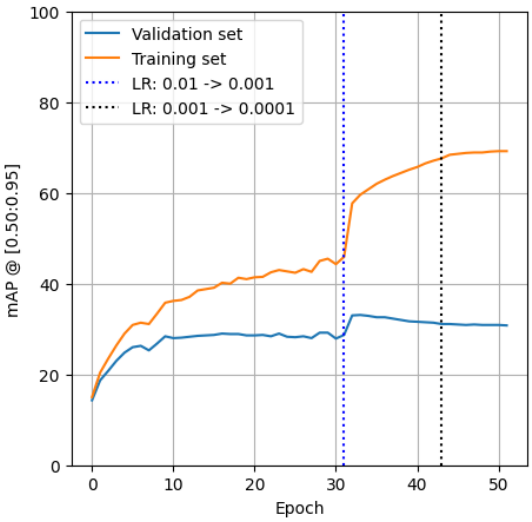
As suggested by the experiment outcomes, potentially owing to the use of pre-trained weights, allowing more parameters to adapt (be trained) to the input data by unfreezing more stages did not seem to lead to a severer overfitting problem. However, as observed in Figure 10, the lower the input data amount was, the less increase in mAP the model

Performance (mAP) of models with 3 trainable stages in 3 *data amount scenarios*

(a) Full set (benchmark)



(b) 50% subset



(c) 25% subset

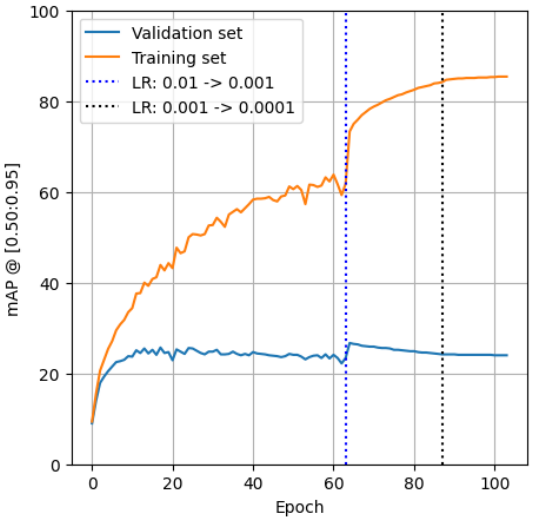


Figure 11. Training (blue curve) and validation mAPs (orange curve) of models with 3 trainable stages throughout Epochs 0-25 with learning rate (LR) changes after Epochs 15 and 21 trained on 100% (a), throughout Epochs 0-51 with LR changes after Epochs 31 and 43 trained on 50% (b), and throughout Epochs 0-103 with LR changes after Epochs 63 and 87 trained on 25% (c) of the COCO dataset.

gained, substantially from 0 to 3 trainable stages. Hence, as shown in the experiments, the advantage of unfreezing stages to improve the model performance appears to reduce as the input dataset size lowers.

In practical terms, due to the unlikely increase in model performance, more stages may be frozen for a smaller input dataset to save computational power. By contrast, on a larger input dataset, unfreezing more stages seemingly leads to a better-performing model. This is most likely because, when more stages are trainable, a model can better adapt to a large dataset that has enough information to provide, while a small dataset can

cause the model to unlearn the information from its pre-training by over-training on the input dataset, which eliminates the advantage of transfer learning.

5.3.3 Effect of Backbone Network

In each of the three *data amount scenarios*, when all residual stages were frozen, the difference between the best and last mAPs was found to be 0pp (0%) on the full set, 0.1pp (0.0034%) on the 50% subset, and 0.4pp (1.6%) on the 25% subset. Despite the aforementioned overfitting problem, such minor variations between the best and last mAPs showed no sign of overfitting; rather, the discrepancy seemed to be due to the slight random variations often observed between epochs.

To validate this suggestion, the models were evaluated with all stages frozen on the corresponding training sets, as shown in Figure 12. In each of the three scenarios, at first, the mAPs on both the validation and training sets increased until the first learning rate decay. Afterwards, the validation mAPs became stagnant on the full set and the 50% subset, whereas on the 25% subset, the validation mAP decreased slightly. This decline indicated a minor overfitting problem, which also corresponded to the 0.4pp difference. Nevertheless, as any of the residual stages became trainable, the overfitting problem was again clearly observed.

As mentioned in Section 2.2.3, the Faster R-CNN ResNet-50 FPN network comprises several components, namely the backbone network ResNet-50 that extracts features from an image, the neck network FPN that improves the detection by using multi-level features, the RPN that generates more accurate bounding box proposals, the RoI pooling that produces a list of feature maps with a fixed size, and the FC layers that function as an object classifier and a bounding box regressor. Note that when there are 0 trainable stages, it is not the backbone network that is fine-tuned to the input dataset but the FPN, RPN, RoI pooling, and FC layers, which still remain trainable.

Hence, our experiments that showed no or little overfitting problems with 0 trainable stages suggest that the overfitting problem is mainly due to feature extraction by the backbone network rather than the effects from the other layers.

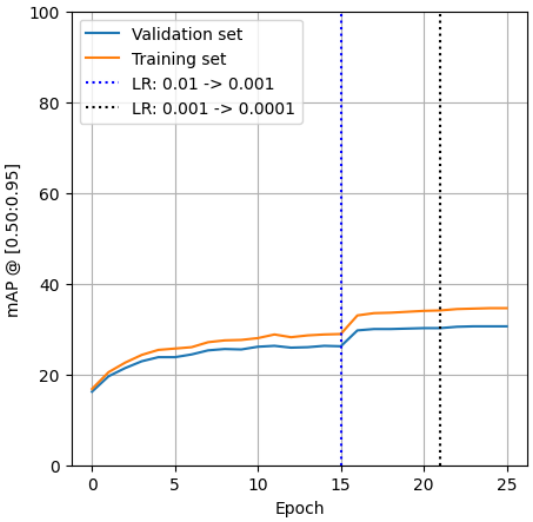
5.4 Model Training Time

Figure 13 demonstrates the training times in all *trainable stage scenarios* combined with *data amount scenarios* with respect to the number of trainable stages. A steady increase in training time could be seen as more stages became trainable. Nonetheless, there were marginal changes in training time between 4 and 5 trainable stages, indicating that the first stage (the stage closest to input) did not increase the training time in the way that the other stages did.

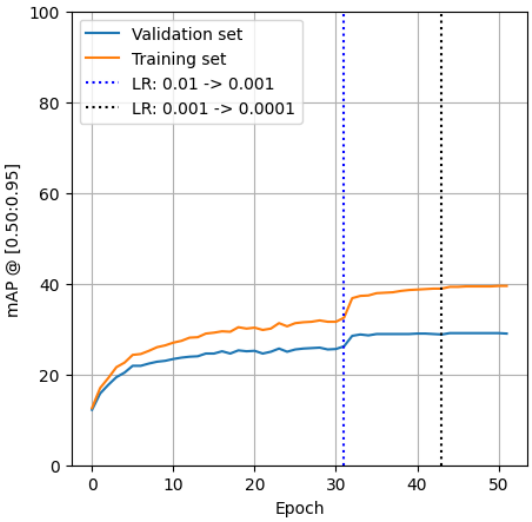
As mentioned in Section 2.2.1, in a ResNet-50 network, the first stage consists of only 1 convolutional layer among other types of layers, whereas each of the other stages

Performance of models with 0 trainable stages in 3 *data amount scenarios*

(a) Full set



(b) 50% subset



(c) 25% subset

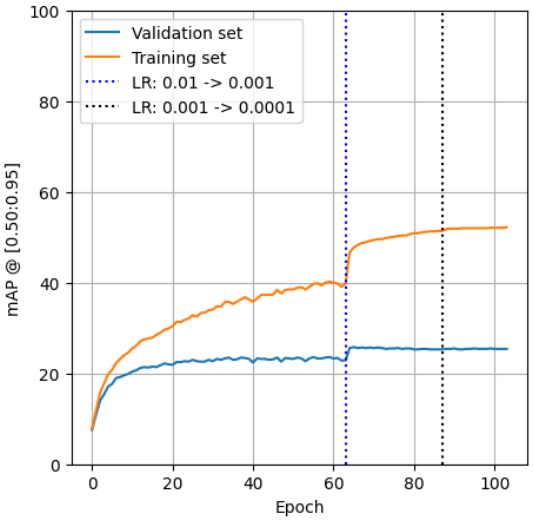


Figure 12. Training (blue curve) and validation mAPs (orange curve) of models with 0 trainable stages throughout Epochs 0-25 with learning rate (LR) changes after Epochs 15 and 21 trained on 100% (a), throughout Epochs 0-51 with LR changes after Epochs 31 and 43 trained on 50% (b), and throughout Epochs 0-103 with LR changes after Epochs 63 and 87 trained on 25% (c) of the COCO dataset.

consists of at least 9 convolutional layers. The vast difference in the structure between the first and the other stages seems to cause inconsistency in the training time they consume.

Although the training times seem to decrease when the first stage is trainable, we conjecture that this phenomenon is due to random variance observed in the previous experiments, as illustrated in Figure 8b. Therefore, as such a decrease is deemed statistically insignificant, we will not draw any conclusions based on this observation.

Hence, it is likely that, except for the first stage, the more trainable stages a ResNet-50 network includes, the longer a training job takes.

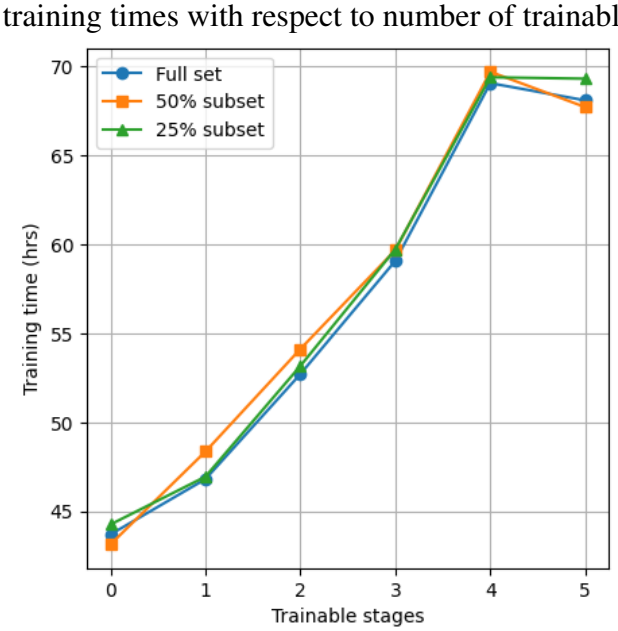


Figure 13. Model training times with respect to 6 scenarios of 0-5 trainable stages when trained on 100% (blue curve with circle marker), 50% (orange curve with square marker), and 25% (green curve with triangle marker) of the COCO dataset.

5.5 Learning Rate Decay

All the models in our experiments benefited from the first learning rate decay, as shown in Figure 14, resulting in an obvious increase in mAP. Table 2 details these increases.

Table 2. Increase in mAP at the first learning rate decay in each scenario.

Dataset	**mAP gain (pp) with trainable stage(s)**					
	0	**1**	**2**	**3**	**4**	**5**
Full	3.5	3.4	5.3	6.3	6.1	6.0
50% subset	2.3	2.6	3.6	4.3	4.6	4.7
25% subset	2.7	1.8	4.1	3.3	3.6	3.5

It is a common practice to change the learning rate so that the gradients descend in a way such that the model improves by taking increasingly smaller steps towards minimising the loss function [55, 57], of which Figure 15 gives an illustration. The learning rate typically starts from a large one and then decays multiple times, which is observed to enhance optimisation and generalisation in previous experiments [57].

The increases in mAP in the outcomes of our experiments appear to agree with

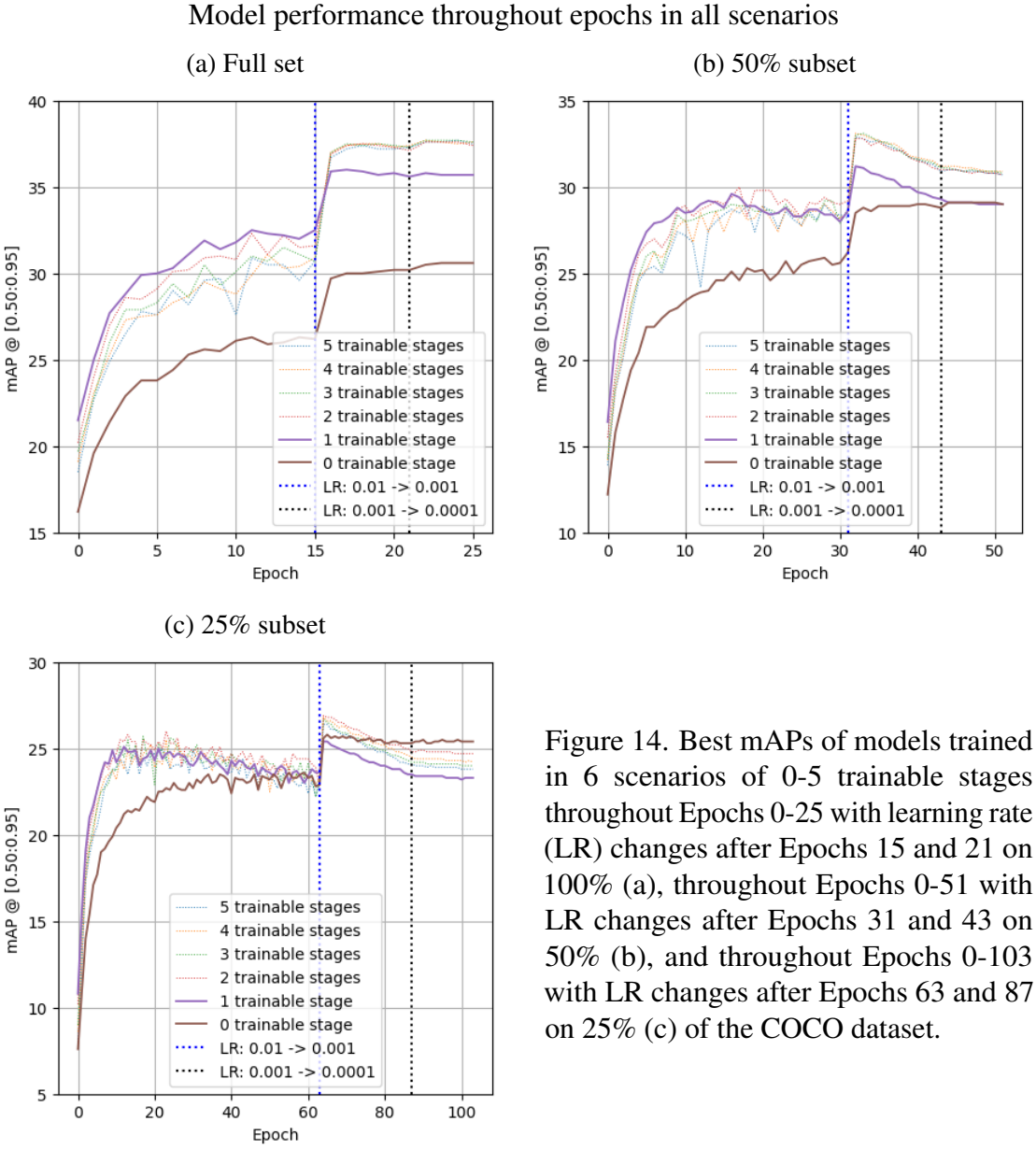


Figure 14. Best mAPs of models trained in 6 scenarios of 0-5 trainable stages throughout Epochs 0-25 with learning rate (LR) changes after Epochs 15 and 21 on 100% (a), throughout Epochs 0-51 with LR changes after Epochs 31 and 43 on 50% (b), and throughout Epochs 0-103 with LR changes after Epochs 63 and 87 on 25% (c) of the COCO dataset.

the advantages of learning rate decay. However, such gains varied between different *trainable stage scenarios*. Within each *data amount scenario*, the learning rate change improved the models with 0 or 1 trainable stage less significantly than the ones with more trainable stages, which was indicative of a smaller effect of reducing the learning rate. This is likely due to the smaller number of active parameters that adapt to the training dataset when a model possesses fewer trainable stages.

Meanwhile, it is also observable in Figures 14b and 14c that the model performance

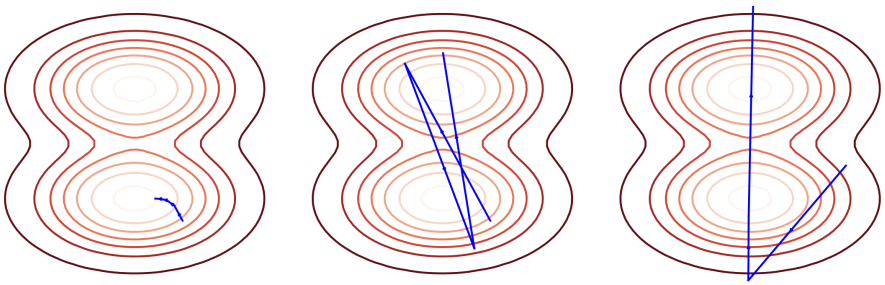


Figure 15. An explanation of gradient descent. From left to right: 1) learning rate is small enough to capture a minimum; 2) slightly too large so that it hops between minima; 3) way too large for convergence [57].

on the 25% and the 50% subsets worsened more rapidly after the first learning rate decay than before, most likely due to overfitting as discussed in Section 5.3.1, which was not seen on the full set in Figure 14a. Figure 16 presents a closer look with a comparison of performance between models with 3 trainable stages as an example. As mentioned in Section 4.3.3, the learning rate decayed at different epochs in different *data amount scenarios* in order to maintain the same proportion of epochs trained at a certain learning rate as in the training recipe, i.e. approx. 61.5% of the epochs were trained at the learning rate of 0.01, 23.1% at 0.001, and 15.4% at 0.0001. Although all three models initially benefited from the first learning rate decay, the mAP decreased swiftly on the 25% and the 50% subsets as opposed to a slight increase on the full set. The second learning rate decay further improved the models on the full set by a limited margin, which was indicative of model convergence since this decay had close to no effect and hence more decays would unlikely be beneficial. However, the second learning rate decay only seemed to decelerate the overfitting on the 50% and 25% subsets without facilitating any performance improvement.

You et al. [57] put forward an explanation that a model does not memorise noisy data when trained at a large learning rate, whereas a small learning rate concentrates the model on learning complex patterns. The outcomes in our experiments seem to correspond to this explanation. As the learning rate decayed, the smaller details, which could be regarded as noisy data in the case of a small training dataset as it was not sufficient to help the model generalise, started fitting into the model and eventually led to a severer overfitting problem.

5.6 Pre-trained Weights

The outcome of the experiment with no pre-trained weights and all stages remaining trainable was compared to the performance of the model with the same input data amount

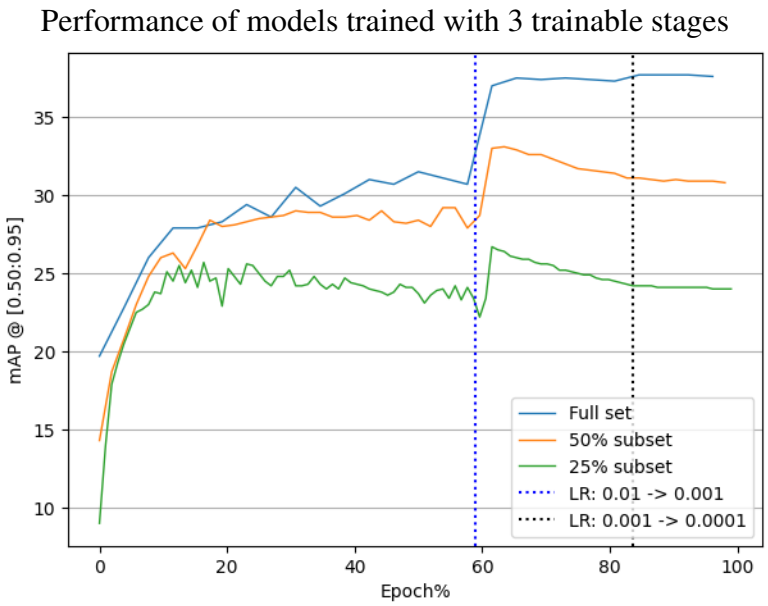


Figure 16. Best mAPs of models trained with 3 trainable stages throughout the epoch% on 100% (blue curve), 50% (orange curve), and 25% (green curve) of the COCO dataset with learning rate (LR) decays after epoch% of approx. 59% and 84%. LR decay lines do not reflect the exact timings of LR decay in the experiments due to rounding errors.

and trainable stages, as illustrated in Figure 17.

The mAP of the model without pre-trained weights initially landed at 4.5%. The model was trained extensively in order to allow it to stabilise before the learning rate was reduced, though the mAP fluctuated between 25% and 28% for 30 epochs from Epoch 20 onwards. After Epoch 49, the learning rate was reduced by 90% from 0.01 to 0.001. Consequently, the mAP rose to 52.2% and remained stagnant, despite another learning rate decay after Epoch 57. The mAP peaked at 32.5% at Epoch 66.

Meanwhile, the model with pre-trained weights was trained in accordance with the training recipe. Its mAP landed at 18.5% initially, increased steadily, and peaked at 37.7% at Epoch 25 after a surge most likely caused by the first learning rate decay after Epoch 15.

As explained in Section 2.3, when pre-trained weights are used, a model has the ability to leverage the knowledge acquired from training on a typically large dataset, especially in the domain of computer vision and image processing [32]. Particularly, a deep convolutional neural network (CNN) pre-trained on a large dataset such as ImageNet is very useful to initialise a model, even if it is intended for a different task [35]. Oquab et al. [39] also achieved a better-performing model with transferred weights of a CNN than the start-of-the-art models at the time.

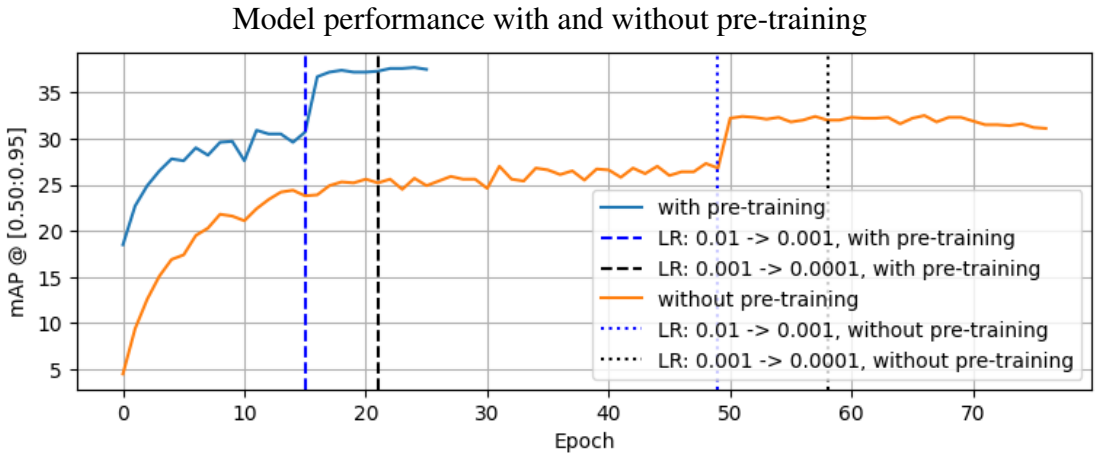


Figure 17. MAP of models trained on the full COCO dataset throughout the epochs. Blue solid curve: with pre-training, learning rate (LR) decays after Epochs 15 (blue dashed line) and 21 (black dashed line). Orange solid curve: without pre-training, LR decays after Epochs 49 (blue dotted line) and 57 (black dotted line).

The comparison of the experiment results seemingly agrees with the findings from the previous work, as it appears that when trained from scratch (without pre-training), a model takes longer to converge and also converges at a lower mAP than when trained with pre-trained weights.

6 Conclusions

This work examines how the input dataset size and the proportion of trainable stages in the backbone network affect the performance and the training time of a state-of-the-art object detection model, Faster R-CNN ResNet-50 FPN.

We show that the overfitting problem becomes severer as the input dataset size decreases. Such an overfitting effect is mainly due to the feature extraction by the backbone network of the model, as shown in Section 5.3. Additionally, freezing residual stages in a ResNet-50 network saves training time (except for the first stage). We also show in Sections 5.3 and 5.4 that although having more trainable stages enables more improvement in the model performance, such an advantage reduces on a smaller input dataset, for which more stages may be frozen in order to save computational power and training time, whereas, on the contrary, a larger input dataset benefits from fine-tuning more stages, which eventually leads to a better-performing model. Moreover, we show in Section 5.5 that while learning rate decay helps with the gain in model performance, it can also result in a severer overfitting problem in the case of a small input dataset. Furthermore, we show that compared to applying pre-trained weights, a model trained from scratch, even with an as big dataset as COCO, takes longer to converge and converges at a worse performance, demonstrating the advantage of transfer learning, as discussed in Section 5.6.

Although more experiments were initially intended in this study, we were not able to achieve them due to time constraints (each execution took approx. 57 hours on average). However, the conducted experiments have allowed us to quantify the intuitive premises made at the end of Chapter 1 for this particular benchmark dataset and suggest that, in fact, it may be feasible to save computational and data-gathering resources in financially constrained industrial applications without major losses in model performance. Nevertheless, these conclusions are, up to this point, particular to this experiment setup, and hence, to be made more generalisable, there are several ways in which this work can be improved or expanded in terms of future works:

1. Experiments on more datasets. Although COCO is widely used for object detection tasks, other datasets, especially those from real business cases in the industry, could help further verify how the findings in this study generalise to other datasets.

2. Experiments on more backbone networks. The same Faster R-CNN architecture with shallower backbone networks (e.g. VGG-19, ResNet-34) and deeper ones (e.g. ResNet-101) could shed more interesting comparisons to investigate if our findings are common with different backbones.

3. Experiments on more architectures. Different architectures (e.g. R-FCN) with ResNet-50 as the backbone network could be beneficial to understand whether our conclusions about ResNet-50 remain consistent.

References

[1] Hussein Abdel-Jaber, Disha Devassy, Azhar Al Salam, Lamya Hidaytallah, and Malak EL-Amir. A review of deep learning algorithms and their applications in healthcare. *Algorithms*, 15(2), 2022.

[2] Ali Borji. Complementary datasets to coco for object detection. *arXiv preprint arXiv:2206.11473*, 2022.

[3] Karanbir Singh Chahal and Kuntal Dey. A survey of modern object detection literature using deep learning. *ArXiv*, abs/1808.07256, 2018.

[4] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Carlos A Ferreira, Tânia Melo, Patrick Sousa, Maria Inês Meyer, Elham Shakibapour, Pedro Costa, and Aurélio Campilho. Classification of breast cancer histology images through transfer learning using a pre-trained inception resnet v2. In *Image Analysis and Recognition: 15th International Conference, ICIAR 2018, Póvoa de Varzim, Portugal, June 27–29, 2018, Proceedings 15*, pages 763–770. Springer, 2018.

[8] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. *Advances in neural information processing systems*, 28, 2015.

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[10] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[11] Dongfang He, Jiajun Wen, and Zhihui Lai. Textile fabric defect detection based on improved faster r-cnn. *AATCC Journal of Research*, 8(1_suppl):82–90, 2021.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[15] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[16] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510, 2018.

[17] Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2):37–50, 1912.

[18] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[19] Faris Kateb, Muhammad Mostafa Monowar, Md. Abdul Hamid, Abu Ohi, and M. Ph. D. Fruitdet: Attentive feature aggregation for real-time fruit detection in orchards. *Agronomy*, 11:2440, 11 2021.

[20] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53:5455–5516, 2020.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[22] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[23] Jiann-Der Lee, Jong-Chih Chien, Yu-Tsung Hsu, and Chieh-Tsai Wu. Automatic surgical instrument recognition—a case of comparison study between the faster

r-cnn, mask r-cnn, and single-shot multi-box detectors. *Applied Sciences*, 11(17), 2021.

[24] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: Design backbone for object detection. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 339–354, Cham, 2018. Springer International Publishing.

[25] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[27] Tsung-Yi Lin, Genevieve Patterson, Matteo Ruggero Ronchi, Yin Cui, Michael Maire, Ross Girshick, and Piotr Dollár. Common objects in context. `https://cocodataset.org/#download`. Accessed: 2023-03-13.

[28] Tsung-Yi Lin, Genevieve Patterson, Matteo Ruggero Ronchi, Yin Cui, Michael Maire, Ross Girshick, and Piotr Dollár. Common objects in context. `https://cocodataset.org/#detection-2017`. Accessed: 2023-03-13.

[29] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[30] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[31] Mohamed Loey, Gunasekaran Manogaran, Mohamed Hamed N Taha, and Nour Eldeen M Khalifa. Fighting against covid-19: A novel deep learning model based on yolo-v2 with resnet-50 for medical face mask detection. *Sustainable cities and society*, 65:102600, 2021.

[32] Jie Lu, Vahid Behbood, Peng Hao, Hua Zuo, Shan Xue, and Guangquan Zhang. Transfer learning using computational intelligence: A survey. *Knowledge-Based Systems*, 80:14–23, 2015. 25th anniversary of Knowledge-Based Systems.

[33] TorchVision maintainers and contributors. Fasterrcnn_resnet50_fpn. https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html. Accessed: 2023-03-13.

[34] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision, 2016.

[35] Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. Question answering through transfer learning from large fine-grained supervision data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 510–517, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[36] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of big data*, 2(1):1–21, 2015.

[37] Alam Noor, Yaqin Zhao, Anis Koubaa, Longwen Wu, Rahim Khan, and Fakheraldin Y.O. Abdalla. Automated sheep facial expression classification using deep transfer learning. *Computers and Electronics in Agriculture*, 175:105528, 2020.

[38] David Olson and Dursun Delen. *Advanced Data Mining Techniques*. Springer Berlin, Heidelberg, 1 edition, 01 2008.

[39] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1717–1724, 2014.

[40] 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)*, pages 237–242, 2020.

[41] A Sai Bharadwaj Reddy and D Sujitha Juliet. Transfer learning with resnet-50 for malaria cell-image classification. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0945–0949. IEEE, 2019.

[42] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[44] Edmar Rezende, Guilherme Ruppert, Tiago Carvalho, Fabio Ramos, and Paulo de Geus. Malicious software classification using transfer learning of resnet-50 deep neural network. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1011–1014, 2017.

[45] Rahmad Sadli. Coco dataset: A step-by-step guide to loading and visualizing. `https://machinelearningspace.com/coco-dataset-a-step-by-step-guide-to-loading-and-visualizing`, Feb 2023. Accessed: 2023-04-25.

[46] Nermin Samet, Samet Hicsonmez, and Emre Akbas. Houghnet: Integrating near and long-range evidence for bottom-up object detection. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*, pages 406–423. Springer, 2020.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Zar Nawab Khan Swati, Qinghua Zhao, Muhammad Kabir, Farman Ali, Zakir Ali, Saeed Ahmed, and Jianfeng Lu. Brain tumor classification for mr images using transfer learning and fine-tuning. *Computerized Medical Imaging and Graphics*, 75:34–46, 2019.

[49] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[50] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312, 2016.

[51] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[52] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104:154–171, 2013.

[53] Francisco Utrera, Evan Kravitz, N Benjamin Erichson, Rajiv Khanna, and Michael W Mahoney. Adversarially-trained deep nets transfer better: Illustration on image classification. *arXiv preprint arXiv:2007.05869*, 2020.

[54] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022.

[55] Yuqing Wang, Minshuo Chen, Tuo Zhao, and Molei Tao. Large learning rate tames homogeneity: Convergence and balancing effect. *arXiv preprint arXiv:2110.03677*, 2021.

[56] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[57] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*, 2019.

[58] Wei Zhao. Research on the deep learning of the small sample data based on transfer learning. *AIP Conference Proceedings*, 1864(1), 08 2017. 020018.

[59] Wei Zhao, Zhiqiang Wang, and Hongda Yang. Traffic sign detection based on faster r-cnn in scene graph. In *Proceedings of the 2018 International Conference on Mechanical, Electrical, Electronic Engineering & Science (MEEES 2018)*, pages 35–42. Atlantis Press, 2018/05.

[60] Mu Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science*, 2, 2004.

[61] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking pre-training and self-training. *Advances in neural information processing systems*, 33:3833–3845, 2020.

Appendix

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Wei Zheng**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Impact of Input Dataset Size and Fine-tuning on Faster R-CNN with Transfer Learning,
   *(*title of thesis)

   supervised by Tomas Björklund and Victor Henrique Cabral Pinheiro.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Wei Zheng
09/05/2023