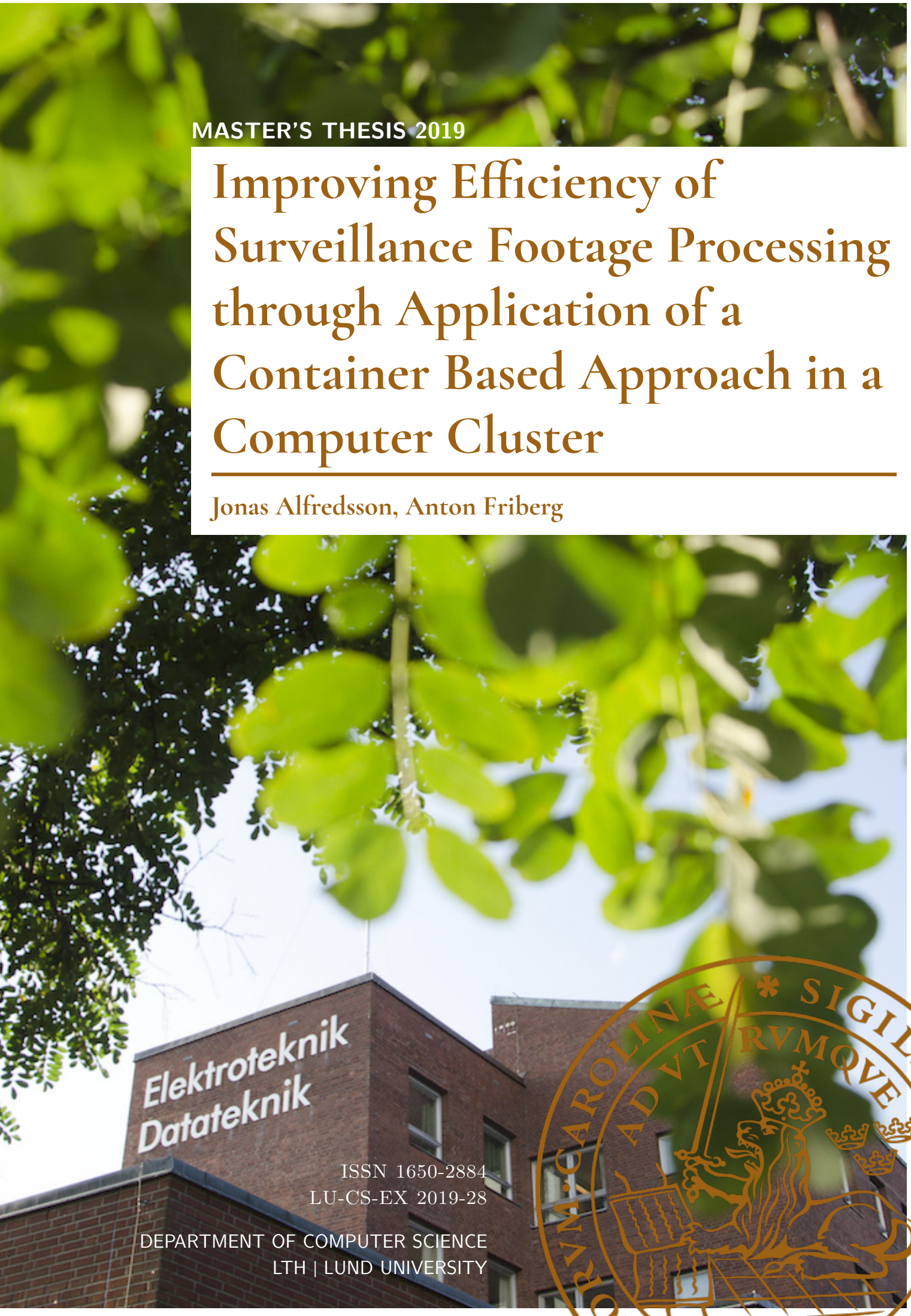


MASTER'S THESIS 2019

Improving Efficiency of Surveillance Footage Processing through Application of a Container Based Approach in a Computer Cluster

Jonas Alfredsson, Anton Friberg

ISSN 1650-2884
LU-CS-EX 2019-28

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

MASTER'S THESIS

Computer Science

LU-CS-EX: 2019-28

Improving Efficiency of Surveillance Footage Processing through Application of a Container Based Approach in a Computer Cluster

Jonas Alfredsson, Anton Friberg

Improving Efficiency of Surveillance Footage Processing through Application of a Container Based Approach in a Computer Cluster

Jonas Alfredsson
jonas.alfredsson@pm.me

Anton Friberg
anton.friberg@outlook.com

October 9, 2019

Master's thesis work carried out at Axis Communications AB.

Supervisors: Alma Orucevic Alagic, alma.orucevic-alagic@cs.lth.se
Christian Colliander, christian.colliander@axis.com
David Wessman, david.wessman@axis.com

Examiner: Martin Höst, martin.host@cs.lth.se

Abstract

Complex hardware and software solutions are often needed in order to process large amounts of media files in parallel. The complexity can make it difficult for teams to scale data processing efforts without additional support or use of cloud based solutions. However, thanks to efforts from some large software companies, like Google and Apple, and open source communities, solutions are now starting to become available outside cloud environments.

A case study has been performed at a major international video surveillance company in order to identify existing bottlenecks and to improve data processing efficiency. An object storage solution is introduced and tested alongside a container based system in an effort to improve software management and collaboration.

Finally, a proof of concept solution is presented, which utilizes a GPU cluster in order to allow easier scheduling, distribution and management of data science experiments.

Keywords: Storage Management, Containers, Graphical Processing Units, Cluster Computing, Data Models

Acknowledgements

First and foremost we would like to thank our supervisor Alma Orucevic Alagic, LTH, for providing excellent guidance and feedback regarding the content of the report, as well as general advice on how to conduct industry relevant research based on empirical data. Her passion for the subject really shone through and was reflected in the discussions throughout the project.

Further thanks are to be extended to our supervisors at Axis Communications. Johan Sandström, for helping us arrange the thesis work at the company together with providing domain expertise on development tools, together with Kenan Kule, Christian Colliander and David Wessman who granted us access to their work and provided us with the necessary hardware and software we needed to complete our research. Additionally, all persons we had to talk to at Axis in different departments have met us with great enthusiasm and interest in what our research would lead to, so we would like to thank them all.

Finally, we want to thank Flavius Gruian, at LTH, who has designed the template used in this report and Martin Höst our examiner and the author of an excellent book on how to structure a thesis work and report.

Contents

Chapter 1

Introduction

Large scale data processing requires complex hardware and software solutions in addition to large amounts of available storage space [1]. This complexity can make it harder for teams who are trying to expand their data processing efforts, particularly in the case of multimedia files [2], [3] or when implementing data-driven learning based algorithms [4], [5]. These difficulties can be due to limitations in the surrounding IT infrastructure, and to overcome these challenges it is often necessary to obtain support from a software engineering teams or, as an alternative, migrate into the locked-in services of a cloud provider [6]–[8].

Today such large scale data processing, often called Big Data Analysis, already drive many aspects of our society, including retail, manufacturing and financial services [9]. For a long time, the analysis or computational model development were made by domain experts with the help of access to the data itself. Today, however, more and more models are developed by learning based algorithms. This is evident in the areas of computer vision [10]–[12], speech recognition [13], [14], machine translation [15], [16] and bioinformatics [17], [18] among others. As a result of the value demonstrated, many companies are starting to look into how to expand their own efforts, with respect to large data collection, in order to draw knowledge or intelligence from it at a later stage. The goal is that, with large scale data analysis in place, decisions can be made from tangible information which previously were often based on guesswork [1], [4].

As such, there exists many tools and frameworks which help teams train computational models at scale by utilizing solutions from the field of large scale data analysis [19]–[21]. Similarly, solutions exists that ease the set up and operation of complex software environments [22], [23] and ease the management of data storage [24]–[28]. In combination these solutions can be leveraged in order to build complete computational systems dedicated to the single purpose of training state of the art computational models [6]–[8], [29]. Such systems have mainly been developed and used by large international data-driven software companies (i.e. Google,

Microsoft, Apple, Uber and others) that have the cross-domain expertise that is required in order to maintain all the hardware and software components. Unfortunately such complete systems are often out of reach for teams that do not have the expertise, or are not yet operating at a scale where dedicated systems can be economically motivated. Additionally, there seems to be a lack of documented solutions into the middle ground, i.e. how to move from small scale manual model development towards automated system solutions incrementally.

A real-world model development system needs to facilitate efficient training of computational models at scale. This requires significant infrastructure engineering in order to handle the changing software requirements, large amounts of required training data and tight dependency on hardware, such as graphical processing units (GPUs), across a growing number of compute nodes. As a consequence, of not having any documented solutions on how to scale up efforts towards systematic model development, it is common to see teams grow into large amounts of technical debt. This can result in considerable effort having to be spent maintaining both the computational and storage infrastructure, greatly reducing the pace at which models can be developed [30].

This report focuses on identifying infrastructure problems related to the training of computational models for problems in the field of computer vision, and explores a variety of possible solutions. The work has been carried out as a case study, at a large international company, in collaboration with a small team that is experiencing difficulties expanding their training efforts. Further details, about the case company, is later presented in Section 2.4. In this report we are not addressing the algorithmic or computational challenges of training computational models, how to distribute or select the model itself or select the best model parameters [5], [31]–[33]. Instead, the focus will be on how to move beyond development on a single machine, towards running experiment jobs on a scalable compute cluster i.e. how to manage compute and data resources, software and dependencies and how to make the experiments easy to monitor while also making the system flexible to allow diverse use-cases.

Previous research, which have looked at the infrastructural challenges, have mainly been looking at a much larger scale [21], [29], [34] or utilized cloud environments [6]–[8]. Instead, this paper will be exploring cost effective scalable solutions from the open source community, that can be deployed on-premise, and already available solutions from within the company. In order to evaluate our work a deep learning example is developed which trains different image classification models in a predictable matter. This example is then utilized to evaluate possible solutions on how to alleviate the problems experiences by the team and serves as a good representation of the kind of model development that is carried out in the daily operation.

The report is structured as follows. In Chapter 2, Background, further details about both the case company and their challenges is given. In Chapter 3, Research Questions and Methodology, we list the research questions and methodology of choice. Following this in Chapter 4, Analysis, the description and evaluation of the current workflow will be presented, to then iterate through possible improvements in Chapter 5. Finally, we present our thoughts and final results within the Discussion and Conclusion chapters.

Chapter 2

Background

2.1 Data Processing Methods

Data processing, or more specifically electronic data processing, is generally defined as the aggregation and manipulation of data by a computer in order to produce new meaningful information [35]. For example, a common operation would be to predict sale volumes of different products from historic financial data. In many cases it may not be feasible for a human to sift through all the available information in a reasonable amount of time, and often the operations performed on the data is suitable for a computer. Such computer applications would then provide the company with valuable indications on how it may set its current price in order to gain optimal profits, or predict how many units to manufacture in preparation for the holidays.

Initially the type of data being analyzed was comprised almost entirely of large amounts of numerical records, such as in accounting and inventory software, since this type of data is easy for computers to process. Later, the success of these early systems paved way for the continued development of both compute power and storage, which in time made it possible to process other forms of information. As a result of those advancements it is now possible to perform extensive data mining on even the most complex forms of data, such as images and video. [36], [37]

At the same time, information gathering devices and storage solutions has continued to decrease in cost, allowing the rate of data collection to grow rapidly. A consequence of this is the increasing complexity of the processing techniques, which is motivated by a need to keep up with the vast amounts of incoming data. These developments were later summed up in a new concept, called *Big Data* [1], which has not yet reached a consensus regarding its exact definition. However, it is most commonly used when referring to manipulation of sets of data

that are considered too large or complex for traditional data storage and processing methods to handle. Such datasets require significant infrastructural engineering to accommodate all the incoming information, and to allow processing within a tolerable amount of time. This has all been particularly true for multimedia data formats, such as images and video, which do not fit within the traditional processing methods easily. The heterogeneity of such data proved to be the main cause of difficulty, as it did not conform with the storage model used up until this point [2], [3]. This growing need for computational power and surrounding infrastructure leads towards further parallelization of computing, as more and more applications outgrow single tenant solutions.

2.1.1 Methods for Data Storage & Retrieval

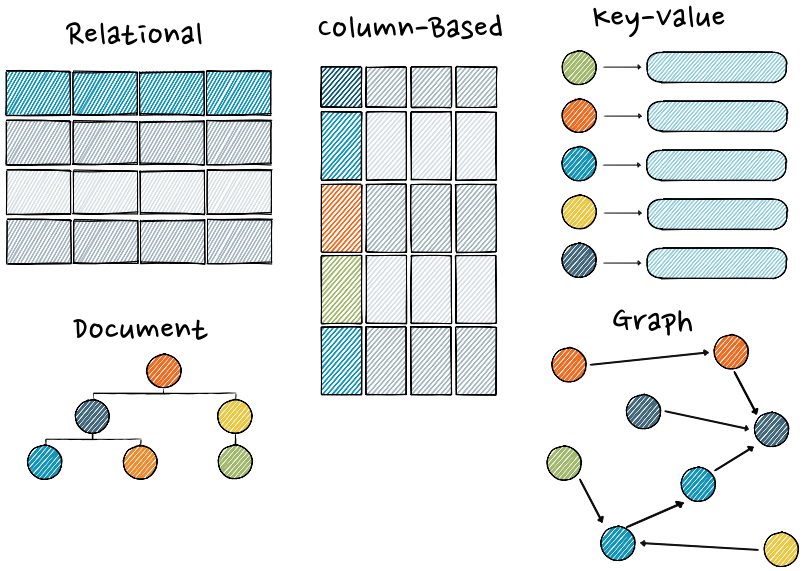


Figure 2.1: Different kinds of data store models.

As increasing amounts of data is consolidated into centralized storage systems, the data processing systems also evolve into new forms. A major driver is the fact that real economic value can be extracted by connecting different data points which previously were spread across multiple systems. These new *Big Data* systems are what put particular high demands on the infrastructural requirements, since the amount of data to be processed was suddenly many times larger than the normal day-to-day operations. Similarly, companies realize that the size of the data itself makes it difficult and expensive to store and sync across multiple locations, which in turn lead to more data being transferred around in the system [1].

Traditional data processing solutions make use of database systems with data structured in fixed formats or fields, often in tables. These started out as spreadsheet applications and later developed into relational database systems with extensive querying capabilities. In general, as long as the data was homogeneous and structured, these solutions worked well and were able to keep up with the increasing amounts of data by simply utilizing more powerful hardware, such as multiprocessor architectures and solid state storage drives. However, to scale beyond a single machine proved more problematic since relational databases are known to be difficult to manage in a distributed setting, which still remain true today [36], [38]–[41].

At the start of the 21st century, alternative data storage and processing techniques appeared, which aimed to offset the limitations experienced with the traditional methods, i.e. the difficulty of handling unstructured data and scale beyond single node deployments. The alternative database solutions, called *NoSQL* in contrast to the relational models' querying language *SQL*, explored different non-relational data models. The data stores that emerged where column-oriented databases [42]–[44], graph databases, key-value stores [45], and document-/object-based stores. These data stores are visualized in Figure 2.1.

Due to significant differences from the relational databases, these alternatives were at first only adopted by users with specific use cases, but later managed to grow into more general adoption [41]. In time, these solutions proved to handle unstructured and heterogeneous data much more efficiently, and also showed better suitability for distributed environments [40]. Today, when discussing *Big Data*, it is often these kinds of storage solutions that are leveraged in combination with advancements in distributed file systems, such as the *Google File System* [24] and the *Hadoop Distributed File System* [25]. Most importantly, it was these advancements in storage models that made way for the large scale data processing that we see today.

2.1.2 Processing Data

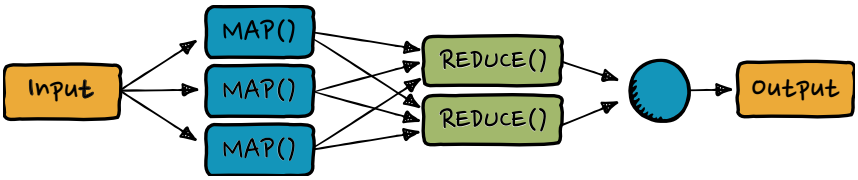


Figure 2.2: Drawing showing the main tasks of the MapReduce programming model.

In similarity to the development of the storage of data, advances have been made in processing of the large datasets. Data processing is a task that includes the structuring and filtering of collected information, which is an important part of gaining knowledge and insight. Early systems utilized strict methods that cleaned, loaded and refreshed incoming data in a controlled fashion in order to limit difficulties processing the data later in the pipeline [46]. Such architectures where scalable, but hard to maintain since they required very intricate knowledge of each site's configuration in order to avoid sever communication, processing and memory bottlenecks.

These solutions were also often difficult to integrate with new data inputs, and a need arose for solutions that could handle vast amounts of unstructured data without needing to change the already established systems. In response, a new programming model was developed at Google, facilitating easier processing and generation of large data sets, called *MapReduce* [47] and shown in Figure 2.2. This new programming model allowed programmers to more easily utilize parallel and distributed system resources. For many tasks this new approach lead to processing times decreasing by significant amounts [48]. Interestingly, this model was presented just one year after Google had introduced its distributed file system and depends on the functionality introduced by it.

2.1.3 Data-Driven Computing

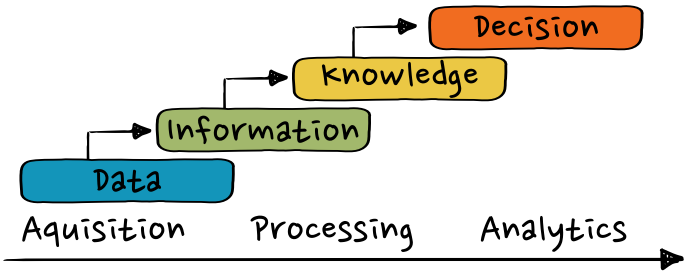


Figure 2.3: The transformation of data into decision inside a large data processing systems.

In parallel with the progress made with large scale data processing, there was also the arrival of data-driven deep computing which utilized advanced learning-based algorithms to be able to make knowledge based decisions [4], [49]. This include machine learning and deep learning, and while such algorithms are not the focus of this thesis, it is still important to show how these differ from other forms of data processing to be able to understand how they affect the entire system and surrounding infrastructure. An overview of the steps involved in taking the data and transforming it into decisions is presented in Figure 2.3.

Learning-based models where first conceived in the late 1950s, but did not see wide range adoption until recent years when an increase in computational power, large scale data processing and commercial interest merged with increased algorithmic solutions and large academic interest [49]. In combination, they proved to be a powerful way to extract actual knowledge, intelligence and even decisions from large and complex sets of data.

In particular, deep learning-based models quickly grew into the most effective learning-based approach for multiple application domains including, but not limited to, computer vision [10]–[12], speech recognition [13], [14], machine translation [15], [16] and bioinformatics [17], [18]. These algorithms and more general machine learning techniques enable computers to learn by example in a similar fashion to humans, and with enough available data some models, such as one for traffic sign classification, have even managed to demonstrate accuracy comparable to, or exceeding, those of human experts [50].

Deep learning algorithms gain their effectiveness by utilizing artificial neural networks to learn how to perform a task by analyzing large amounts of training examples. Increasing the amount of training data, and/or creating models with larger neural networks, is often the most effective way of increasing prediction accuracy. Unfortunately, this also means that as the complexity of the tasks increases, so does the size of the neural network. Larger neural networks generally require more training data than smaller models and also takes significantly longer for each training step. This can produce additional challenges when trying to implement these algorithms as solutions to complex problems, such as driving a car or analyzing multimedia files, at scale [3], [5], [19]. Figure 2.4 shows a very simplified process of how a neural network identifies the contents of a picture from features it has learned are important.

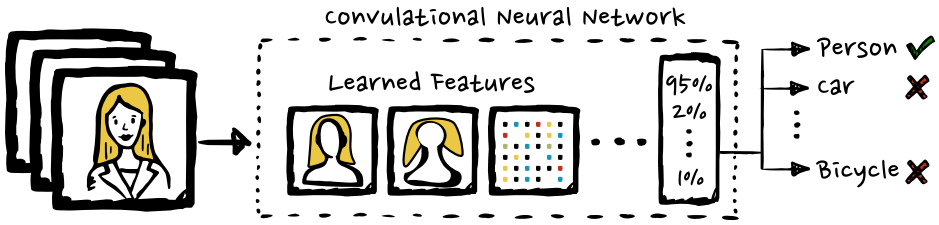


Figure 2.4: Overview of how deep learning models extract features in order to make a decision.

The challenges involved can be divided into distinct parts [29]. Firstly, the networks pose a significant computational challenge during training and, in some cases, even during application on new data [31]. This means that large amounts of raw computational power is needed. Secondly, they pose an algorithmic challenge for the reason that when the workload is broken apart into small batches, the model's ability to generalize often degrades [51] and the additional compute nodes add significant communication overhead [21]. Lastly, a real-world deep learning system at scale requires a significant infrastructure engineering effort to facilitate the ever-changing software requirements among the distributed compute nodes, the transport and storage of large amounts of training data, and the often tight dependency on specific hardware requirements such as graphical processing units (GPUs). These all contribute towards technical debt as the system scales up [30], [52].

Even though the above mentioned challenges can be reduced by applying certain techniques [5], [29], [30], they still pose a large obstacle for small to medium-sized teams looking to scale up their efforts into data-driven computing. This is true for all kinds of large scale data processing systems [3], especially for those handling complex forms of data such as multimedia files, e.g. images and video. For these forms of data the infrastructural challenges are particularly prominent [3].

2.1.4 Multimedia Processing Complexities

Figure 2.5: Common examples of multimedia data formats.

Multimedia data formats, examples displayed in Figure 2.5, have been proven to be particularly challenging for both *Big Data* analysis and as training input for learning-based models [2], [3]. As such, the difficulties needs to be taken into careful consideration during the entire thesis work since the team currently work with images and wants to move into the processing of video files.

Multimedia data is unstructured and heterogeneous by nature and, in the case of video with sound, even multimodal. Additionally, the handling of large amounts of multimedia data also entails the need for considerably more resources in order to acquire, store, transmit, present and process the data, which includes, in some instances, a need for a GPU. Even with application of compression and efficient storage techniques the volume of media files takes significantly more space than storage of text or numerical files. In most real-world scenarios it is simply not possible to store all the generated multimedia data for long. [2], [3]

In addition to the difficulties in managing the characteristics of the multimedia data itself, there is also a lot of problems surrounding the mere handling of the data due to the sometimes sensitive nature of the information. Audio recordings, pictures and recorded video is much more regulated than traditional forms of data, and handling it comes with both security and trust concerns. This is especially true within the European Union after the introduction of the General Data Protections Regulations [53] (GDPR), which require companies to facilitate a process where a user may demand the removal of any personally identifiable information that is stored [53]. These additional legal considerations limit teams ability to move into cloud based solutions and affects the choice of solutions throughout this thesis.

2.2 Deep Learning Workflow

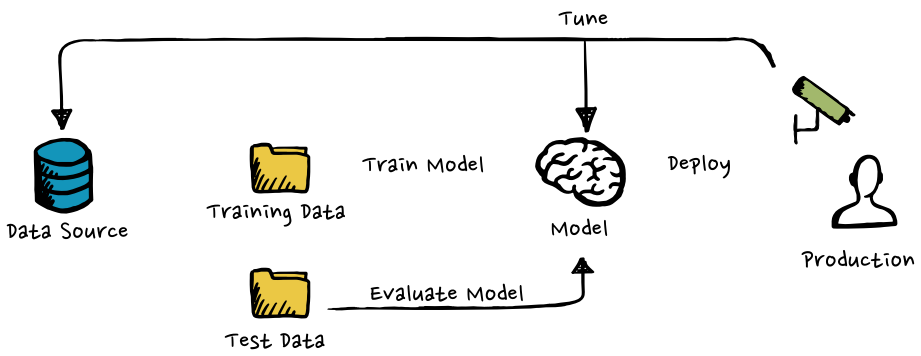


Figure 2.6: A diagram showing a generalized deep learning workflow.

Deep learning model development differs significantly from the development of traditional mathematical models and is usually divided into three distinct steps: *Preprocessing*, *Training* and *Validation*. However, alongside preprocessing there is also the matter of storing the data, which is to be processed, persistently. This is a separate problem which will be touched upon further in Section 2.8. An overview of the development life cycle is visible in Figure 2.6. Data is collected by for instance a camera and put into storage, which is then processed before being used as training data for the deep learning model. The model is then trained and evaluated on separate batches of data and after sufficient training the resulting model is validated. The model is then either put into production or discarded depending on how it performs.

2.2.1 Preprocessing

The part of the deep learning workflow denominated as preprocessing, is a broad definition which includes multiple different procedures that are applied to the data before it is used to train the actual algorithms [54]. Because of this broad definition there does not seem to be a consensus of what is specifically included, as it may be interpreted to encompass tasks from the very early stages, when data is collected and assembled to a dataset, to slight alterations done within milliseconds before being used in the training step [54]–[57].

2.2.1.1 Data Collection

Normally the first step within data preprocessing involves the collection of data. Here it is important that the data provides an honest representation of situation that the model will be expected to act upon. What this means is that if the model is intended to be able to recognize human faces, training it on a collection of pictures depicting only vehicles will yield unsatisfactory results [54].

The data may be newly created for this particular training, or can make use of previously prepared large collections of images. However, these may require removal of invalid data, adding missing data and/or structuring it to conform to the desired annotation method of the learning algorithms used [55], [56]. This structure varies between different implementations, but is required by the algorithm for it to be able to successfully learn the desired features. In Figure 2.7 one method of annotation is illustrated, where the objects of interest are highlighted by having the coordinates of the colored boxes written in an accompanying text file.

Figure 2.7: Annotating the input data may be done by placing coordinates around the parts of the image that are of importance.

These preparatory steps are usually only performed once, and the remaining structured data is what is stored, in a persistent manner, as the dataset that is to be used during training. How it is stored is not a concern of the preprocessing step, but, depending on the type of data in use, there may exist many solutions which can vary in how efficiently it is able to save the data for later retrieval.

2.2.1.2 Data Augmentation

One might think that the most accurately annotated training data results in the most accurate models. However, it has been shown that the amount of input given to the algorithms have significantly greater impact than the quality of it [34]. Researchers [58] trained a convolutional neural network, specialized in fine grained recognition (i.e. identify species of birds), firstly on a well annotated set of data and then again after having extended this set by adding "noisy" data.

Noisy data, in this regard, is data that has not been verified to actually contain what it is annotated as. In this particular case the researchers used a Google image search for the all the different species included in the well annotated set, and then downloaded all the results without confirming that they actually contained the correct species of bird (between 35% and 90% of the images were actually of the subject depending on the search). The result was an increase in accuracy of the trained model from 84% to 93%, which is close to how well human experts perform [59].

As these algorithms prefer volume over quality of data, research has been conducted into efficient methods of extending datasets that already exist. Advanced solutions for this is available, such as using Generative Adversarial Networks [60], but those remain outside the scope of this report. However, the following three simple methods of alterations, also illustrated in Figure 2.8, are enough to significantly extend image datasets and increase the accuracy [61], [62]. The effectiveness of the data augmentation methods are well-known and all three of the mentioned augmentations below are utilized by the case company in combination with others when needed. The data augmentations are particularly important for deep learning techniques since the augmentations increase the trained models' ability to generalize to unseen data and makes it more robust against over-fitting [54], [61], [62].

1. **Cropping**
   Cropping images needs to be done with care, as to still retain the object of interest inside the remaining area. If the images are small, cropping to make them smaller might cause a problem for the network as it expects a minimum size of the input. Hence, there are methods that keep the crop size the same as the original picture size, but move the picture and pad the empty area using the average color of the image or more content aware techniques [63].

2. **Rotating**
   Rotation of pictures is done to make networks able to identify objects that are not perfectly aligned. Having rotational diversity is desirable, however, it is important to keep the rotation parameters relevant for the data to be analyzed. A 180 degree rotated picture of a stop sign might not be of relevance for a self-driving car, and a picture of the number "9" will create confusion as it is dangerously similar to the number "6".

3. **Flipping**
   As in the case of rotations, flipping will also help in generalizing the identification of objects, but care needs to be taken to consider how the direction affects the meaning of the input. A horizontal flip of the letter "b" might be identified as a "d".

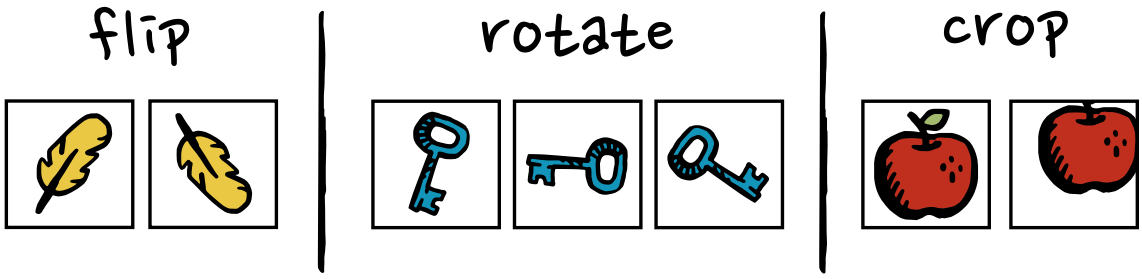


Figure 2.8: Examples of preprocessing augmentations used to artificially inflate a dataset.

These alterations may be done at the same time as the preparation of the dataset, and then stored to maintain a static setup of any changes that has been made. However, this requires additional disk space which may not be available. Therefore, some deep learning frameworks have built-in methods that allow for the above mentioned transformations to be performed when the image is being read from the long term storage. As this is performed just before the neural network is allowed to learn from it, it is still considered preprocessing.

2.2.2 Training

Entering the step where the training of the network is to be performed, the need for deeper knowledge about the actual algorithms becomes prominent, however this is not within the scope of the thesis. This description will instead remain at a high level, by not emphasizing the type of neural network that is trained. The reason for this is that neural networks exist in many shapes and forms that have different areas where they perform optimally. It may be possible to train a general image classifier network to differentiate between trucks and cars, but it can not identify which brand it is. Other, more specialized networks, may then be able to tell you what brand a car is, but have no understanding of objects beyond the car brands it knows. The exact science behind these networks are beyond the scope of this thesis work, and the network used will be mostly irrelevant for the training process as it is explained here.

Nevertheless, there is usually a large amount of code associated with the training of the neural network model. To alleviate the necessity for everyone to write their own unique code, a number of frameworks have been implemented to abstract many of the more tedious parts of training, and make it easier to set up a new experiment session. These frameworks are mostly agnostic to the neural network being used and make it easy to ingest different forms on input data with only minor alterations to the underlying code.

These frameworks are mainly responsible for loading the data, which the network is to be trained on, from persistent storage into memory and apply augmentations to improve the results. It is also possible to tune the hyperparameters, which define how the network is learning, in an easier and more observable way through the framework. However, this is a topic complex enough to warrant its own paper and hence beyond the scope of this thesis work.

Some examples of well-known, free and open source frameworks are:

- **TensorFlow** [64]
  This is the most popular machine learning framework and is officially supported by Google. It has distributed training support, scalable production methods and support for many devices, such as Android and iOS, as well as a large established community. It provides Python and C++ implementations and has excellent support for Compute Unified Device Architecture (CUDA), allowing efficient utilization of Nvidia GPUs in order to accelerate training.

- **PyTorch** [65]
  One of the newest open source deep learning frameworks which has gained popularity due to its simplicity and ease of use. It provides both Python and C++ implementations and has excellent CUDA support.

- **Keras** [66]
  This framework is growing quickly in popularity due to its light-weight and quick implementation and its ability to make it simpler to run other deep learning frameworks such as TensorFlow. It provides only a Python implementation and has CUDA support.

- **Darknet** [67]
  A deep learning framework in the C programming language that is fast and easy to install. It is popular among embedded devices programmers since it is entirely implemented in C.

- **Caffe** [68]
  A deep learning framework implemented in C++ and Python, and developed by the University of Berkeley. It is one of the oldest and most widely supported frameworks and is popular choice to enable embedding of deep learning models into embedded devices and also provides great CUDA support.

In this thesis the primary focus will be spent on the PyTorch [65] framework, which currently have support for distributed training and GPU acceleration, and provides many excellent examples of object detection training available under MIT license. Its style of the coding is more compact and use function names that are descriptive, which will make it easier for the reader to follow the process without having to understand all the underlying mathematics. The reason for choosing this framework was mainly because it allows for quick iterative modifications, due to its simplicity, and that our supervisors at the case company already have experience from using it.

2.2.3 Validation

The training of a neural network is an iterative process which may be continued forever if so is desired. However, the goal is usually to arrive at a state where the performance of the network is deemed to be satisfactorily, and then halt any further computations. At the case company, it is currently up to the engineers to manually monitor the output metrics of the experiments, and then decide whether to complete the training based on what is presented.

The exact circumstances required for making these educated decisions is left to the engineers, and are outside the scope of this thesis report.

There are however issues with a manual monitoring processes. The close attention required to monitor the outputs means that productive hours of the days are wasted. The alternative however, is to risk degradation of the model's performance as the training continues beyond the most optimal point. Aborting too early will result in an underperforming model, while continuing to train on the same data for too long will cause it to become overfitted. What this means is that the network has iterated over the training data so many times that it remembers every single detail, and not only the identifying features. This will result in an almost perfect accuracy score when the network is analyzing the training data itself, but will perform poorly when it is to be deployed in production on unseen data.

It is therefore important to have all monitoring metrics of the training easily accessible, and presented in clearly and easily understood way, for the engineers to be able to make quicker and better decisions on how to proceed. Fully automated systems may be implemented to trigger when there are indications on when the model is about to become overfitted, and pause further computations, but human intervention will be necessary for the final verdict concerning whether to implement the model into production.

2.3 Related Work

There have been a lot of research conducted on how to handle data processing tasks in a distributed environment, however, from our extensive research it seems like it is mainly in the last four years when papers regarding how to scale the training of deep learning models have begun to surface. Unfortunately the proposed systems are often presented without details, and utilize complex infrastructure and software that needs a large team of software engineers to implement and maintain. It has only been very recently that the solutions, proposed in these systems, has become available to smaller teams with less resources [23], [29], [69], [70]. The unique infrastructural and algorithmic challenges, related to the deep learning sub field of data processing, seems to have been solved in small steps. Hence, it is not until now that any complete, and working, system solutions have been made available outside cloud environment as open source software. In the following subsections some of the incremental solutions which have contributed significant advances are presented in chronological order.

2.3.1 Adam — Distributed Training on CPUs

One of the early advocates of the need to distribute the training of deep learning models, in addition to Google, was Microsoft. Microsoft implemented a distributed system called *Adam* [19] on top of traditional CPUs without the support of GPUs. At the time, one could not utilize more than four GPUs in parallel for the training of deep learning models, which limited the size of the model that could be trained. The authors showed that larger deep neural networks increase the accuracy of the model, but require significant amounts of training data and the amount of computing cycles, to train the model, becomes proportional to the product of model size and the training data volume.

As proposed, the implemented system utilized a distribution method similar to Google's DistBelief [20] shown two years earlier which used parameter servers to allow distributed training by keeping track and propagating network parameters. This solved the challenge of needing the knowledge learned by the computations available to all without causing too much delay. In Figure 2.9a an overview is presented in how parameter servers share the calculated results from each processing unit. An interesting observation from this paper was that the models' accuracy when trained asynchronously increased rather than decreased, which was contrary to what many people believed. Unfortunately, the parameter server still needed expensive hardware to facilitate the communication overhead introduced which did not scale well above 20-30 processing units.

2.3.2 Nvidia Docker — GPUs Inside Containers

In November 2015, Nvidia open sources experimental software bindings [69] to make it easier to utilize GPUs in a container environments [71] such as Docker [22]. Container environments had been popularized years earlier and provides an efficient way of isolating applications and handling software dependencies without relying on virtualization techniques. This contribution allowed the complex software and hardware requirements of deep learning model development to be managed much easier. Although, a full release would not arrive until February 2016.

2.3.3 TensorFlow — Library for Distributed Training

In November 2015, a year after Microsoft's paper on *Adam* [19], Google open sources TensorFlow [72] which is a framework for machine learning (including deep learning) model development. It utilizes Nvidia's CUDA technology [73] to enable efficient training on GPUs, and quickly received support for NVIDIA Collective Communications Library [74] (NCCL). *NCCL* optimizes the communications between Nvidia GPUs, allowing more units to work in parallel.

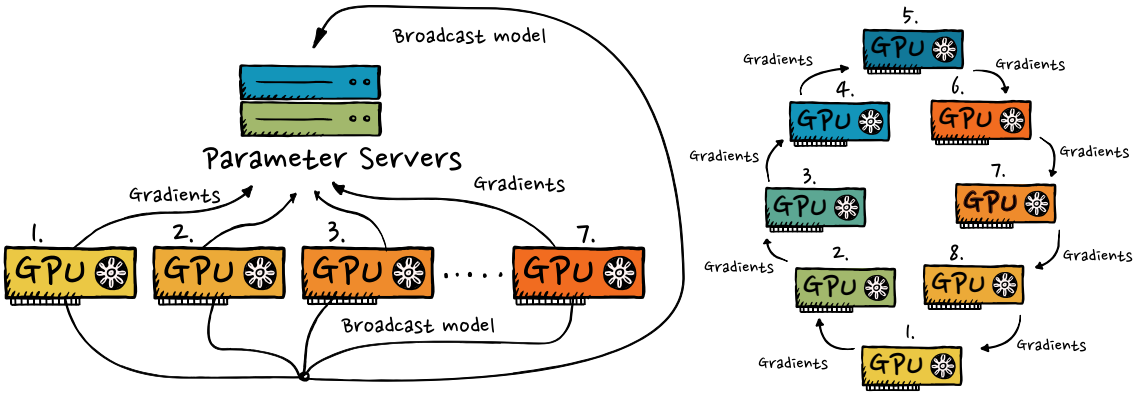
The TensorFlow library allowed for much easier development of machine learning models and was quickly adopted across both the academic and commercial sector. Unfortunately, it was still difficult to train models across multiple compute nodes due to the use of parameter servers, which required expensive network hardware due to communication overhead, and a need for major changes in the training software (the models themselves needed to be fully aware of the distribution environment).

2.3.4 Horovod — Efficient GPU Communication

Uber Technologies Inc. presented, and open sourced, its library called Horovod [21] in late 2017, which allows the training of deep learning models to scale better across multiple GPUs and compute nodes. They utilized earlier research from Facebook [32] and Baidu [75] to improve the existing distribution methods supported by the TensorFlow [64] library. The advances removed the need for dedicated parameter servers and reduced the communication

overhead by instead utilizing a strategy called ring-allreduce, displayed in Figure 2.9b. This meant that the GPU scaling improved to upwards of 70% from the earlier TensorFlow implementations which only reached around 30-50% in the best case above a certain number of nodes. Details of how this ring communication is set up is considered out of scope, but is available in the original paper [21] and in the source code they provide.

The software has, since its publication, been further developed to include distribution methods for both Keras [66] and PyTorch [65] frameworks. The library does introduce some complexity since it depends on the installation of external software and changes to the training code itself to allow distributed training. But importantly the reduced communication overhead of the ring-allreduce strategy meant that teams could integrate more nodes into the computation without needing to invest in expensive network hardware.



(a) Centralized parameter server.

(b) Horovod's ring-allreduce.

Figure 2.9: High level overview of the differences between the ring architecture used by Horovod, and the old method of using centralized parameter servers.

2.3.5 Kubernetes — Orchestration of GPU Resources

Kubernetes is a container orchestration system which aims to reduce the complexity of managing containers [71] across clusters of hosts in diverse network environments [23]. Originally released in 2015 it has quickly grown into the most popular method of managing container orchestrations in both cloud and on premise environments. At the start of this thesis work, in September 2018, the Kubernetes device plugin functionality was officially released. This allows engineering teams to much easier set up a computational cluster environments with GPUs as long as the software environment is packaged in a container environment such as Docker. The included scheduling and cluster networking features within Kubernetes finally provides a robust solution for common infrastructure problems experienced by many engineering teams which have not yet developed solutions of their own.

2.3.6 Alchemist — End-to-End Implementation

In parallel to the Kubernetes device plugin release, research by Apple Inc. discussed the design and implementation of an internal service called *Alchemist* which was built to provide easy, fast and scalable distributed deep learning training [29]. Case studies have been provided, showing how internal adoption lead to a 10X reduction in training times on very large datasets in the development of autonomous systems. The authors [29] recognize that training deep neural networks quickly, and at scale, still pose significant challenges despite their widespread adoption, and that these can be divided into computational, algorithmic and infrastructure engineering challenges.

In order to address the computational challenge they propose a method to parallelizing the computation across multiple GPUs, hosted across multiple compute instances, as the only viable scalable method since the development of hardware has not seen fast enough advancements. The algorithmic challenges have, according to the authors [29], already been mostly solved and the algorithms are able to be distributed over multiple nodes. This is possible by utilizing either the older method of parameter servers, used by the TensorFlow framework, or a more recent approach of utilizing the previously mentioned Horovod library.

The infrastructure engineering challenge is recognized as needing the most work and is further divided into four distinct areas:

1. **Interacting with compute resources:**
   As the training moves from an interactive environment to a dedicated training environment there is a need to keep track of IP addresses, starting and stopping processes on multiple machines, managing synchronization of multiple processes and handling storage space.

2. **Interacting with data resources:**
   Large datasets, that are shared and modified between users and teams, needs to be stored on something that is both performant and is easily accessible by multiple different frameworks [29]. There is also regulations related to storage of data, which companies must comply with [53].

3. **Managing software and dependencies:**
   Deep learning requires maintaining a complex software stack ranging from low-level GPU libraries (CUDA) to high level machine learning frameworks (PyTorch, Keras, TensorFlow). The required software needs to be deployed to the correct compute node and the training environment needs to be the same as the interactive environment i.e. the cluster environment should closely match the environment of the developers own workstation in order to make the software easier to debug.

4. **Availability of monitoring tools:**
   The user that sets up the training requires an ability to monitor metrics, both on the application level and the system level, in order to confidently decide when the training has completed and see how it performs.

It was to address these four infrastructural challenges that the *Alchemist* system was built. The system is portable between private and public cloud and supports multiple training frameworks and distributed training paradigms. It is managed by Kubernetes [23] and employs a container based approach [71] to handle the software environment. The storage is handled by distributed file systems and object storage depending on the size of the dataset. The distributed training is either handled by separate parameter servers or by Horovod jobs. They conclude that distribution with Horovod scales much better without needing dedicated communication nodes.

Using Alchemist the training time was able to be reduced from days down to a few hours, which significantly accelerated the development cycle of the algorithms. Unfortunately for us, the paper lacks any implementation details or runnable code, making it impossible to reproduce the results. The hardware nodes are also made up of state-of-the-art components with 64 virtual CPU cores, 488 GB of memory and 8 NVIDIA Tesla V100 GPUs that communicate over a 25GbE network, minimizing possibilities of communication bottlenecks. The paper is still interesting and relevant since it provides details regarding more aspects of the infrastructural challenges of training deep neural networks, and discusses solutions in areas often neglected, such as storage and dependency management.

2.4 Case Company

This thesis work has been carried out at Axis Communications AB [76], at their main headquarters in Lund, Sweden. The company is a market leader in network video, with its main source of revenue coming from video surveillance solutions.

Axis has utilized machine learning technology for many years in order to equip their camera technology with analytical capabilities, such as recognition of persons in a video stream to alert observers about potentially undesired activities, as well as high order of storage compression. It is within the company's interest to continue to expand and scale the efforts regarding machine learning research in order to maintain a competitive edge within the video surveillance business, especially within the subfield of deep learning.

There are currently machine learning efforts expanding at multiple departments simultaneously, using various forms of data, but the main efforts are primarily related to images and video. Due to the sensitive nature of the majority of the data being stored, it is important to be able to maintain strict control over its storage location considering recent GDPR [53] legislation. Axis has worked hard to secure compliance, but the current deep learning projects are experiencing difficulties adhering to the centralization requirements of sensitive data which removes the possibility of storing image data in multiple places.

The day to day deep learning workflow differs greatly depending on the maturity of the model implementation and the experience of the development team but an overview of the included steps are presented in Figure 2.10. However, it is common, during the early stages of a deep learning model construction, to only experiment on a small subset of the available input data. This is to quickly be able to test with different configuration options, preprocessing steps and parameter selections in an interactive environment which will return a result quickly.

This environment could be the command line of the engineer's personal workstation, inside a Docker [22] container or in a Jupyter notebook [77]. The input data is usually temporary stored locally and inspected manually to explore particular features and interpret the result. This fast changing training environment is usually referred to as *interactive training* and is displayed in Figure 2.11a.

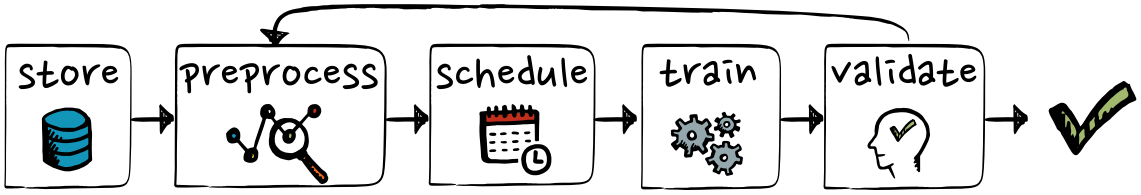
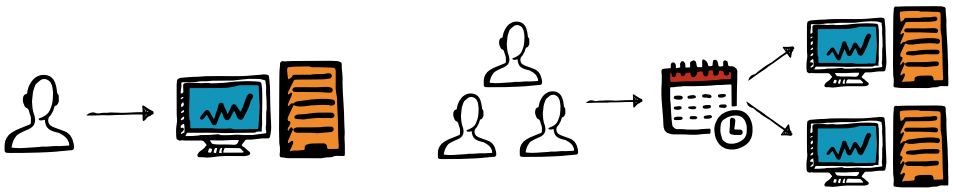


Figure 2.10: Current machine learning development workflow.

During later stages of the model construction, the amount of input data is increased in order to improve the accuracy. This often means more preprocessing steps and many times more data. The training is executed on dedicated training computers, which are much faster, and monitored manually to evaluate how they perform and when to stop the training process. These separate training computers is available for use through a manual process that includes making an entry into a calendar booking system. Additionally, the transfer of all the required data, as well as properly configuring the code for the new environment, are also manual processes. This separate training environment is referred to as *batch training* and displayed in Figure 2.11b. Compared to readily available cloud-based solutions from Amazon or Google, this manual process is inefficient and the engineers are looking for alternatives to better scale their current efforts.



(a) Interactive training.

(b) Batch training via booking system.

Figure 2.11: The different training environments used for machine learning development at the case company.

There is no unified workflow between the departments, used to streamline the process, and different solutions are used by everyone. There is a strong desire to create a case of best practices that can be used as guidelines for new projects, to significantly reduce the cost of initial setup. A unified, and centrally controlled, data repository is also something that has been actively discussed for a long time to further ease the task of GDPR compliance.

2.5 Problem Description

To help us identify the specific problems, experienced during the deep learning experiments at the case company, a preliminary unstructured interview was conducted with employees in one department which conducts machine learning model development. At this department they are mainly focused on training neural networks to be able to identify objects in video feeds coming from surveillance cameras. Concerns and problems mentioned here might therefore be centered on this department, but most of these are shared with other projects.

Marcus Skans is Lead Engineer at aforementioned department, and has summarized the problems most commonly expressed by the engineers that he manages.

1. **The experiments are highly dependent on the hardware and software.**
   When the engineers set up their experiments, specific configurations need to be made to ensure that the machine learning network can run on their computer. What this entails is that the code is not easily portable between machines, as other engineers needs to spend a considerable amount of time re-configuring the network for it to be compatible with their hardware. Furthermore, sometimes different software versions may be incompatible with each other, and replacement of an entire hardware driver might be necessary. A method of automating these necessary modifications would significantly reduce setup time and cooperation abilities.

2. **The experiments are difficult to scale.**
   This is partially related to problem number 1. If additional, better performing, hardware is introduced to the machine running the calculations, the configuration needs to be modified for it to take advantage of these new resources. Using larger datasets has also proven to require more than desired manual labor to adapt to these new conditions. An easy method of scaling both up and down would provide better allotment of compute resources between the experiments.

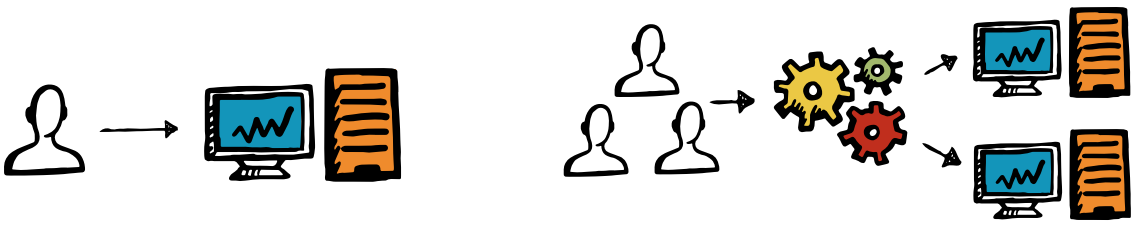
3. **The data sets are not easily transferred from storage.**
   The manner in which the data is currently stored requires the engineers to download everything to their local computer before being able to select the subset of files that are of interest. It is desirable to be able to download the data sets in a composable (modular) way as this would significantly reduce the time preparing the experiments.

4. **There are problems versioning the data sets.**
   The current method of keeping a journal of modifications to the datasets is inefficient and does not enforce the researcher to record what version they are using. This cause confusion to what exact files were used during a previous experiment and makes reproducibility difficult. When conducting research it is important to be able to reproduce the results at a later time, which is why a solution to this problem is necessary.

5. **There is no link between footage metadata and the corresponding data sets.**
   There needs to be a standardized way to link metadata with corresponding data sets in order to easier facilitate future automation. If possible, it would be desirable to combine the actual data with the corresponding metadata information in a robust way.

6. **It is not possible to prioritize or schedule long-running calculation tasks.**
   The data processing tasks take a long time to complete and there is currently no method implemented that allows to prioritize and schedule these tasks. To be able to prioritize critical steps to finish during business hours would enable the engineers to catch errors on the same day instead of the morning after. This would allow a more efficient use of the active workday and much better hardware utilization.

The problems expressed mainly revolve around the batch training environment where the computers are shared and larger datasets are regularly downloaded and manipulated in addition to the problems revolving around the data storage method. The wish is to introduce some form of automated system to handle the data and compute assets and the complex software environments involved with machine learning development. In comparison, the interactive environment should be kept mostly intact, i.e. as flexible as possible, while focusing on making the data resources easier to access with a focus on modularity. The changes are visible in Figure 2.12.



(a) Interactive training.

(b) Batch training via automated system.

Figure 2.12: The different training environments where instead of a booking system and manual processes the batch environment is handled by automated processes.

2.6 Scope

This thesis work is limited to analyzing a single example of deep learning experiments from the work being conducted at the video analytics department at the case company. From this example proposed solutions to workflow inefficiencies is evaluated. It is presumed that this single example of deep learning development is not deviating significantly from other experiments being conducted, and that solutions which improve the situation in our experiment will also improve them for all deep learning experiments. This is important as it is our intention to provide solutions that could be used by other departments in to improve their turnover time as well.

Furthermore, improvements to the experiment is limited to modifying and/or replacing the software around the neural network currently in use, as optimization to the core algorithms are complex enough to warrant a separate thesis work. Our pre-configured example is therefore approved by the engineers who are experts in said area. However, higher level settings, such as defining GPU instead of CPU processing, needs to be adjusted to allow for specific software to be compatible with said network.

Proposed solutions are mainly of open source nature, but corporate policies and incompatible licenses did in some cases require proprietary alternatives. Solutions that are readily available at the case company, but not currently used by the projects, was also prioritized as this allows for faster integration into the daily workflow since the infrastructure or licenses are already in place. However, this still requires careful consideration to weigh the preexisting tool's feature set to what is available on the market, and decide if it satisfies the needs expressed.

2.7 Contributions

In recent years companies have begun to prioritize the establishment of departments that are dedicated to deep learning research, in order to keep up with the trends of the industry. However, most of these companies soon realize that there are a lot of problems regarding the logistics necessary to facilitate a system capable at performing at a competitive level, and thus have a hard time scaling up the research.

There are papers published regarding how to set up a prototype experiment on a single local computer, and detailed explanations of the largest compute clusters that are maintained by the industry leading experts, such as Google and Amazon. However, there does not seem to exist any good information about the incremental growth procedure, from single computer to large datacenter.

It is our intention to provide an analysis to what difficulties a medium to large sized company will have to undertake when the experimental deep learning divisions are to be scaled up, if starting from similar circumstances as the case company. The workload of these projects are unlike other large scale data processing tasks, and the requirements on the underlying infrastructure differs significantly. This work aim to give an indication to what is necessary to facilitate on-premises development, and provide solutions to some of the more fundamental requirements, such as storage solutions and software management. Further details of the research question this paper answers is listed in Section 3.2.

Furthermore, there does not seem to be a unified method of sharing fully trained networks or configuration files used when setting up these experiments. While improving the efficiency of the workflow at the case company, our hope was that software solutions solution could be outlined and establish best practices which would be useful for the internal projects, but might also be helpful for the deep learning community at large.

2.8 Storage Technologies

Apple's *Alchemist* paper [29] mentions that the issue of interacting with the data resources is an important part of the machine learning workflow, and is something that has a performance impact proportional to the size of the datasets being used. The engineers at the case company have already experienced serious issues with the currently employed storage solution, even though the overall data in use is relatively small, and therefore find this to be a high priority issue.

At the foundation of any storage solution there will be a hard disk or a solid state drive that is ultimately tasked with storing the data in a persistent manner, and will allow for recovery in case of loss of power. These two types of drives use different technologies to physically store the data, but both may be referred to as *block devices*, since the core functionality of them is to store *blocks* of data [78]. These low level differences are not relevant for the use cases of this thesis work, but instead it is the abstraction layer used, when interacting with these block devices, which is of interest.

When saving a file on the computer, it is written to the storage device as a collection of many smaller blocks, and it is the task of the abstraction layer to signify that a cluster of blocks constitutes that specific file. The drive itself does not track how the blocks are related to each other [79], [80], and without an abstraction layer it becomes difficult to locate and retrieve the data of interest from the device.

As technologies evolve, different methods of interacting with the physical devices have been created to optimize for various workloads. While one alternative might be able to read and write data to the disks efficiently, it may be limited in how many files it is able to store without becoming prohibitively slow. It might therefore be possible to achieve significant performance gains by changing method of storage to one that is more suitable for the task at hand. For this thesis work, the following four storage technologies have been researched to later determine which is optimal for our use case:

- **File Storage**
- **Block Storage**
- **Object Storage**
- **Databases**

At the foundation of all alternatives presented here, there will be a hard disk drive or a solid state drive that is ultimately tasked with storing the raw data bits of interest. However, it is the interface the user interacts with, to store and retrieve data, which differ significantly between these alternatives. It is important to choose a solution that implements the best interface for the type, and the amount, of data that will be used by the engineers at the case company, in order to provide the fastest and most reliable access to the data.

Additional care will also have to be taken to not introduce a solution that has an interface that is too difficult to use, or even incompatible, with the current workflow. A short description of the fundamental functionality will therefore be presented for each of the storage technologies, with notes on the main advantages and disadvantages.

2.8.1 File Storage

This is the most common method used to abstract how data is physically stored on a device, and is used by almost every desktop computer. Opening the operating system's native file explorer will present the block device as a tree structure of folders, inside which files can be placed or further sub-folders can be introduced to add additional levels into the structure. To obtain a file the computer will have to traverse the path through all the folders to reach the one that contains the requested data [81].

A popular analogy to visualize this system is to imagine the block devices as normal physical cabinets that contain manila folders, which in turn contain paper files. This is, for humans, a logical way of storing items, but will become unwieldy when the amount of data increases. The cabinets can only house a limited number of files, and if the folders needs expanding the solution is to move them to separate cabinets that are larger [81], [82]. A simple illustration of this can be seen in Figure 2.13.

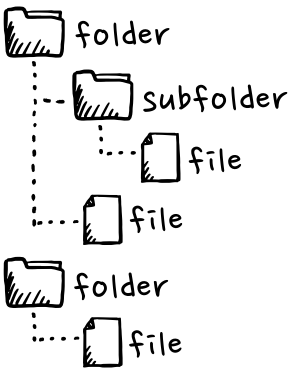


Figure 2.13: Example of a basic folder structure in a file system.

Advantages of this storage technology is that it is trivial to set up, and has wide adoption, through operating systems' file managers, have made the general population understand how it works and feel comfortable using it. There also exist well-known and tested protocols to share these file based storage systems across a network (NFS and SMB/CIFS [83]).

The disadvantages are limitations to the folder structure, and how the storage space should be distributed. Without additional tools used to disperse the data, a file will only exist as one instance on one drive with all the redundancy and speed implications that entails. Storing files this way will work for smaller projects, but will be difficult to scale towards multiple millions of files [80], [81].

For the engineers at the case company it is primarily the limitations to the number of files that makes this a bad choice of storage. The backend storage servers, in use today, are already struggling with the number of files in each dataset, and this will only increase in the future.

2.8.2 Block Storage

Block storage solutions are, as the name suggests, closely related to raw block storage devices, as it presents the computer with an interface very similar to normal storage drives [79], [81]. However, the drive interface is an abstraction created by a storage controller that has access to multiple physical drives, on which it may address all data blocks individually [80]. The controller then expose this collection of blocks outwards, trough protocols such as iSCSI [84], which allows computers to directly access them over a network. See Figure 2.14 for a simplified visualization of this.

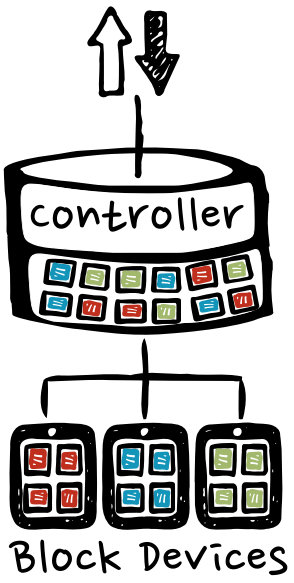


Figure 2.14: A controller exposes a collection of disks as a single block device.

What this entails is that it is possible to create a virtual block device that can be mounted to an external computer as a normal hard drive [85]. Furthermore, as was mentioned in the introduction of this chapter, since block devices are agnostic to the type of data it stores it is then possible to apply a file system on top of the drive. This gives it the same ease of use benefits as the file storage method mentioned above [86], while also being able

to distribute the data over multiple drives at the controller, which allows for more available storage and faster transfer speeds. It is also possible to create backup copies of the data transparently for the user.

Furthermore, as the blocks are addressed directly on the underlying storage devices, it allows for fast and granular access to writing and reading to the disks, and incremental updating of files is possible [87]. Applications that do a substantial amount of reading and writing benefit greatly from this type of storage, e.g. databases of which some prefers access to the blocks directly [88].

Disadvantages with this technology are that complexity grows quickly if the underlying physical block devices become numerous, and even worse if they are located in different data centers that are miles apart. Applications that have their file systems placed on drives that are far away may experience too high latencies when reading or writing files and may start to misbehave [81], [87], [88].

For the use case of storing large amounts of multimedia data, which is usually only saved once but read multiple times in the case of machine learning, the main benefits of this solution is not utilized. The complexities related to the use of many storage drives also makes this expensive if the total storage size grows large. This price premium will have to be weighed against cheaper, possibly slower, options.

2.8.3 Object Storage

In contrast to block and file storage, object storage does not directly expose neither files nor blocks towards the users. Communication with this storage solution is more akin to either downloading or uploading a file through a web browser, as the server side software responsible for the physical storage usually implement a RESTful API [89] over HTTP as the interface towards the clients [81], [90]. The most common API is Amazon S3 [91], which allows clients to download a file by navigating to a URL inside the web browser, or commence batch transfers using programs [92] that are installed on the computer.

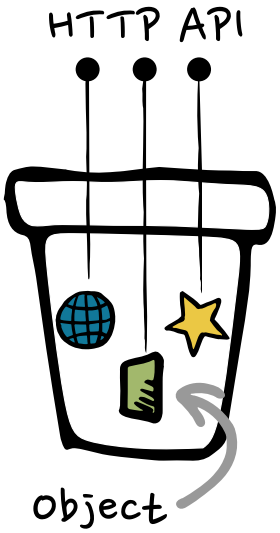


Figure 2.15: Object storage allows for a flat address structure, and the RESTful API make requesting data easy.

Furthermore, this technology implements the feature of assigning a unique identifier to each object, which the storage software uses to locate where the object is physically located. Because of this identifier, the objects can be stored in something that is called a *flat address structure* or a *flat memory structure* [90]. The user then only sees the available storage as a big "bucket", where the unique URL allows for direct and immediate retrieval regardless where this object is physically located, see Figure 2.15 for an illustration.

When a file is uploaded to the storage solution, it is written to the backend storage devices as an object. An object is a self-contained piece of information that consists of the main data of

the file uploaded, as well as whatever metadata that is desired to be added to it. This metadata may include information such as a human readable name of the file, time stamps, permissions and tags to what the file includes [81].

Having a flat address structure can be equated to storing all objects in a single root folder, with the unique identifier ensuring that there are no name conflicts. Saving files, using this method, enables the storage software to treat every drive connected as an expansion of the same root folder, and thereby spread the data over the available storage space. In contrast to block storage solutions, object storage has the ability to more easily scale as the design of most server software allows for distribution and coordination over multiple physical computers. Using multiple system, connected as a cluster, can then increase the storage space or balance the load of requests between them by duplicating the data which also adds the benefit of redundancy [87], [90].

The disadvantage with this is that the system is not designed to handle applications that require fast read and write operations [88]. The reason for this is that it does not support incremental updating of objects, so any modification requires the entire file to be re-uploaded. Instead, this is more suited to store a massive amount of files that are downloaded often and written once [80], [81], [85]. This is close to the use case of the machine learning workflow at the case company, where the datasets are uploaded once to a central repository, for long term storage, and then downloaded many times to the different workstations. What might be a concern is that the RESTful API, used to communicate with the servers, is a new concept for the engineers, and may need some time to fully learn.

2.8.4 Database

The final technology to be discussed is databases. These usually provide excellent performance that is faster than storing the information as files directly on a physical drive [93]. However, the caveat with using databases is that it is important to differentiate those who are fully relational (SQL) from those who are also able to handle non-relational data (NoSQL). The former has the capability to be ACID (Atomicity, Consistency, Isolation, Durability) compliant [94], which means there will not be any inconsistencies between data entries when there are many queries coming to the database. However, these capabilities place strict requirements on the data that is to be stored, as it needs to be structured in rows and columns, and every row must have the same number of columns. This does not allow for new data formats which do not conform to this schema [95].

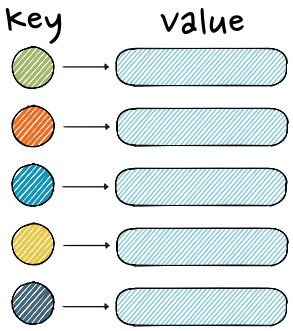


Figure 2.16: Visualization of a key-value database.

To move beyond this limitation, NoSQL databases offer schema design that allows for the input data to be non-uniform. These types of databases sacrifice the strict consistency capability for the ability to store larger structures of data that may differ significantly from each other [96]. This schema design also allows for sharding of the database, which means that it is able to be split over multiple storage locations and/or multiple services. This will

then allow for significantly better scalability than its counterpart [97]. Figure 2.16 is a simple visualization of a key-value database structure which highlights the simplicity of the model that makes it easy to place the keys and values across multiple computers.

This might be the deciding factor when considering a database as storage solution. If the input data vary greatly in both size and in attached metadata a NoSQL schema design will be required. Furthermore, this feature is necessary if the database is to be efficiently sharded over multiple drives and/or nodes and allow for cost-effective expansion and redundancy [97].

Disadvantages with databases are that, when storing a large amount of files of various formats, the general recommendation would be to store the large binary files in a normal file system and only store the links and metadata inside the database [98], [99]. This requires two separate solutions which might not always be desirable. Another disadvantage, in similarity to object storage, is that the database usually has its own query language that needs to be used when interacting with it [100], [101].

The data used by the engineers vary greatly in both size and type, which is not ideal for a database storage solution. It is also not as easy as object storage to scale in size, which makes it difficult to see this as a good suggestion for storage solution.

Chapter 3

Research Questions & Methodology

In this chapter the goals of the thesis project will be presented, alongside the research questions that are to be answered throughout this report. The reasoning behind the choice of methodology will also be mentioned, together with a description on how the problems are approached in order to obtain relevant results.

3.1 Thesis Goals

In conjunction with recent efforts into expanding the machine learning model developments, at the case company, the previously manageable inefficiencies of the current workflows have become more obtrusive, and the overall productivity have suffered as a consequence. In order for the case company to remain competitive, the turnover time for each experiment needs to be reduced, so that the researchers can iterate through different input parameters and develop more accurate models faster.

The list presented in Section 2.5 was expressed by the lead engineer of the video analytics department, and it provides a description of what the affected employees believe to be the most prominent problems. These statements were later used as the foundation when defining the research questions of this thesis report. It was our intention that, by providing answers to these questions, we would be able to reduce the problems expressed with the current workflow, and thus decrease the turnover time of the experiments.

As of today, the engineers at the video analytics department did not have a well-defined and organized documentation of the workflow. Hence, there was no easy method of analyzing which of the steps that are actually causing the most significant inefficiencies. The desired

outcome of this thesis work was therefore to provide the engineers with a structured analysis of the current workflow, and present our findings of the most prominent bottlenecks.

The goal was to research what software solutions are available which can be used to alleviate the problems expressed in Section 2.5. The ambition was that improvements for multiple steps of the workflow could be solved with either a small collection of software solutions, or a system solution which would tie together a collection of useful tools in a user-friendly way. An implementation of such a system was considered less complicated and preferable to developing our own software solutions and did turn out to solve a lot of the identified problems.

Focus was put towards actually implementing the solutions found into the workflow to showcase actual improvement results and not only provide a case study of the possibilities that are available. Our desire was that our suggestions would be viable to be deployed into production at the case company, and that they should benefit not only the video analytics department but all similar development efforts being conducted at the case company.

3.2 Research Questions

This thesis work has been conducted at a large international company that is industry leading in the field of video surveillance. The research questions have therefore been formulated so that focus is put towards solving the problems specific to this company, and more precisely the problems identified at a single department. However, care has been taken in order to make these questions generic as to make the research relevant for a wider audience.

Our belief is that the following questions provide a good balance between academic and corporate interests. Research of these issues will touch upon general problems of machine learning workflows, while at the same time inciting solutions that solve concrete problems currently present at the case company. Answering these research questions will alleviate the issues outlined by the engineers in Section 2.5.

1. What are the current processes (in analytics) for data storage and processing at the case company?

2. Which parts of the company process cause the most significant inefficiencies?

3. What approach, including method and tools, could be applied in order to minimize the identified inefficiencies?

4. Which of the improvement proposals would be most applicable to this problem space and why?

5. How does the improvement proposals identified, under question number 4, improve the actual process efficiency?

3.3 Methodology

Through careful consideration it was decided that *action research* was the most suitable methodology for us to follow in order to be able to answer our questions in a formal and scientific manner. The methodology centers around four main steps - studying, planning, action and learning [102]. These are then continuously repeated, as is illustrated in Figure 3.1, for each change that is made until the final desired outcome is reached.

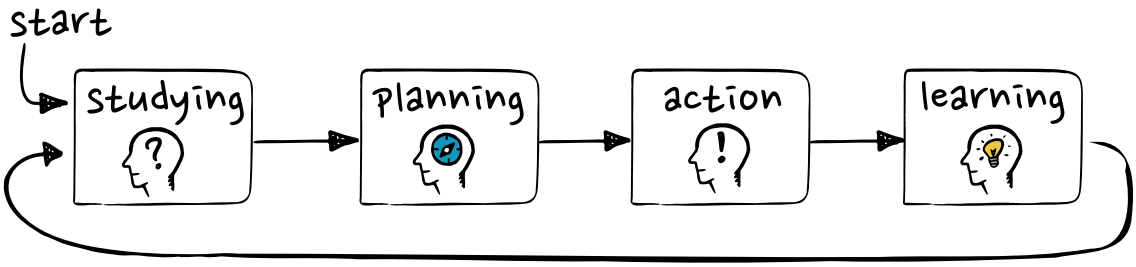


Figure 3.1: The four basic parts of action research methodology.

The methodology states that the researcher should be an active user of the process that is to be studied, before beginning to investigate any improvement options. By experiencing the process firsthand, the researcher will have a better grasp of what actions would change the workflow the most, and will also have an easier time planning how these modifications may be best implemented.

When a plan has been devised, it is time to take action and implement the suggested changes into the actual process. Since the researcher is familiar with the previous performance, it should be immediately noticeable if the modifications provided any significant improvements. By observing the outcome it should also be possible to learn what parts had the most significant impact, and help identify if more modifications can be introduced. The cycle then repeats.

By continuously going through the entire process numerous times during the research, immediate feedback can be obtained and suggested solutions can be discarded or approved rapidly. Keeping the changes small will also allow for more iterations, and a higher chance of finding the best solution for the problem at hand.

Iterating through the workflow multiple times with small alterations every time yield fast results, but this iterative process can go on for a long time. Because of this, it is of great importance to keep a detailed journal of the differences experienced by any of the modifications. This will help in the step where these are to be studied, as any regression of features will become apparent immediately, and a rollback to a previous iteration can be done.

By following this method it should, towards the end, be straightforward to understand the evolution of the project and how it reached its final state. Along the way any solutions that was tested should be mentioned and explained as to why they were discarded or implemented, but the focus should be on how much the process improved as the result of the applied modifications. A comparison is then made between the final product and the original, where the modifications that made the largest differences are highlighted.

3.4 Approach

In line with the action research methodology, the approach to solving the overarching problem, of an inefficient workflow, was initiated by breaking it down into smaller more manageable pieces that could be analyzed in isolation. As such, the effect each individual piece had on the overall process was more easily measured, and could be iterated through more quickly.

After an initial semi-structured interview with the engineers, who are the ones performing the machine learning experiments, a better overview was received on how the entire process is structured. The workflow of the current development cycle included in machine learning model development was from that information divided into five individual components, as shown in Figure 3.2.

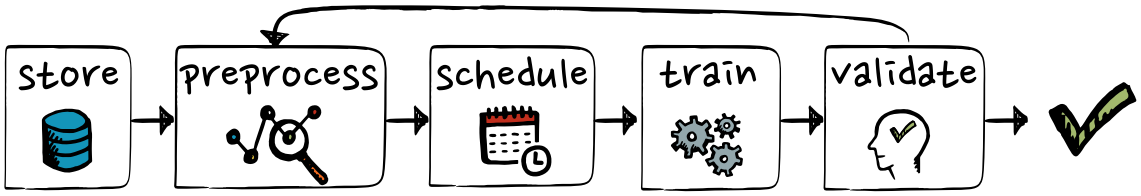


Figure 3.2: Simplified schema of the primary components that the current machine learning workflow consists of.

The storage section consists of solutions that are responsible for physically storing the data of interest in the long term, as well as the methods used to move this data from different locations. In the preprocessing step, the input data is selected in order to later be transformed and modified though various techniques which both extends the data set, and formats it according to the preferences of the machine learning framework and models chosen.

The scheduling and training steps are closely related, and even though it is difficult to discuss one without mentioning the other, there is a necessity to distinguish between them. The scheduler is responsible for taking the computational tasks, from the succeeding step, and plan when and on which compute resources those are to be executed. Having taken these decisions, it is now possible for the training step to start. This is often the most time-consuming part of the workflow, since the algorithms will need to iterate trough a minimum amount of input data before the output metrics shows how it may perform in a real world scenario. This is, of course, highly dependent on how fast the compute resources are.

Finally, in the validation part of the workflow, the results are evaluated to check whether the model performs according to the engineer's expectations. If not, one might want to reiterate with different input data or training parameters and start the process once more. If the model performs according to expectations and requirements it is stored for future integration in a production environment.

From studying the process performed by the engineers, the infrastructural components that were affected during each step were identified, along with software solutions currently used. It was then possible to measure performance metrics and test new solutions that reduced the friction experienced.

Since the process was divided in such a way that it was possible to introduce alternative tools without affecting the other parts, multiple iterations of changes was made in parallel in order to more quickly find better solutions. However, it was important to define how a solution is deemed to be more efficient than the one currently in use.

3.4.1 Suitable Metrics

Since the majority of the thesis work revolves around evaluating software based solutions, there needs to be a template that can be used to help determine which of the proposed alternatives is the better choice. This thesis utilized an official ISO standard [103], as it provides general guidelines on how to assess features of a particular software, and define them in a way which allows for quantifiable comparisons between different solutions. This standard describes software by six characteristics, which may be further sub-divided if necessary:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Through these six categories, it should be easier to analyze the options in a more objective manner. Even though *Usability* might be subjective, it should be easier to quantify the feature as *better* or *worse* than the other alternatives. Depending on the use case, the points presented here may also have different order of importance. An interface designated for the end user might rank usability over efficiency, while an evaluation of a database might value reliability over everything else.

However, since the main goal of this project was to decrease the lead time of the machine learning processes at the case company, it was the characteristic of efficiency which was the main point of our focus. To be even more specific, the primary unit of measurement was how long it took to complete a task (wall-clock time), with some regard to how easy it was to execute. Even if execution of the new solution is faster, the process required to make it work might be overly complicated and would make it slower overall. It is therefore necessary to be very specific when defining the moment a task begin, and when it ends.

Since elapsed real time is an objective point of reference, it was easy to use it to compare the current process to our modified ones. Faster set up and execution was immediately noticed, and showed a distinct and measurable effect regarding the turnover time for the experiments at the video analytics department.

3.4.2 The Process

In order to be able to collect the metrics of relevance, work began by observing the engineers conduct an example of the machine learning workflow from beginning to end. During this process any thoughts and comments the engineers had were recorded regarding which tasks

were necessary and what software and data needed to be used. This was done in order to understand how the engineers preferred to work, so that this could be taken into account when evaluating alternative solutions. Introducing solutions that would be incompatible with the current way of working would most likely cause more harm than good with regard to the goals of this thesis work.

This information allowed us to quickly filter out any of our solutions that had obvious features missing, due to being incompatible with the current workflow, and priorities those which was both compatible and improved the situation the most. It was therefore important to get time measurements on all the tasks that were performed during the machine learning process, in a way that followed the structure defined in Figure 3.2. The most natural breakpoints, signifying when to start and when to stop the timer, was decided through discussions with the machine learning engineers at the company.

When we felt confident about a recommendation for an alternative solution, we notified the persons that would be affected and had a short semi-structured meeting with them, to make sure the solution would be approved before a more in depth analysis commenced. This eliminated the probability of us implementing anything that would immediately be dismissed by the engineers at a later stage. A time measurement was then taken on the new software solution, to be able use this as an improvement metric. The starting and stopping points was kept as similar as possible to the original process, to make the comparison as valid as possible.

In regard to the storage solution, which needed to be replaced, we had to perform some additional steps beyond the normal procedure. Several semi-structured meetings with decision makers at the IT department were conducted, since an alternative storage solutions would replace a company wide service. As expected, this required more afterthought and planning from our side, as well as from the IT department, who were keen on ensuring the chosen option would be compatible with the local network infrastructure. For this reason we were provided with stricter feature criteria, which needed to be fulfilled, as it would be deployed for large scale production in an enterprise environment.

3.4.3 Threats to Validity

In regard to the validity of the time measurements that have been performed, there were usually some small variations in the timings between otherwise identical runs. Our belief is that this is because the software is running on hardware that has the ability to dynamically change its performance if the power supply is powerful enough, and the thermals are within reasonable limits. This did not affect the tests performed on the local workstation significantly, but would prove to be more of an issue with the multi-GPU node. However, for the proof of concept measurements that was made, where these issues played a role, the thermal limitations is clearly expressed as an important factor which degraded the results significantly.

Regarding the local workstation used, we made sure that the hardware setup was as close as possible to what the engineers are working with on a day-to-day basis, to better reflect a real-world scenario. However, since not every engineer had identical hardware, our computer was instead composed of representative hardware of the general case. Therefore, the results

provided will not be representative for every unique situation, but should provide a valid comparison to the average case.

Throughout this report there have also been a strong bias towards software solutions that are open source. This was an active choice for the reason that it allowed a greater possibility to explore in depth how the solutions worked, and that these would be much quicker to have up and running to show real world performance metrics. Open source solutions are also important as without them the reproducibility of our results would become much harder.

There is also an overall positive attitude towards open source solutions at the case company, which meant that our suggested solutions were usually well-received when presented to the engineers. The overall impressions were that it was mostly the amount of support that differed from closed versus open source solutions, which only became a significant issue when it came to the discussions about deploying a production ready solution. It was primarily in the case of deciding the type of storage solution, which greatly involved the IT department, when compromises to this preference had to be made. Service agreements with third parties made a proprietary solution the easiest one to actually deploy at the case company.

3.5 Contribution Statement

This thesis work was created as a joint effort between two departments at the case company, since the difficulties experienced by one is within the responsibility domain of the other. Anton Friberg was working part-time at the latter, and was therefore able to help shape this project into fulfilling the requirements of a master thesis work. Jonas Alfredsson was working at another department at the time, and was therefore not present in the very initial stages, but joined soon after the main goals were outlined.

However, from that point onward the workload was split as evenly as possible, with many of the steps requiring the presence and attention of both the thesis workers. Since interviews and workshops required knowledge about all the components of the process, to be able to keep up with the discussion, sharing all of our knowledge between us have been paramount. This was necessary, as otherwise we would not have been able to provide the employees with credible arguments to why our solutions were motivated.

Nevertheless, it is inefficient to have two persons always working on the same item, hence there are subjects in the report that have been researched more by one student than the other. Jonas has a great interest in computer storage, as well as networking, while Anton found the art of distributing computation exciting. It was therefore natural to having the former putting more time into researching storage solutions, while the latter placed a greater effort into finding tools to ease the distribution of the computational tasks.

As new information was surfaced by one of us, the other was soon after informed. Decisions regarding what solutions would be best to implement, were therefore taken together after careful reasoning and discussion originating from the information we had gathered.

Chapter 4

Analysis

In this chapter the current machine learning workflow, at the video analytics department, is presented and described in a detail. The tools and methods mentioned during this process demonstrate the current circumstances, and will be the performance and usability baseline to which proposed efficiency improvements are to be compared against.

The subsequent section presents research into possible solutions to the problems outlined in Section 4.2. The reasoning behind the decisions to implement one or more of our suggested solutions are also explained, however, how it impacted performance will be presented first in Chapter 5.

To follow the action research methodology, outlined in Chapter 3, the report will be focusing on evaluating steps of the workflow in isolation. The idea is to evaluate problems and solutions for each part of the workflow and in the end present the overreaching result of the proposed solutions.

4.1 Preparations

During the thesis work different software solutions have been tested and compared in order to see how much they increase the efficiency of the workflow in comparison to the current alternatives. In order for these measurements to be valid representations of a real world scenario, both the hardware and software setup used needs to be as identical as possible to what the engineers are currently using. Therefore, the components, version numbers and other specification used is carefully defined and will remain static throughout the testing. This information will be of interest if one would like to reproduce the tests performed in this thesis.

4.1.1 Hardware

The most common setup of a workstation, used by the engineers, is an Intel CPU paired with a top of the line consumer grade Nvidia GPU. This is complemented with a large amount of system memory and fast NVMe storage. The collection of hardware that was deemed to be a good representation of the average machine is listed in more detail in Table 4.1.

Table 4.1: Hardware specifications of the local workstation used for taking time measurements.

Component	Brand	Version	Specification
CPU	Intel	i7 8700K	6x3.7GHz
GPU	Nvidia	GTX 1080Ti	1 531 MHz, 11GB
RAM	Corsair	DDR4	2x8GB, 2666MHz
PSU	Corsair	CX750	750W
Storage	Intel	660p NVMe SSD	1TB
Motherboard	Gigabyte	H370 HD3	rev 1.0
Network	Intel	I219-V	1GbE

In the later part of the thesis work we will also run some experiments on a more powerful multi-GPU compute node. These are shared between many engineers, and are not used in the day-to-day workflow. In comparison to the normal workstation, this compute node has a more powerful CPU from AMD, more system memory, more storage and four GPUs instead of one. The complete hardware setup of the node is listed in Table 4.2.

Table 4.2: Hardware specifications of the multi-GPU compute node used during measurements.

Component	Brand	Version	Specification
CPU	AMD	Threadripper 1900X	8x3.8GHz
GPU (x4)	Nvidia	GeForce GTX 1080 Ti	1 531 MHz, 11GB
RAM	Corsair	DDR4	4x16GB, 3200MHz
PSU	EVGA	Supernova T2	1600W
Storage	Samsung	970 EVO NVMe SSD	2TB
Storage (x2)	Western Digital	7200 rpm	4TB
Motherboard	Gigabyte	X399 Designare EX	rev 1.0
Network	Intel	I211-AT	1GbE

4.1.2 Software

As was mentioned in Section 2.6, the central training software used during measurements is limited to a single machine learning example. Our example is based on a common supervised deep learning task, that was suggested by the engineers, which is the training of a model for object recognition on the `CIFAR-10` [104] dataset.

Supervised deep learning is the most common type of machine learning task that is performed by the engineers, and the general workflow of this was introduced in Section 2.2. This object recognition task provides a suitable starting point, since implementations exists as a part of the official documentation for most of the machine learning frameworks, including PyTorch [105] and TensorFlow [106]. Another advantage is that this example works on the same kind of input data as the engineers themselves normally work on, i.e. binary images, which comes with additional challenges in comparison to text based information, as was mentioned in Section 2.1.

However, our example modifies the original PyTorch implementation in order to increase the computational complexity and make it easier to configure input parameters. This was mainly done to better represent the real world scenario, of having the actual training itself being the most demanding task, but also to put additional stress on the infrastructure. The increase in computational complexity was achieved by replacing the `CIFAR-10` dataset with a drop-in replacement called `CINIC-10` [107]. This increased the number of input images in the dataset from 60,000 to 270,000.

The addition of letting the training example be configured via command line arguments was done to make it easier and faster to set up and configure the experiment for different scenarios. The complete written implementation has been made available on GitHub [108], and has mainly been tested with Python 3. All Python dependencies are outlined in the `requirements.txt` file inside the repository.

The machine learning frameworks, used by the video analytics department, all utilize the included functionality of being GPU accelerated thorough the Nvidia CUDA API [73]. This requires both the base Nvidia graphics driver and two CUDA add-ons to be installed in order to have full functionality. The complete software rundown used during testing is displayed in Table 4.3.

Table 4.3: The name and version of the relevant software that was installed on the machines used for training.

Software	Native Version	Docker Version
Linux Distro	Ubuntu 16.04	Ubuntu 16.04
Nvidia Driver	384.130	410.78
Nvidia CUDA	8.0.61	10.0.130
Nvidia cuDNN	6.0.21	7.5.0
PyTorch	1.0.0	1.0.0
TorchVision	0.2.1	0.2.1
Python	3.5.2	3.6.7

4.1.3 Datasets

In addition to the `CINIC-10` dataset, utilized by our machine learning example, two additional datasets were also used in order to stress the storage and data transfer infrastructure. One is called `CUHK03` [109], which contains a collection of JPEG images of full body shots of people walking along streets. The other dataset is called `OpenImagesV3` [110] and is one of the largest datasets that is both openly available for download, and being actively used by the department. This dataset contains multiple different categories with thousands of pictures in each of them. This was suggested as an example of a set of data that should prove itself to be very difficult to download to the local computer. Relevant data about the datasets used are shown in Table 4.4.

Table 4.4: The number of files in each dataset, as well as different size measurements of them.

Dataset	Nbr of Files	Size on Disk	Real Size	Average File Size
`OpenImagesV3`	1,759,154	531,516 MB	528,081 MB	300 kB
`CINIC-10`	270,003	1,208 MB	727 MB	2.7 kB
`CUHK03`	28,193	218 MB	167 MB	5.9 kB

Something that should be noted, in Table 4.4, is that there are two sizes given to the datasets, some of which differs significantly from each other. The reason behind this is related to how physical disk drives work, and how files are stored in *sectors* on the drive. A disk sector is the smallest size a single disk can write, and if a file is smaller it will simply be padded with zeroes to fill up the additional space. With larger files the amount of storage wasted is negligible, but with many smaller files this adds up to a significant amount.

The disk used, during testing, has sectors of 4 KB, and the average size of the files in the `CINIC-10` dataset is around 2.7 KB, which would waste about 32% of the sector size. This can be compared to `OpenImagesV3` and `CUHK03` which waste 0.7% and 25% respectively.

When measuring transfer speeds, these calculations will be based on the *real size*, since this will be closer to what is actually transferred over the network. However, when comparing the storage size of the datasets, the *size on disk* will instead be used to represent the actual storage space required to host them.

4.2 Current Workflow

Following the action research methodology, outlined in Chapter 3, the initial step is to perform a baseline experiment in order to obtain results that can be used to verify if a suggested solution really do improve the workflow. Thus, we will begin by performing the entire workflow once, just as the engineers are usually doing it, with the subsections being named in accordance to the general structure shown in Figure 3.2. This is done to make it easier for the reader to understand what part of the process that is being discussed.

4.2.1 Storage

The first step the engineers perform, when a new project is started, is to choose what set of data that should be used when training the machine learning algorithms, and then proceed to download this to the local computer. Relating this part to a segment in Figure 3.2, would place this inside the part called *Store*, with the process of downloading being closer to *preprocessing*.

All the relevant details about the datasets, which we had at our disposal, are listed in Section 4.1.3. We were recommended, by the engineers, to try to download both `CHUK03` and `OpenImagesV3`, with the former being an example of a small and easy dataset to obtain and the latter being a large and difficult one.

4.2.1.1 Current Storage Solution

Before proceeding with downloading of these two datasets, a short explanation of how the current storage solution works is needed to fully understand what goes wrong later. Currently, all the datasets used are stored on an on-premise hosted version of Gerrit Code Review [111], which is a version control system that extends the usual Git [112] software. However, while Git is efficient at tracking small text files, it is not the optimal solution when it comes to tracking larger binary files. To mitigate this limitation there exist the Git LFS extension [113], which allows for the binary files to be saved on an external storage solution, while the main Git repository only retains a small text file with a pointer to where the real data can be downloaded from.

Figure 4.1 displays how this dual storage solution works. The local client keeps a folder structure consisting of both small text files, e.g. code, and images that have been manually marked as *large binary files* via the LFS extension. When committing changes and uploading them to the remote repository server, these binary files are replaced with pointer files, while the real data is being uploaded to a separate storage service that is better suited for storing binary files. The procedure in reverse has the LFS extension interpret the text inside these pointer files as links, and downloads the correct files from the file storage. The files used as links are then replaced with the intended binary files as they are downloaded to the client.

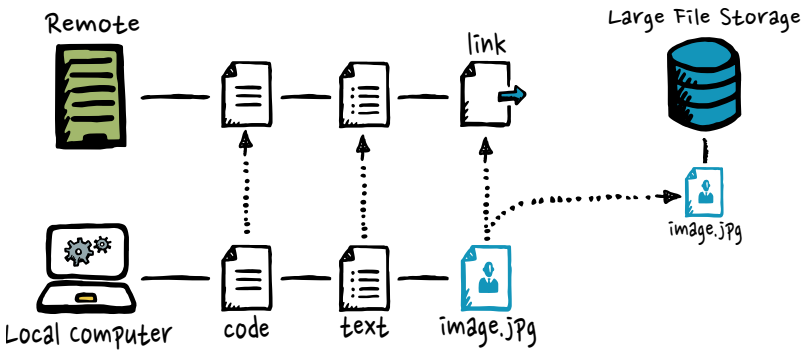


Figure 4.1: A diagram on how the LFS extension integrates with Git.

The external storage solution, used by the LFS extension within the case company, is an Artifactory [114] instance running on a separate server. Artifactory is an object storage solution which has native support for the LFS protocol [115], and is currently only in use as there was an absence of other alternatives at the time of introduction.

4.2.1.2 Current Data Retrieval Method

Due to some commands using file paths that might be considered confidential, these will be replaced with generic statements. However, it should be simple to understand how to modify the command to work if it were to be executed outside the case company's network. A high level overview of how the files are requested, when using Git LFS, is shown in Figure 4.2.

To begin with we tried to clone the repository containing the dataset `OpenImagesV3` to the local computer. However, execution of the following download command failed fatally after about an hour.

```
1 git lfs clone <repository>
```

The reason for the failure was a combination of the large amount of files that needed to be downloaded, optimizations done by recent Git LFS versions (`>2.3.0`), and rate limit settings on the Artifactory server. When Artifactory initiates these rate limits it notifies the client responsible by responding with the `HTTP 500` status code, however, it seems these are not handled as expected by the LFS extension. After multiple attempts, with the same `500` response, the client comes to the conclusion that the requested resources do not exist and then fail with an undescriptive error message.

To be able to effectively troubleshoot the issue further, we shifted the focus to try to download the `CHUK03` dataset instead. Downgrading the client used, and tweaking parameters related to simultaneous transfer streams, the download managed to complete in about 17 minutes. However, this is a relatively small collection consisting of less than 30 thousand image files that collectively take up around two hundred megabytes of disk space. The challenge of downloading the large dataset still remained.

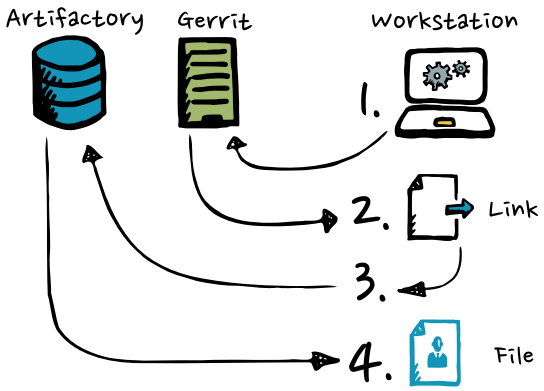


Figure 4.2: Overview of the current download procedure where Git LFS communicate with two storage services to obtain one file.

Despite the previous workarounds implemented, we were still not able to get the transfer of `OpenImagesV3` to complete in one continuous process. Instead, this had to be downloaded in multiple smaller batches, which also failed multiple times and had to be restarted. It took us slightly more than seven working days to finally have a complete copy of this dataset on our local computer. Out of this time, the client was only actively downloading data for about 19 hours, the rest was wasted on troubleshooting or restarts.

One of the workarounds was to only download the absolute most recent version of the dataset, since otherwise Git would include the history of all the files that have been changed. Even though this was done, the folder occupied 1.1 terabytes of disk space when the dataset itself was expected to be around 530 gigabytes. Similar increases could be seen on `CHUK03` as well, which took up 505 megabytes on disk while it should only be 218.

This storage overhead is introduced by Git and the LFS extension, which also seems to inflate the total number of files that are located inside the repository folder. `OpenImagesV3` increased from almost two million to around four million files. This meant more than half of our retrieved data was superfluous.

The engineers themselves were surprised by this revelation, and expressed concern that this problem had not been identified sooner. The employees at the case company, that are tasked with maintaining Gerrit and Artifactory, noticed the significant strain the download attempts had put on the respective systems, and expressed concern about it affecting the rest of the company. All the affected parties agreed that this is not a sustainable solution, especially since the sets of data are expected to grow significantly in the near future.

4.2.2 Preprocessing

In Section 2.2.1 a number of tasks which are part of the preprocessing step is discussed. The datasets used in this thesis have already been filtered and assigned metadata in a chosen structure. As such, the initial part of the preprocessing step is complete, and the input data is ready to be used by the machine learning frameworks of interest.

At the time of writing there is also no process in place to produce internally sourced sets of data with standardized structuring of metadata. The engineers are currently actively testing different methods, which would allow easier management of metadata, but these efforts are outside the scope of this thesis project and are therefore not discussed further.

Regarding extending the dataset used during training, the engineers at the video analytics department do use all the simple augmentation methods mentioned in Section 2.2.1, and more where deemed appropriate. The major improvements possible, with these simple modifications, are widely known [61], [62] and, since both the TensorFlow and PyTorch framework have includes native support for such augmentations, these are almost always utilized. The minimal additional work needed to gain added accuracy and robustness is considered a fair trade-off.

However, in the training example constructed, only horizontal flipping and cropping of images will be applied. It was decided to limit the number of augmentations to keep the code simple and easier to understand. For the sake of simplicity only PyTorch will be discussed,

as the procedure in TensorFlow is much more verbose. The two mentioned alterations are implemented by simply including line 2 and 3 as shown below.

```
1 train_transform = transforms.Compose([
2     transforms.RandomCrop(32, padding=4),
3     transforms.RandomHorizontalFlip(),
4     transforms.ToTensor(),
5     transforms.Normalize(mean=cinic_mean, std=cinic_std)
6 ])
```

In order to analyze if there might be any settings that are improperly configured, a closer look at the internal functionality was taken. It was noticed that the PyTorch framework is designed so that, when its internal `DataLoader` class reads the original image from disk, it applies the desired alterations randomly just before loading this new image into GPU memory. The altered file is never stored on disk, and this randomness creates additional diversity of the input data, since reading the same file twice should yield a slightly different result of the final image that is used for the training.

An interesting feature of the `DataLoader` class is its asynchronous and threadable nature, which allows for multiple instances of it to load and prepare images in parallel. Executing the training computations on a GPU would then make it possible for the augmentation tasks to be offloaded to the CPU, to fully utilize all available resources. However, for this to be viable it is important to have storage that is fast enough to support multiple simultaneous file reads.

Furthermore, there needs to be enough augmentation workers to successfully provide the GPU(s) with sufficient amounts of input data to keep them fully utilized. As most of the experiments are done on the engineers' local machines, with only one GPU, multithreaded data loading might not be of benefit. The engineers have not experimented with anything else than default settings, which only use a single worker, and thus the effect of this parallelism is unknown as of yet. However, if experiments are to be distributed to the multi-GPU workstations it may become a significant bottleneck, and it should be analyzed how this affects overall performance on a training batch.

As was stated earlier, it is known that it is beneficial to extend the datasets by augmenting them through processes that involve randomness. However, this use of random parameters cause a concern regarding the reproducibility of the experiments, since the data used to train the network is not stored after being altered. The original pictures on disk are randomly modified before being handed over to the neural network, which means that the experiment will not be able to be reproduced with the exact same result. This will become a problem if it is a requirement to be able to have reproduce experiments, and a solution should be found.

Since all the relevant preprocessing tasks are executed at runtime of the training code, we can not provide a specific time it takes to execute at this point in the process. Impact comparisons, in both time and accuracy, by using different techniques are discussed further in 4.3.2, with results presented in 5.2.

4.2.3 Scheduling & Training

The next step was to explore the most time-consuming part of the workflow, i.e. the scheduling and training of the deep learning tasks. Utilizing the example code, that was constructed by the engineers, the training experiment was run on both the personal workstation and on the more powerful multi-GPU compute node, in order to fully experience both methods of scheduling and training.

However, before this is done, it is necessary to discuss the thoughts and issues the engineers have with the hardware and software that is used. Version numbers and specifications are presented in detail in Section 4.1, and are therefore absent from this section.

4.2.3.1 Hardware

For deep learning tasks it is primarily the graphics cards that are of interest, since these allow for faster and more efficient computations of these workloads [116]. If it is desirable to perform the training in a shorter time span, the solution has been to equip the machines with more and/or faster GPUs. However, since it is only possible to fit four graphics cards inside most computers, this kind of vertical scaling is limited in how much that can be achieved.

Furthermore, the most powerful hardware is disproportionately expensive for the performance offered, especially in comparison to the consumer grade equipment that is used in the current workstations [117]. If all engineers are to be given equally powerful computers, it becomes prohibitively expensive in the long run. This is one of the reasons why there exist only a few shared nodes that has the additional compute power. Having the hardware running at full utilization is also more economical than allowing them to idle, which would happen if personal multi-GPU workstation where provided.

The engineers are not worried about the current computational limitations of the local workstations, since it is more desirable to perform long running experiments on the powerful nodes either way. Additionally, having this work offloaded to another computer allows the engineers to continue to use the local one for further experimentation in the meantime. Instead, it is mainly the storage requirements that has become a larger problem.

From Section 4.2.1.2 it was noted that the entire dataset is downloaded with a significant overhead, in both size and number of files, because of the Git history and the LFS protocol. In the case of `OpenImagesV3`, it occupied around twice the expected disk space, and only half of the files were actually of interest in the experiment code. This results in low utilization of expensive high performance storage, while also causing unnecessary delays and is annoying for the users. Either larger disks are needed on all workstations, and/or there needs to be a more efficient method of storing the datasets.

4.2.3.2 Software

What was noticed, during the configuration of the software environment, was that the installation of the CUDA components is not a trivial task, and helping instructions are available

in B.8. This is also a prominent problem that was brought up by the engineers, as both upgrading and downgrading the versions is a difficult and often problematic experience. This is a more significant issue than what it may seem at first glance, and the reason is that the machine learning framework in use is required to be compiled with the version of CUDA that is currently installed. This means that these software components needs to be removed and reinstalled when a new experiment, using another framework, is to be tested. For the shared multi-GPU compute nodes this cause significant friction each time the environment needs to be reconfigured.

Furthermore, the software repositories used by different versions of Linux do not necessarily provide the same versions of all the programs offered. For example, the default version of Python on Ubuntu versus Debian is not equal, which can cause incompatibility when code is to be transferred between workstations and the instructions written by the engineers only state that Python was used.

These problems are also the cause of the current situation where reproducibility of experiments is severely limited. There are too many parameters to keep track of to recreate the environment used during development. Often the external dependencies of the software used is not recorded, hence uncertainty arise when time comes to install older versions again.

The issue of reproducibility is something that the engineers have been discussing for a while, but have not yet agreed on the best solution. One alternative proposed was just to record all the version numbers of all the software all the time, but the fast and agile development process used does not encourage the user to do so. The obligation to constantly write in a journal would easily be forgotten by the engineers, or it may risk slowing down the development. Furthermore, if the version of the software used is not available on the current OS's repositories, it is unfeasible to expect that the engineers should compile it themselves. Therefore, another method of packaging the current environment is needed, one that constantly saves the state and can be effortlessly saved for the future if reproducibility is needed.

4.2.3.3 Training

To better understand the scheduling part of this section, which will be presented later, we will evaluate the training procedure before mentioning how time is allocated for these experiments.

As a first step, the machine learning engineers conduct something often called *interactive training*, by themselves on their own workstations. During this part of the training procedure, the local environment is configured as the engineer prefers, and the training code is being actively modified. Usually only a small subset of the entire dataset is utilized at this stage, to get preliminary training results faster. This is conducted to have as short of a turnover time as possible, and to be able to fine tune the parameters used and make sure the network is performing optimally.

The exact details of what is being done during this step requires more in depth knowledge about how neural networks work, and how they are best trained. This thesis work does not go into detail about how these are tuned, instead the parameters used were given to us by the engineers and the software were set up to make these easily configurable. Our benchmark

uses the PyTorch framework to be able to train a couple of state of the art neural networks from scratch in order to make them able to classify images as belonging to one of ten object categories [105].

This was confirmed to be a good benchmark that would properly stress the hardware, while still being fast enough to allow us to iterate trough it multiple times to test with different settings. Tweaking input parameters, such as how many preprocessing augmentations that should be applied and how many epochs (iterations of input data) the network should be trained for, gave us a glance of the work that the engineers do and allowed us to explore the difficult relationship between these parameters. However, no further details of this will be given as it is not within the scope of this thesis work.

Referring back to the issues discussed regarding the software used, it was noted that all the working environments of the engineers differs significantly. We experienced difficulties properly setting up and configuring the training example on our workstation, since it was not entirely identical to the environment from which we got the training example. Further difficulties with drivers and CUDA made the setup process take almost a week in total before it worked as intended. However, when the network was finally able to run, the training procedure was quite simple and the results of interest were easy to read. Changing the preprocessing parameters yielded significant changes in the final accuracy of the network, and increasing the number of *DataLoader* workers in the framework made a clear difference in execution time.

After iterating trough different settings of the neural network setup, during the interactive training, the engineers usually reach a point where it is of interest to run a much larger test to get more conclusive results. For these long-running experiments it is often a requirement to offload them to the more powerful multi-GPU compute nodes. This is mostly due to higher requirements on GPU memory, but also to allow the training to complete faster. This offloading to different hardware is usually called *batch training* and is normally what produces the final results which is put into a product.

During batch training, the complete dataset is often used and the experiment will be allowed to continue uninterrupted for multiple days, if necessary. The model's accuracy at the end of the training will then be an indication into how it will perform in a real world scenario, and a more informed decision can be made regarding how to proceed. At this step a flaw might be noticed or it might not perform as expected, and the process will have to return to either the step of interactive training, or as far back as to change the dataset in the preprocessing step. The uncertainty of where the problem might originate from, require the steps performed up until now to be well documented and allow an engineer to return to any point of interest.

4.2.3.4 Scheduling

During interactive training the majority of the experiments are executed on the engineers' local computers. This limits the scheduling to simply keeping track of what is currently being run on one's own machine. Starting and aborting training sessions in this environment can be done as pleased, and the setup of the experiment can be quickly modified if it is noticed that

the result is diverging in an undesired direction. In order to notice when the model is diverging, the engineer usually monitors the terminal which outputs the performance numbers, or external software may be used to log the performance into graphs visible in the browser.

After further discussion with the engineers, it was understood that it is common for them to have a good idea of what the output is expected to be from training sessions. As such, they usually make a note of what succeeding training sessions are to be prepared, and then, depending on the result of the currently running session, they immediately know which to run next. However, it is also not uncommon for the first session to take longer than expected, and the engineer forgetting their train of thought of what should follow. This causes unnecessary delays along with annoyance and wasted time. As such, a method to automatically execute an entire matrix of training experiments and easily abort bad performers is highly desired.

In regard to the shared multi-GPU work nodes, there are additional scheduling issues present. As these nodes are limited in numbers, a booking system was put into place in order to allow the resources to be shared between users and to give the ability to reserve resources for a specified amount of time. The booking system is implemented by letting the work nodes have a dedicated calendar, where users create events to signal that the computer is occupied. A simple visualization of this system is presented in Figure 4.3.

The engineers themselves have noted that this method is sub-optimal. Users forgets to schedule an event and instead just makes sure to use an available node not currently being used, blocking the user that have scheduled the time. This, among other issues with the booking system, results in under-utilized hardware, risk of losing of training results and communication difficulties among the engineers which gets worse as the team grows in size.

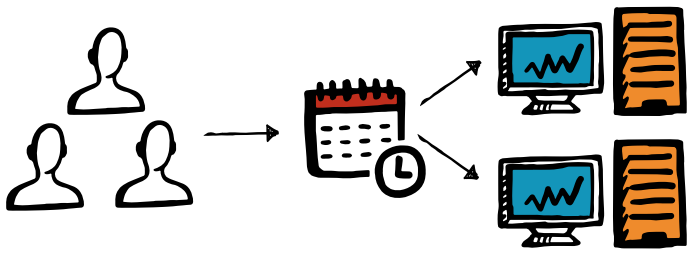


Figure 4.3: The powerful compute nodes are booked by entering an "event" into a shared calendar that was set up for this specific purpose.

This issue can be traced back to the original problem where the engineers are making a subjective decision regarding if a training has reached a satisfying target. Instead, a method which is able to queue more tasks, and stop them based on performance, would solve many of the expressed problems. In the current situation, experiments that finished early will idle the computer until the booked time slot expires or a new user comes along.

4.2.4 Summary of Current Workflow

The current situation has now been explained in detail, and an answer to the first research question from Chapter 3, i.e. what are the current processes for data storage and data processing, can be summarized with the following items.

1. Assemble dataset along with metadata.
2. Upload dataset with Git LFS to Artifactory and keep metadata in Git.
3. Install software dependencies for training.
4. Download dataset and metadata using Git LFS.
5. Perform preprocessing (if separate from training).
6. Select training parameters.
7. Start training.
8. Monitor performance and stop manually.
9. Upload resulting model if satisfactory or reiterate from previous step.

These processes differ depending on if one is performing *interactive training* on the personal workstations or *batch training* on the dedicated computation nodes. But overall these are the discrete steps that are performed in end-to-end training of a machine learning model currently.

Additionally, an answer to the second research question, i.e. which parts of processes listed cause significant inefficiencies, is also starting to become clear since some of the above steps are more time-consuming and cause more busywork than others. Currently, for the day to day work, the upload and download method and the training and scheduling cause very large inefficiencies which are presented in Chapter 5.

4.3 Proposed Solutions

The current machine learning process has now been performed from the beginning to the end, with no modifications, just as the engineers are used to doing it. This section discusses possible solutions to the problems outlined in Section 4.2, and describes which alternative solutions that can be implemented at the case company. Like with Section 4.2, the naming of the subsections follow the structure defined by Figure 3.2. Solutions, and information about them, are to be presented in the order of the workflow, with the results of the improvements announced in Chapter 5.

4.3.1 Storage

After the difficulties discussed regarding the versioning and storage of the datasets, from Section 4.2.1.2, it is clear that the current storage solution is incapable of managing these large sets of data, and needs to be replaced. Currently, the amount of load the data retrieval process

place on the network and backend servers negatively impacts other departments, which is why the engineers have requested that focus is placed on finding a sustainable solution to this issue.

Additionally, a new storage solution would also directly affect the ease of introducing other improvements. In fact, some of the researched solutions require certain storage technologies, and might contribute more significantly to the overall improvement than the storage solution itself.

4.3.1.1 Desired Features

To limit the search scope of potential storage solutions, the initial research only looked at three primary points of interest. Our hope was to locate a limited number of solutions that fulfilled all the requirements, which could then be more thoroughly tested to see how much the overall efficiency improved.

First, and most importantly, the desired functionality of the storage solution had to be decided. The second point of consideration was maintainability. The engineers at the video analytics department is not expected to maintain their own storage, which meant that the solution either requires low degrees of maintenance or allows the IT department to take over operation. Finally, the usability of the storage client itself was evaluated to make sure that it was able to be integrated with the training environment. In addition, both storage reliability and cost efficiency was also taken into consideration due to being fundamental parts of a long term solution.

1. **Functionality**
   The functionality evaluation started with looking into the most suitable type of storage technology. In Section 2.8 the four major storage technologies of relevance are presented. Analyzing the structure of the given datasets it was noted that they contain large amounts of files of which most are smaller than half a megabyte in size. However, in conjunction there are a few files in these that are approaching the gigabyte size range, which is expected to grow as the department starts processing more video. Additionally, there is no standard form of naming convention or file structure between the data, as it is usually only enforced by the one constructing the dataset.

   This variation of data composition made a database storage solution a poor fit, especially concerning the scalability of storage space. The unstructured nature of the datasets could be better handled by NoSQL type database systems, which would allow for easier sharding and distribution of the database. However, sharding is a non-trivial task and some of the more promising alternatives, such as Cassandra [118], have limits on item sizes which can be stored. This combination could lead to undesired restrictions in the future.

   Block storage provides the best available performance when accessing the data repeatedly with small reads and writes. However, the datasets are usually only uploaded once in its entirety, and then repeatedly downloaded, making this performance increase less of a factor in our case. More importantly, block storage solutions have a tendency to

become prohibitively expensive as the data grows making it an unattractive option in the long run.

In contrast, object storage solutions looks to be the most promising considering the functionality offered. Firstly, it allows easy distribution of data across multiple machines. Secondly, this technology allows diverse types of data, with the added benefit of allowing metadata to be integrated inside the object itself [91] [119]. Comparing this technology to classical file storage systems it has much greater possibility to scale to future functionality and storage needs.

Since the engineers see themselves using all the storage space available to them, scaling of storage space becomes the most important factor regarding choice of solution. Additionally, since the data used is usually only written once to the long term storage, but read multiple times, object storage was deemed the best fit of the explored storage technologies.

2. **Maintainability**
   Maintainability is the second point of interest. The engineers have extensive knowledge regarding machine learning and other forms of data modeling, however, they lack experiences in maintaining storage solutions. A better use of resources would be to assign the storage solution to the IT department. Another option could also be to utilize a cloud storage service. However, this exposes new problems related to GDPR and other legal issues. As such, an on-premise hosted storage solution is therefor currently the only available option and should, in the best case, be operated by the IT department. With this said, it is important to look into how easy the recommended solution would be to maintain and what the IT department demands of a solution to be willing to take over daily operation.

3. **Usability**
   Regarding usability, we equate this to evaluating the client or `API`. With this we mean the interface provided by the storage solution, to allow external programs to communicate with it and obtain desired data assets. If the data of interest is stored on a file system, as it is with file storage, the `API` would mean the assignment of the desired file's path inside the program that would like to obtain the data. This would also be the case with block storage, since the most common solution is to place a file system on top of the virtual block device that is created.

   Using file paths, as a method of obtaining data, is what most people are familiar with and are comfortable using. Of the machine learning frameworks that have been studied, all expect the data to be available on a local file system by default. This makes file storage solutions the most compatible method in this regard, since a NFS share would allow for easy mounting of the remote file system. However, this is not a very sustainable model for expanding the storage capacity in the future as a large number of files would potentially slow down the system or hit limits in the file system or block devices.

   In the case of object storage there exist multiple different interfaces that are less known than the file system. The three major cloud storage providers, which offer object storage solutions (Amazon, Google and Microsoft), implement their own RESTful API [89] which use HTTP methods such as `PUT`, `GET` and `DELETE` to interact with objects stored.

Between these, the Amazon S3 API [91] has grown to become more or less the de facto standard [120] when it comes to cloud storage. Python, the preferred language used for machine learning at Axis, has an official software development kit [121] which allows simple interaction with data assets stored on solutions offering S3 API access. As a result, object storage solution that offers S3 API will be more valuable for the engineers than other less widely used APIs. An S3 API also has the added benefit of making migration of solutions into the cloud easier, if this should become a possibility in the future [122].

Databases are another form of data storage that provides implementation specific interfaces, often called query languages [123], that are used to ask the database to find, update or retrieve specific data. Structured Query Language (SQL) is the language used to interact with popular relational databases, such as MySQL or PostgreSQL, while the column based Cassandra database has its own language called *The Cassandra Query Language* (CQL). These query languages allow more advanced data retrieval options, but also require more setup and knowledge to interface with. In conclusion, it is therefore in our opinion better to focus on the S3 API, instead of these query languages, as this allows easy scripting without any database knowledge.

To summarize, it was decided that an *object storage solution*, with *S3 API* support, would provide the best features to combat the current storage problem. Such a solution would scale to however much storage would be required while also allowing easy migration to the cloud. However, it will be necessary to coordinate with the IT department, at the case company, to make sure they are able, and willing, to deploy and maintain our suggested solutions.

4.3.1.2 Researched Solutions

Originating from the three most prominent feature specifications outlined in Section 4.3.1.1, the following storage solutions have been researched and evaluated:

- Arvados [124]
- Cassandra [125]
- Ceph [26]
- GlusterFS [126]
- Minio [28]
- MooseFS [127]
- Pithos [128]
- RiakCS [129]
- RiakKV [130]
- SeaweedFS [131]
- Swift [27]
- *Proprietary Solution*

Before proceeding, a clarification is required here regarding the *Proprietary Solution*, which is mentioned at the very end of the list above. At the case company there is currently a different storage solution deployed, and maintained, by a third party company whose real name was requested to be omitted. This company also offers a closed source object storage solution. Since all other alternatives we have researched are open source, this particular solution will henceforth be referred to as either *Proprietary Solution* or simply *Proprietary*.

Preliminary analysis of the advertised capabilities of these solutions made them all seem promising. However, the most desired feature we were searching for was an object storage solution. Looking in Table 4.5, the feature set of all the aforementioned solutions are listed.

Table 4.5: Storage technology feature comparison.

Storage Solution	Storage Technology			
	File	Block	Object	Database
Arvados	x		x	
Cassandra				x
Ceph	x	x	x	
GlusterFS	x			
Minio			x	
MooseFS	x			
Pithos			x[1]	
RiakCS			x[2]	
RiakKV				x
SeaweedFS	x			
Swift			x	
Proprietary			x	

Eliminating any solution that does not provide object storage functionality leaves seven possible alternatives. Out of those remaining Pithos will be discarded, as this project prominently displays a notice that it is not under active development [132]. Similarly, one should be wary against the two Riak solutions since the parent company has recently declared bankruptcy [133].

The remaining solutions offer a varying set of interface options, which are displayed in Table 4.6. *Filesystem* refers to access to the storage solution via a direct interface into to the client's filesystem using either NFS or CIFS/SMB. *Block Device* allows the storage solution to expose the available space as raw block devices, which allows for similar manipulation as a disk volume attached to the client system. *Amazon S3* and *Swift API* are RESTful APIs that are used to communicate with supported object storage solutions. These RESTful APIs provide easy interaction with the stored items with not just the officially supported SDKs, but every HTTP capable solution.

In Table 4.6, all the solutions that claim to support Amazon S3 are listed. However, none of them follow the official specifications perfectly, with even Amazon themselves having deviations from what they have defined as their API [134]. However, the ones listed indicate where they differ from the official documentation [135]–[137], Arvados does not, which is why it will be discarded.

[1] Uses RiakKV as backend.
[2] Uses Cassandra as backend.

Table 4.6: API availability comparison.

Storage Solution	API/Interface			
	Filesystem	Block Device	Amazon S3	Swift API
Arvados	x		x	
Ceph	x	x	x	x
Minio			x	
Swift			x	x
Proprietary			x	x

Comparing Minio to the remaining alternatives unfortunately place it at a disadvantage. The reason is that the philosophy of the Minio project is to be a single tenant service. This means that the recommendation is to limit each Minio server to a limited amount of people or a single application [138]–[140]. This is often the preferred solution in a cloud environment, which is why Minio considers itself to be *cloud-native* [141]. Unfortunately, it is less suitable when one does not have any ability to abstract away the hardware infrastructure as easily as in a cloud environment. For now this will provide too large of an obstacle to maintain as a standalone service.

With the remaining three solutions it was finally important to evaluate the considerations from the IT department in order to make sure they had the ability to take over the daily operation. One important aspect put forward was to consider what kind of support could be expected in case of problems or issues. In this area both Ceph and the Proprietary Solution have the development and support of large respectable companies while Swift is developed and supported by the OpenStack Foundation [142] [143]. As such they are all actively maintained and well-supported with an advantage to the Proprietary Solution as its surrounding infrastructure and support agreements are already in place at the case company.

4.3.1.3 Implemented Solutions

Having limited our search of possible storage solutions down to two alternatives, which we felt confident in recommending, we contacted the main person responsible for the storage at the case company. During discussion, it was then revealed that an S3 compatible object storage solution had been requested by multiple departments before, but, due to other more critical issues, no investigation into a possible solution had yet begun.

With help from our research a consensus was formed that the testing of the proprietary solution would be made a higher priority. Negotiations with the third party storage company was immediately initiated, and a fully functional trial version of their object storage solution was deployed at the case company. Important to note is that the hardware allocated for this trial is below the minimal recommended configuration for a production environment. The storage company informed us that their solution will not perform optimally in this configuration, but should provide an opportunity to ensure that all features necessary are present.

At the time of writing it seems likely that the Proprietary Solution will be available as a production ready offering operated by the IT department in the coming six months. It is both a highly requested feature and the solution that can be deployed in the least amount

of time, in addition to having the best support of all options analyzed. A figure showing the function of the production ready setup is visible in Figure 4.4. The multiple backend storage servers would theoretically allow for a higher throughput.

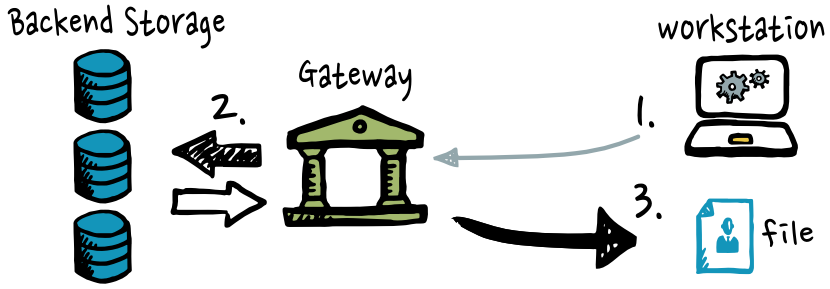


Figure 4.4: Overview of the production ready object storage solution with the Proprietary Solution.

However, continued discussion with additional people inside the IT department have raised concern that this proprietary storage may become unnecessary expensive for the storage requirements that are expected in the future. Based on the recommendations from our research, there is therefor also a project in place which aim to implement and evaluate a Ceph storage cluster in parallel to the proprietary solution. This is a larger task, compared to the externally supported proprietary solution, and no functional implementation was able to be brought up for testing before the end of this thesis work. However, their preliminary evaluation does indicate that, over time, it would be a more economical solution for large data storage that does not require the same amount of external support.

Due to not having the ability to test against a Ceph based solution a temporary solution involving Minio was instead utilized. This is to provide comparable numbers regarding what transfer speeds an object storage solution on real hardware, is capable of, since the Proprietary Solution will be somewhat limited in its current setup. Additionally, our Minio setup disregards data integrity and redundancy in favor of faster access and transfer speeds to better represent the performance of a full production setup.

Another reason to utilize Minio for additional testing is to extend the results with the differences between methods of retrieving the data. Object storage solutions require the use of a client software to download the data assets and, as can be seen in the Results chapter, a significant improvement in download times can be achieved by using another client and tweaking the settings. Thus, Minio will be used as a side comparison to more thoroughly test the clients' performance allowing us to issue a recommendation regarding the client setup as well.

4.3.1.4 Storage Solution Summary

Our research and testing suggest that the deployment of either Minio, Ceph or Proprietary Solution at the case company would solve half of the problems raised by the engineers in 2.5d. In our experience the transfers to and from the long term storage has been much easier and faster, while also completing without errors. These solutions introduce no extra storage overhead on the clients meaning the disk size required is drastically reduced.

By using the Amazon S3 API specifications, it is also possible for the clients to attach the metadata of the images into the data item that is uploaded. This could be an appreciated feature when data is to be annotated in the future. Furthermore, the suggested solutions can be configured to allow for easy versioning of the objects stored. Importantly, when a new item is uploaded with the same path as an already existing one, the old data will not be deleted, but rather hidden and replaced and may be accessed by specifying version number in the `GET` request. There is also no method of performing partial rewrites of uploaded items as each change requires the entire item to be replaced further enforcing simpler versioning of the files.

4.3.2 Preprocessing

Without diving too deep into the training code itself, it was found that there is not much to be altered in the current process regarding the preprocessing tasks. One method which improves the accuracy and robustness of an image classifier model is to artificially inflate the training data through simple, or advanced, augmentations [62]. The most common augmentations, presented in Section 2.2.1, are both known and used by the engineers, meaning that our only suggestion is to continue with the current processes.

Having studied the PyTorch implementation in particular, the available augmentation functions are applied just after the original data has been read from disk. This significantly reduces the amount of physical storage space needed, with the important drawback being that the exact data used by the network is random in nature and reproducibility is lost. This is a known problem, and to combat this PyTorch allows a `seed` to be set for the generator used to create randomness. From this seed the random generator will produce the same sequence of numbers on succeeding runs. This will allow for reproducibility if the input seed is used, but may differ between different versions of PyTorch [144]. Future experiments should therefor record both the seed and the PyTorch version to be able to be able to accurately trace how their resulting network came to be. Similar functionality is also available in the TensorFlow framework meaning our recommendation is also valid there.

Furthermore, both of these frameworks support batch loading of the data from disk into GPU memory, with asynchronous processes allowing multiple batches to be preprocessed simultaneously by the CPU. What needs to be assured is that there are enough worker threads to saturate the bandwidth to the GPU(s). There are no guidelines regarding this, but a forum post suggest four worker threads per graphics card used [145]. Our results with different amount of workers are presented in 5.2, but needs to be evaluated more in depth for the case where multiple GPUs per computer are involved.

Regarding preprocessing tasks that are done in the early creation stages, such as collecting and sanitizing the datasets, these are left for future projects to improve. The reason for this is that there is currently no consensus on how data should be collected and annotated, and creating such a workflow is too complicated to be completed in the time for this thesis work.

4.3.3 Scheduling & Training

From the beginning of this thesis work, our most ambitious goal have been to make it possible for the machine learning tasks to be automatically scheduled and run in parallel on a distributed compute cluster. To allow such automation one requires the tasks themselves to be both schedulable and agnostic towards the underlying hardware and software setup, which are two of the problems expressed by the engineers in Section 2.5. Our hope was that solutions in these areas would also allow easier ability to run the training tasks in a compute cluster.

The engineers have been looking at complete system solutions which would allow better distributed training efforts for a long time and were surprised that no solution outside cloud environments seemed to exist. Our research did not find a complete answer to this but the many difficulties with large scale data processing, particularly with multimedia data, has been outlined in greater detail in Chapter 2. One thing is however clear, the components of a system solution seems to have surfaced very recently as is indicated by the solutions outlined by the related works in Section 2.3. Thankfully many of the solutions presented in the background chapter have been made available by open source efforts finally allowing complete system solutions to become available to the general public.

A good indication of how far the efforts have come is *Alchemist* [29], i.e. the system allowing easy, fast and scalable distributed deep learning by Apple previously presented in Section 2.3. Published in November 2018, it arose at the later stage of our research into possible solutions and outlined a complete end-to-end system for easy training of deep neural networks. In this paper the authors utilize object storage in combination with container technology to create reproducible and portable training jobs. This is then combined with the scalability and management capabilities of Kubernetes to allow easy scheduling and distribution of the jobs across a compute cluster. In the end a full system is presented almost entirely built from open source components.

Luckily, the case company had recently deployed Kubernetes cluster of their own allowing the possibility to test similar solutions. In order to be able to test the training example needed to be package in a container allowing us to explore how well that solves the problems of software versioning and what performance penalty could be expected.

4.3.3.1 Software

From Section 4.2.3 it was identified that the most trouble and delays, associated with scheduling and training, stems from incompatibility issues between all the software components. The engineers mentioned that the most frustrating part is managing the software dependencies which are necessary for the GPU acceleration of the training frameworks. As these dependencies require a reboot after reinstallation of a different version they are both time-consuming and limits any ability to run different versions in parallel.

Since the engineers prefer to use different frameworks, and those frameworks require different versions of CUDA, it is easy to understand the time wasted on reinstalls do eventually

add up. If the department is to continue to scale up their efforts in the future, this will become prohibiting factor, as the shared computers will have to be reconfigured more often. The incompatibilities also makes it difficult to collaborate with colleagues, since they may require disruptive modifications to the local environment in order to run.

On a similar note, it was noticed that the software dependencies are often not recorded along with the training code in the version control system. Unfortunately, it is the interactive training that is partly to blame for this, as it encourages quick and small incremental changes without enforcing that all required parts are in place in the version control system itself. This allows a lot of flexibility for the engineers themselves, but makes it difficult to recreate the experiments on the powerful workstations and collaborate on the same experiment. Consequentially this also results in poor, or nonexistent, methods of reproducing the results of earlier versions.

These issues could be solved by utilizing a solution that can create isolated environments for each experiment, and allow them to run in isolation without modifying the local environment. These environments should also be able to be portable between computer hosts, and/or stored for long periods of time, to better satisfy the requirement of reproducibility.

One solution, that fulfills the above mentioned requirements, would be to use virtual machines. This is a well know and proven technology that creates a virtual environment, isolated from the main system, inside which it possible to install a completely separate operating system. Having multiple virtual machines running in parallel would allow for many different software environments to coexist on the same hardware. These virtual machines can be stored and transferred to a new host without modifying the guest system. This allows for very good reproducibility and portability, however, such a solution infers computational overhead, as an entire virtual operating system is running in addition to the programs themselves. Furthermore, this solution will generate a significant amount of additional data that needs to be stored, since each virtual image has all the files of an entire operating system included.

A more modern method, with similar results, would instead be to use *containerization technologies* [71]. These create isolated environments on a program level, instead of the operating system level, making the overhead smaller at the sacrifice of compatibility regarding the underlying system. This means is that the programs running inside the container must be able to run on the operating system that is installed on the host machine, i.e. a Windows native program can not run inside a container on a Linux system without some form of translation layer in between. See Figure 4.5 for a simplification on how the different isolation solutions works.

Examples of such containerization solutions are LXC [146], LXD [147], OpenVz [148] and Docker [22]. Out of these, Docker has risen to become the most popular, while also being one of the best supported solutions to deploy on a Kubernetes cluster. The wide adoption of this open source tool makes for isolated environment with relatively low overhead. The setup consists of install the *Docker daemon* on the host system (instructions in Appendix B.1), and all containers compatible with the native operating system should be able to run without issues. Additionally, isolation on application level makes it possible to have two containers, running in parallel on the same host, which are running on completely different software environments. When a container finishes it can be completely removed from the host, which subsequently removes any trace of it.

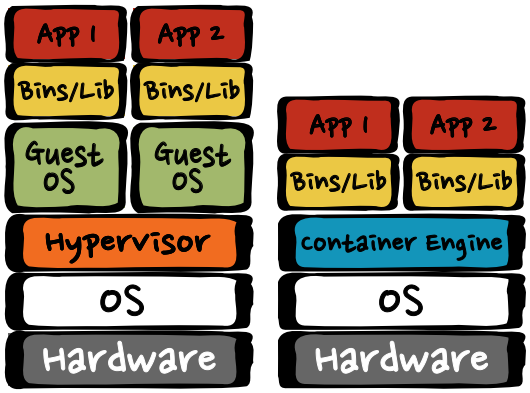


Figure 4.5: Visualization of the principle behind a program running inside a virtual machine (to the left) and a program running in a container (to the right).

One of the best features of Docker in our opinion is the method of how this isolated container environment is created by the user. The Docker daemon builds the entire container from instructions that are explicitly written inside a recipe file, which in this case is called the `Dockerfile`. Thus, the engineers will have to write down all installation steps, and all modifications performed, into this file to be able to have the desired environment in which the experiments should run. Even though this may require additional efforts by the engineers, it will be a significant improvement regarding both reproducibility and portability resulting in much easier collaboration. The `Dockerfile` is also suitable for inclusion along with the code itself in the version control system used since it is essentially a simple text file. By including this `Dockerfile` in the same repository as the experiment code, the issue about reproducibility and portability would be almost entirely solved.

Another positive aspect of using Docker is that Nvidia provides official support of having the GPU dependencies themselves also be isolated in the container environments [69]. The host will still be required to have the GPU drivers installed, but since CUDA is essentially a runtime binary utilizing the hardware via the drivers it has no problem being isolated in the container environment. This means two experiments, using different frameworks that require different CUDA versions, may run in parallel on the same host machine. Making it possible for multiple containers, that require GPU accelerated computations, to coexist on a single powerful multi-GPU node, which may be included in a Kubernetes cluster.

Through this single feature, one of the most irritating and time-consuming steps, of setting up the training environment, has been removed. The only thing that might be of a concern is how much of an overhead this containerization technology introduce on the experiments. The isolation methods used by the Docker daemon may make the execution of the experiments slower, however, we believe that this overhead will be severely outweighed by the advantages offered. Furthermore, this solution allows one improve the training processes further through distribution with a Kubernetes cluster. But, for this to be possible, it is also necessary for the code, running inside the containers, to be autonomous enough to both be started by another program, and have a clear termination point. Otherwise, the container will continue to live indefinitely on the cluster.

4.3.3.2 Scheduling & Training

Even at the relatively small scale, at which the engineers are currently working at, the issues of having to manually schedule experiments have become noticeable. During the interactive training phase the experiments are started without any triggers determining completion, and it is up to the engineer to continuously monitor training metrics and manually terminate training once satisfactorily values has been reached. This is a prominent source of wasted resources, since either the engineer is occupied with monitoring the output or the training will continue longer than necessary, risking degrading the models' ability to generalize while prohibiting the system to be utilized by other experiments.

To allow the experiments to be able to run in a distributed environment, and be automatically scheduled, the first step is to prohibit the manual intervention in the termination of the training efforts. Additionally, some method of automated monitoring, and triggers on certain metrics, would allow the engineers to focus their attention on other areas. Examples of metrics that are important to monitor are the current epoch and the accuracy of the model. An automated system could potentially stop the model when it reaches a certain number of epochs, or earlier if a certain level of accuracy is achieved, thus allowing other experiments to run. The next experiment should ideally be started as quickly as possible, to increase the overall utilization of the expensive hardware.

In searching for solutions to the above mentioned issues, taking guidance from the Apple paper [29] discussed in Section 2.3, an open source solution called *Polyaxon* [70] was found. This framework is, in the words of the creators, "a platform for reproducing and managing the whole life cycle of machine learning and deep learning applications". It is deployed on to a Kubernetes cluster, and acts as an abstraction layer from the otherwise complicated command line interface of pure Kubernetes management, even going as far as abstracting away most of the container implementation as well and implementing its own recipe files.

In order to offer a complete end-to-end solution, to the workflow of the machine learning experiments, Polyaxon is composed of multiple smaller services in a microservice architecture, utilizing the many features available in both Docker and Kubernetes. In contrast to pure Kubernetes, it only exposes a simple command line interface along with a good-looking web interface towards the end users. The web interface allows the ability to log in and monitor, or stop, currently running tasks, and get results of completed experiments. A complete system overview is visible in Figure 4.6, and a full set of screen captures are available in Appendix A.

The workflow promoted by Polyaxon is constructed to allow very flexible *interactive training*, just like it was previously, i.e. the engineers manage their own personal environment and iterate and modify the code as they wish while monitoring it in the terminal. The main difference is how the *batch training* is managed, executed and monitored.

Explained in short, the Polyaxon framework utilizes its own version of a recipe file called `polyaxonfile.yml` which specifies how to run the experiment or group of experiments. Here one specifies if performing a group or single experiment, resources to request (Compute node, CPUs, memory, GPUs, etc.), data sources and outputs, what command line arguments to pass to the application, what Docker environment to use as a base, what extra dependencies to install and how to execute the experiment. With the experiment or experiment group thoroughly defined it is then uploaded to the Polyaxon scheduler. The scheduler makes sure

that a container is built according to the specification and made available, a compute node that meets the resource request is found and retrieves the built container and as a final step is started according to the inputs given.

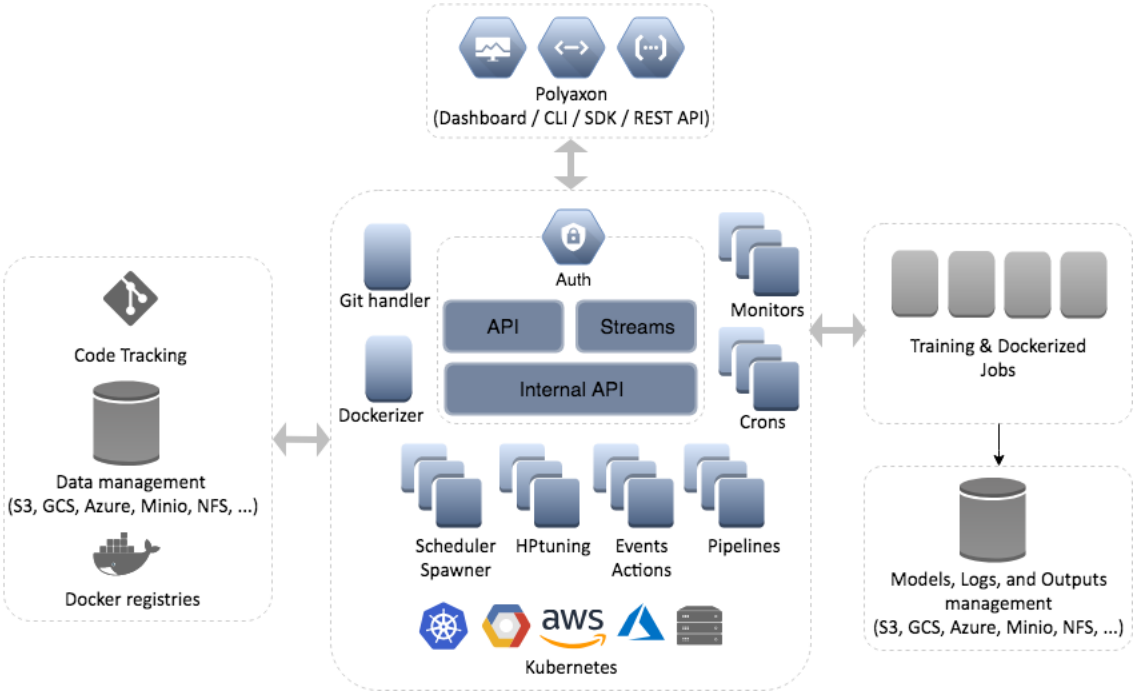


Figure 4.6: Image providing an overview of the architecture of Polyaxon from the official documentation [70].

If the requested resources are unavailable or simply missing the experiment is placed in a queue, waiting to be built and transferred, until a matching node is made available. When running, the Polyaxon Client will continuously report back to the scheduler how the monitored variables are progressing, until the cutoff point is reached this is done by utilizing the official Python Module and telling Polyaxon which values to monitor in the training code itself. All the metrics are then displayed, in almost real time, with graphs on the user facing web interface, making it easy for engineers to see how the training progresses over time. This provides a better overview of how the different parameters change the overall results of the model, and allows for faster comparisons between different training sessions. Furthermore, it is possible specify an entire experiment group in the recipe file directly, making it much easier to schedule the entire set of input parameters one wishes to evaluate. Polyaxon will then schedule all specified combinations of the parameter intervals defined. An example use case which benefited our own evaluation was to test how the number of `DataLoder` worker threads, from 1 to 4, affects both a large and a small dataset. This can be setup by simply defining the range of 1-4 as the specified `DataLoder` and the two datasets as input variations. From this it will automatically create eight experiments to encompass all the possible combinations.

This will significantly speed up the testing methodology for the engineers, as this is a process that is often undertaken developing new machine learning models. As a good example of how this could improve testing in the future, we decided to test how well different neural

networks could be trained on the `CINIC-10` image set. Initially we started each test manually, and waited until it had finished before starting another. Then we tested the same procedure again, but this time with Polyaxon as being responsible for starting all the tests. Unexpectedly, this was much more effective as is seen in the results of Chapter 5.

With the monitoring and scheduling now having a working solution for *batch training* one wonders if similar systems could be set up to provide more control over the *interactive training* as well. Currently, this is not possible but the authors of Polyaxon has started development of the ability to run major parts of Polyaxon without having any Kubernetes dependency. This would allow many monitoring and scheduling features to work in a single node environment.

4.3.3.3 Hardware

One of the most significant aspects affecting the training process is the hardware being used. The classical method of making the training task faster is to invest in faster and more powerful hardware. However, this is costly to do every time a new revision is released, and having to do it for every engineer's workstation will force compromises on the budget. At the same time this hardware is mostly underutilized, since when the engineers are not actively training, the hardware is sitting idle. This is known and as such there are a few very powerful, multi-GPU workstations that are shared by the entire department which they have to book in a calendar to be able to use.

By leveraging Kubernetes a much more robust infrastructure could be constructed as the inclusion of new hardware, either on premise or in the cloud, would be very fast and only requires network access to the master node of the cluster. This means that all the hardware utilized for *batch training* could be managed by the IT department instead of the engineers themselves. This would potentially make it easier to make sure that the hardware is both properly cooled and placed physically closer to the storage servers, further decreasing the turnaround time of each experiment. The one in charge of the hardware could also be allowed to monitor the degree of utilization and order more hardware as the queue of experiments starts to become too long. The Co-locating tasks that are CPU intensive with those who are GPU intensive on the same node would also allow for maximum utilization of the available computational power.

Another added benefit of the cluster solution is that it allows easier distribution and parallelization of the training computation across multiple GPUs and even multiple nodes. The main factor that simplifies such horizontal scaling of hardware is that Kubernetes handles the network configuration, managing its own internal network with virtual IPs and domain names, completely separate from the outside network. This makes it much simpler to connect nodes together and as the machines themselves can be located elsewhere they can be placed in data centers with higher bandwidth communication between them.

Chapter 5

Results

Chapter 4 began with a thorough analysis of the machine learning process at the case company, in order to delineate the current workflow and thus answering the first and second research question when the most prominent bottlenecks were identified. In Section 4.3 an investigation into possible solutions for these identified problems was initiated, with a focus on finding alternatives that would be feasible to implement at the case company. The solutions that were identified as the most suitable were subsequently deployed, which would answer research question 3 and 4, and allowed further tests regarding their impact on performance.

This Chapter will answer the final research question, on how much the implemented solutions increase the efficiency of the current workflow. Similarly, to the previous chapter, the results of the improvements are divided into the sections presented in Figure 3.2. However, the Discussion, in Chapter 6, will follow up with additional points regarding the subjective thoughts of improvements from the engineers.

5.1 Storage

The Proprietary Solution was running on less performant virtualized hardware during testing, which means that the measurements obtained will not reflect the performance of a properly configured system. Therefore, its transfer times will be complimented with the results from a Minio setup on dedicated hardware, to showcase how a single node best case scenario would look like for a similar storage system. More information regarding how the experiment was set up, and what commands that were issued, can be found in Appendix D.1.

5.1.1 Speed Improvements

The first measurements made were the time it took to upload and download the datasets to and from the different server solutions. Since the client software is a part of the variables, both the `aws-cli` and the `rclone` clients, presented in Appendix D.1, were used to observe how those impacted transfer rates. The current storage solution requires the use of the Git client, with the Git LFS extension, which is why no other clients were tested with this.

All the transfer times obtained are presented in Table 5.1, and these are all measured from a single computer to the server solution in question. This was to keep outside variables to a minimum. To better illustrate the differences, the download and upload times were plotted in a bar graph for each dataset. These can be observed in Figure 5.1, 5.2 and 5.3.

Table 5.1: Transfer times to and from the different solutions with different clients. Time format is HH:MM:SS.

	`OpenImagesV3`		`CINIC-10`		`CHUK03`	
Storage Solution	Download	Upload	Download	Upload	Download	Upload
Git LFS	18:37:00	x	02:15:00	~3 days	00:12:37	00:09:24
Propr. (`awscli`)	04:27:00	07:58:00	00:09:09	00:23:28	00:00:58	00:02:42
Propr. (`rclone`)	03:40:00	09:11:00	00:04:46	00:18:59	00:00:43	00:02:41
Minio (`awscli`)	02:41:00	04:12:00	00:09:39	00:10:05	00:00:59	00:01:04
Minio (`rclone`)	01:29:00	03:57:00	00:01:59	00:03:11	00:00:05	00:00:15

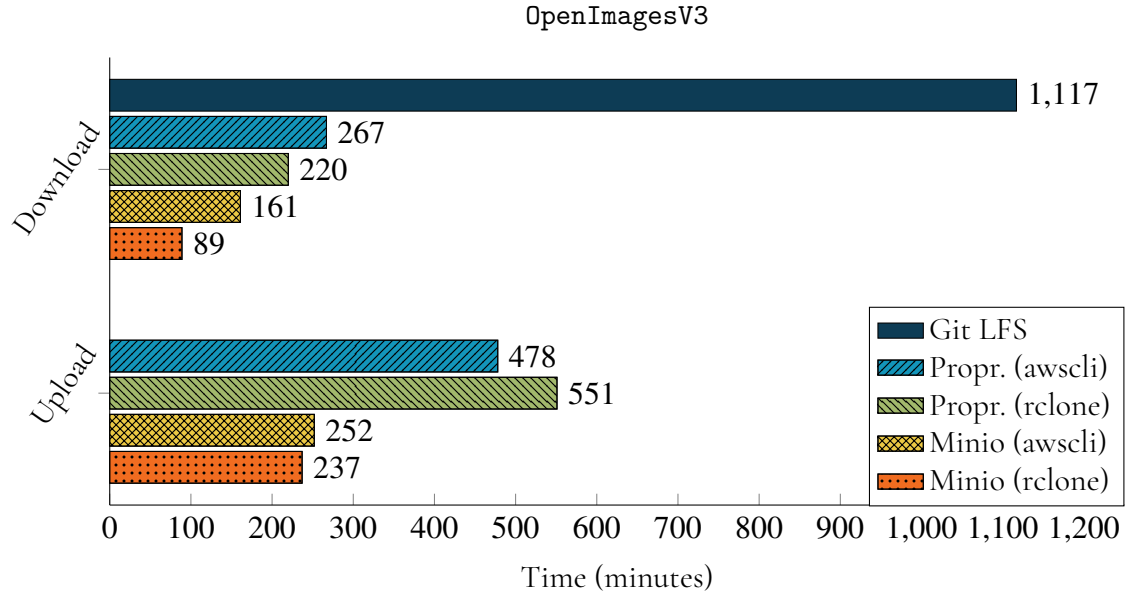


Figure 5.1: Bar graph displaying, in minutes, how long it took to download and upload the `OpenImagesV3` dataset to the different storage solutions with the different clients. Please note that the upload bar for Git LFS is absent, since it was impossible for us to complete that transfer.

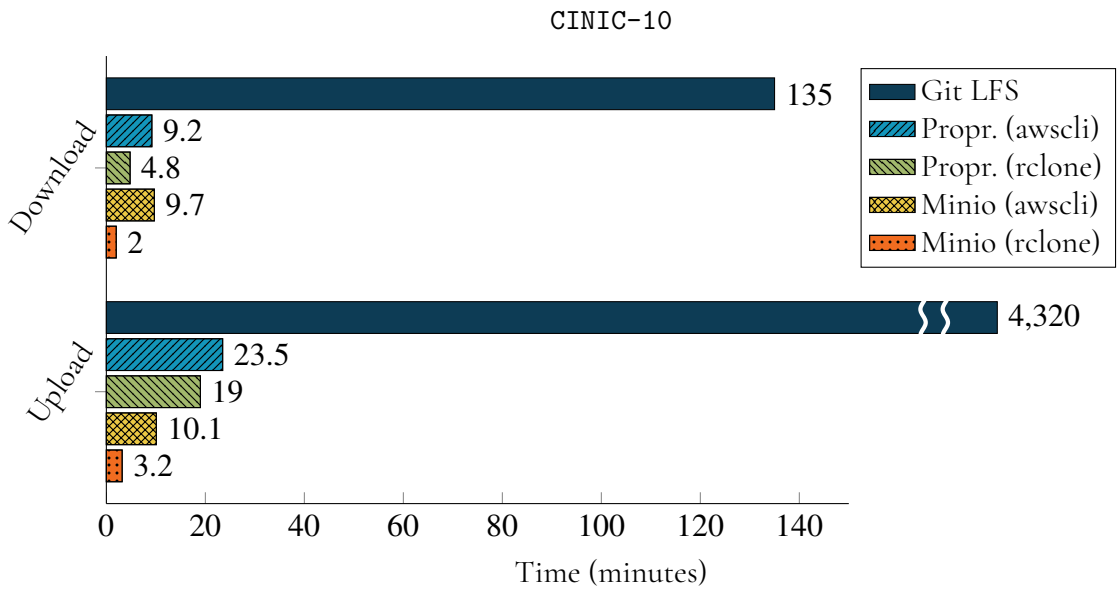


Figure 5.2: Bar graph displaying, in minutes, how long it took to download and upload the `CINIC-10` dataset to the different storage solutions with the different clients. Please note that the bar for the Git LFS upload has been significantly scaled to fit on the page.

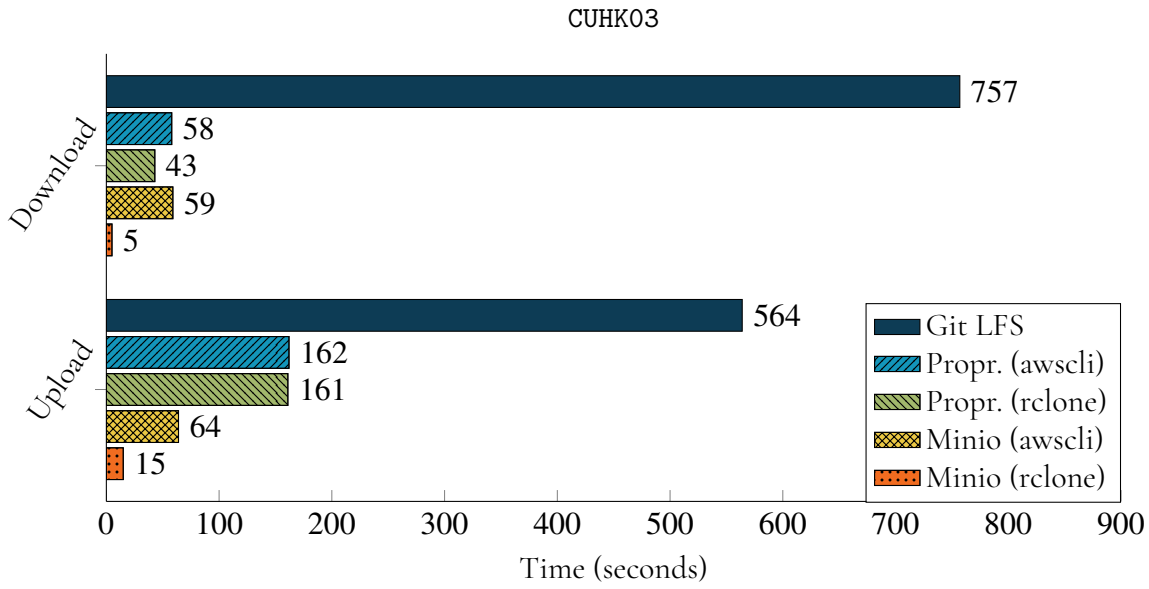


Figure 5.3: Bar graph displaying, in seconds, how long it took to download and upload the `CUHK03` dataset to the different storage solutions with the different clients.

5.1.2 Size Improvements

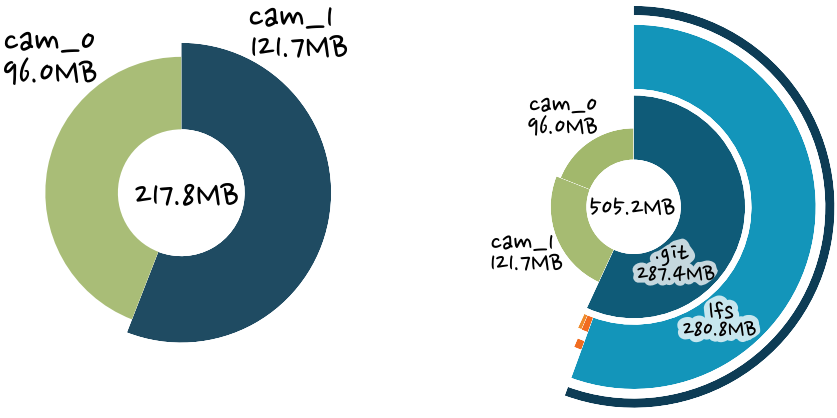
Beyond the time efficiency measurements, it was noted that the storage overhead introduced by the Git LFS extension was significant. Due to failure in uploading the largest dataset with Git LFS, the other two dataset are instead used to visualize how much extra data the LFS extension contributes. The following sunburst diagrams have been created with the program `duc` [149], which is able to calculate the total size of a folder on a computer, and then display this along with the sizes of any existing subfolders.

In Figure 5.4a, a visualization of the sizes of the two subfolders `cam_0` and `cam_1`, that together compose the `CHUK03` dataset, can be seen. In total these two folders occupy 217.8 MB of disk space, and out of this the `cam_0` folder is responsible for 96 MB, while `cam_1` takes up 121.7 MB. The circular sectors, that are adjacent to the folder names in the figure, are then representing the percentage of the total size each folder is responsible for. In this case `cam_0` is responsible for ~44%, while `cam_1` contributes ~56% of the total size. The exact percentage numbers are not important, as these diagrams just serve as visual aid for the reader.

After downloading the same dataset from Git LFS the 287.4 MB large `.git` folder is added inside the root folder, which can be seen as the blue part in Figure 5.4b, and brings the total size on disk up to 505.2 MB. The circular sectors are then shrunk to indicate that `cam_0` is now only responsible for ~19% of the total size, and `cam_1` is ~24%, with the `.git` folder now being responsible for ~57% of the total space taken up on disk.

Inside the `.git` folder there are two further subfolders, called `lfs` and `objects`, whose names are more easily discerned in Figure 5.5b. These are visualized by adding a circular layer further out from the center. The circular sectors of these subfolders are then also proportional to the percentage of the total size of the root folder.

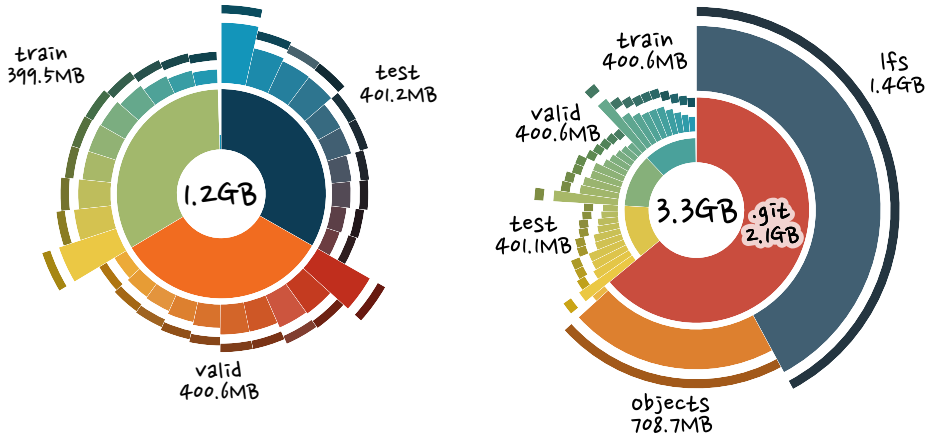
In Figure 5.4 the visualization of the relative sizes of each folder that compose `CHUK03` can be seen, while Figure 5.5 shows the `CINIC-10` dataset. The names on most of the subfolders have been omitted in favor of more easily readable diagrams.



(a) The raw size on disk.

(b) Size on disk after Git LFS.

Figure 5.4: A visualization on the relative sizes of the folders that are included in the `CHUK03` dataset before and after being downloaded from Git LFS.

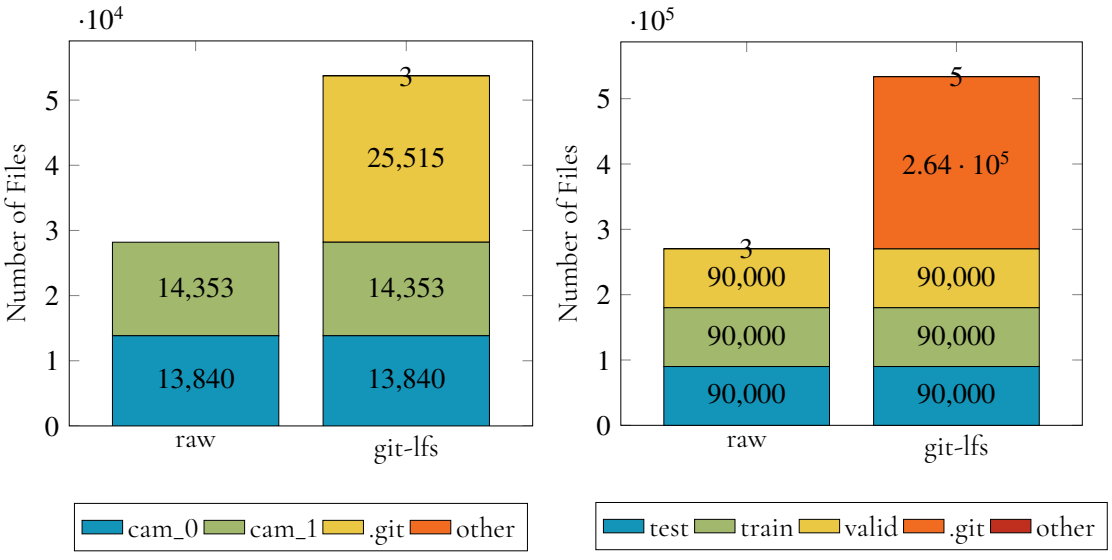


(a) The raw size on disk.

(b) Size on disk after Git LFS.

Figure 5.5: A visualization on the relative sizes of the folders that are included in the `CINIC-10` dataset before and after being downloaded from Git LFS.

As can be seen, the total size of the folder containing the dataset has almost tripled in both cases, after the dataset was downloaded from Git LFS, with the `.git` folder being the culprit. This folder contains information about the versioning of the files and will store the history of changes, which means it can only be expected to grow in size as more versions of the data are tracked. In Figure 5.6, a stacked bar graph is presented of the number of files inside each of the main folders before it was uploaded and then after it was downloaded from Git LFS.



(a) Number of files in `CHUK03`.

(b) Number of files in `CINIC-10`.

Figure 5.6: A stacked bar graph of the number of files inside the folders that are included in the `CHUK03` and `CINIC-10` dataset, before and after being downloaded from Git LFS.

For both datasets the total number of files almost doubled, with the `.git` folder being the one responsible. Comparing this to the object storage solutions, the downloaded data only contains the raw data itself, from the chosen version of the dataset, and none of the overhead is present on the client side. However, on the server side the files uploaded are encapsulated inside an object which is slightly inflated as it incorporates accompanying metadata. Furthermore, the service may want to store multiple copies of the objects, to guarantee high availability and integrity, which will multiply the total storage requirements. This is transparent for the user, and will only be a concern for the server maintainers.

5.2 Preprocessing

As the number of alterations done to the data, before it is being trained on, was limited to two simple augmentations, the results here might not be completely representative of all forms of modifications. Some augmentations not mentioned may inflict a significantly higher performance impact, but those studied are still deemed to be representative of the current situation at the case company. The method used to isolate the impact of these preprocessing steps can be read in D.2.

The only parameter tuned, during this testing, was the number of worker threads used when loading and augmenting the input data. The training example was then left to iterate over the entire `CINIC-10` dataset, for the default of 300 epochs, after which the final accuracy was recorded along with the time it took for the training task to complete. Our results are presented in Table 5.2.

Table 5.2: Time to complete 300 epochs of training for the `VGG16` object classifier network, on the `CINIC-10` dataset, with different amount of worker threads. Last column is the resulting accuracy.

Nbr of workers	TTC	Accuracy (%)
1	04:13:00	84.82
2	03:12:00	84.74
3	03:25:00	84.83
4	03:16:00	84.50

To demonstrate how these small augmentations to the input data impacts the accuracy of the trained model, another training process was done without any augmentation to the input data. In Table 5.3 it is possible to compare how much the accuracy was impacted when these augmentations are turned off. Both of these tests were performed with two worker threads for loading and augmenting the input data.

Table 5.3: Accuracy of the `VGG16` object classifier network after 300 epochs with and without preprocessing augmentations to the input data. Both are using two worker threads for data loading.

Augmentation	TTC	Accuracy (%)
None	03:22:00	76.14
Rotation & Cropping	03:12:00	84.74

5.3 Scheduling & Training

It is necessary to preface this section by mentioning that improving the scheduling and training required adjustment of multiple variables that interact with each other in intricate ways. These dependencies, on other components, make it unfair to judge the performance of every individual alteration, introduced in Section 4.3.3, by itself since some may make one part of the process slightly slower while then making the process significantly more efficient when combined with another tool.

The improvement results are therefore presented here as isolated as possible, to then be discussed in Chapter 6 in their entirety. The order of how the results are presented is kept the same as in Section 4.3.3, and information about how the measurements were made can be found in Appendix D.3.

5.3.1 Software

Introducing Docker to the engineers provided a major change to the workflow, as it required the engineers to learn a new tool that affects how one develops and deploys the model training. At first, this is expected to cause longer development times, until enough experience has been gained and the benefits of the tool becomes noticeable.

Modifying an environment without Docker, by following the guide written in Appendix B.8, took about 30 minutes. Comparing this to having the Docker daemon download and build the same environment inside a container only took about 5 minutes resulting in a six times decrease in environment setup time.

Running the code inside a container does affect the time it takes to execute. In Table 5.4 are time measurements of how long it took for different object classification models to complete 300 epochs running on the bare metal (Native) versus inside a container (Docker). The percentage difference, how much slower/faster the containerized code is, is presented in the final column where a positive number equals more time added, i.e. slower. These differences are also presented in a bar graph in Figure 5.7, where the total execution time is converted into minutes.

Table 5.4: Comparison of the training times of different models, using the `CINIC-10` dataset, running on the native operating system versus inside a Docker container.

Model	Accuracy	Training Time		Difference
		Native	Docker	
VGG16	84.74%	03:12:00	03:40:00	+15.6%
ResNet18	87.45%	04:41:00	04:24:00	-6.0%
ResNet50	88.40%	15:12:00	15:27:00	+1.6%
ResNet101	88.42%	24:57:00	25:19:00	+1.4%
MobileNetV2	83.99%	06:19:00	14:50:00	-2.4%
ResNeXt29(32x4d)	88.67%	14:21:00	14:59:00	+4.4%
ResNeXt29(2x64d)	89.09%	14:06:00	14:47:00	+4.8%
DenseNet121	88.58%	16:34:00	17:03:00	+2.9%
PreActResNet18	87.01%	04:19:00	04:19:00	+0.0%
DPN92	88.18%	38:33:00	43:19:00	+12.3%

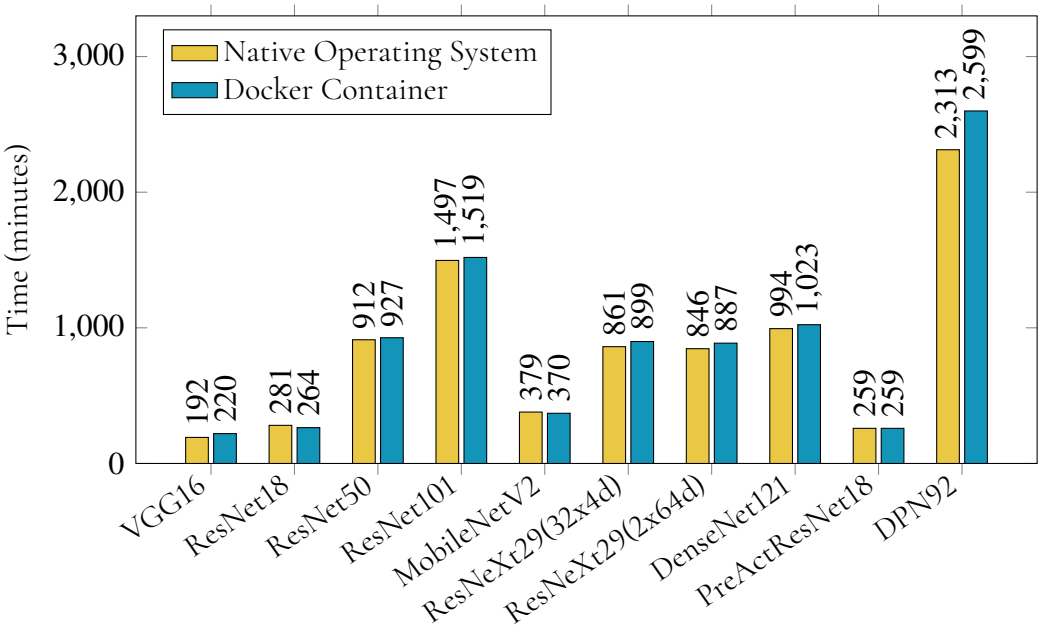


Figure 5.7: Bar graph displaying, in minutes, how long it took for the different network models to complete 300 epochs of training on the `CINIC-10` dataset, running on the native operating system and inside a Docker container.

What should also be mentioned in this section are the improvements in portability and reproducibility, however, these can not be measured in quantifiable time. Instead this is considered to be more of a binary result, since previous solutions did not offer easy portability or reproducibility, while the suggested improvements does. These features have been shown to allow both easier collaboration and troubleshooting, meaning it can potentially save a lot of time in the long run.

5.3.2 Scheduling & Training

Adding up the raw numbers, of how long each Docker experiment from Table 5.4 took to run, the minimum time it would be possible to complete all of them is 8,967 minutes, or slightly more than 6 days. Since it was impossible for us to align the beginning of the next experiment perfectly with the end of the previous, the actual process of completing that table stretched on for 12 days. This included using a remote connection on weekends to start experiments from home, which is probably not something the full time employees are interested in during their free time.

Defining each and every experiment, previously run with manual scheduling, in a way that would be schedulable by Polyaxon, allowed us to complete everything in about the time it took for the two longest running experiments to finish, i.e. just under 3 days. A simple graph of the total difference in completing the experiments is available in Figure 5.8.

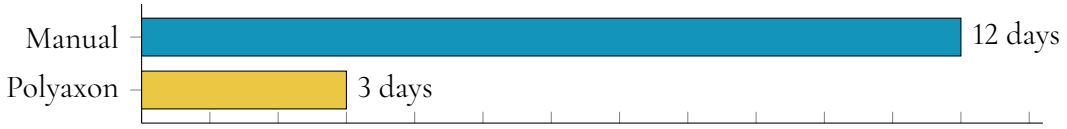


Figure 5.8: Time to execute all containerized training experiments, from Table 5.5, by manually scheduling versus utilize Polyaxon's scheduler.

The reason to why Polyaxon is able to complete it under the theoretical minimum time is because it has access to five GPUs, which is further explained in the setup procedure in Appendix D.3. This was not considered a problem, but rather a display of how much more efficient the process can be made with a software scheduler instead of manually starting each experiment.

The time it took each individual experiment to complete is visible in Table 5.5, together with the previous results of running them as a single instance on a single computer within a Docker container. Here it is important to point out that some experiments took significantly longer to complete within the Polyaxon environment, which turned out to be because of thermal limitations with the computer used for batch training (i.e. the one outlined in Table 4.2) due to having too many GPUs without adequate cooling. The results from `ResNet101` and `DPN92` where scheduled on the less powerful computer (i.e. the one in Table 4.1) and did not experience thermal issues and as such performed much better.

Table 5.5: Comparison of the training times of different models, using the `CINIC-10` dataset, running on inside a Docker container on a single host versus in the Polyaxon proof-of-concept solution on multiple hosts.

Model	Training Time		Difference	Thermal Issue
	Docker	Polyaxon		
VGG16	03:40:00	05:01:00	+36.8%	yes
ResNet18	04:24:00	05:03:00	+14.8%	yes
ResNet50	15:27:00	21:28:00	+38.9%	yes
ResNet101	25:19:00	25:58:00	+2.6%	no
MobileNetV2	06:10:00	07:23:00	+19.7%	yes
ResNeXt29(32x4d)	14:59:00	26:13:00	+75.0%	yes
ResNeXt29(2x64d)	14:47:00	32:13:00	+117.9%	yes
DenseNet121	17:03:00	31:05:00	+82.3%	yes
PreActResNet18	04:19:00	06:23:00	+47.9%	yes
DPN92	43:19:00	46:22:00	+7.0%	no

Something that is also important to point out is that, for each experiment running on Polyaxon on the Kubernetes cluster, the `CINIC-10` dataset was downloaded each and every time a new test was started. This was done to simulate that the nodes in the cluster will not always have this data locally cached, and will need to fetch it from a central storage solution.

To show the impact that the download procedure may have on the process, the download times are combined with the training times for all experiments and visualized in Figure 5.9. The left bar for each experiment is the time it took to download the `CINIC-10` dataset with Git LFS and run the training with a normal Docker container setup. The right bar is the time it took to download the same dataset with `rclone` on our Minio setup (which is not even visible due to taking < 3 minutes) and train on the Polyaxon cluster. Once more it is noted that all experiments except `ResNet101` and `DPN92` had thermal issues as mentioned previously.

Furthermore, the Polyaxon framework also keeps track of input arguments, used to define the experiments, as well as record all the output created by the tests, with the ability to review them through the web interface. This means it is possible to return to old experiments to see how those compared to new ones. This will also provide easier cooperation between colleagues as results are more easily accessible by everyone.

Screenshots on how the overview of running, stopped and finished experiments looks is visible in Figure 5.10 and the single experiment view in Figure 5.11. Additional screenshots are located in Appendix A.

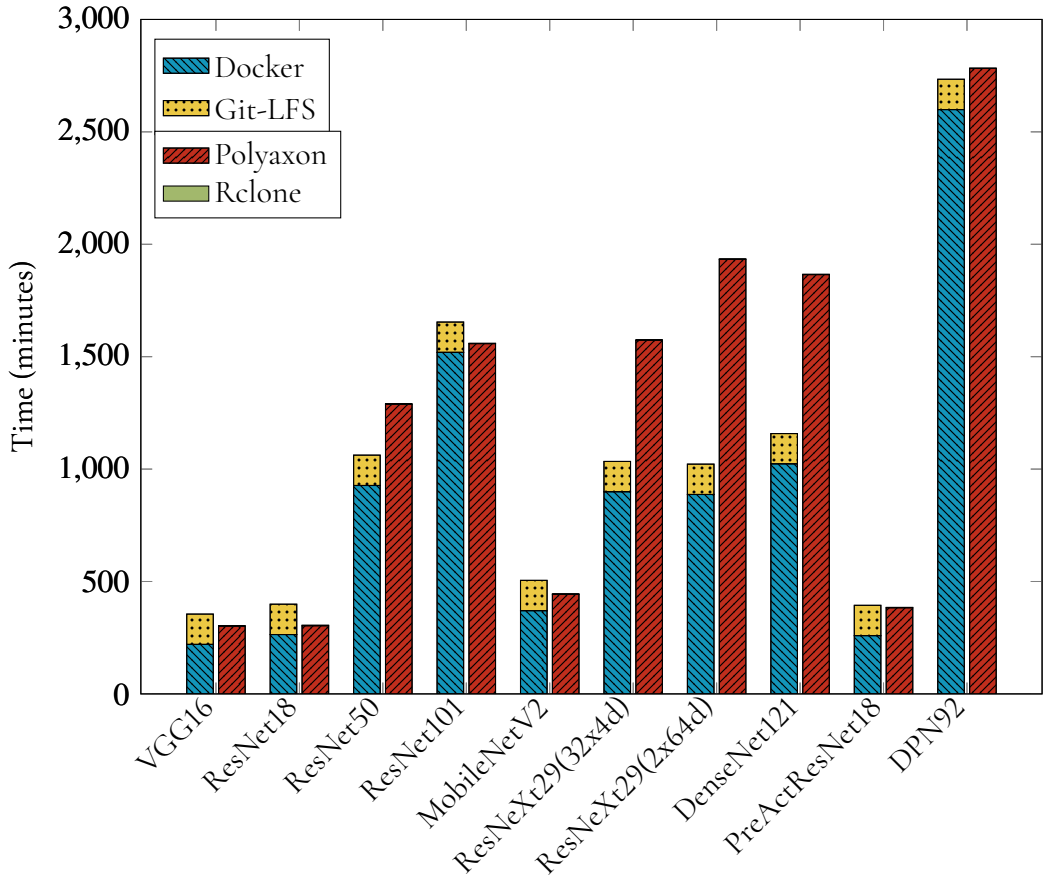


Figure 5.9: Bar graph displaying, in minutes, downloading with Git LFS [113] and training with Docker [22] versus downloading with Rclone [150] and running with the Polyaxon [70] proof of concept.

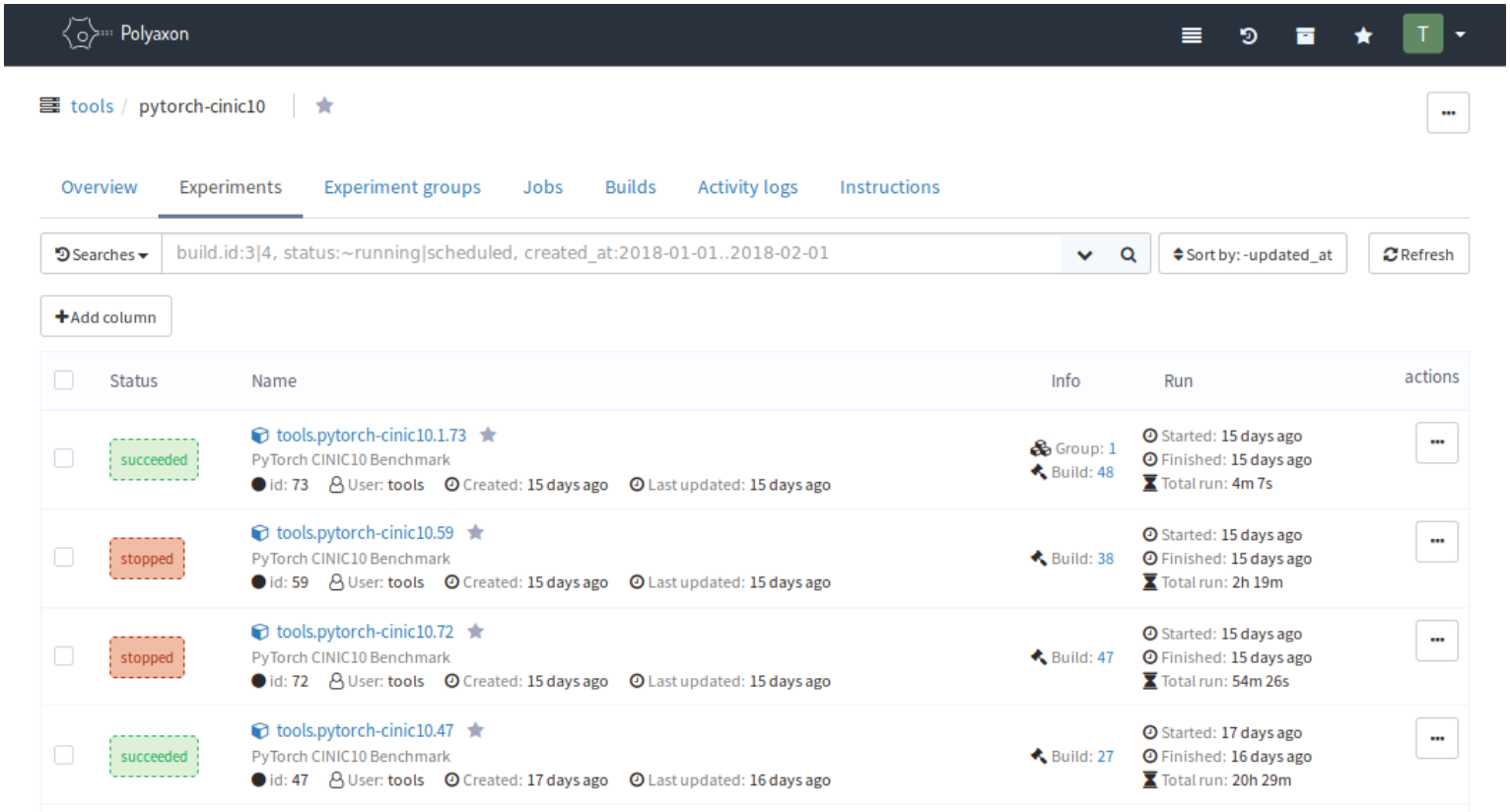


Figure 5.10: Experiments view of the Polyaxon framework [70].

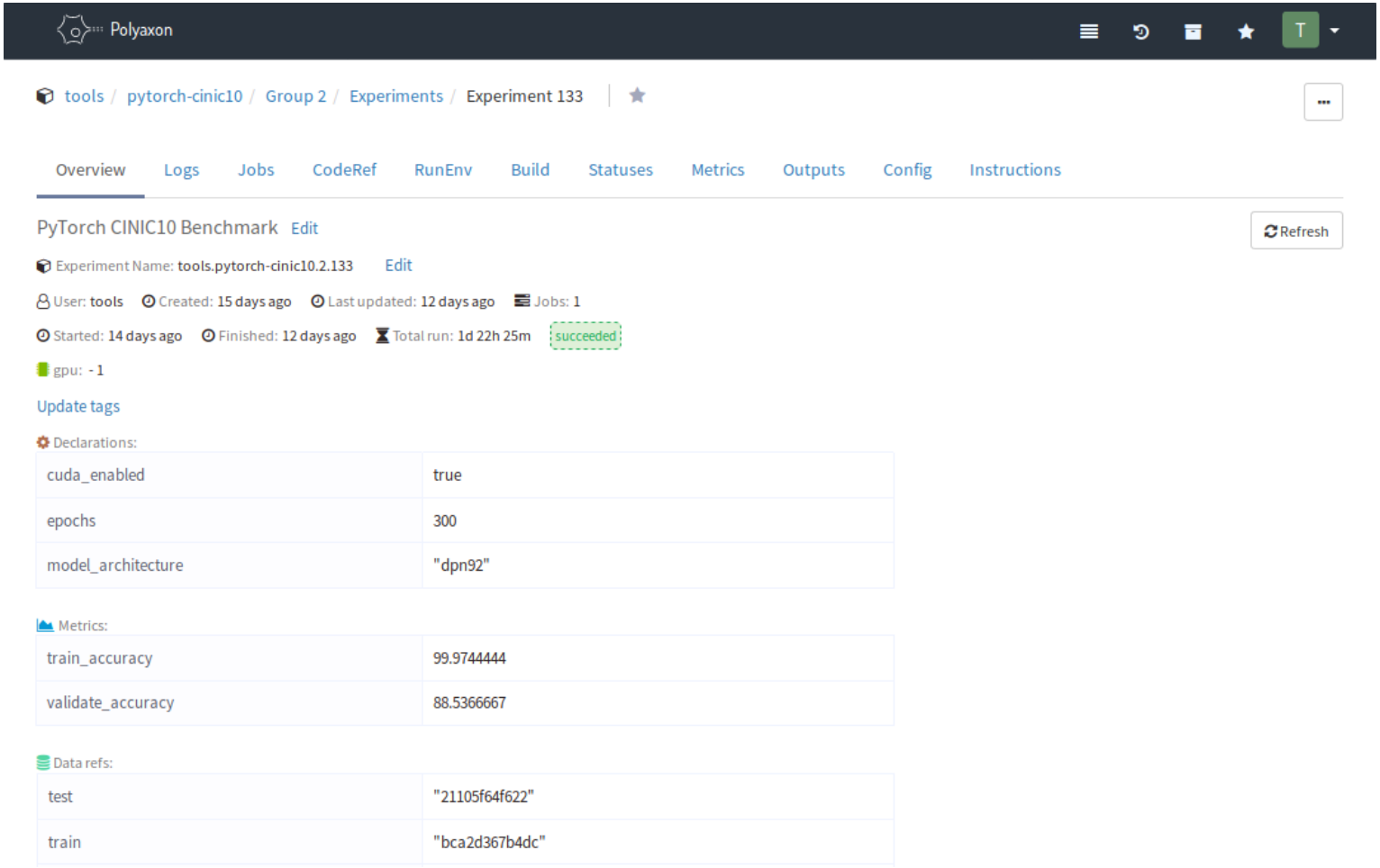


Figure 5.11: Overview of a single experiment in the Polyaxon framework [70].

5.3.3 Hardware

Hardware is a more difficult topic to define a time measurement on. The most concrete example of improvement here would be how the hardware is sourced. As it is now, the video analytics department have to order the computers via the IT departments which retrieves the hardware and sets up the initial environment. The hardware is then moved by the engineers to their own offices where additional setup is performed. This entire process could be much improved and streamlined if the IT department handled the entire process of ordering, setup and integration into the cluster while the engineers simply specify their hardware requirements in a few lines of code, which should be possible with Kubernetes.

Having the powerful hardware included in a Kubernetes cluster would also allow for other departments to take advantage of the available compute resources, which means pooling of compute resources allowing better handling of spike utilization. By offloading the maintenance of the cluster to the IT department will also entail that the machines used are included in a shared budget possibly allowing for fewer but more powerful nodes, instead of multiple less powerful ones for each department doing machine learning.

However, no concrete results are able to be presented here as they would require more thorough investigation and discussion between departments.

Chapter 6

Discussion

In the Analysis chapter, the current workflow was thoroughly analyzed and the most prominent inefficiencies were identified. Possible solutions to these were researched, and subsequently implemented in order to obtain measurements on how these affected the turnover time of the training experiments. The succeeding results were then presented in Chapter 5, with the explanation on how these were obtained in Appendix D.

This chapter will start out with a discussion about the results from Chapter 5 and continue with a recap of the research questions and goals of the thesis work.

6.1 Storage

The first component of the machine learning process that was analyzed was the storage solution for the input data. The current Git LFS solution was not able cope with the amount of files and data that was being stored, which was very noticeable during transfers. A replacement was considered high importance by both the engineers at the video analytics department and the maintainers of the current storage infrastructure.

Multiple alternative solutions were researched, and many of them seemed promising initially. However, it soon became clear that the storage space the engineers required, for the current datasets, would be the limiting factor. The amount of data is also expected to increase substantially in the near future, which is why the aspect of scalability became the most important aspect to focus on. From this it was concluded that the only storage technology that would be able to scale to arbitrary capacities, without becoming prohibitively expensive, is object storage solutions.

The fact that other departments, at the case company, had also expressed desire for a company wide object storage solution helped when our suggestions were expressed to the IT department. It was important that solutions, intended for long term usage, could be deployed and maintained by this department for it to retain full support in the future. This is why the *Proprietary Solution* had the best chance of being implemented the quickest.

Research papers and industry leaders [6]–[8], [29] recommended that these types of solutions allow communication and data management of assets over an HTTP interface, preferably the Amazon S3 API, an argument which is further reinforced by the wide use of this API by the machine learning frameworks themselves. The S3 specifications defines desirable features, such as metadata integrated into the objects, easy version management and powerful permission controls. Referring back to the problem description in Section 2.5, the engineers specifically commented on the issues with versioning the datasets and not having a strong link between the metadata and the actual data. These two problems are possible to solve with an Amazon S3 capable object storage solution, which also has the additional benefit of being a centralized solution capable of restricting access on the object level, thus allowing better compliance with GDPR legislation.

Another point that was brought up by the engineers was the difficult in uploading and downloading the current datasets by using the current Git LFS solution. During testing of our suggested object storage solutions, not a single transfer had trouble completing or experienced any errors, something that could not be said about the current Git LFS solution. The reliability aspect alone makes the suggested storage solution highly preferable over the current one, and the suggested solution have been proved to solve the issue of transferring the datasets from storage, thus solving the issue with the highest priority expressed by the engineers.

However, a negative aspect of our suggested solutions is that there is a significant difference in transfer speeds between the client solutions tested, with each client's individual settings also playing a significant role. Without any of our special modifications to `rclone`, mentioned in B.6, the download of `CINIC-10` would take almost 28 minutes. This is 14 times slower than what is presented in Figure 5.2 and three times slower than the tweaked `aws-cli`. This means that the clients themselves do not perform optimally out of the box, and instead requires some additional tweaks which increases complexity.

The performance difference between client configurations was mainly relevant for datasets containing many small files as the difference was significantly less pronounced with datasets containing fewer larger files. However, since the overwhelming majority of the datasets used are composed of mainly small files, modifications are recommended as it provides much better performance in most machine learning situations.

What is most noteworthy is the upload times of the larger datasets to Git. The relatively small `CINIC-10` required us to upload it in batches over three whole days, even though it is only around an order of magnitude larger than `CHUK03`. The transfer times does not seem to have a linear correlation towards size, which is why we will not be able to speculate on how long `OpenImagesV3` would theoretically have taken to upload. It is therefore difficult to say exactly how much an object storage solution improves the transfer times over the current git LFS solution, since the failure of uploading `OpenImagesV3` would equate to an infinite speedup. However, this would not be representative of the average use case, which is why we will base our improvement statistics on the transfer times of the `CINIC-10` dataset from the

Proprietary Solution with the `rclone` client. Even though the trial setup is in an underperforming configuration, the current download time is reduced from around two hours to a mere five minutes, a 28 times speedup. This is very respectable, and may move towards the 67 times speedup Minio was able to achieve, for the same test, with a more optimal configuration.

This improvement is also coupled with a significant reduction in the size that the dataset takes up once downloaded to the client. The `.git` folder, used by the Git LFS client, inflated the size of everything three times the raw values, and doubled the amount of files. The object storage solutions, and their clients, do not add anything on top of the raw data of interest, which makes for a much better user experience. This will also allow easier use of larger datasets in the future.

Besides the obvious improvements in reliability, scalability and transfer speeds, this type of solution has also opened up the possibilities for other quality of life improvements for the engineers, which we are not able to quantify in time. The most impressive example is a *Dataset Browser* that was developed, by the engineers, after it became clear that an Amazon S3 object storage solution would be deployed. This tool indexes available images, along with attached metadata, and allows for significantly faster searches over the datasets than the older file system based approach, with additional tagging functionality. This software has made it possible to, for example, find all images of cars, provide the download links and then compose a new specialized dataset based on these results. The search service is accessed via a web browser, and the images that match the search query are displayed on the same page for the engineer to check that the metadata is actually correct. This feature is made possible thanks to the fact that the storage solution use an HTTP interface, and the web browser can access the stored data through the same interface as all other assets on the page. Solutions such as these greatly improve the workflow of the engineers and the new storage solution allows more possibilities over the current solution.

The flexibility of this solution will make a significant difference, not only for the departments performing machine learning, but for any developer that is interested in long term storage of vast amounts of data. Similar solutions has been highly anticipated, and it seems certain that an object storage solution will be rolled out to be available to all engineers at the case company in the future, especially among the teams performing Big Data analysis.

6.2 Preprocessing

As was stated in Section 4.2.2, preprocessing is a wide topic that include a large amount of tasks that can be performed on the data before it is being trained on. However, the scope was limited to only show how two augmentations could be used to extend the dataset and how these impacted the performance of the training.

In accordance with other research [34], [58], our experiments showed that by only using two simple augmentations a significantly better accuracy could be obtained. Table 5.3 shows that the accuracy fell more than eight percentage points when all augmentation were turned off. Comparing the time it took to complete the tests it can also be noted that the two alterations that are made have no discernible impact on the computation time. The minor differences between the two runs may be attributed to the dynamic speed of the CPU and GPU based on

thermal and power limitations, or what kind of random alterations are made. The differences are deemed to be within the margin of error and augmentations should therefore be used simply for the improvements in accuracy gained.

Another discovery, was that the default settings, in the PyTorch framework, only used one worker thread to read data from disk and apply the specified augmentations. This was not enough to saturate one GPU and more than two did not showing any additional benefits according to the numbers presented in Table 5.2. However, by going from the default settings, to using two worker threads, gave about a 33% increase in execution speed, which reduced the time to complete with 25%. Interestingly, the saturation level of the GPU could easily be monitored as the GPU utilization increased close to 100%.

Since this process contain randomness, small variations in the final accuracy and time to completion is expected, but it is the difference in the execution time between one and two worker threads that is of interest. That difference is significant, which is why our recommendation is that this setting should be changed from the default. Details on how the difference scales with more GPUs will have to be more thoroughly researched in the future.

The main concern related to this topic was how reproducibility can be ascertained when randomness is involved. This was solved through the use of a seed to the random number generator that is built into the frameworks used. This will make sure that the stream of "randomness" is repeated identically between runs using the same seed.

6.3 Scheduling & Training

It is necessary to preface this section by mentioning that improving the scheduling and training has required adjustment of multiple variables that all interact with each other in intricate ways. These dependencies, on other components, make it unfair to judge the performance of every individual alteration, introduced in this section, by itself.

A prime example of such a case would be our recommendation to run the training code inside Docker containers, even though it will result in a slower execution time than running on bare metal. It is still worth recommending as it opens up the possibility to run the experiments, in a more efficient way, on a Kubernetes cluster. Furthermore, in the future, when experiments needs to be reproduced, the container environments make up a significant amount of time that previously was spent installing dependencies and debugging run-time errors.

6.3.1 Docker

In the problem description, Section 2.5, it was mentioned that a major issue with the current experiments is that they are highly dependent on the hardware and software setup, and thus not easily transferable between systems. This was also clearly noticed when trying to import the example machine learning task, given to us by the engineers, onto our local workstation. Many conflicting versions of software had to be removed and installed again, with the GPU dependencies causing the majority of the difficulties.

It was noted that the current methods of working did not provide adequate portability and reproducibility, which are paramount to allow collaboration and give legitimacy to the research being conducted. Our suggestion was therefore to introduce Docker containers, since these make the transfers of experiments between compute nodes much easier. Additionally, the fact that the environments needs to be meticulously defined in a `Dockerfile` significantly improves reproducibility, which can be further improved by saving a copy of the produced Docker image that contains all needed components of the experiment.

The introduction of Docker alone solved the mentioned issue of portability, from the list in Section 2.5, and provides an important component for solving the issue of scaling the training tasks. The fact that the GPU dependencies are able to be isolated to the container, makes it possible to transport the experiment setup with minimal friction to remote compute nodes that have access to much more powerful hardware. This is a prerequisite for being able to distribute these tasks on to a Kubernetes cluster.

Containerization is an established technology and continues to see an increase in use due to the popularity of microservice architectures for complex software solutions. Docker is not just the most widely known, but was also required by Polyaxon, the scheduling framework that is evaluated later, which is why it was preferred over alternative container solutions. As shown in the Table 5.4, there is a very slight performance impact when it comes to execution times, but since the performance impact is small it can easily be motivated by the portability and reproducibility gained by adopting container solutions.

Something noteworthy was that two models had faster execution time inside the Docker environment, versus running on the native machine. This was unexpected but shows that it the overall performance impact of the Docker isolation may be less than the random execution time differences that are inherent to each experiment. The average performance penalty, with all networks included, was ~3.5%, but this is increased to ~5.4% if the outliers of `ResNet18` and `MobileNetV2` are ignored. The worst performing experiment model, `VGG16`, was actually the one used as default for the preprocessing measurements, being 15.6% slower inside the container environment.

According to previous research [151] the performance impact should not be greater than 1% after the initial epochs where the impact could, in the worst case, be as high as 75% due to startup overhead. The initial performance impact shown by previous research could explain why the model with the shortest overall runtime also performed the worst. The large difference measured in our worst performing model could also be attributed to a difference in ambient temperature between times of measurements, since the GPUs processor speed dynamically change depending on thermal situation.

The results of the Docker performance impact are presented without taking into account the 33% increase in speed that was gained by defining the use of two worker threads, instead of the default one, during the preprocessing step. This speedup offsets even the worst case scenario here, but the limited testing with other frameworks makes us hesitant of stating that this gain will be consistent across all possible combinations.

Introducing Docker to the engineers was highly appreciated, and was immediately applicable to the current situation by eliminating the need to reconfigure the shared compute nodes every time the machine learning framework changed. The total time savings related to the

increase of this portability is difficult to appraise, as it will most likely have an even greater impact when more people are employed and actively using the shared nodes. The process of reconfiguring the environment is, with Docker, only the time it takes for the daemon to build the container, which is measured in seconds instead of the tens of minutes it takes do reinstalls and reboots.

Additionally, the software environment can be recorded explicitly by having the Dockerfile version controlled alongside the experimental code itself allowing much better reproducibility. However, the engineers did express that working with Docker introduces a significant change to how they usually work, which will require some time to get adjusted to. However, from our own experience it should not take more than a couple of days to become proficient. A compromise would be to still run everything outside of containers on the local workstations, for interactive training, when the machine learning framework itself is often kept the same. But utilize Docker for any experiment that is moved to batch training on the shared compute nodes. This ensures that the benefits of Docker is used where it has the most impact, i.e. in the collaborative and shared compute environment, while still enforcing the documentation of software environment to ensure reproducibility in the long run.

6.3.2 Kubernetes & Polyaxon

The most ambitious task undertaken during this thesis work, was the deployment of Polyaxon on the Kubernetes cluster that is present at the case company. From the research presented in the related works, Section 2.3, it was clear that the only sustainable method of scaling the machine learning tasks is by distributing them over an orchestrated compute cluster.

At the leading tech companies, outlined in Section 2.3, similar infrastructures have been leveraged to allow single experiments to be distributed across multiple compute nodes, but such an implementation would require additional efforts during both setup and cluster configuration. Instead, a middle ground was studied, where the machine learning tasks are scheduled across the entire compute cluster, but not distributed in such a way that allowed computations to utilize more than a single node's resources. This significantly reduces the complexity to a level that was manageable during the given time, but still include the most vital parts that are necessary for enabling the distribution and scaling of training tasks. With the information gained from our solution it should be easier to transition to an even higher degree of distributed computing in the future.

We would like to clarify is that the work conducted in this thesis report did not include the setup of the Kubernetes cluster itself, but instead relied on previous efforts at the case company. Deploying a production ready cluster requires large efforts during initial setup, and it is a complicated piece of software to both maintain and for developers to interact with. In order to be a solution to the scaling difficulties experienced by the engineers at the video analytics department, it is necessary to make the interaction with the cluster as easy as possible.

With that in mind, there are a few solutions available that aim to decrease the complexities for the end user by reducing the flexibility and featureset exposed by Kubernetes itself. Usually these solutions are targeted towards specific workflows, such as web development, and for a long time no such solutions existed for machine learning development. This was true until

the arrival of Polyaxon, which provides a machine learning developer just the functionality needed from Kubernetes, while reducing the complexity to acceptable levels. The potential of this framework is substantial, but in its current early state there are still some breaking bugs which was experienced by us when running it on an on-premise environment. From open source community discussion it seems like it works better when deployed on a cloud environment provided by either Google, Microsoft or Amazon.

However, after being able to circumvent the problems (with the help of the author of the framework and the open source community) and deploying it on the cluster, both we and the engineers were impressed by how much of the desired functionality was provided by this solution. The method of having the experiment setup declared in a recipe file, similar to a `Dockerfile`, and having the environment automatically be built and placed in a queue to later be started on an available compute node was exactly what was desired. The web interface allows easy access from any computer, and the queue is easily managed. In the recipe file it is possible to define what type of compute resources are required by the experiment, and Polyaxon will let Kubernetes find the first available compute node that is free and fulfills these requirements. Thus, the need for the user to manage the hardware and scheduling itself is completely removed.

The automatic scheduling significantly reduced the friction when running many different experiments after each other. To illustrate the power of this automatic and distributed scheduling, all the tasks from Table 5.4 was run, one after the other, on our local workstation with a single GPU first without the help of Polyaxon and Kubernetes. This took a long time, mostly due to the manual monitoring to see when the tasks had finished. This shows that there are difficulties managing the scheduling of the current solution with even a single compute node.

In contrast, Polyaxon was given five graphics cards across two nodes to schedule the same tasks, and was able to execute all the specified experiments as specified with minimal downtime in between. This level of efficiency is only possible with automated systems, which is why this is highly recommended to be introduced if the department wishes to scale more easily. The twelve days it took us to successfully work though the full list of experiments was completed in just three days by having them scheduled over more hardware. This is a four time speedup, and would be even greater if the longest running experiments had been scheduled to execute in the beginning.

Unfortunately, Polyaxon does not currently fix the situation of scheduling experiments on the local workstation itself. However, this is less of an issue since the interactive training environment of the local workstation is mainly used for quick experimentation where the need for automated scheduling is less crucial. Nevertheless, the developers behind Polyaxon have said that they plan to allow the entire solution to run outside Kubernetes by only utilizing Docker itself. Such a solution would allow the same functionality for the local workstation without needing to connect it to the cluster.

Still, the fast and agile development cycle used during the interactive training was deemed ill-suited for the strict scheduling restrictions that are put in place by Polyaxon. Our belief is that there needs to be a higher degree of freedom for the engineers to make more widespread changes to their local experiments with as little fiction as possible during the interactive training, to encourage exploration and curiosity. This was also something that resonated well with the engineers themselves.

Referring back to the high priority issues stated in Section 2.5, Polyaxon can be seen as the solution to the scheduling troubles, and by relying on the functionality provided by Kubernetes and Docker it is also able to take the first steps towards massively scaling the training across the entire compute cluster. Polyaxon is still in early stages, but our work provides a proof of concept of what benefits it brings to machine learning development teams that would have access to such a solution. We see this as a great opportunity to continue the research, and see what possibilities will arise as the Polyaxon framework slowly grows into a mature product.

6.4 Combination of Solutions

All the components introduced have overall made each and every part of the workflow faster. To create a final verdict, on how much the actual turnover time has decreased, the entire process of running the `CINIC-10` benchmark in both the old and the new method will be quickly summarized.

First the `CINIC-10` dataset needs to be downloaded, which was reduced to five minutes, using the object storage and `rclone`, compared to the two hours it took with Git LFS. After this it would be necessary to properly configure the environment the experiment is to run in, which used to require a significant amount of effort, but is now reduced to a simple build command given to the Docker daemon. This was a reduction from about half an hour to five minutes, which is not that significant for a single experiment, but will add up over time.

Running the training code on the computer natively is slightly faster than from inside a Docker container on average, but by using two worker threads during the preprocessing step still managed to save an additional hour over the almost four hours it took to complete normally. This would mean that the total time of downloading the dataset and executing the experiment could be finished in three and a half hour instead of six, a speedup of 71%.

This is without the added benefits of scheduling, which is difficult to summarize with a single experiment. A comparison could be that during a workday the engineers would be able to start two of the slow experiments manually before it is time to go home, while Polyaxon would be able to schedule and execute seven of our optimized tasks during a 24-hour period. This would then make a complete implementation of all our solution three times more time efficient than what the process is today.

In addition, our solutions assures that the input data is stored in a centralized solution in accordance with GDPR legislation. It also provides much better reproducibility by making sure that the seed parameter and entire software environment is documented alongside the code itself before being able to run on the compute cluster. With Polyaxon the results are available for all developers that has access and together with the other mentioned benefits the entire development process provides much easier collaboration and a better ability to scale with both additional computational hardware and additional colleagues.

6.5 Thesis Goals & Research Questions

At the beginning of the thesis work, five research questions were defined, in Section 3.2, in order to be able to identify and improve the inefficiencies outlined by the engineers in Section

2.5. From these questions the thesis goals were outlined, in Section 3.1, where it was stated that our minimum expectations were to at least document the current workflow, identify the most prominent inefficiencies and implement a working solution to one of them. The ambition was to be able to provide suggestions of solutions to all the identified issues, with the final goal being to successfully complete a fully working example of the training process with multiple of our alternative solutions implemented.

To be able to achieve the goals that were set up, we found that action research was the most suitable methodology, which works by dividing the entire process into smaller, more manageable, parts. Using this approach, the different issues were able to be isolated into separate problem realms, thus limiting the scope of the research necessary and making the discussion more focused.

The methodology also state that the researcher should be an active user of the process that is to be improved, which is why the workflow first needed to be documented in its entirety (Section 4.2) without any improvements. After this it was easier to both understand and identify the most significant inefficiencies, thus answering research question 1 and 2.

Since the issues experienced during the current workflow had already been divided into isolated problem spaces, it was straightforward to split the work into separate investigations of possible solutions for each of these. Through thorough analysis of the feature sets of the tools found, it was not difficult to identify which one that would be the most promising solution. However, the engineers had further requirements, in addition to the limitations of company resources, which needed to be taken into consideration when deciding on which solution to actually deploy. Each of the solutions presented during Section 4.3 would therefore be an answer to question 3 and 4.

Ultimately, the Results chapter answer question number 5, by providing factual time improvements, wherever possible, from the solutions that were implemented, thus wrapping up the processes of answering all the research questions of the thesis work.

6.6 Ethical Aspects

The field of machine learning is still a relatively new area of research, which is why there are many social and ethical aspects of the technology that have not yet been thoroughly discussed and had their dangers assessed [152]. There are currently valid concerns about engineers accidentally, and unexpectedly, inserting biases into the algorithms by not being careful when selecting the training data. An example would be cases were there have been efforts to remove the biased humans from the hiring process, of a company, by using an automatic filtering algorithm. However, the historical input data may have indicated that men have a higher probability of being promoted, which is then interpreted as a preferable trait in a candidate, and now the algorithm has suddenly become discriminatory against women. This is a real issue, and today there are already papers being written about what measurements that should be taken to prevent this [153].

Another topic, that is also a cause for concern, is the advancements of facial recognition technologies and the surveillance applications that are made possible with this. When algorithms

can identify a person on surveillance footage, and has the ability to track them when they are moving from camera to camera, it becomes difficult to remain anonymous when roaming public places. While there are preferable aspects with this technology, such as pickpocketers being identified and tracked across the city, human rights activists, in more totalitarian states, might be tracked and prosecuted using the same methods [154]. The technology can be used for both purposes, and it is up to the country to decide what is allowed and what is deemed illegal. But there are currently little regulation for making companies liable for their inventions, so Microsoft have publicly called for government intervention in this field in order to stop the bad practices before it becomes too late to easily impose new regulations [155].

These are all valid concerns and issues that needs to be considered by the engineers working with such technologies, however, it is not something we feel is within the scope of this thesis to discuss further. While we have worked alongside a team that do research with image recognition, we have not involved ourselves with the actual algorithms and their final use cases. We have studied tools and methods that would allow a better workflow for the engineers that work with large amounts of image data, which is not limited to just machine learning and image recognition.

We do not feel there are any questionable ethical issues with the implementation of the object storage solution, Docker containers, Kubernetes or Polyaxon. While we did make the training and preprocessing steps of the learning algorithms faster, the settings changed were not specific for just image recognition, and these can be used to speed up other learning tasks as well.

6.7 Future Work

A limitation of the proposed object storage solution is that the input data needs to be stored intermittently on the computer during training. A better solution would be to stream the input data directly as it is being used on the GPUs. The functionality needed is already implemented by the machine learning frameworks TensorFlow and PyTorch, but how this would be implemented and how it would affect the training time compared to the current setup is left to future work.

Additional testing of Polyaxon and similar solutions is also left as future work. Especially concerning distributing the execution of experiments across multiple GPUs and nodes while limiting the amount of communication overhead and infrastructure difficulties. Kubernetes and solutions such as Polyaxon should help alleviate some difficulties but more work is needed to properly test how well such distribution methods scale and how easy they are to implement on given experiments.

Additionally, more methods of producing the underlying datasets in an automated fashion are needed and solutions to this problem which leverage the benefits of object storage solutions have been difficult to find. Exploring such possibilities are also left for future work.

Chapter 7

Conclusion

At the case company there has recently been significant efforts into expanding the departments that are conducting research in the area of computer vision. However, amid the increase in both employees and computer hardware, it became clear that the current infrastructure is not able to support this sudden surge in resource demands. It was realized that it would be necessary to introduce alternative solutions that are able to manage the needs of today, while also allowing for future scaling.

Together with the engineers at the video analytics department, six high priority issues were identified that needed to be solved as soon as possible. Three of these could be alleviated by introducing an object storage solution with an Amazon S3 API. The specific choice of API was because of the version management and access controls that are defined in its specifications, and that native clients are available for most scripting languages. Even though it was not the fastest alternative researched, our suggested solution still reduced the download time by a factor of 28 for the `CINIC-10` dataset, which represents a difficult task for most storage solutions due to the large amounts of files. In concrete numbers this equated to a reduction from slightly more than two hours down to five minutes.

Data processing of multi-media files is complicated to effectively scale up. Vertical scaling of compute resources is the easiest method of decreasing the execution time for the relevant tasks, but this will reach a limit when there are no faster components available to buy. To be able to reach beyond this limit it is necessary to instead scale horizontally, which is why this was another point of interest explored in this thesis work.

It was not until recently that open source efforts, along with new research into this subject, made it possible for teams outside the industry leading companies, such as Apple or Google, to deploy on-premise solutions that could compete with cloud based alternatives. One of the important open source components, used to enable this capability, is Kubernetes, which

obtained the ability to orchestrate clusters with GPU nodes first in 2018. The fact that Kubernetes were already available at the case company allowed us to test the possibility of running machine learning tasks with GPU acceleration in a distributed and parallel fashion.

A machine learning management system, called Polyaxon, was located and deployed on the on-premise Kubernetes cluster. This framework leverages Kubernetes and abstracts the complexities of running computations on a cluster. This allowed easy scheduling of tasks, which was not just a desired feature by the engineers, but a requirement for us to be able to provide efficient distributed computing. After our testing and demonstration of the capabilities included in this system, both the requirements for scalability and schedulability expressed by the engineers in the initial problem description was considered satisfied.

However, a prerequisite which was necessary in order to utilize the capabilities of Kubernetes was Docker. This tool removes the hardware and software dependencies of the tasks that are to be executed, and allows for much easier transportation of experiments between different computers. This portability is required by Polyaxon, and a necessity when deploying any application on Kubernetes, since the programs needs to be easily transferred between nodes and isolated from each other. Docker will not only be useful in the future, but saw immediate adoption by the engineers as it significantly reduced the friction on the transportation of code between the computers at the department. This was another of the initial issue expressed by the engineers which reduced collaboration and slowed down development time. Removing the need for reinstallation of incompatible software every time a new experiment is to be run might not be too significant for each instance, but these small improvements will add up over time.

Additional speedups were possible to achieve by tuning the default settings of the machine learning frameworks used. These parameters do not have anything to do with the actual network, but rather how data is loaded and transported. With simple modifications the execution times decreased with as much as 25%, and these settings will be used in future training sessions.

Through the process of solving the issues expressed by the engineers, our thesis goals were satisfied and all the research questions outlined in the beginning of the work were able to be answered. With all our solutions combined we were able to reduce the turnover time of a standard experiment from six to only three and a half hours, i.e. a 71% speedup. This is a significant amount of time saved, which may be increased even further in the future when the solutions mature a bit more.

Bibliography

[1] A. Labrinidis and H. V. Jagadish,
Challenges and Opportunities with Big Data, *PVLDB - The Proceedings of the Very Large Data Bases Endowment*, vol. 5, no. 12, pp. 2032–2033, 2012.

[2] W. Zhu, P. Cui, Z. Wang, and G. Hua,
Multimedia Big Data Computing, *IEEE MultiMedia*, vol. 22, no. 3, p. 96, 2015.

[3] Z. Wang, S. Mao, L. Yang, and P. Tang,
A survey of multimedia big data,
China Communications, vol. 15, no. 1, pp. 155–176, 2018.

[4] A. L'Heureux, K. Grolinger, H. F. ElYamany, and M. A. M. Capretz,
Machine Learning With Big Data: Challenges and Approaches,
IEEE Access, vol. 5, pp. 7776–7797, 2017.

[5] T. Ben-Nun and T. Hoefler,
Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, *Computing Research Repository (CoRR)*, vol. abs/1802.09941, 2018.

[6] Google. (2018).
Cloud AI products,
[Online]. Available: `cloud.google.com/products/ai` (visited on 2018-11-30).

[7] Amazon.com, Inc. (2018).
Machine Learning on AWS, [Online]. Available:
`aws.amazon.com/machine-learning` (visited on 2018-11-30).

[8] Microsoft Corporation. (2018).
Azure AI,
[Online]. Available: `azure.microsoft.com/en-us/overview/ai-platform`
(visited on 2018-11-30).

[9] O. Y. Al-Jarrah, P. D. Yoo, S. Muhaidat, G. K. Karagiannidis, and K. Taha,
Efficient Machine Learning for Big Data: A Review,
Big Data Research, vol. 2, no. 3, pp. 87–93, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton,
ImageNet classification with deep convolutional neural networks,
Commun. ACM, vol. 60, no. 6, pp. 84–90, 2017.

[11] K. Simonyan and A. Zisserman,
Very Deep Convolutional Networks for Large-Scale Image Recognition,
Computing Research Repository (CoRR), vol. abs/1409.1556, 2014.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan,
V. Vanhoucke, and A. Rabinovich,
Going Deeper with Convolutions,
Computing Research Repository (CoRR), vol. abs/1409.4842, 2014.

[13] E. Battenberg, J. Chen, R. Child, A. Coates, Y. Gaur, Y. Li, H. Liu, S. Satheesh,
A. Sriram, and Z. Zhu,
Exploring neural transducers for end-to-end speech recognition,
in *2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2017, Okinawa, Japan, December 16-20, 2017*, IEEE, 2017, pp. 206–213.

[14] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger,
S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng,
Deep Speech: Scaling up end-to-end speech recognition,
Computing Research Repository (CoRR), vol. abs/1412.5567, 2014.

[15] I. Sutskever, O. Vinyals, and Q. V. Le,
Sequence to Sequence Learning with Neural Networks,
in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*,
Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds.,
2014, pp. 3104–3112.

[16] S. Jean, K. Cho, R. Memisevic, and Y. Bengio,
On Using Very Large Target Vocabulary for Neural Machine Translation,
Computing Research Repository (CoRR), vol. abs/1412.2007, 2014.

[17] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey,
Deep learning of the tissue-regulated splicing code,
Bioinformatics, vol. 30, no. 12, pp. i121–i129, 2014.

[18] M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk,
Connectomic reconstruction of the inner plexiform layer in the mouse retina,
Nature, vol. 500, no. 7461, p. 168, 2013.

[19] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman,
Project Adam: Building an Efficient and Scalable Deep Learning Training System,
in *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, J. Flinn and H. Levy, Eds.,
USENIX Association, 2014, pp. 571–582.

[20] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng,
Large Scale Distributed Deep Networks, in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*,
P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1232–1240.

[21] A. Sergeev and M. D. Balso,
Horovod: fast and easy distributed deep learning in TensorFlow,
Computing Research Repository (CoRR), vol. abs/1802.05799, 2018.

[22] Docker Inc. (2019).
Docker - Homepage, [Online]. Available: `docker.com` (visited on 2019-02-18).

[23] Kubernetes Contributers. (2014).
Kubernetes: Production-Grade Container Orchestration,
[Online]. Available: `kubernetes.io` (visited on 2018-11-07).

[24] S. Ghemawat, H. Gobioff, and S. Leung,
The Google file system, in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*,
M. L. Scott and L. L. Peterson, Eds., ACM, 2003, pp. 29–43.

[25] K. Shvachko, H. Kuang, S. Radia, and R. Chansler,
The hadoop distributed file system, in *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010*,
M. G. Khatib, X. He, and M. Factor, Eds., IEEE Computer Society, 2010, pp. 1–10.

[26] Red Hat, Inc. (2018).
Ceph - Homepage, [Online]. Available: `ceph.com` (visited on 2018-11-12).

[27] The OpenStack Foundation. (2018).
OpenStack - Homepage,
[Online]. Available: `openstack.org` (visited on 2018-11-15).

[28] Minio, Inc. (2018).
Minio - Homepage, [Online]. Available: `minio.io` (visited on 2018-10-09).

[29] M. Ma, H. P. Ansari, D. Chao, S. Adya, S. Akle, Y. Qin, D. Gimnicher, and D. Walsh,
Democratizing Production-Scale Distributed Deep Learning,
arXiv preprint arXiv:1811.00143, 2018.

[30] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison,
Hidden Technical Debt in Machine Learning Systems,
in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*,
C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015,
pp. 2503–2511.

[31] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi,
Scaling for edge inference of deep neural networks,
Nature Electronics, vol. 1, no. 4, pp. 216–222, 2018.

[32] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He,
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour,
Computing Research Repository (CoRR), vol. abs/1706.02677, 2017.

[33] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang,
Ease.ml: Towards Multi-tenant Resource Sharing for Machine Learning Workloads,
PVLDB - The Proceedings of the Very Large Data Bases Endowment, vol. 11, no. 5,
pp. 607–620, 2018.

[34] C. Sun, A. Shrivastava, S. Singh, and A. Gupta,
Revisiting Unreasonable Effectiveness of Data in Deep Learning Era,
in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, IEEE Computer Society, 2017, pp. 843–852.

[35] C. S. French,
Data Processing and Information Technology (10th ed.)
Cengage Learning Business Press, 1996, p. 2.

[36] H. Garcia-Molina, J. D. Ullman, and J. Widom,
Database systems - the complete book (2. ed.) Pearson Education, 2009.

[37] R. Kee,
Data processing technology and accounting: A historical perspective,
Accounting Historians Journal, vol. 20, no. 2, pp. 187–216, 1993.

[38] M. Stonebraker and L. A. Rowe,
The Design of Postgres, in *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 28-30, 1986.*,
C. Zaniolo, Ed., ACM Press, 1986, pp. 340–355.

[39] A. Silberschatz, M. Stonebraker, and J. Ullman,
Database systems: Achievements and opportunities,
Communications of the ACM, vol. 34, no. 10, pp. 110–120, 1991.

[40] N. Leavitt,
Will NoSQL Databases Live Up to Their Promise?,
IEEE Computer, vol. 43, no. 2, pp. 12–14, 2010.

[41] R. Cattell,
Scalable SQL and NoSQL data stores,
SIGMOD Record, vol. 39, no. 4, pp. 12–27, 2010.

[42] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik,
C-Store: A Column-oriented DBMS, in *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*,
K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P. Larson, and B. C. Ooi, Eds.,
ACM, 2005, pp. 553–564.

[43] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber,
Bigtable: A Distributed Storage System for Structured Data (Awarded Best Paper!),
in *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, B. N. Bershad and J. C. Mogul, Eds.,
USENIX Association, 2006, pp. 205–218.

[44] D. J. Abadi, S. Madden, and N. Hachem,
Column-stores vs. row-stores: how different are they really?,
in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, J. T. Wang, Ed., ACM, 2008, pp. 967–980.

[45] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels,
Dynamo: amazon's highly available key-value store,
in *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*,
T. C. Bressoud and M. F. Kaashoek, Eds., ACM, 2007, pp. 205–220.

[46] S. Chaudhuri and U. Dayal,
An Overview of Data Warehousing and OLAP Technology,
SIGMOD Record, vol. 26, no. 1, pp. 65–74, 1997.

[47] J. Dean and S. Ghemawat,
MapReduce: Simplified Data Processing on Large Clusters,
in *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, E. A. Brewer and P. Chen, Eds.,
USENIX Association, 2004, pp. 137–150.

[48] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker,
A comparison of approaches to large-scale data analysis,
in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*,
U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, Eds., ACM, 2009, pp. 165–178.

[49] Y. LeCun, Y. Bengio, and G. Hinton,
Deep learning, *Nature*, vol. 521, no. 7553, p. 436, 2015.

[50] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber,
Multi-column deep neural network for traffic sign classification,
Neural Networks, vol. 32, pp. 333–338, 2012.

[51] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang,
On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, *Computing Research Repository (CoRR)*, vol. abs/1609.04836, 2016.

[52] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young,
Machine learning: The high interest credit card of technical debt, 2014.

[53] European Union,
Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation),
Official Journal of the European Union, vol. L119, pp. 1–88, 2016.

[54] S. Garcia, J. Luengo, and F. Herrera,
Data Preprocessing in Data Mining, ser. Intelligent Systems Reference Library. Springer, 2015, vol. 72.

[55] J. Brownlee. (2013).
How to Prepare Data For Machine Learning,
[Online]. Available: `machinelearningmastery.com/how-to-prepare-data-for-machine-learning` (visited on 2019-01-04).

[56] N. S. Gill. (2017).
Data Preparation, Preprocessing and Wrangling in Deep Learning,
[Online]. Available: `xenonstack.com/blog/data-science/preparation-wrangling-machine-learning-deep` (visited on 2019-01-04).

[57] F. Chollet. (2016).
Building powerful image classification models using very little data, [Online]. Available: `blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html` (visited on 2019-01-07).

[58] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and F. Li,
The Unreasonable Effectiveness of Noisy Data for Fine-Grained Recognition,
Computing Research Repository (CoRR), vol. abs/1511.06789, 2015.

[59] S. Branson, G. Van Horn, C. Wah, P. Perona, and S. Belongie,
The Ignorant Led by the Blind: A Hybrid Human–Machine Vision System for Fine-Grained Categorization,
International Journal of Computer Vision, vol. 108, 2014-05.

[60] O. Carey. (2018).
Generative Adversarial Networks (GANs) - A Beginner's Guide,
[Online]. Available: `towardsdatascience.com/generative-adversarial-networks-gans-a-beginners-guide-5b38eceece24` (visited on 2019-01-07).

[61] T. N. Minh, M. Sinn, H. T. Lam, and M. Wistuba,
Automated Image Data Preprocessing with Deep Reinforcement Learning,
Computing Research Repository (CoRR), vol. abs/1806.05886, 2018.

[62] L. Perez and J. Wang,
The Effectiveness of Data Augmentation in Image Classification using Deep Learning, *Computing Research Repository (CoRR)*, vol. abs/1712.04621, 2017.

[63] J. Hays and A. A. Efros,
Scene completion using millions of photographs,
Commun. ACM, vol. 51, no. 10, pp. 87–94, 2008.

[64] TensorFlow Contributers. (2015).
TensorFlow Homepage,
[Online]. Available: `tensorflow.org` (visited on 2018-12-14).

[65] PyTorch Contributers. (2016).
PyTorch Homepage, [Online]. Available: `pytorch.org` (visited on 2018-11-30).

[66] Keras Contributers. (2015).
Keras Homepage, [Online]. Available: `pytorch.org` (visited on 2018-12-14).

[67] Darknet Contributers. (2015).
Darknet Repository on GitHub,
[Online]. Available: `github.com/pjreddie/darknet` (visited on 2018-12-14).

[68] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell,
Caffe: Convolutional Architecture for Fast Feature Embedding,
arXiv preprint arXiv:1408.5093, 2014.

[69] NVIDIA Corporation. (2019).
NVIDIA Container Runtime for Docker, [Online]. Available:
`github.com/NVIDIA/nvidia-docker` (visited on 2019-02-19).

[70] M. Mourafiq,
Polyaxon: A platform for reproducible and scalable machine learning and deep learning on kubernetes, Web Page, 2017.
[Online]. Available: `github.com/polyaxon/polyaxon`.

[71] Red Hat Inc. (2019).
What's a Linux container?, [Online]. Available:
`redhat.com/en/topics/containers/whats-a-linux-container` (visited on 2019-02-18).

[72] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng,
TensorFlow: A System for Large-Scale Machine Learning,
in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, K. Keeton and T. Roscoe, Eds.,
USENIX Association, 2016, pp. 265–283.

[73] NVIDIA Corporation. (2018).
CUDA FAQ,
[Online]. Available: `developer.nvidia.com/cuda-faq` (visited on 2019-02-14).

[74] ——, (2019).
NVIDIA Collective Communications Library (NCCL),
[Online]. Available: `developer.nvidia.com/nccl` (visited on 2019-02-20).

[75] Baidu Research. (2017).
TensorFlow AllReduce,
[Online]. Available: `github.com/baidu-research/tensorflow-allreduce`
(visited on 2018-12-17).

[76] Axis Communications AB. (2018).
Axis Communications Homepage,
[Online]. Available: axis.com (visited on 2018-11-09).

[77] Jupyter Contributers. (2019).
Jupyter - Homepage, [Online]. Available: jupyter.org (visited on 2019-04-07).

[78] A. Wittig and M. Wittig,
Amazon Web Services in Action, in. Manning Publications, 2015, pp. 204–206,
ISBN: 1617292885.

[79] DigitalOcean Inc. (2018).
An Introduction to Storage Terminology and Concepts in Linux, [Online].
Available: digitalocean.com/community/tutorials/an-introduction-to-storage-terminology-and-concepts-in-linux (visited on 2018-11-20).

[80] M. Nuncic. (2018).
The Evolution of Storage: File Storage vs. Block Storage vs. Object Storage – Part 1,
[Online]. Available: ontrack.com/blog/2018/02/22/the-evolution-of-storage-file-storage-vs-block-storage-vs-object-storage-part-1
(visited on 2018-11-08).

[81] Red Hat, Inc. (2018).
File storage, block storage, or object storage?, [Online]. Available:
redhat.com/ko/topics/data-storage/file-block-object-storage
(visited on 2018-11-08).

[82] Cloudian Inc. (2018).
Object Storage vs. File Storage: What's the Difference?,
[Online]. Available: cloudian.com/blog/object-storage-vs-file-storage
(visited on 2018-11-20).

[83] Red Hat, Inc. (2018).
What is network-attached storage?, [Online]. Available:
redhat.com/en/topics/data-storage/network-attached-storage
(visited on 2018-11-20).

[84] M. Rouse. (2014).
iSCSI (Internet Small Computer System Interface), [Online]. Available:
searchstorage.techtarget.com/definition/iSCSI (visited on 2018-11-20).

[85] Nitheesh Poojary. (2018).
Understanding Object Storage and Block Storage use cases, [Online]. Available:
cloudacademy.com/blog/object-storage-block-storage (visited on 2018-11-08).

[86] Techopedia Inc. (2018).
Block Storage, [Online]. Available:
techopedia.com/definition/31924/block-storage (visited on 2018-11-08).

[87] Druva. (2014).
Object Storage versus Block Storage: Understanding the Technology Differences,
[Online]. Available: druva.com/blog/object-storage-versus-block-storage-understanding-technology-differences (visited on 2018-11-20).

[88] NetApp, Inc. (2018).
Block Storage Vs. Object Storage in the AWS Cloud, [Online]. Available:
cloud.netapp.com/blog/block-storage-vs-object-storage-cloud
(visited on 2018-11-08).

[89] Margaret Rouse. (2014).
RESTful API, [Online]. Available:
searchmicroservices.techtarget.com/definition/RESTful-API (visited on 2018-11-20).

[90] Contel Bradford. (2018).
Storage Wars: File vs Block vs Object Storage, [Online]. Available:
blog.storagecraft.com/object-storage-systems (visited on 2018-11-08).

[91] Amazon Web Services, Inc. (2018).
Amazon S3 REST API Introduction, [Online]. Available:
docs.aws.amazon.com/AmazonS3/latest/API (visited on 2018-10-23).

[92] ——, (2018).
AWS Command Line Interface,
[Online]. Available: aws.amazon.com/cli (visited on 2018-11-30).

[93] The SQLite Consortium. (2018).
35% Faster Than The Filesystem,
[Online]. Available: sqlite.org/fasterthanfs.html (visited on 2018-11-30).

[94] Techopedia Inc. (2018).
Atomicity Consistency Isolation Durability (ACID),
[Online]. Available: techopedia.com/definition/23949/atomicity-consistency-isolation-durability-acid (visited on 2018-11-30).

[95] Neon Rain Interactive. (2018).
MySQL vs. MongoDB: Looking At Relational and Non-Relational Databases,
[Online]. Available: neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases (visited on 2018-11-30).

[96] Alon Brody. (2018).
SQL Vs NoSQL: The Differences Explained, [Online]. Available:
blog.panoply.io/sql-or-nosql-that-is-the-question (visited on 2018-11-30).

[97] MongoDB, Inc. (2018).
Relational Vs Non Relational Database, [Online]. Available:
mongodb.com/scale/relational-vs-non-relational-database (visited on 2018-11-30).

[98] Abu Thahir. (2018).
Which is Better ? Saving Files in Database or in File System,
[Online]. Available: habiletechnologies.com/blog/better-saving-files-database-file-system (visited on 2018-11-30).

[99] Christian Smith. (2018).
What is the difference between a file system and a database?, [Online]. Available: quora.com/What-is-the-difference-between-a-file-system-and-a-database/answer/Christian-Smith-2 (visited on 2018-11-30).

[100] Abuthahir Sulaiman. (2018).
File System vs. Database, [Online]. Available: dzone.com/articles/which-is-better-saving-files-in-database-or-in-fil (visited on 2018-11-09).

[101] Jacqueline Homan. (2018).
Relational vs. non-relational databases: Which one is right for you?, [Online]. Available: pluralsight.com/blog/software-development/relational-non-relational-databases (visited on 2018-11-09).

[102] M. Höst, B. Regnell, and P. Runeson,
Att genomföra examensarbete, swe. Studentlitteratur AB, 2006.

[103] ISO Central Secretary,
Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, en,
International Organization for Standardization, Geneva, CH,
Standard ISO/IEC 25010:2011, 2011.

[104] A. Krizhevsky and G. Hinton,
Learning multiple layers of features from tiny images, Citeseer,
Tech. Rep. CIFAR-10, 2009.

[105] Torch Contributors. (2018).
Training a Classifier, [Online]. Available:
pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
(visited on 2019-02-18).

[106] TensorFlow Contributors. (2018).
Advanced Convolutional Neural Networks, [Online]. Available:
tensorflow.org/tutorials/images/deep_cnn (visited on 2019-02-18).

[107] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey,
CINIC-10 is not ImageNet or CIFAR-10,
Computing Research Repository (CoRR), vol. abs/1810.03505, 2018.

[108] A. Friberg. (2018).
Train CINIC-10 with PyTorch, [Online]. Available:
github.com/AntonFriberg/pytorch-cinic-10 (visited on 2019-02-14).

[109] W. Li, R. Zhao, T. Xiao, and X. Wang,
DeepReID: Deep Filter Pairing Neural Network for Person Re-identification,
in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[110] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy,
OpenImages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*, 2017.

[111] Gerrit. (2018).
Gerrit Code Review,
[Online]. Available: gerritcodereview.com (visited on 2018-10-22).

[112] Git. (2018).
GIT - Homepage, [Online]. Available: git-scm.com (visited on 2018-11-29).

[113] The Git LFS Team. (2018).
Git Large File Storage,
[Online]. Available: git-lfs.github.com (visited on 2018-10-22).

[114] JFrog Ltd. (2018).
Artifactory,
[Online]. Available: jfrog.com/artifactory (visited on 2018-10-22).

[115] ——, (2018).
Artifactory, [Online]. Available:
jfrog.com/confluence/display/RTF/Git+LFS+Repositories (visited on 2018-11-05).

[116] D. Strigl, K. Kofler, and S. Podlipnig,
Performance and Scalability of GPU-Based Convolutional Neural Networks,
in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 317–324.

[117] T. Dettmers. (2019).
Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning, [Online]. Available:
timdettmers.com/2018/11/05/which-gpu-for-deep-learning (visited on 2019-02-21).

[118] Datastax, Inc. (2018).
Blob type, [Online]. Available:
docs.datastax.com/en/cql/3.3/cql/cql_reference/blob_r.html
(visited on 2019-01-14).

[119] The OpenStack Foundation. (2018).
Object Storage API overview, [Online]. Available:
docs.openstack.org/swift/latest/api/object_api_v1_overview.html
(visited on 2018-11-15).

[120] Cloudian Inc. (2018).
S3 API & Extensions for Enterprise Object Storage,
[Online]. Available: cloudian.com/blog/s3-api-extensions-for-enterprise-object-storage (visited on 2018-11-21).

[121] Amazon Web Services, Inc. (2018).
Boto 3 - The AWS SDK for Python,
[Online]. Available: github.com/boto/boto3 (visited on 2018-11-21).

[122] Marc Staimer. (2018).
Using the RESTful API as an onramp to object storage,
[Online]. Available: searchstorage.techtarget.com/tip/Using-the-RESTful-API-as-an-onramp-to-object-storage (visited on 2018-11-09).

[123] Techopedia Inc. (2018).
Query Language, [Online]. Available:
techopedia.com/definition/3948/query-language (visited on 2018-11-21).

[124] Veritas Genetics, Inc. (2018).
Arvados - Homepage, [Online]. Available: arvados.org (visited on 2018-10-10).

[125] The Apache Software Foundation. (2018).
What is Cassandra?,
[Online]. Available: cassandra.apache.org (visited on 2018-10-24).

[126] Red Hat, Inc. (2018).
What is Gluster ?,
[Online]. Available: docs.gluster.org/en/v3/Administrator%20Guide/GlusterFS%20Introduction (visited on 2018-11-13).

[127] Core Technology Sp. z o.o. (2018).
MooseFS - Homepage, [Online]. Available: moosefs.com (visited on 2018-12-17).

[128] Exoscale. (2018).
The Pithos Guide, [Online]. Available: pithos.io (visited on 2018-12-17).

[129] Basho Technologies, Inc. (2018).
Riak CS,
[Online]. Available: docs.basho.com/riak/cs/2.1.1/tutorials/fast-track/what-is-riak-cs (visited on 2018-01-15).

[130] ——, (2018).
Riak KV,
[Online]. Available: basho.com/products/riak-kv (visited on 2018-01-15).

[131] The SeaweedFS Contributors. (2018).
SeaweedFS,
[Online]. Available: github.com/chrislusf/seaweedfs (visited on 2018-12-17).

[132] Exoscale. (2018).
Pithos: Cassandra object storage,
[Online]. Available: github.com/exoscale/pithos (visited on 2018-11-13).

[133] Basho Technologies, Inc. (2018).
Riak KV 2.2.5 Release Notes, [Online]. Available:
github.com/basho/riak/blob/riak-2.2.5/RELEASE-NOTES.md (visited on 2018-11-14).

[134] R. Johnson. (2018).
RGW S3: Features Vs Deep Compatibility - Robin Johnson,
[Online]. Available: youtu.be/pYf_27YR2P0?t=360 (visited on 2019-01-30).

[135] The OpenStack Foundation. (2018).
S3/Swift REST API Comparison Matrix,
[Online]. Available: docs.openstack.org/swift/latest/s3_compat.html
(visited on 2018-11-15).

[136] Minio, Inc. (2018).
Minio Server Limits Per Tenant, [Online]. Available:
docs.minio.io/docs/minio-server-limits-per-tenant (visited on 2019-01-30).

[137] Red Hat, Inc. (2018).
Ceph Object Gateway S3 API, [Online]. Available:
docs.ceph.com/docs/master/radosgw/s3 (visited on 2019-01-30).

[138] nitisht. (2017).
How can I extend the storage size after deployment?, [Online]. Available:
github.com/minio/minio/issues/4364#issuecomment-302471048 (visited on 2018-11-21).

[139] fwessels. (2017).
How can I extend the storage size after deployment in same instance ?,
[Online]. Available:
github.com/minio/minio/issues/4712#issuecomment-318525347 (visited on 2018-11-21).

[140] Minio, Inc. (2018).
Multi-tenant Minio Deployment Guide, [Online]. Available:
docs.minio.io/docs/multi-tenant-minio-deployment-guide.html
(visited on 2018-10-09).

[141] ——, (2018).
Minio Deployment Quickstart Guide, [Online]. Available:
docs.minio.io/docs/minio-deployment-quickstart-guide (visited on 2018-11-21).

[142] OpenStack Foundation. (2018).
OpenStack - Homepage,
[Online]. Available: openstack.org (visited on 2019-01-30).

[143] SwiftStack Inc. (2019).
SwiftStack - Homepage,
[Online]. Available: swiftstack.com (visited on 2019-01-30).

[144] PyTorch Contributors. (2018).
Reproducibility, [Online]. Available:
pytorch.org/docs/stable/notes/randomness.html (visited on 2019-01-10).

[145] H. Vardhan. (2018).
Guidelines for assigning num_workers to DataLoader,
[Online]. Available: discuss.pytorch.org/t/guidelines-for-assigning-num-workers-to-dataloader/813 (visited on 2019-01-10).

[146] The Linux Container Project Contributors. (2019).
LXC - Homepage, [Online]. Available:
linuxcontainers.org/lxc/introduction (visited on 2019-02-18).

[147] ——, (2019).
LXD - Homepage, [Online]. Available:
linuxcontainers.org/lxd/introduction (visited on 2019-02-18).

[148] Virtuozzo. (2019).
OpenVz - Homepage, [Online]. Available: `openvz.org` (visited on 2019-02-18).

[149] I. Doornekamp. (2019).
duc - Homepage, [Online]. Available: `duc.zevv.nl` (visited on 2019-02-13).

[150] N. Craig-Wood. (2017).
Rclone - Homepage, [Online]. Available: `rclone.org` (visited on 2019-01-31).

[151] P. Xu, S. Shi, and X. Chu,
Performance Evaluation of Deep Learning Tools in Docker Containers, in *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, 2017-08, pp. 395–403.

[152] O. Torres. (2018).
7 Short-Term AI ethics questions,
[Online]. Available: `towardsdatascience.com/7-short-term-ai-ethics-questions-32791956a6ad` (visited on 2019-08-18).

[153] W. Ben-Hassine, N. Bueno, P. M. Chaudhary, M. Coninsx, S. Crown, P. Dawson, T. Ermacora, M. Gorbis, P. Hicks, A. Koenig, S. Kumar, E. Piraces, A. Popper, M. H. Posner, E. Santow, and A. Webb,
How to Prevent Discriminatory Outcomes in Machine Learning,
Global Future Council on Human Rights, World Economic Forum, Tech. Rep., 2018.

[154] Y. Kuflinski. (2019).
How Ethical Is Facial Recognition Technology?,
[Online]. Available: `towardsdatascience.com/how-ethical-is-facial-recognition-technology-8104db2cb81b` (visited on 2019-08-18).

[155] Microsoft Corporation. (2018).
Facial recognition: It's time for action, [Online]. Available: `blogs.microsoft.com/on-the-issues/2018/12/06/facial-recognition-its-time-for-action` (visited on 2019-08-18).

[156] Docker Inc. (2018).
Get Docker CE for Debian,
[Online]. Available: `docs.docker.com/install/linux/docker-ce/debian` (visited on 2018-09-26).

[157] ——, (2018).
Releases,
[Online]. Available: `github.com/docker/compose` (visited on 2018-11-28).

[158] ——, (2018).
Release Versions, [Online]. Available: `github.com/docker/compose/releases` (visited on 2018-11-28).

[159] The Kubernetes Authors. (2018).
Running Kubernetes Locally via Minikube, [Online]. Available: `kubernetes.io/docs/setup/minikube` (visited on 2018-10-05).

[160] ——, (2018).
Overview of kubectl, [Online]. Available:
kubernetes.io/docs/reference/kubectl/overview (visited on 2018-10-05).

[161] ——, (2018).
Minikube,
[Online]. Available: github.com/kubernetes/minikube (visited on 2018-10-05).

[162] ——, (2018).
Driver plugin installation,
[Online]. Available: github.com/kubernetes/minikube/blob/master/docs/drivers.md#kvm2-driver (visited on 2018-10-05).

[163] ——, (2018).
Install and Set Up kubectl,
[Online]. Available: kubernetes.io/docs/tasks/tools/install-kubectl
(visited on 2018-10-05).

[164] ——, (2018).
Minikube, [Online]. Available: github.com/kubernetes/minikube/releases
(visited on 2018-10-05).

[165] NVIDIA Corporation. (2019).
CUDA Toolkit Archive, [Online]. Available:
developer.nvidia.com/cuda-toolkit-archive (visited on 2019-02-21).

[166] ——, (2019).
cuDNN Archive, [Online]. Available:
developer.nvidia.com/rdp/cudnn-archive (visited on 2019-02-21).

[167] ——, (2019).
Nvidia CUDA Installation Guide for Linux,
[Online]. Available: developer.download.nvidia.com/compute/cuda/8.0/secure/Prod2/docs/sidebar/CUDA_Installation_Guide_Linux.pdf
(visited on 2019-02-21).

[168] ——, (2019).
Deep Learning SDK Documentation,
[Online]. Available: docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html#installlinux-tar (visited on 2019-02-21).

[169] Zsh Web Page Maintainers. (2019).
Z shell, [Online]. Available: zsh.sourceforge.net (visited on 2019-02-21).

[170] mkasberg. (2018).
Cannot Increase open file limit past 4096 (Ubuntu), [Online]. Available:
superuser.com/questions/1200539/cannot-increase-open-file-limit-past-4096-ubuntu/1200818#1200818 (visited on 2018-10-15).

Appendices

Appendix A

Polyaxon Screenshots

In this appendix you will find a collection of screenshots that are taken from the Polyaxon web interface, as it looked like when we did our experiments. This is done in order for the reader to get a better understanding on how the workflow looked like. However, we would like to inform the reader that the Polyaxon project moves very fast, and these screenshots might not be representative at the time of reading.

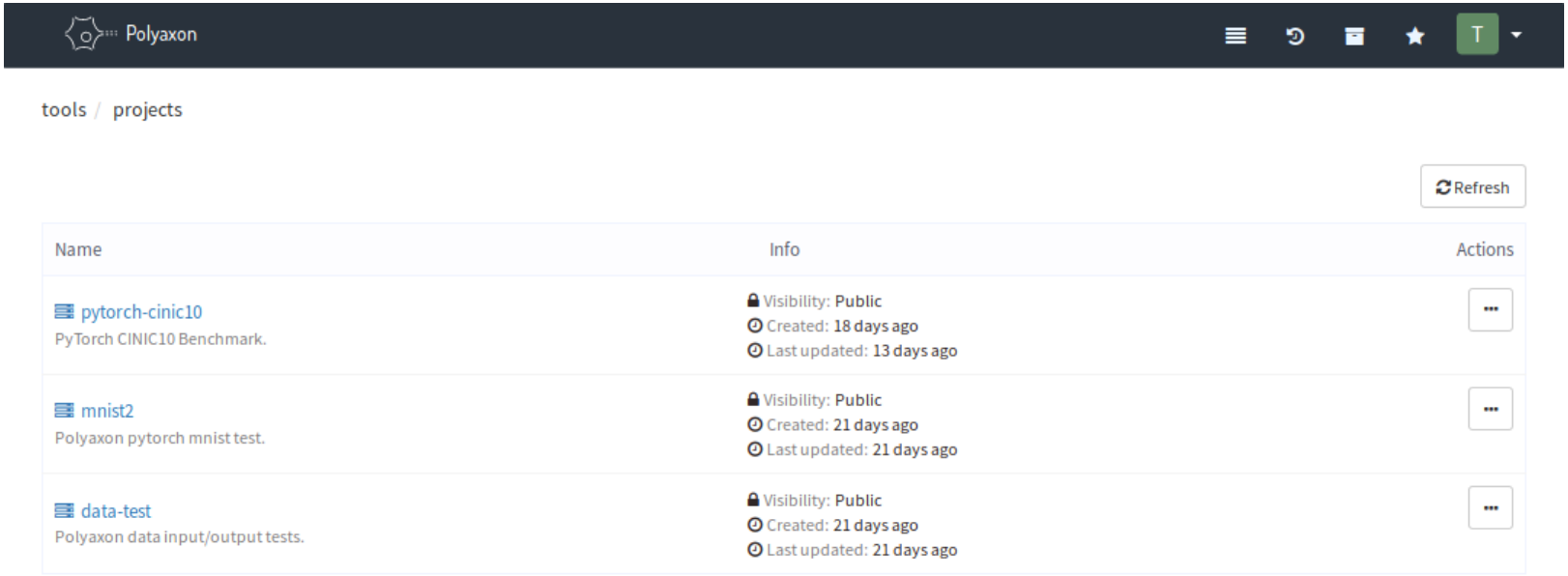


Figure A.1: Projects view of the Polyaxon framework [70].

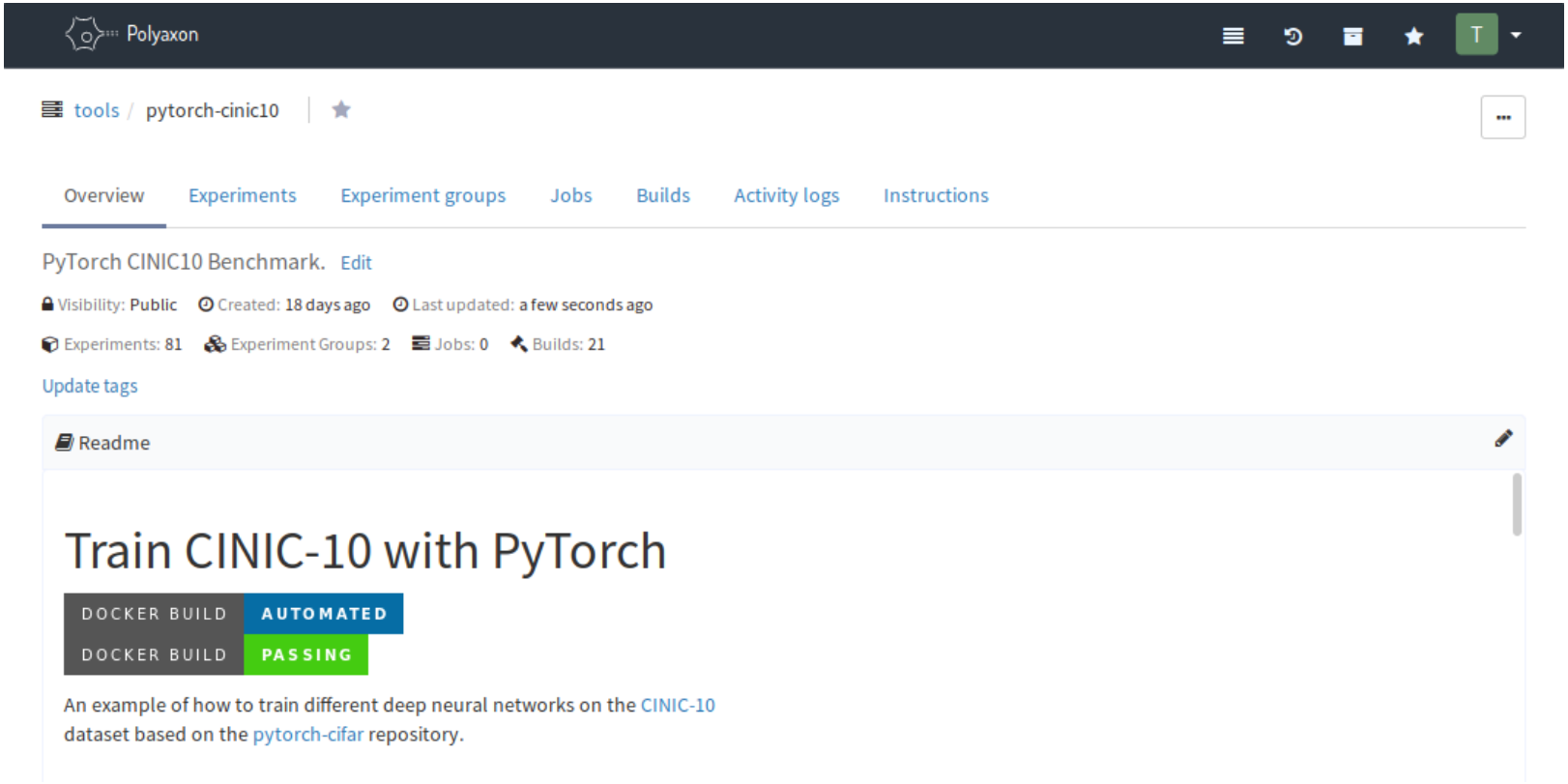


Figure A.2: Project Overview of the Polyaxon framework [70].

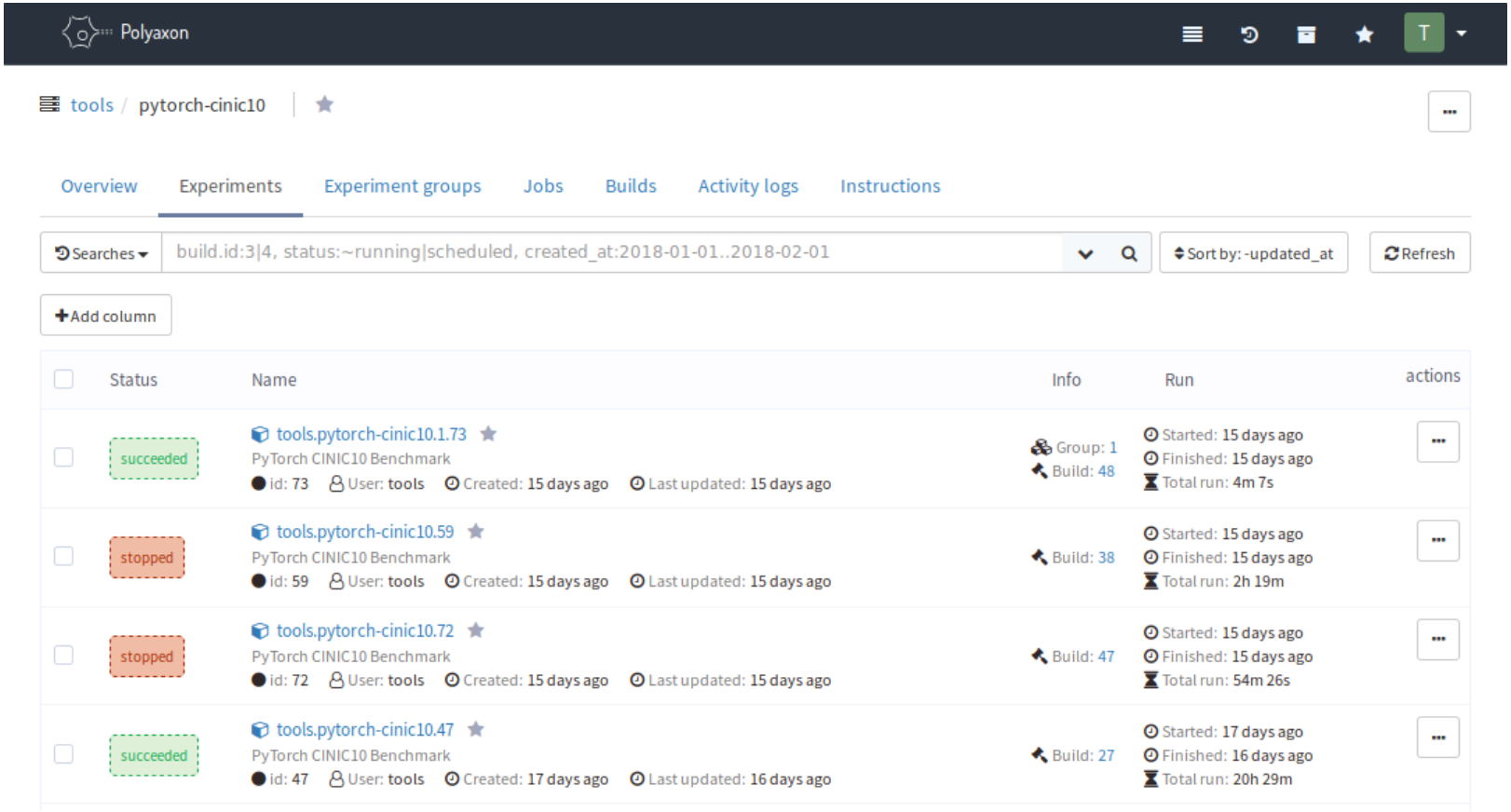


Figure A.3: Experiments view of the Polyaxon framework [70].

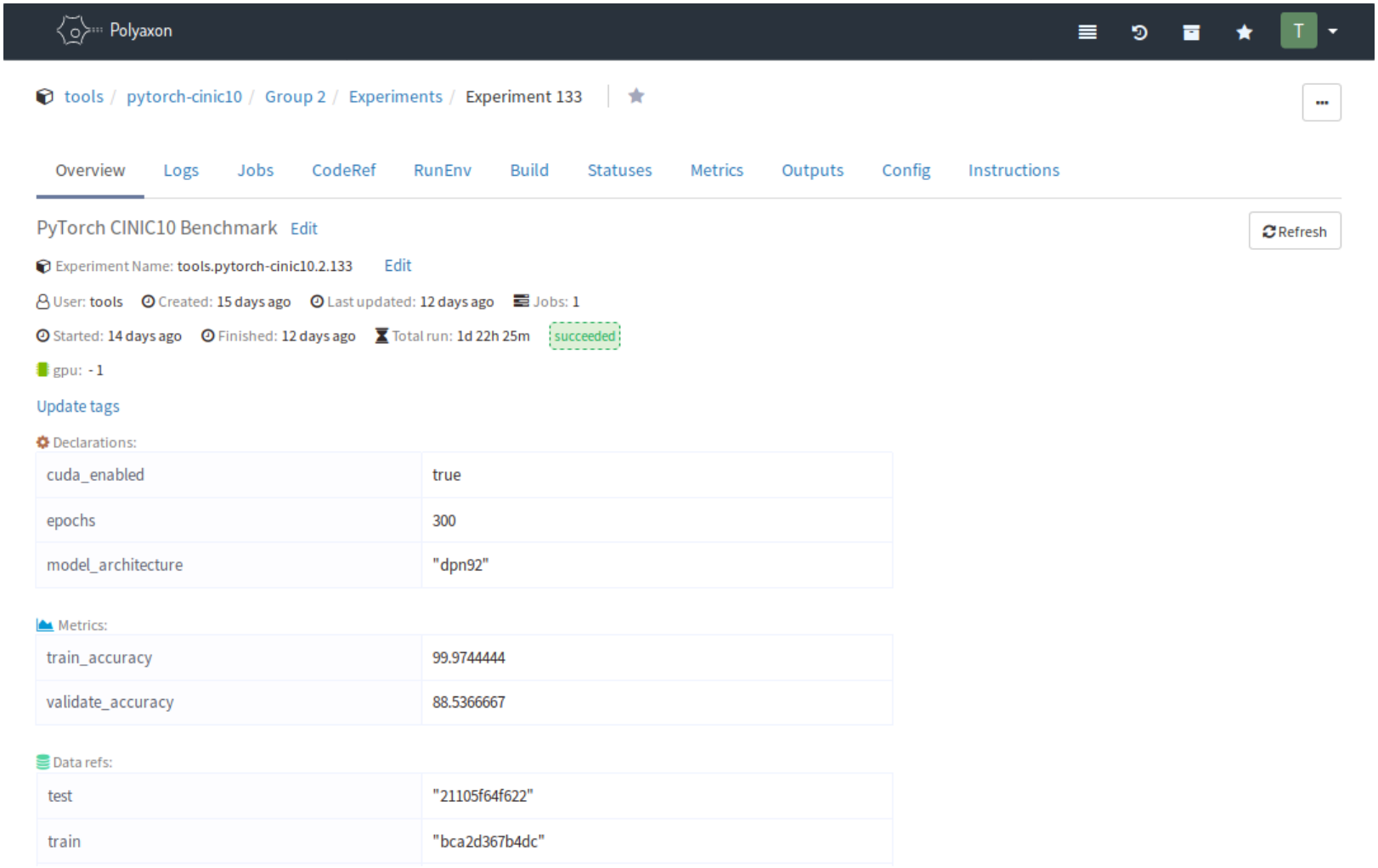


Figure A.4: Experiment view of the Polyaxon framework [70].

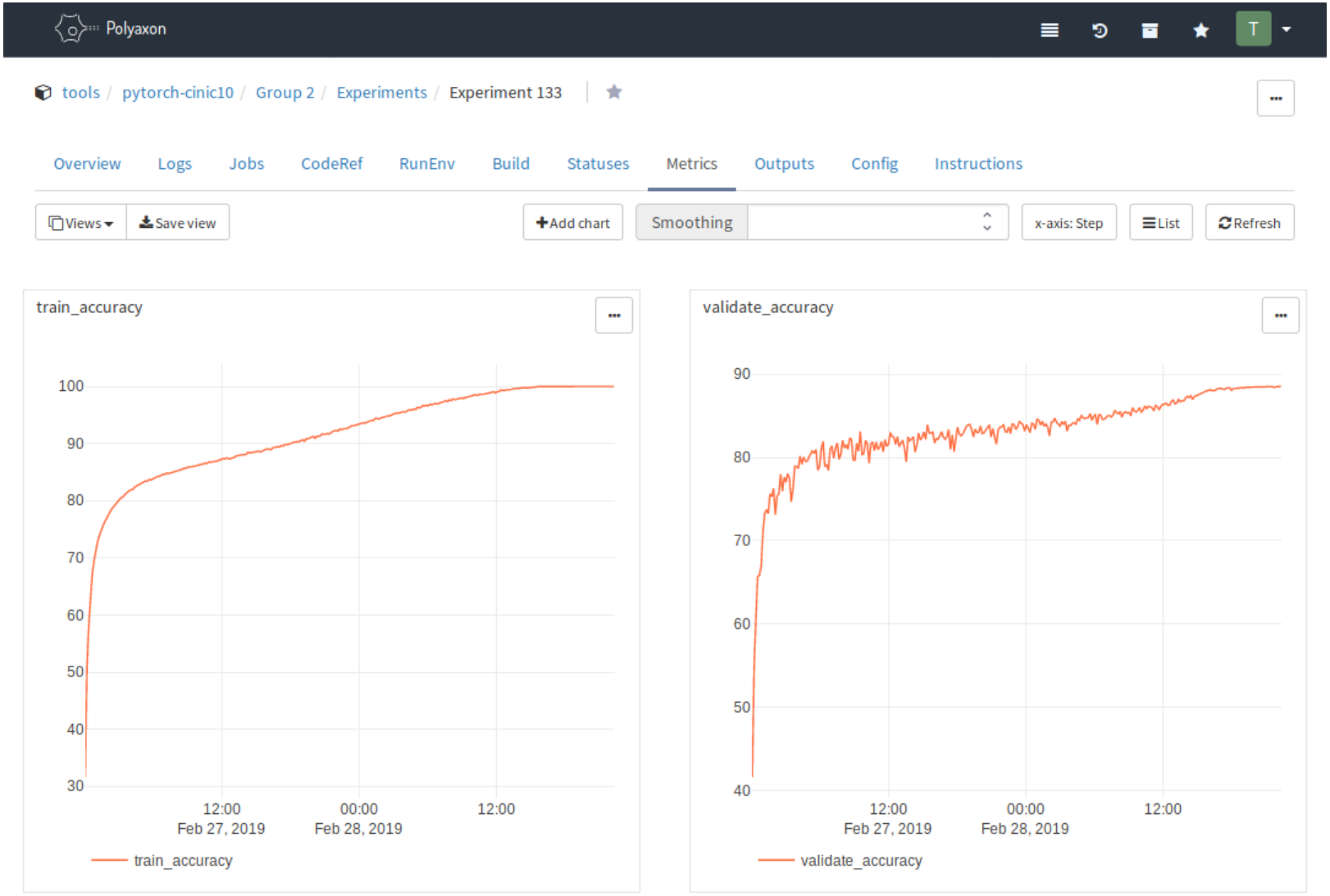


Figure A.5: Experiment metric view of the Polyaxon framework [70].

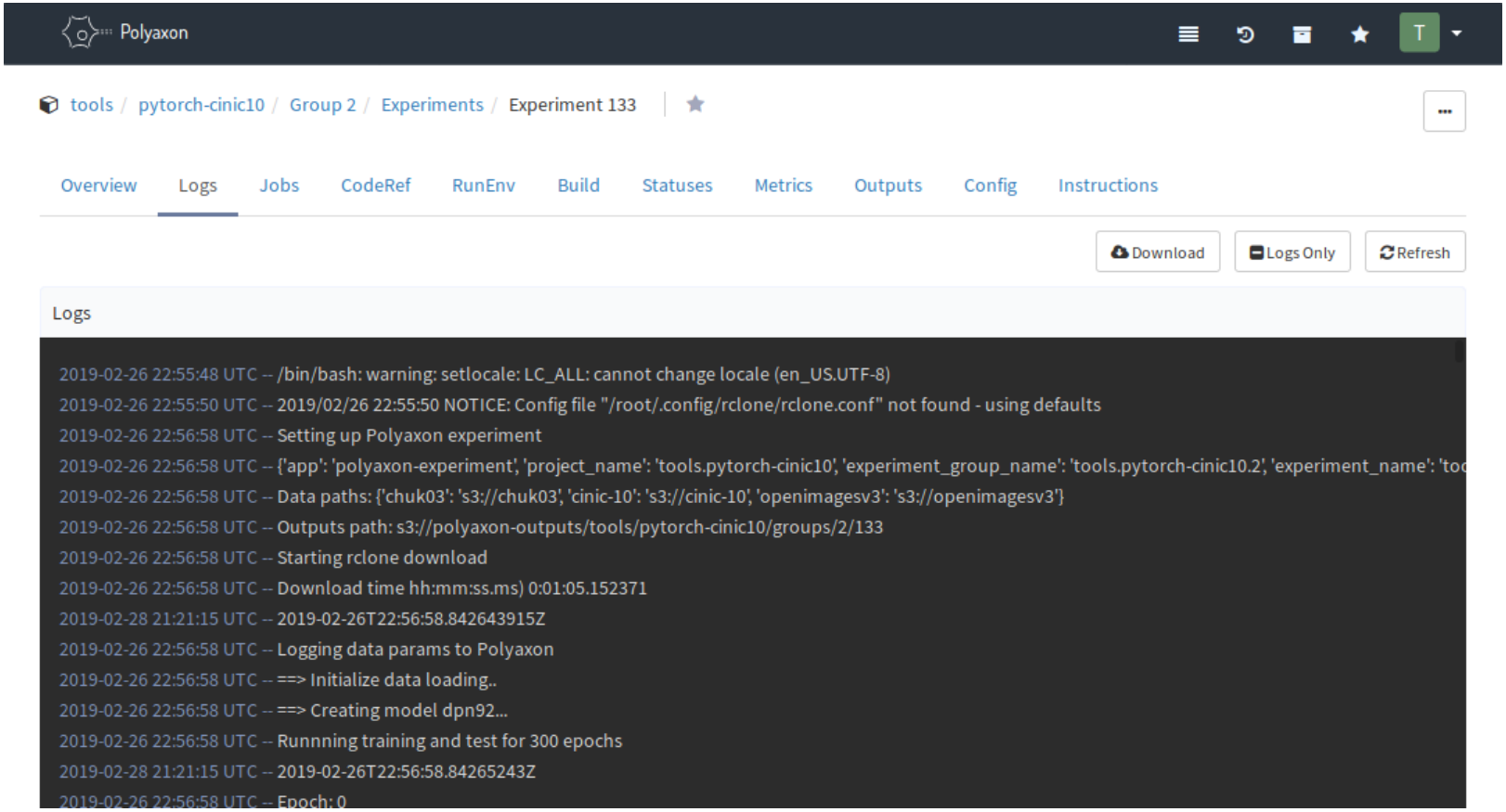


Figure A.6: Experiment logs view of the Polyaxon framework [70].

Appendix B

Installation Procedures & Version Info

During the course of this project there will be a couple of different computer tools used. Since these programs continually get updates it is important to record the versions and installation methods used at the point of time this report is written. This appendix will not contain any information that is of importance for those who just want to read about the findings, but might be interesting for those who want to replicate the methods.

B.1 Install `Docker CE` for Debian

Before starting, it should be verified that the operating system used is supported by the Docker program. We are using Debian 9.5 Stretch (x86_64), running the Linux 4.9.0-8-amd64 kernel, which is part of supported versions at the moment [156]:

- Buster 10 (Docker CE 17.11 Edge only)
- Stretch 9 (stable) / Raspbian Stretch
- Jessie 8 (LTS) / Raspbian Jessie
- Wheezy 7.7 (LTS)

What is about to follow is a quick installation guide to have Docker up and running on a similar computer to the one presented above. For detailed information about these steps we refer to the official installation guide [156].

```
sudo apt-get remove docker docker-engine docker.io
sudo apt-get update
sudo apt-get install \
     apt-transport-https \
     ca-certificates \
     curl \
     gnupg2 \
     software-properties-common
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
sudo add-apt-repository \
     "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
```

To be able to run docker as the current user, without root privileges, a group called `docker` needs to be created, and the username of interest added to it. Otherwise, `sudo` needs to be prepended to all Docker commands. Going forward it is expected that the user has been added to this "`docker`" group by running the following commands:

```
sudo groupadd docker
sudo usermod -aG docker <your-user>
```

Log out and back in again so that group memberships are re-evaluated. The version of Docker installed at the time of writing is `18.06.1-ce, build e68fc7a`. This number can be obtained by running the following after installation.

```
docker --version
```

If a version number is returned, the official `hello-world` container should be run to verify that everything is working correctly.

```
docker run hello-world
```

This should return a message stating something along the following lines if Docker was correctly installed.

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
...
...
```

B.2 Install `docker-compose` for Debian

Docker-Compose is a python script used to simplify a single deployment of multiple Docker containers and setting up an isolated network environment in which these can communicate with each other. This script relies on a fully functional and up to date Docker daemon. Installation instruction for this can be found in Section B.1.

Installation of `docker-compose` requires only two lines of code. The first one downloads the script from the main GitHub repository [157]. Please note that the following line downloads version `1.23.1`, for a complete list of releases see their repository [158] and change the number in the link below.

```
sudo curl -L \
"https://github.com/docker/compose/releases/download/1.23.1/\
docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Then this file needs to be made executable.

```
sudo chmod +x /usr/local/bin/docker-compose
```

Verify the installation with the following command.

```
docker-compose --version
```

B.3 Install the Minikube Environment

Minikube is a tool which enables a Kubernetes cluster to be run on your local computer for development and testing proposes without the need of multiple machines [159].

There are three parts to this setup that are necessary; first a hypervisor is needed, secondly `kubectl` [160] which is the command-line interface tool to manage Kubernetes before finally the actual `Minikube` [161] tool. This guide also expect `Docker CE` to already be installed, if not the guide in B.1 should be followed before proceeding.

A word of warning is that the projects mentioned here move at a significant speed, and by the time of reading this there will be newer versions available which may differ significantly. This installation will work for the version numbers presented, but not all versions of these three components are compatible with each other. Unfortunately possible future problems with this installation guide can not be accounted for since trial and error has been the only way to find out these conflicting version numbers up until now.

B.3.1 Install a Hypervisor

The Minikube tool can run naively on the computer by utilizing Docker directly, however, this requires root privileges which is not ideal. The log output from running `Minikube` with the option `-vm-driver=none` will yield the following warning:

```
WARNING: IT IS RECOMMENDED NOT TO RUN THE NONE DRIVER ON PERSONAL WORKSTATIONS
         The 'none' driver will run an insecure kubernetes apiserver as root that
         may leave the host vulnerable to CSRF attacks
```

To get around this a hypervisor is needed, and for this setup the `kvm2` driver is chosen. This is the one recommended by the Minikube development team and it can be installed accordingly [162]. The (Intel) `VT-x` or `AMD-v` virtualization technology must be enabled in your computer's BIOS before continuing. Install the prerequisites.

```
sudo apt install libvirt-clients libvirt-daemon-system qemu-kvm
```

Add your `user` to the `libvirt` group and update the session.

```
sudo usermod -a -G libvirt <your-user>
newgrp libvirt
```

Install the actual driver.

```
curl -Lo docker-machine-driver-kvm2 \
https://storage.googleapis.com/minikube/releases/v0.29.0/docker-machine-driver-kvm2

chmod +x docker-machine-driver-kvm2
sudo cp docker-machine-driver-kvm2 /usr/local/bin/
rm docker-machine-driver-kvm2
```

B.3.2 Install `kubectl`

The command-line interface tool `kubectl` [160] is available for download via the Debian package manager, however, a newer version is desired. Here version `v1.12.0` is used [163].

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/\
v1.12.0/bin/darwin/amd64/kubectl

chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

B.3.3 Install `Minikube`

Installation of Minikube can now commence [164]. At the time of writing version `v0.29.0` is the latest.

```
curl -Lo minikube https://storage.googleapis.com/minikube/\
releases/v0.29.0/minikube-linux-amd64

chmod +x minikube
sudo cp minikube /usr/local/bin/
rm minikube
```

B.3.4 Verify the `Minikube` Installation

We experienced some boot inconsistencies when launching Minikube which seems to be related to the networking components in the `virsh` module. The `default` net did not register properly with Minikube, but a reset of it before first launch solved our troubles. This step might not be necessary for you.

```
sudo virsh net-destroy default
sudo virsh net-define /etc/libvirt/qemu/networks/default.xml
sudo virsh net-autostart default
sudo virsh net-start default
```

Verify the installation by first starting Minikube. This step might take a long time if a slow computer is used, with the step "`Starting cluster components`" taking the longest. Unfortunately this is also the step that gets stuck if the network problem described above happens, so one has to judge on their own if it is stuck or to wait a bit longer. Furthermore, there were more network problems when trying to run version `v1.12.0` of Kubernetes inside the virtual machine, therefore version `1.11.0` is selected in the command below.

```
minikube start --vm-driver=kvm2 --kubernetes-version=v1.11.0
```

If the command completes it should be possible to obtain some status information from `kubectl` by running the following.

```
kubectl cluster-info
```

A successful command should produce something along these lines.

```
Kubernetes master is running at https://192.168.39.34:8443
CoreDNS is running at https://192.168.39.34:8443/api/v1/namespaces/kube-system\
/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

B.3.5 Remove `Minikube`

If there are any problems with the install, or a desire to upgrade the Minikube version, we strongly recommend completely erasing all Minikube dependencies and starting over from scratch to avoid strange problems. Here is a quick script which will completely wipe anything Minikube related if the above guide was followed.

Paste the following into a file and run it for quick and easy removal of Minikube.

```
#!/bin/sh
echo "=== Running minikube stop ==="
minikube config set WantReportErrorPrompt false
minikube stop
echo "=== Running minikube delete ==="
minikube delete
echo "=== Deleting local .minikube and .kube folders ==="
rm -rf ~/.minikube ~/.kube
echo "=== Uninstalling local minikube and kubectl ==="
sudo apt purge kubectl minikube -y
sudo rm /usr/local/bin/localkube /usr/local/bin/minikube /usr/local/bin/kubectl
echo "=== Uninstalling kvm2 and minikube kvm2 driver ==="
sudo rm /usr/bin/docker-machine-driver-kvm2
sudo apt purge libvirt-clients libvirt-daemon qemu-kvm -y
sudo rm -rf /var/lib/libvirt /etc/libvirt
echo "=== Cleaning up unused dependencies ==="
sudo apt autoremove -y
```

For good measure it might be good to reboot the computer as well, but it should not be required. To install Minikube again, just begin form B.3.1 again.

B.4 Install & Configure `Minio`

The easiest method of installing Minio is to run it from inside a Docker container. The guide in B.1 should therefore have been completed before continuing here. After this it is very simple to have a well configured Minio server up and running. Execute the following line in a terminal, but be careful to change the path of the mounted folder to something that exist on your system and to not overwrite anything.

```
docker run --rm -p 9000:9000 --name minio \
                -v /path/to/external/folder:/data \
                --env MINIO_ACCESS_KEY=accesskey \
                --env MINIO_SECRET_KEY=supersecret \
                minio/minio server /data
```

It should now be possible to navigate to `http://<hostname>:9000` to reach a webpage for the server which was just started. Enter the access key and secret key defined in the command above to be able to log in. Minio should now be ready to be used as a testing environment. If you are interested in a production ready set up of this, we recommend reading the guides on the homepage [28].

B.5 Install & Configure `AWS CLI`

The AWS Command Line Interface tool [92] is the official client, written by Amazon themselves, used to communicate with S3 compatible services. Both Debian and Ubuntu should have it available in the official repositories, and can be installed by simply running the following command.

```
sudo apt install awscli
```

After this it is important to configure the client to use the correct access key and secret key for the service to which you are trying to connect to. To edit the configuration file you type the following into the terminal.

```
aws confgure
```

If we use the same keys as in B.4, the following options should be set.

```
AWS Access Key ID [None]: accesskey
AWS Secret Access Key [None]: supersecret
Default region name [None]: us-east-1
Default output format [None]:
```

After this you may use your favorite text editor to open up the file `~/.aws/config` and make sure it looks like this:

```
[default]
region = us-east-1
s3 =
  max_concurrent_requests = 100
  max_queue_size = 10000
  multipart_threshold = 64MB
  multipart_chunksize = 16MB
  payload_signing_enabled = false
```

When this is done you will have the same settings as us when we tested the solutions.

B.6 Install & Configure `rclone`

Rclone [150] is a tool which is compatible with many different protocols, one of which is Amazon S3. This is usually not available on the normal package repositories and needs to be

downloaded from an external site. An installation script can be downloaded and piped[1] into bash like this:

```
curl https://rclone.org/install.sh | sudo -E bash
```

This will then need to be configured to properly connect with the storage service you are targeting. The configuration procedure may be accessed by typing:

```
rclone config
```

We will be using the keys and addresses from B.4, and below is how we configured it where at each new line, after the ">", is what option we chose.

```
New Remote> n
name> minio
type of storage> 4
s3 provider> 6
credentials> 1
access_key_id> accesskey
secret_access_key> supersecret
region> us-east-1
endpoint> http://<server_hostname>:9000
location_constraint>
acl> 1
advanced> n
save> y
quit> q
```

B.7 Install Git & the LFS Extension

This guide will install both `git` and the `lfs` extension via the package manager on Debian 9.6. The two necessary programs can be installed by executing the following command:

```
sudo apt install git git-lfs
```

The `lfs` extension then needs to be activated inside git as well.

```
git lfs install
```

The installations can be verified with the following two commands.

```
git version
git lfs version
```

This should result in the following output.

```
git version 2.11.0
git-lfs/2.6.0 (GitHub; linux amd64; go 1.10.3)
```

A note regarding the use of this extension is that if an error stating that "there are too many files open" is displayed, you may need to follow the guide found in C.1 to be able to continue.

[1] It is usually a very bad idea to pipe unknown scripts from the internet directly into bash, analyze what you download.

B.8 Install Nvidia Driver, CUDA & cuDNN

In order to recreate the same software environment, as the engineers of the video analytics department used, we let the IT department of the case company reformat our computer and install Ubuntu 16.04 LTS on it. This automatically installed some Nvidia drivers, but we needed to make sure only the specific versions outlined in Table 4.1 were the one to be installed. Thus, we had to remove anything Nvidia related by running the command below. This might break the visual output to the screen you are using, so it is advisable to test that SSH connections are accepted by the host before doing this.

```
sudo apt-get purge 'nvidia-*'
```

We then updated the remaining packages and installed build dependencies.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install gcc make g++ build-essential
```

To install the correct versions of the Nvidia components, we utilized the archive repository on their official website to obtain the graphics driver, CUDA toolkit [165] and cuDNN [166]. This was then installed using the provided documentation [167], [168]. In order to download the cuDNN dependencies it was necessary to sign up to their website.

```
# Download CUDA Toolkit
wget https://developer.nvidia.com/compute/cuda/8.0/Prod2/local_installers/\
cuda-repo-ubuntu1604-8-0-local-ga2_8.0.61-1_amd64-deb -O "cuda-deb"

# Install CUDA and the Nvidia Driver
sudo dpkg -i cuda-deb
sudo apt-get update
sudo apt-get install cuda

# Download cuDNN and install it
wget https://developer.download.nvidia.com/compute/redist/cudnn/v6.0/\
cudnn-8.0-linux-x64-v6.0.tgz -O "cudnn.tgz"
tar -zxf cudnn.tgz
sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
sudo cp cuda/include/cudnn.h /usr/local/cuda/include/
sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn
```

We added the needed environment variables and rebooted the system. Please note we are using the `Z` shell [169], which make the paths persistent over reboots by including them inside the `.zshrc` file. If another shell is used, these commands will need to be changed appropriately.

```
echo "export CUDA_HOME=\"/usr/local/cuda\"" >> ~/.zshrc
echo "export LD_LIBRARY_PATH=\"/usr/local/cuda-8.0\":\$LD_LIBRARY_PATH" >> ~/.zshrc
echo "export LD_LIBRARY_PATH=\"/usr/local/cuda-8.0/lib64\":\$LD_LIBRARY_PATH" >> ~/.zshrc
echo "export PATH=\"/usr/local/cuda-8.0/bin:\$PATH\"" >> ~/.zshrc
sudo reboot
```

Finally we verified that the correct versions were installed.

```
# Nvidia driver info
nvidia-smi

# CUDA version
nvcc --version

# cuDNN location
locate libcudnn
```

Appendix C

Host OS Modifications

This appendix contain information regarding specific modifications that were required for certain parts of the installation process or for a program to perform better. If there are not specifically addressed in a guide these modifications should be ignored as they could make a system more unstable.

C.1 Increase `ulimit`

To increase the limit of maximum number files that is allowed to be open simultaneously, as shown by the command `ulimit -n`, three files are needed to be modified [170]. Firstly the line `DefaultLimitNOFILE=65535` should be added to the following two files:

- `/etc/systemd/user.conf`
- `/etc/systemd/system.conf`

This setting affects sessions using the graphical login, so for non-GUI login the following two lines:

```
* hard nofile 65535
* soft nofile 65535
```

needs to be added to the file

- `/etc/security/limits.conf`

After this at least a logout and login again is necessary, but a reboot is recommended. Verify that this limit has increased by running `ulimit -n` again.

Appendix D

Experiment Setup

This appendix will provide additional information regrading how the experiments were set up and performed in order to obtain the results that are presented in Chapter 5. The section titles will follow the same naming scheme for easy cross-reference.

D.1 Storage

Since it was requested that the information regrading the Proprietary Solution was to be kept to a minimum, the configuration steps for this can not be disclosed for the readers to reproduce. These instructions are therefore limited to the Minio server solution used, which firstly requires Docker to be installed, see Appendix B.1. The guide on how to install Minio can then be followed in Appendix B.4, to be able to have an identical setup as the one used in the testing.

The clients, used to download the data assets from the object storage solutions, are comprised of the `aws-cli` [92] program, which is written and maintained by Amazon, and the more versatile `rclone` [150] program. Additional details on the optimized `aws-cli` setup used during testing can be found in Appendix B.5, while the corresponding guide for `rclone` is located in Appendix B.6.

The command used to download using `aws-cli` is presented below. The same command is also used for uploads, but with the source and destination swapped around.

```
aws --endpoint-url http://<remotehost>:<port> s3 sync s3://<bucket> /<localfolder>
```

The `rclone` download command differs slightly, and the `<remoteService>` is the name of the profile that was created in the setup guide seen in Appendix B.6. To upload, the source and destination arguments are reversed.

```
rclone copy <remoteService>:<bucket> <localfolder> -v --transfers 100 --checkers 100
```

Using the `time` utility, which comes preinstalled with Debian, the above commands are wrapped with it to report back how long time it took the computer to finish executing them. This is then the time that is presented in the results.

Something to take note of is that all communications between the *Proprietary Solution* and the client was encrypted (`HTTPS`), which introduces an overhead compared to the non-encrypted setup used in the Minio instance. The computer hosting the server instance of Minio is the same machine as the one outlined in Table 4.1. For these experiments the hardware information of interest is mainly CPU performance, disk transfer speeds and network transfer speeds since the other components are sparsely utilized. Importantly, results gained through additional external testing made sure that the NVMe disk was fast enough to not be a bottleneck in the tests.

Further tests that were performed, with multiple simultaneous downloads, showed that the Proprietary Solution was mainly limited by its shared hardware, while the Minio server was mostly limited by its single gigabit uplink. A single downloading stream from the Minio solution would therefore instead show the limitations of the client software used.

Regarding the results displaying the improved storage sizes, these were obtained by first storing the dataset in its raw form, then upload it to Git LFS, for it to be directly downloaded again to a separate directory on the same computer. Both the size on disk and the number of files were then documented for the separate folders, which was then plotted in a sunburst diagram by utilizing the software `duc` [149] with some minor color changes. The same was done for both of the object storage solutions, but these did not add any additional data and are identical to the raw dataset.

D.2 Preprocessing

There is no easy method of testing the time improvements of each of the preprocessing steps in isolation, which is why the object identification example on the `CINIC-10` dataset [108], introduced in 4.2.3, was utilized in full. In order to be able to identify the performance impact of any of these steps, all settings, except those involving preprocessing, will remain unchanged throughout this section. Thus, any variation to time or accuracy should be accounted to these preprocessing steps.

At the time of writing, the engineers are only using the default configuration of a single worker thread for the data augmentations. By increasing the number of workers the augmentation workload can be conducted in parallel on the CPU and speed up the time to complete the training. Changing the number of workers are as easy as modifying the `<nbr_of_workers>` parameter in the code below and is in the tested code example exposed as input parameter to the program. When it came to testing the impact of having no augmentations at all, the lines of code enabling these were removed.

```
trainloader = torch.utils.data.DataLoader(trainset,
                                          batch_size=64,
                                          shuffle=True,
                                          num_workers=<nbr_of_workers>)
```

D.3 Scheduling & Training

There are a couple of different measurements that are performed in this section, and these are divided into the same subsections as in Section 5.3.

D.3.1 Software

The first improvement that was measured was how long it took to tear down an old test environment and set up a new one. This represents how it currently works with the shared compute nodes, where engineers conducts experiments that has software dependencies that are incompatible with the other frameworks used. When a new experiment is to be conducted, it is therefore necessary to remove these old components and reinstall the correct ones.

Since both the current *native* reinstallation process, and the Docker process, requires some practice for the user to become proficient, it is assumed that the engineers have performed this procedure many times before. It is also expected that both the base Nvidia graphics driver, and the Docker daemon, is already installed and properly configured on the system. Since these two prerequisites are a one-time installation occurrence, it was not deemed fair to include these steps in the measurements. Furthermore, the experiment code, as well as the dataset, is also already present on the machine.

The *native* reinstallation is then performed according to the guide written in Appendix B.8. The timer started when the first command was entered, and ended when the last one completed execution. The Docker process was only composed of building a new container of our test environment [108]. However, this does not include the time it takes to create the `Dockerfile` in the first place. But similarly, the time it takes to register on the Nvidia homepage, to be able to download the needed software, is also not included in the other measurements.

To determine how much overhead the Docker isolation layer add to the training time, a couple of different openly available object classification models were integrated into the developed `CINIC-10` training example [108]. This was quite simple as many examples were readily available for the PyTorch framework that was being used, and switching between the models was made simple by creating an input argument where the model name is specified. In similarity to the benchmark performed in the preprocessing step, all other settings were set to the defaults and the network was left to train for 300 epochs on the entire `CINIC-10` dataset.

By only changing the model used, the idea was to discern how different computations are affected by running inside a Docker container environment. All the networks are commonly used as object classifiers, which is why the final accuracy is also included. The *native* procedure was conducted in exactly the same fashion, except that there was no simple input argument that could be changed, so instead the setup code had to be manually changed between each run.

D.3.2 Scheduling & Training

Regarding scheduling, our method of measuring improvements is to compare the time it took to run all the containerized tests, presented in Table 5.4, by first manually scheduling everything, and then by letting Polyaxon schedule them on the Kubernetes cluster. Since Polyaxon will have access to five GPUs in the cluster, it will not be an apples to apples comparison to the single GPU workstation used to run the manual scheduling. However, it does represent the current situation where one should benefit by running the experiments on the cluster and is designed to showcase the power of having a software scheduler instead of doing everything manually.

To provide a baseline, for how long time manual scheduling takes, one of the Docker tests from Table 5.4 was started on our local training computer, which still has the setup defined in Table 4.1. Then the terminal output was intermittently monitored to see if it had finished. If that was the case, the results were recorded and the next task in line was started. It was known that some models would take a significantly longer time to train than others which is why these were chosen to start when we left for the evening in order to complete before the next morning. This was done to accurately represent the current situation. Unsurprisingly, this would not always align as well as one would have hoped, leading to long stretches of time when the hardware was simply not used. The timer stopped when we noticed that the final experiment had finished.

The dataset used during this time was already present on the training computer, and did not need to be downloaded for each new network that was used. However, such was not the case with the Polyaxon experiment. The configuration created, that would properly set up and distribute the training tasks across the Kubernetes cluster, could not guarantee that the dataset was cached on the compute note it executed on, and was therefore forced to download a new copy each time.

After the Polyaxon scheduler was started, we did not have any control on how and where the different training tasks were executed. The experiment was deemed over when we noticed that the every scheduled task was showing up as completed on the web interface of Polyaxon. It is in this interface it is possible to view old experiments and compare previous results for similar training tasks.

D.3.3 Hardware

Obtaining hardware was an issue that we did not focus on, and the only experience we have regarding the lead times of ordering hardware is the time it took for our workstation, presented in Table 4.1, to be assembled. This was also not deemed to be of interest at the moment, by the engineers, and no further efforts were made into obtaining concrete time measurements of this.