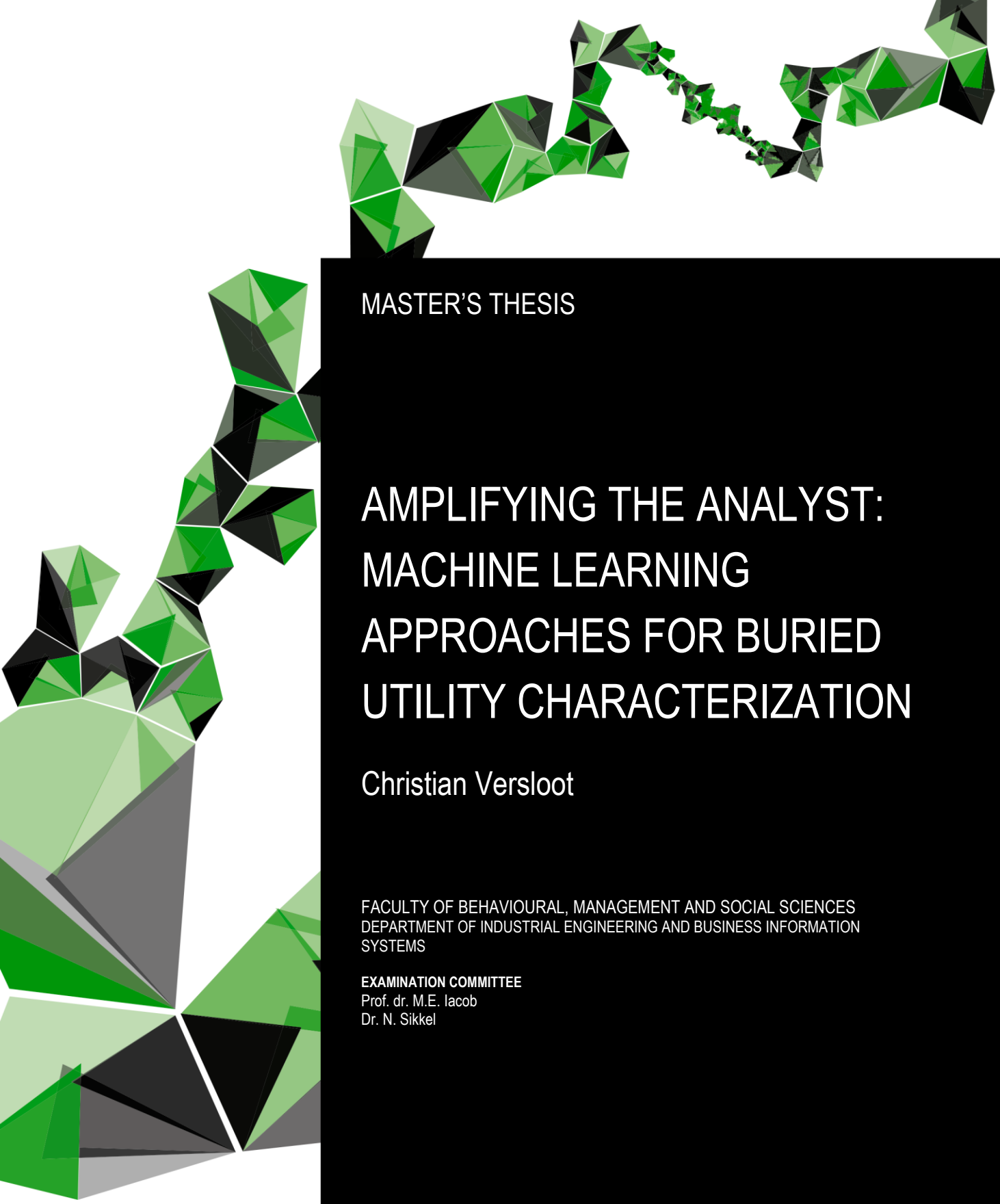


MASTER'S THESIS

AMPLIFYING THE ANALYST: MACHINE LEARNING APPROACHES FOR BURIED UTILITY CHARACTERIZATION

Christian Versloot

FACULTY OF BEHAVIOURAL, MANAGEMENT AND SOCIAL SCIENCES
DEPARTMENT OF INDUSTRIAL ENGINEERING AND BUSINESS INFORMATION SYSTEMS

EXAMINATION COMMITTEE
Prof. dr. M.E. Iacob
Dr. N. Sikkel

UNIVERSITY OF TWENTE.

Executive Summary

In the Netherlands, during excavation activities, underground utilities such as cables and pipelines are struck every three to four minutes. Those strikes result in annual material damages of 25 million Euros and originate from a multitude of reasons. Foremost, until the 2010s, administering underground utilities was not mandatory in the Netherlands. Neither was checking one's excavation activities against the registered infrastructure. Also considering the inaccuracies of outdated geolocation devices used years ago renders a simple conclusion, being that substantial safety risks for workers, residents and infrastructure are present during excavation work.

Today, Dutch legislation requires an adequate registration as well as a proper inquiry into the KLIC registry prior to starting excavation work. The KLIC is maintained by the Dutch land registry and provides an overview of Dutch underground infrastructure. Its inaccuracies still present today require organizations performing excavation activities to be rather safe than sorry. Consequently, a business model has emerged for services of so-called underground mapping companies. Using geophysical technologies like Ground Penetrating Radar (GPR) and Radio Detection, they can accurately map the infrastructure. Services like those prevent many strikes and by consequence damages to underground utilities. This makes the problem manageable.

TerraCarta B.V. is one such company. It employs analysts mapping the underground for major clients. However, their analysis tasks are complex and time-consuming. Acquiring the necessary geophysical knowledge takes years and specialized personnel is scarce. Consequently, an analyst's tasks are rather cost-intensive. Besides, they are also repetitive – possibly reducing the analyst's work satisfaction.

Recently, machine learning (ML) algorithms, today especially the deep learning (DL) ones, have been used to eliminate repetitive work in various business domains. By training ML algorithms using many examples, hidden patterns underlying the data set can be identified and harnessed for making predictions for new data. Therefore, it is possibly worthwhile to apply them to the work of an underground mapping analyst as well. Fortunately with respect to this observation, TerraCarta is open to the adoption of ML based tools into its working processes. It therefore served as the primary industry partner for this work.

In our work, we identified well-performing ML algorithms for predicting the material type of underground utilities registered with GPR, considered by TerraCarta to be an open problem for automation. In doing so, we explicitly took an Intelligence Amplification (IA) approach in which the analyst is not replaced. Rather, through our ML models, our goal was to amplify their intelligence. This allows analysts to quickly identify the material types of simple objects while using more creative, human intelligence-based approaches for complex ones. It would reduce the repetitiveness of one's work while making on-the-shelf knowledge available repetitively.

Our work first reviews the literature available within this specific niche. It introduces Intelligence Amplification by discussing the recent trend towards Industry 4.0 and smart working and introduces IA frameworks conceptualizing its main concepts. We then link IA to the recent advances in ML and DL, which are subsets of the broader Artificial Intelligence field. We narratively review its salient concepts and prominent algorithms used today before moving on to a narrative review of the GPR and other mapping techniques used in practice. It is finally followed by a systematic review of the literature about applying ML to GPR imagery. The results suggest that object detection is a trivial problem, while indeed object material recognition remains an open problem, together with estimating object dimension and object size.

Subsequently, this work proposes, designs and validates three novel classes of ML algorithms for object material recognition. Inspired by previous studies, traditional histogram-based and Discrete Cosine Transform (DCT) based approaches for feature extraction are combined with novel Convolutional Neural Networks (CNNs). A third class using preprocessed but original GPR B-scan data is then also combined with CNNs. Our efforts demonstrate that the histogram and DCT based approaches are ineffective with accuracies plateauing around 60%. This is explained through the possible blindness of the CNN due to double feature extraction. The approach using original GPR data shows promising results with maximum accuracies of 81.5% on average across 10 training folds. Outliers of 86% are reported by training many variations. Our results also suggest that ML models for GPR are sensitive to noise and gain applied to the input data, besides regular ML sensitivities like the model's architecture, activation functions and hyperparameters such as the learning rate.

Our work also demonstrates the technical feasibility of the IA scenario. It designs and develops a web application into which an analyst can upload GPR imagery for subsequent analysis using an arbitrary ML model. Finally, beyond technical feasibility, it also discusses its business value by showing its relationships to a generic GPR data acquisition and analysis process and gathering early feedback from project stakeholders.

Acknowledgements

It was April 2018 when I was first introduced to the concept of a Ground Penetrating Radar and its utilization for mapping utilities. It was in the Bastille at the UT when I was asked to participate in a project that researched opportunities for applying machine learning to automate parts of the GPR based utility mapping process. The data intrigued me: the noisy, inscrutable hyperbolic signatures that were entirely random to me could nevertheless be interpreted by GPR analysts. It defined which ML projects I appreciate most, namely the ones with noisy, industrial data where the solution is far out of sight.

In the first place, I thus wish to thank the people from *De Gasfabriek* in Deventer who back then introduced me to this topic. It was the good time, good place for me to transition from an unsuccessful software project to the field of machine learning. One year later, this master's thesis is the end product of more than a full year of deeply acquiring machine learning knowledge. In parallel, the project has allowed me to work together with Gertjan and Jeroen on various data related projects. Thanks to them and once again the *Gasfabriek* people who have supported this parallel trajectory with all kinds of advice related to technology and business. It has taught me a lot.

Since January 2019, when my actual thesis work started, I was given the opportunity to further develop my ideas about applying machine learning to GPR imagery. Together with De Gasfabriek, TerraCarta B.V. provided the opportunity for me to learn, to experiment, to fail and – eventually – to succeed in finding promising machine learning models and developing a tool which can be used by end users. This was an interesting yet intense time, with all kinds of other software projects running in parallel.

In particular, I wish to thank Karel Meinen and Harjo Claus from TerraCarta for having the confidence that this relatively challenging project could be brought to a successful end. Your efforts have allowed me to explore the GPR even further, and especially its use in practice. It was such a revelation to experience how the GPR is treated by many as a miracle method, while in fact it needs to be applied with care! Without TerraCarta's background expertise, my work would not have been possible.

Another TerraCarta employee whose efforts to help me cannot go unnoticed is Michael Fehér. Michael has been a great help in exploring the geophysical and by consequence mathematical thus theoretical aspects of the GPR. The quick success with the GprMax simulations could not be achieved without Michael's work, who provided the electromagnetic configurations underlying the simulations and managed the simulation process on TerraCarta's hardware. A characteristic example of the way in which we worked together is that afternoon when we phoned about converting that GprMax waveform from a discrete set of amplitudes into a smooth, continuous function. Dropping everything out of your hands to hop into your car to pay me a visit in Deventer from Hoogeveen is not what everyone would do.

Cecill, thanks must go out to you as well for those afternoons and email sessions in which we discussed the theoretical suitability of the DCT for machine learning feature extraction. Additionally, thanks for reading those papers yourself to help me move forward. Your extensive knowledge on signal processing has allowed me to understand in depth what the DCT does and why it could be useful to my project.

I am also grateful for the trust put into my work by my graduation committee, being Maria Iacob and Klaas Sikkel. Graduation projects like those often go beyond one's area of expertise due to the depth of the topic – and projects related to partially automating the interpretation of radio waves are not the most common ones in the BIT program. Nevertheless, they always remained confident that it would be brought to a successful end. Additionally, they provided valuable advice for my literature review, provided advice on my thesis in general and helped me navigate the waters of a sometimes bureaucratic university. Thank you for this.

Finally, I wish to thank Shell Projects & Technology for providing TerraCarta with relevant hardware for their geoscience projects, which could be used for my research project as well.

If you are not included – fear not. Although I tried, it may be the case that I have forgotten you. Shame on me, then, but know that I am grateful for your efforts.

Table of contents

List of Figures

List of Tables

List of Acronyms

Acronym	Full meaning
Adam	Adaptive Moment Estimation
ADR	Action Design Research
AI	Artificial Intelligence
ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
BI&A	Business Intelligence & Analytics
BIE	Building, Intervention and Evaluation
CLR	Cyclical Learning Rate
CNN	Convolutional Neural Network
CSV	Comma-Separated Value
DCT	Discrete Cosine Transform
DL	Deep Learning
DN	Binnendiameter (*Dutch for ID; Inside Diameter)*
DNN	Deep Neural Network
DSRM	Design Science Research Methodology
FDTD	Finite-difference time-domain
FL	Fuzzy Logic
GPR	Ground Penetrating Radar
GPU	Graphics Processing Unit
GSSI	Geophysical Survey Systems, Inc.
HDPE	High-density Polyethylene
H-LAM/T	Humans and their Languages, Artefacts and Methods in which they are Trained
IA	Intelligence Amplification
ICT	Information and Communication Technologies
ID	Inside Diameter
IoT	Internet of Things
KLIC	Kabels en Leidingen Informatie Centrum
kNN	k-Nearest Neighbors
LoA	Levels of Automation
LOOCV	Leave One Out Cross Validation
LoRaWAN	Long Range Wide Area Network
LRRT	Learning Rate Range Test
LSTM	Long Short-Term Memory
MABA-MABA	Men Are Better At – Machines Are Better At
ML	Machine Learning
MLP	Multilayer Perceptron
MQ	Main Question
PCA	Principal Components Analysis
PEC	Perfect Electricity Conductor
PoC	Proof-of-Concept
RD	Radio Detection
RDBMS	Relational Database Management System
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC AUC	Receiver Operating Characteristics Area Under Curve
SGD	Stochastic Gradient Descent
SLP	Single-layer Perceptron
SQ	Sub Question
SVM	Support Vector Machine
Tanh	Tangens hyperbolicus
WIBON	Wet informatie-uitwisseling bovengrondse en ondergrondse netten en netwerken
YOLO	You Only Look Once

1: Introduction

The Netherlands is a densely populated country. With a population of approximately 17.15 million people in July 2018 and an area of approximately 41.500 million square kilometers, population density exceeds 400 people per square kilometer [1]. The distribution of population density is additionally rather skewed, with a large number of people living in the Randstad area in the western part of the country [1].

It is the 10th highly developed country with a Human Development Index of 0.931 [2]. Unsurprisingly, and perhaps consequently, The Netherlands has a highly developed underground infrastructure [3]. The total length of cables and pipelines put underground exceeds 1.7 million kilometers [4]. Examples of such cables and pipelines are telecommunications cables, gas pipelines and sewage pipelines.

One can imagine that these conditions result into a highly complex infrastructural network. If it were static, this would not be problematic, since it would remain untouched. However, the contrary is true: Dutch infrastructure is highly dynamic, with 2016 revenues of construction companies exceeding 88 billion Euros [5].

Various activities can be shared under the umbrella term *construction*. For example, masonry, carpentry, road maintenance, sewage maintenance; they all relate to construction work. Many construction projects employ a combination of these activities. However, only a subset of them is relevant for this work. It will specifically focus on construction activities that interfere with underground infrastructure, and especially excavation work. Annually, in about 5.7% of all excavation work, damage is caused to underground cables and pipelines [4]. In numbers, this means 33.000 incidents annually, and thus approximately one incident every 3 to 4 minutes.

Economically, these numbers convert to at least 25 million Euros in annual damages, with additional non-economic consequences like direct dangers (in case of gas leaks) and outages (e.g. in case of damaged power cables), rendering even higher indirect costs [4].

1.1: Why Damage: Administering Underground Cables and Pipelines

The post-World War II era was a period of economic expansion. In the Netherlands, it was triggered by increased production capacity during the war that could then be fully utilized [6]. It was further accelerated by the European Recovery Program also known as the Marshall Plan [7]. Quickly rebuilding ground-level and underground infrastructure were the initial focus points in this reconstruction era.

At the time, however, no proper method existed for administering new cables and pipelines. As a consequence, distribution network operators (Dutch: netbeheerders) did either not administer the location of their cables and pipelines, or did not use a standardized method. This became problematic in the 1960s, when construction companies could no longer accurately assess where underground infrastructure was located. Construction became rather chaotic.

As a result, in 1967, the first *Kabels en Leidingen Informatie Centrum* (Cables and Pipelines Information Center, KLIC) was founded in the province of Groningen [8]. The organization served as an information management hub for cables and pipelines. Distribution network operators could administer their cables and pipelines in a shared information repository. Subsequently, before commencing any excavation work, construction companies could request information about underground infrastructure from the KLIC repository, and add information about their work to it.

Prior to 2008, the KLIC was self-regulatory, which means that construction companies were stimulated but not required to request or to add information. The Dutch government, although initially stimulating the system of self-regulation, believed that excavation damage remained unacceptably abundant [9]. As a consequence, in 2008, it introduced legislation that requires the proper administration of underground cables and pipelines [9]. In 2018, this legislation was extended and now also includes surface- and aerial-based infrastructure, and is known as *Wet informatie-uitwisseling bovengrondse en ondergrondse netten en netwerken* (WIBON) [10].

The main consequences of the legislation are as follows [9] [10]:

- The KLIC information repository has become part of the Dutch land registry (Dutch: Kadaster).
- Network operators are required to register all cables and pipelines with the Dutch land registry, including maps and drawings.
- Network operators are required to provide all necessary information to parties which aim to perform activities that may interfere with underground infrastructure.

- The protection of cables and pipelines transporting hazardous contents (e.g. gas) or having high economic value (e.g. telecommunications backbones) must be physically protected by network operators. This means that during excavation activities, a representative of the operator must be present to physically identify the cables and/or pipelines.
- Parties aiming to interfere with underground infrastructure (Dutch: grondroerders) are required to report their activities to the KLIC repository 3 to 20 working days in advance, requesting all available information regarding underground infrastructure for the construction site.
- Those parties are required to have maps and drawings available at the construction site.
- Parties that find previously unknown cables and pipelines (Dutch: weeskabels/-leidingen) during excavation activities are required to report them in the KLIC information repository.
- The Dutch land registry will attempt to find the owner of this cable or pipeline, or solicits with municipalities to report it as an orphan. Municipalities are subsequently required to provide such information based on a new KLIC request prior to any underground infrastructure work.
- Unused cables and pipelines must also be registered, unless they are physically removed.
- When damage occurs, the party interfering with underground infrastructure is required to immediately report the damage to the network operator.

These consequences indicate that information about the location of cables and pipelines is now managed in accordance with an intuitively proper framework. The location of new cables and pipelines and previously found orphans is accurate and, given the national scope of the information repository, relatively complete. However, the number of orphans, given previous inadequacies with regards to infrastructure administration, remains unknown. Although much progress has been made with regards to information management, incidents continue to occur today [4].

A direct consequence of this legislative activity is the emergence of a valid business case for private companies to specialize in underground mapping. Using various methods and techniques, which often originate from geophysics, analysts can identify and segment underground objects in a non-destructive way. TerraCarta B.V. is a Hoogeveen-based underground mapping company which specializes in this domain. The company embraces innovations and attempts to continuously improve its working practices. It will serve as the primary industry partner for this work.

1.2: Detecting Underground Objects with Ground Penetrating Radar

One of the specialisms of TerraCarta B.V. is applying ground penetrating radar (GPR) to detect and analyze underground objects. GPR is a geophysical technique that can be used to scan the Earth using radio waves [11]. Initial work on GPR has emerged in the 1950s and has expanded over time, both in terms of technology and applications. The technology has improved incrementally and application domains like land mine detection, road analysis and utility monitoring have emerged in parallel.

Generally speaking, a GPR device is equipped with a transmitter and a receiver. The transmitter emits radio waves, which will be partially reflected by underground objects [11]. This way, the presence of these objects can be registered. Since different types of underground objects reflect the radio waves differently, experts can distinguish various object types. For example, they can segment gas pipelines from telephone cables.

Conclusions drawn to the analysis by experts close the loop with regards to the business case introduced above. In sum, the legislative framework requires construction companies to accurately assess the underground infrastructure they intervene with. This provides the need for underground mapping, for which technologies and experts have emerged. Their analyses provide the answers that construction companies need, and serve as the commercially viable basis for underground mapping companies.

1.3: Augmenting the Analysis Process with Machine Learning

TerraCarta B.V. describes analyzing GPR data as a labor-intensive and thus time-intensive process. Data on a GPR must be loaded onto a computer, uploaded into GPR analysis software and subsequently scrutinized by experts. Possibly, identified signals must be compared with other sources, such as the KLIC. Although feasible, it is time inefficient.

Another pressing matter is the actual level of expertise that is required before one can successfully analyze GPR imagery. GPR itself is a very accessible method to obtain underground intelligence [11]. Converting the raw output files into a so-called radargram is not difficult. However, at the same time, it is the pitfall of the method, as “(…) many inexperienced practitioners fail to fully appreciate the true nature of GPR wave

propagation and its interaction with the subsurface materials" [11]. Successful GPR data analysis and interpretation thus relies heavily on either theoretical or practical, but intimate knowledge about electromagnetic fields and radio frequencies. For example, signal strength may be disrupted by the electromagnetic properties of the subsurface materials, whereas the direction of moving the GPR device during measurement may result in skewed reflections [11]. Depending on whether the GPR device automatically applies filters to reduce these disruptions, the GPR analyst may need to apply those themselves, recognizing such physical disruptions in a rather noisy radargram [11].

This necessity of expert knowledge for GPR data analysis can be a substantial risk for any private organization that provides underground mapping services. Operations would be disturbed if, for example, an expert leaves the organization. However, if somehow a computer could emulate the expert's knowledge, the risk would be mitigated and perhaps new business opportunities would arise.

Recent developments in the fields of Artificial Intelligence (AI), with machine learning and deep learning in particular, render such emulation theoretically feasible. Machine learning is a discipline that attempts to "construct computer systems that automatically improve through experience" [12]. Within the broader discipline of AI, it has emerged as one of the key research areas, since allowing machines to learn themselves is significantly more efficient than attempting to manually program complex behavior into an algorithm [12]. Consequently, for automating GPR analysis, machine learning algorithms could be our way forward.

Academic works related to the detection of underground objects are manifold [13]. However, the characterization of underground objects – e.g., accurately predicting the material of an underground object – remains a candidate area for future research. In this work, we therefore design and validate novel machine learning algorithms that can be used to accurately predict the characteristics of underground objects.

However, anecdotal evidence put forward by TerraCarta B.V. during initial conversations related to our work results in an understanding that GPR data is too noisy for fully automated analysis. We therefore take a different point of view than many previous initiatives, in which full automation was the goal: rather, we aim to amplify the intelligence of an analyst. Machine learning models trained for this particular goal therefore take a more advisory role compared to the ones designed and validated in recent studies [14] [15] [16]. This stance is called Intelligence Amplification (IA). With respect to this work, it means that we also show the technical feasibility of an IA-based tool for GPR analysis and its embedding in a generic analysis process.

1.4: Problem Statement and Research Objectives

With 25 million Euros in annual direct damages and even larger indirect damages, underground interference is a clear issue in today's civil engineering practice. Private companies that specialize in underground mapping have flourished, but – besides benefiting from their experts' knowledge – face the inherent drawback of the scarcity of such knowledge. Additionally, for workers, analysis tasks may become repetitive.

Machine learning techniques can mitigate this drawback by amplifying the analyst's intelligence by feeding expert knowledge from previous analyses into machines. Given the scant amount of works related to underground object characterization, the potential for applying machine learning to this problem remains unclear and thus a possibly viable path for new research. The rest of this work will work towards exploration of this research problem. By exploring both formal knowledge (from available literature) and practical knowledge (about GPR and underground infrastructure intelligence through our industry partner TerraCarta B.V.), we hope to advance the relevant academic fields with fresh ideas and new findings. We consolidate our findings into a Proof-of-Concept web-based artefact useful to TerraCarta B.V. and similar companies. However, before this is possible, a thorough review of the available literature is imperative.

1.5: Research Questions

The introduction above provides the justification for our work. It was summarized in section 1.4. Following from those is a set of research questions that can be aggregated into one main research question (**MQ**):

> ***MQ.*** *How can machine learning techniques theoretically and practically amplify human intelligence in characterizing buried cables and pipelines?*

It is composed of multiple sub research questions. The first three questions will attempt to build a thorough understanding of the relevant literature:

> ***SQ1.*** *What are the most salient concepts in the field of Intelligence Amplification?*

SQ2. *How does underground intelligence through wave emission and reception work; especially, what data is produced?*

SQ3. *What are the state-of-the-art machine learning architectures used in characterizing underground objects using underground intelligence techniques; especially, GPR?*

Based on the literature review, we design and validate novel ML models for utility characterization. They serve to answer the next sub question:

SQ4. *Which machine learning model architectures are suitable for identifying and segmenting underground objects with ground penetrating radar?*

A small Proof-of-Concept Analysis Tool will be built to demonstrate technical feasibility of the IA scenario. This tool will take as its basis the ML algorithms designed when exploring answer paths for SQ4. In doing so, we attempt to answer the following sub question [17]:

SQ5. *How can a PoC improve the GPR analysis processes at TerraCarta B.V. for underground cables and pipelines through amplifying human intelligence?*

1.6: Structure

The remainder of this work is structured as follows:

2. **Background** presents the results of the literature analysis performed prior to the design activities. It synthesizes the answers to the theoretical research questions (SQ1-3) into a coherent discussion.
3. **Research Methodology** outlines the methodology for the design and validation phase.
4. **Novel Algorithms for Utility Material Characterization** presents the rationale and design for three novel algorithms for utility material characterization, i.e. classification of material type given a hyperbola.
5. **Validation Strategy** discusses and presents the strategy used to validate the algorithms discussed in chapter 4. Specifically, it discusses the various configuration options for the neural networks used.
6. **Validation Results** presents the results found during validation. It also suggests variations which could optimize performance of the models trained initially and presents their performance.
7. In the **Machine Learning Driven IA Tool for GPR Analysts** chapter, we propose the software design and instantiation of a small Proof-of-Concept Analysis Tool which benefits from the machine learning algorithms validated in the previous chapters. This chapter also embeds the tool into a generic GPR analysis process through an architectural analysis.
8. In the **Discussion** chapter, we interpret and explain our results.
9. **Conclusion and Future Work** concludes this thesis and suggests paths that are worthwhile for future research activities.

2: Background

This chapter synthesizes the results of our literature review. It provides the theoretical background with respect to the research goals outlined before, and thus attempts to answer research questions SQ1-3. By answering these questions, the subsequent work planned to attain the research objectives is justified.

Our analysis starts with a historical timeline of industrial revolutions, which aims to provide the wider context of our work. We especially focus on the recent movement towards smart industries and the Industry 4.0 trend, and identify that its Smart Working business application is of interest for our research objectives.

With a pragmatic eye towards the research objectives, we subsequently argue about the dilemma traditionally present within the automation literature. This dilemma entails the question “when should we automate?”. We discuss the binarity of the initial approach towards answering this question. We then present an alternative approach, identified by analyzing literature on human-machine symbiosis, more recently known as Intelligence Amplification (IA). It includes a discussion on how machine learning techniques play an important role in the amplification of human intellect and why this may be worthwhile for our work.

Having obtained a theoretical path towards our research objectives, we then proceed with an analysis of the salient concepts within the field of machine learning, including a historical timeline. It is followed by an analysis of salient geophysical concepts and techniques on underground intelligence. Finally, we combine our analyses into a systematic review of the available literature on using machine learning to amplify intelligence in identifying the presence and structure of underground cables and pipelines.

2.1: Smart Industries and Automation

The world is never a static place. In science, the continuous practice of theorizing results in the emergence of new theories on a daily basis. When new theories gain substantive traction, they may alter collective beliefs among scholars about prominent theories, a process known as a paradigm shift [18]. Sometimes, these shifts are radical enough to be denoted as *scientific revolutions*.

Such shifts do however not only occur in science. Rather, economies are susceptible to such shifts as well. For example, in industry and especially manufacturing, new technologies have changed production processes before – paradigm shifts known as *industrial revolutions* [19].

The first revolution changed agrarian and handicraft economies into machine-dominated ones by introducing steam-powered machines that could mechanize production processes [20] [21]. In the second revolution, the increased availability of electricity ensured that machines could be used more intensively, whereas the third revolution revolved around digitization: the advances in computing from the 1950s onward offered organizations widespread access to information [20].

Today, industries are facing the fourth industrial revolution [20]. This revolution can be characterized as a “global [modernization] movement in the manufacturing industry towards the adoption of recent advances in the ICT realm” [22]. These advances are different than those which triggered the third revolution. For example, organizations can now not only harness the power of information sources (i.e., browse the world wide web), but also gather and interpret raw data in real-time. Mobile and compact communication devices combined with steeply increasing computational power make this feasible [22]. Data generated by measurement devices in the field can subsequently be used to train artificially intelligent systems, which spawns a wide range of predictive applications. Together, these developments open a substantially new world to business incumbents and new entrants.

2.1.1: Essentials of Industry 4.0

All these observations can be traced back to 2011, when the term Industry 4.0 (or, initially: *Industrie 4.0*) was coined in Germany [23]. It is the result of a strategic proposition for securing the competitiveness of the German manufacturing industry. Recognizing the consequences of emerging technologies like the Internet of Things (i.e. sensor technology), cloud services (i.e. computing being commoditized), big data and analytics (e.g. advances in ML), it was introduced in an attempt to maintain the market leadership of German production and to preserve it in the future.

Since then, the term – or at least its meaning – has been adopted globally, and research activities have intensified. This section provides an outline of the essentials of the Industry 4.0 principle. Starting from the observation that it extends the industrial revolutions of the past using new technologies and business

applications, we separately analyze the technology and the business essentials. Subsequently, we argue that the *integration* of the two has become critical for today's business innovations.

2.1.1.1: Technology Essentials

Technology has been an increasingly integral part of the world since World War 2. The first computers and the nascence of the transistor have created an unprecedented volume and growth of technology utilization [19]. Starting as massive pieces of technology only available at the most sophisticated research institutes, computers have moved into peoples' homes and more recently into peoples' hands, as smartphones. Together with the increased interconnectivity between devices through the internet, this transition was the driver of the third industrial revolution that we discussed before, in which the availability of information transformed manufacturing processes and eventually the way of doing business [20, 19].

In the fourth industrial revolution, the role of technology is similarly significant, albeit that this time different technologies serve as the primary driver. Primarily, the advent of the Internet of Things, cloud services, big data and analytics make the transition from the third into the fourth industrial revolution possible [23]. We will now briefly discuss them individually.

The **Internet of Things** (IoT), in [23] known as the Internet of Things and Services and in the Anglo-Saxon countries as the Industrial Internet, is "an information network of physical objects (sensors, machines, cars, buildings, and other items) that allows interaction and cooperation (…) to reach common goals" [24].

The first part of this definition – physical objects that are part of a network and distribute their state – implies that they must somehow be capable of producing data. Indeed, the definition provides the insight that *sensor technologies* allow physical objects to measure object state. Through extensions of the regular internet such as LoRaWAN, these sensor-equipped objects are connected to the internet [25]. Today, objects can thus measure variables at the edge, while transmitting them to centralized processing units over the internet.

Attempts to build such a network date back to the 1970s, but researchers and practitioners faced many challenges [24]. For example, at the time, the internet was nascent and widely accepted protocols did not exist, rendering widespread adoption infeasible. However, due to technological progress, with one of the primary and recent examples being the new Internet protocol IPv6, prior limitations were overcome [23]. In fact, significant efforts both from academia and practice have made apparent its business potential [24].

The IoT is not the only technology that drives the fourth industrial revolution. One of the consequences of the IoT allows us to bridge to one of the other technologies, namely that the IoT produces "large quantities of diverse data" [23]. Data sets that have a substantially large volume are also known as big datasets, or **big data** in short. However, these data sets are not only characterized by volume. Different heuristics for designating a data set as 'big' co-exist with volume. Although no unified view of big data exists, business oriented practitioners often use the so-called 5Vs to describe big datasets [26]:

- Big datasets have a large *volume*, a characteristic that we discussed previously;
- These datasets also have a large *variety*, which means that the data generated is of a substantially varying structure;
- Big datasets can also be characterized as having a large *velocity*, which means that data is generated at a fast and continuously accelerating pace;
- *Veracity* means that these datasets are possibly unreliable and that one should not assume causal relationships to be present by default, even when quantities are large;
- Finally, big datasets are also *variable* in the sense that they have variable data flow rates.

Traditional organizations often have ill-equipped on-premise IT landscapes to support data sets characterized by these 5Vs. Using anecdotal evidence from experience with practice, we can state that often datasets are too *large*, too *rapid* and too *varied* to be supported by traditional relational databases. Additionally, traditional organizations often do not have in-house personnel trained for handling such datasets.

The final V – the variability of the data – is often a technological challenge for organizations. Being used to possessing and maintaining an on-premise IT infrastructure, organizations traditionally cannot cope with this variability. Their hardware simply does not scale well with the characteristics discussed above, either due to the cost structure of the existing infrastructure or due to its technical architecture. This often results in inadequate trajectories towards the adoption of 'big' data and the failure of many such projects.

However, both pioneers from the dot-com era as well as today's modern technology startups do no longer use on-premise IT infrastructures [27]. Rather, they use third-party **cloud services** that "enable ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [28]. Organizations using such services, for example to run their applications and/or store their big datasets, can access them on-demand. They also benefit from rapid provisioning, which means that instantiation of new cloud-based resources (e.g. in case data sets grow too big for the current subscription) is a quick process, especially when comparing it with traditional on-premise data center management. By offering a shared pool of resources (e.g. hardware), cloud service vendors have built a business model around the nascence of those technologies, one that has recently grown significantly [23].

One key aspect of the technologies driving the Industry 4.0 principle is still missing from our discussion, namely the importance of **analytics** [23]. Whereas the IoT produces big data that can be stored and processed by cloud services, data sets must be analyzed before they can provide any business value to organizations. As a consequence, the field is also known as Business Intelligence & Analytics [29] (BI&A). Chen et al. outline a historical timeline of BI&A that contains three phases. In BI&A 1.0, that "has its roots in the long-standing database management field", the focus lies on data extracted from centralized databases, often relational ones (RDBMS). As a consequence, the data itself is often centralized as well, and is often produced by the organization itself, being used for business improvement. Analytical techniques used are mostly statistical in nature [29]. Data warehousing and business performance management using visualization in dashboards are the key analytical activities.

From the 2000s onward, BI&A 2.0 presented the next wave in analytics [29]. The trend from Web 1.0 (i.e. static websites providing information) towards Web 2.0 (i.e. websites on which user interaction is triggered) triggered the production of large quantities of user-generated content. The sheer variety of data produced in BI&A 2.0 over 1.0 is unprecedented. Consequently, together with non-user-generated content like web server traffic logs, user-generated data sets comprise the big datasets that we discussed before. Various new techniques (e.g. social network analysis and applied natural language processing) have augmented the traditional BI&A 1.0 techniques [29]. Additionally, new and often non-relational technologies (e.g. Hadoop and Apache Spark) extend the BI&A 1.0 technologies in order to handle the volume, variety, velocity, veracity and variability of the BI&A 2.0 datasets. Advances in AI, specifically machine learning and more recently deep learning, have democratized creating predictive models. Today, given a suitable data set, creating such models does no longer require an advanced statistical degree.

In 2011, the number of active mobile phones and tablets surpassed the number of laptops and PCs for the first time, creating a "highly mobile, location-aware, person-centered, and context-relevant" network of devices [29]. These technological advances – that primarily occurred in terms of technology adoption – opened up the era of BI&A 3.0, in which analytics is becoming distributed.

The Internet of Things is one of the examples of this trend towards distributed data collection and analytics. Together with the other technologies – cloud services, big data and analytics – organizations transition from being purely information-driven into becoming data-driven. Successful utilization of these technologies spawns a wide array of new business opportunities, some of which we will cover next.

2.1.1.2: Business Essentials

According to Frank et al., the technology essentials discussed in the previous section comprise the so-called *base technologies* of the fourth industrial revolution [30]. According to them, the base technologies support front-end technologies. These are technologies that are visible to end users and may also be viewed as applications of the base technologies. The initial document describing the *Industrie 4.0* principle considered one front-end technology, namely Smart Manufacturing [23]. Frank et al. extend this application with three others, which results in the following four application areas for Industry 4.0 [30]:

- **Smart Manufacturing**, i.e. improved internal production operations by deploying base technologies;
- **Smart Working**, i.e. improved operational activities by deploying base technologies;
- **Smart Products**, i.e., improved end products by deploying base technologies;
- **Smart Supply Chain**, i.e. improved integration within supply chains to optimise processes by deploying base technologies.

According to Frank et al., the Smart Working application area "considers technologies to support worker's tasks, enabling them to be more productive and flexible to attend [their] (…) requirements" [30]. This definition

and by consequence the concept of Smart Working aligns with the research objectives outlined in section 1.4. If the intelligence of GPR analysts is amplified, it might be the case that their operational activities are improved. Consequently, the subsequent parts of this work focus on Smart Working using (a subset of) the base technologies discussed in the previous section.

2.1.1.3: Integration of Technology and Business through Automation

Traditionally, organizations have developed in a siloed fashion [31]. Organizations would distribute specific responsibilities to departments which would guard these strongly. As a consequence, one can view traditional organizations as systems of departments that would work together, but which would constitute their own smaller organizations within the larger one. These departments may also be viewed as silos, hence the name siloed organization. Other factors – such as mergers and acquisitions – increased the siloed organizational structure.

This silo structure impacted organizational technology landscapes as well. Vendors would produce proprietary software that ran on proprietary hardware. Departments possessed or claimed a mandate to decide about their own IT infrastructure themselves [31]. Central IT policies did not or scantily exist. When it became apparent that working cooperatively with other departments would improve overall results, many organizations faced difficulties with integrating their technology landscapes. As a consequence, integration was seen as a curse rather than a blessing.

The Industry 4.0 principle radically breaks with this traditional viewpoint. Rather, it prescribes that technologies are integrated both with other technologies and into production processes [22]. This integration occurs at various levels:

- **Horizontally**, which means that distinct technologies are combined to create a technologically sound solution for production processes (e.g., the combination of IoT and machine learning for analysis of livestock health);
- **Vertically**, which means that technologies are integrated over the full spectrum from hardware through software to business IT relevance (e.g. sensor measurements becoming visible in a geographical information system, providing accurate insights to government officials).
- **Circularly,** which means that horizontal and vertical technology integration is combined to create a technologically sound and organizationally relevant IT solution for production.

The circular integration produces a so-called *Cyber-Physical System*: “smart machines, (…) systems and (…) facilities that have been developed digitally and feature end-to-end ICT-based integration” [23]. According to the German strategic report, such systems are capable of autonomously exchanging information, triggering actions in other machines and controlling each other independently.

As a consequence, for the research objectives of this work and the Industry 4.0 observation that state-of-the-art benefits may benefit Smart Working, we may view automation as a critical element towards an artefact that benefits TerraCarta B.V. Literature on automation however presents a dilemma, which is the question “how much should we automate?”. Next, we will therefore analyze this dilemma by reviewing the traditional binarity present in automation decisions and how it has transformed into an automation spectrum.

2.1.2: Automation and its Adoption: Socio-technical Systems

In the dawn of the computing era, technologists considered automation to be a binary choice: either the path towards solving a problem was fully automated or solving the problem was fully left to human beings [32]. This viewpoint shifted in the early 1950s. Researchers, in an attempt to characterize the field of human-computer interaction, proposed a set of heuristics to distinguish between what “men are better at” and what “machines are better at” (the so-called MABA-MABA heuristics). Table 1 presents these heuristics based on the original work ([32]) and its analysis in [33].

Criticism, however, ensued from these heuristics, since traditional engineers are trained to produce a system that minimizes variance in its performance [32]. Given their goals and the MABA-MABA heuristics, critics argued that engineers would be biased towards producing systems that are fully automated, replacing human beings and leaving them without a job. In an essay demonstrating her expert opinion, Mary Cummings argues that “indeed, [engineers are] trying to “capitalize on the strengths [of automation] while eliminating or compensating for the weaknesses”” [32]. Practice thus demonstrates that the MABA-MABA insights would not reduce the binarity of the automation dilemma; rather, it produced an increased tendency towards ill-considered automation.

Attribute	Machine	Human
Speed	Superior	Comparatively slow
Power output	Superior in level in consistency	Comparatively weak
Consistency	Ideal for consistent, repetitive action	Unreliable learning and fatigue are factors
Information capacity	Multichannel	Primarily single channel
Memory	Ideal for literal reproduction, access restricted, and formal	Better for principles and strategies, access is versatile and innovative
Reasoning computation	Deductive, tedious to program, fast and accurate, poor error correction	Inductive, easier to program, slow, accurate, and good error correction
Sensing	Good at quantitative assessment, poor at pattern recognition	Wide ranges, multifunction, judgement
Perceiving	Copes with variation poorly, susceptible to noise	Copes with variation better, susceptible to noise

Table 1: *MABA-MABA heuristics.*

In response to this trend, new research activities transformed the binarity between automation and human problem-solving into a spectrum named Levels of Automation (LoA) [34]. These levels aim to "help (…) understand the nuances of how humans could interact with a (…) system in a decision-making capacity" [32]. The levels extend the binary view of automation and allow automation projects to partially include human beings in artefact operation. The LoAs, freely derived from [32], are presented in Table 2, representing the degree of automation in increasing order.

LoA	Automation description
1	The computer offers no assistance: human must take all decisions and actions.
2	The computers offers a complete set of decision/action alternatives.
3	The computer narrows the selection of alternatives down to a few.
4	The computer suggests one alternative out of the alternatives.
5	The computer executes the alternative suggestion if the human approves.
6	The computer allows the human a restricted time to veto before automatic execution.
7	The computer executes automatically and necessarily informs humans.
8	The computer executes automatically and informs humans only when asked.
9	The computer executes automatically and informs humans only when it decides to do so.
10	The computer decides everything and acts automatically, ignoring the human.

Table 2: *Levels of Automation.*

That a spectrum-based approach towards automation projects does allow for more flexibility can be demonstrated by introducing the concept of a *socio-technical system*. Such systems "consist of a cluster of elements, including technology, regulation, user practices and markets, cultural meaning, infrastructure, maintenance networks and supply networks" [35]. Freely interpreted, socio-technical systems are networks of technology, social actors (e.g. human beings or organizations) and social structures (e.g. economies and cultural values within these economies) that influence each other. It can thus be used as a starting point for studying how technology impacts societal actors and structures, and how these impact technology in return.

Taking this viewpoint and combining it with the binary stance initially employed towards automation, it becomes clear that full automation may not always be viable. For example, Döppner et al., in an attempt to construct an air transportation logistics artefact, found that fully automating air transportation processes was impossible due to regulatory aspects [36]. They thus concluded that, although technology itself is capable of sufficiently performing tasks related to these processes, the socio-economic aspects of the socio-technical system disallowed full automation. Since when using the binary approach the other option would be *no*

automation, this example clearly demonstrates its limitations, and why the flexibility provided by e.g. LoAs is relevant for practice.

Taking the LoA stance towards automation projects, Döppner et al. continued their pursuit of an artefact that utilizes advances in digitization in resource allocation of air transportation logistics containers. Recognizing that a case for full automation was not realistic, they produced an artefact that benefitted from all MABA-MABA strengths by joining them. For this, they used an approach that is today known as Intelligence Amplification (IA).

2.1.3: Intelligence Amplification

In 1956 in *An Introduction to Cybernetics*, William Ross Ashby presents the vision that human intelligence can be increased using an amplifier; that is, when "given a little of [it], [the amplifier] will emit a lot of it" [37]. Ashby argues that problem solving is "largely, perhaps entirely, a matter of appropriate selection", thus presenting the vision that solving a problem would be mostly equal to picking one solution out of a set of alternatives.

He then argues that tests of intelligence "are scored essentially according to the candidate's power of appropriate selection", and by analogy argues that intellectual power would thus be equal to one's *power to select a solution appropriately*. Since, he argues, devices can amplify one's power of selection (i.e. assist with picking a solution), by analogy Ashby argues that intelligence can be amplified by machines. Using the definition of an amplifier, it would mean that by giving a little bit of intelligence to a machine, it will emit a lot of it in return. Ashby's comparisons by analogy would be the basis of further research activities and the nascence of a research area that is today known as IA.

If intelligence can be amplified, how to do it? For insights on this question, we turn to Joseph Licklider, who in 1960 presented his vision that the biological process of symbiosis can be applied to the interaction between men and machines [38]. In the work, he introduced biological symbiosis as "living together in intimate association, or even close union, of two dissimilar organisms". With the example of a tree and an insect that would not be able to thrive if they did not co-exist, he shows that two separate organisms can share a "productive and thriving partnership" that is beneficial *and* required at the same time [38].

Grounded in ideas similar to the MABA-MABA heuristics discussed before, Licklider subsequently argued that machines have specific strengths and weaknesses that are perpendicular to those of human beings. However, by coupling "human brains and computing machines (...) together very tightly, (...) the resulting partnership [may] think as no human brain has ever thought (...)" [38]. Freely interpreted, his argument is thus that by producing a symbiotic relationship between human beings and machines, perpendicular strengths and weaknesses would balance out and average performance of the unity would increase. This idea is shared by multiple scholars and practitioners [39, 40].

Both human-machine symbiosis and IA overlap in meaning. We can illustrate this by joining the two visions and combining them with the Levels of Automation discussed previously. In the middle of this spectrum, i.e. the combination between men and machines, we identify the interaction patterns suggested by both Licklider and Ashby. We will use the term Intelligence Amplification in the rest of our work to cover both visions. Given the complexity of GPR data discussed in section 1.3, we deliberately choose the strength of the unity between men and machines for the developments pursued in the rest of this work. For this reason, we will therefore next analyze conceptual IA frameworks, identifying salient concepts to be used in our design activities.

2.1.3.1: Intelligence Amplification Frameworks

If one intends to build artefacts that amplify intelligence, a good starting point would be to identify frameworks that provide a conceptual overview of IA and its relevant aspects. As a result of U.S. government research efforts, one such framework was published in 1962 by Doug Engelbart [41]. In this framework, IA was approached in a systems engineering fashion, in which humans "do not exist singularly but rather as part of a larger system consisting of a Human using Language, [Artefacts], and Methodology in which he is Trained" [42]. Engelbart calls this the *H-LAM/T system*. Image 1 presents our interpretation of Engelbart's work.

The framework explicitly focuses on problem solving [42]. When humans perform problem solving tasks, they use processes (or "little steps of actions" [42] [41]) which they break apart in sets of smaller subprocesses. This results in what Engelbart calls a *process hierarchy*.

When humans (either consciously or unconsciously) break down a process into subprocesses, they can use subprocesses that they previously used in other higher-order processes [41]. Eventually, their experience with certain subprocesses becomes stronger – and humans become specialized in performing certain tasks –

something Engelbart calls a *repertoire* [41]. Repertoires can also be broken apart in more atomic sub repertoires, creating a repertoire hierarchy.

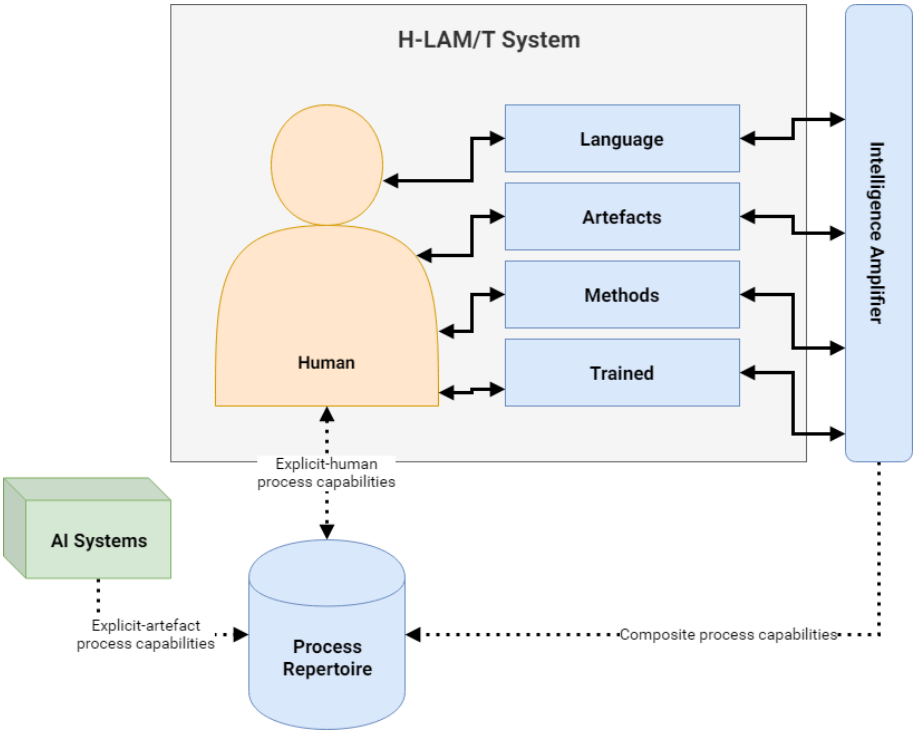


Image 1: *Engelbart's IA framework, freely interpreted.*

Processes within one's repertoire have certain capabilities, freely interpreted as strengths [41]. These capabilities fall within one of three general categories:

- **Explicit-human process capabilities**, which means that a (sub) process is fully executed by human beings;
- **Explicit-artefact process capabilities**, which means that artefacts fully execute the (sub) process, and that human beings may use these machines to fully automate their tasks;
- **Composite capabilities**, which means that (sub) processes are hybrid and executed by human beings and machines in a cooperative or symbiotic fashion.

In the first case, human beings use their H-LAM/T system to solve a problem. In the second case, they outsource solving the problems to machines, in which case they do not use their H-LAM/T system at all. The third case becomes more interesting, however, because machines amplify the strengths of human beings. However, what part of human problem-solving should be amplified? According to Engelbart, one must see a human's problem solving strategy as an interplay between the components of the entire H-LAM/T system [42]. Consequently, the amplification of one's intellect would be possible by improving either language, artefacts, methodologies or training practices. This illustrates that human problem-solving is not improved by simply creating a piece of technology supposed to aid their work. Rather, one must study the process hierarchies, the capabilities present in an individual human's repertoire, and in case these capabilities are partially human or composite, identify where in one's H-LAM/T system improvements would be most optimal.

Using Engelbart's IA framework, the conceptual relationships between humans, their problem solving strategies and machines become clear. However, when one takes a systems viewpoint towards the concepts presented in the framework, what remains unclear is the temporal development of the system and its components. That is, today's world is not static and language, artefacts, methods and training practices are

altered continuously. As a consequence, a human being adapts their process repertoire continuously as well. At the same time, technology improves, possibly disrupting aspects of peoples' process repertoires.

Engelbart recognizes this temporal relationship and by analogy reasons about it. According to him, the human-technology relationship resides in a continuous state of *co-evolution* [42]. That is, technology influences human behavior by offering different paths of least resistance, which humans would take in solving any problem, and therefore alters human evolution. Similarly, because human beings choose to utilize and subsequently improve useful technologies, Engelbart argues that every action made by humans is reflected in technological developments. In short, the relationships between men and machines therefore "create a feedback loop between human actions and technology development" [42].

In a conference contribution, Xia et Maes extend Engelbart's IA framework in multiple ways. They recognize that technology has advanced significantly since the 1960s [42]. For example, mainframes have been replaced by personal computers, tablets and smartphones, some of which are equipped by sensors that make devices highly personal [42]. As a consequence, they argue that *problem solving* should no longer be the only focus of Intelligence Amplification. Rather, taking Engelbart's framework as a starting point, they include additional application areas for IA: it can be used to augment one's memory; to increase motivation; to assist in decision making, and to improve one's mood. This expands the range of IA research to new applications.

Additionally, Xia et Maes argue that due to technology and societal developments, technology has never been more personalized before [42]. According to them, the principle of co-evolution remains valid but becomes personalized. Rather than the question "how to change mankind?" that one could ask about the centralized human-technology relationship, today, the question "how to change myself?" becomes increasingly important.

Building on top of Licklider's vision about human-machine symbiosis, Dobrkovic et al. present an IA framework for enhancing scheduling processes in synchro modal logistics [43]. In their work, they make key assumptions that are both in line with the MABA-MABA heuristics and Engelbart's ideas discussed previously. They argue that "every problem can be decomposed in two or more [subproblems]", which is the essence of Engelbart's process hierarchies. The complexity and workload related to a process are factors that determine whether the capabilities of a process are, in Engelbart's terms, explicit-human, explicit-artefact or composed. They then convert these key assumptions into the notion that high complexity, low workload problems are often explicit-human, whereas low complexity, high workload problems are often suited for machines [43]. This is in line with the MABA-MABA heuristics that introduced us to the concept of IA in the first place [33].

Whereas Engelbart's framework is highly abstract, Dobrkovic et al.'s framework is significantly more pragmatic. It does not distinguish between the components of the interaction between men and machines; rather, it makes suggestions about the division of tasks over human beings, machines and amplifiers (defined as *intelligent agents* in their work) [43]. In it, human beings are given the master role, overseeing the artificially intelligent systems, which assist the human being in doing their work. During run-time, responsibilities vary: human beings have the responsibility for creative tasks and strategic decision making whereas AI systems perform tactical and operational tasks, which often require much data processing. Human beings can designate a level of autonomy to the AI system, rendering it more of an amplifier or a true (autonomic) AI system. In all cases, however, human beings are able to overrule conclusions drawn by the AI system.

Consequently, both frameworks play an important role with respect to our research objectives. Engelbart's framework can be utilized to understand human-technology interaction and human needs, for improving the artefact. Additionally, the framework by Dobrkovic et al. can be used to decide about the degree of amplification that is necessary in supporting certain human processes, once they have been identified.

2.1.3.2: The Role of Machine Learning in Intelligence Amplification

Dobrkovic et al. explicitly mention "the AI" in their framework [43]. We can thus argue that Artificial intelligence techniques can be used for amplifying intelligence. However, there exists no clear boundary as to what is Artificial Intelligence. One class of algorithms that is generally considered to compose Artificial Intelligence techniques is *machine learning*. A machine learning approach, being part of the field of pattern recognition, can be used to "[automatically discover] regularities in data through the use of computer algorithms and with the use of these regularities to take actions" [44]. Example actions are "classifying the data into different categories". Especially for our research objectives – to design and develop an artefact that amplifies one's intelligence by providing judgements about the structure of underground cables and pipelines, i.e. by classifying data – this class of algorithms proves to be an interesting candidate for further analysis.

2.2: Machine Learning Techniques

In this section, we analyze today's practically relevant machine learning (ML) techniques. Our analysis begins with the essentials of ML that are relevant for any such project. Based on an analysis of machine learning algorithms primarily used today, we cover two recent classes of algorithms: Support Vector Machines and Artificial Neural Networks [45]. We assume basic mathematical knowledge to be present with the reader.

2.2.1: Machine Learning Essentials

This section covers the essentials of any machine learning project. We will first discuss common machine learning objectives. We then continue and cover four high-level machine learning approaches, namely supervised, unsupervised, self-supervised and reinforcement learning. Subsequently, we discuss the required split between training, validation and testing data. The analysis then moves forward by highlighting critical aspects of data pre-processing and is concluded by recognizing the challenges faced in ML projects.

2.2.1.1: Common Machine Learning Objectives

By definition, one always approaches a ML problem with a set of objectives in mind. These objectives are often similar to those used by humans in problem-solving situations. For example, if a human would need to put pictures in two semantically different buckets, the ML problem would have the same objective. In the case of the pictures, the objective would be to create a model that *classifies* images with a high accuracy. However, there exists a wide array of machine learning objectives [46]. We present them in Table 3.

Objective	Task at hand
Binary classification	To categorize an input sample into two exclusive categories.
Multiclass classification	To categorize an input sample into more than two categories.
Multilabel classification	To assign a variable number of labels to an input sample.
Scalar regression	To predict a continuous scalar value based on an input sample.
Vector regression	To predict a set of continuous scalar values based on an input sample.
Sequence generation	To predict the caption of an image. It is a higher level objective, often making use of lower-level classification techniques.
Syntax tree prediction	To decompose a sentence into a syntax tree.
Object detection	To detect certain objects in a picture by drawing a bounding box around them. It is a higher level objective, often making use of lower-level techniques, like classification and regression, e.g. vector regression.
Image segmentation	To classify every pixel in an image to a certain class, in order to sharply segment objects in images.
Dimensionality reduction	To reduce the complexity of an input data set.
Clustering	To identify certain groups in an input data set.
Autoencoding	To e.g. generate target labels for a supervised training set.

Table 3: *Machine learning objectives.*

2.2.1.2: Machine Learning Approaches to Attain ML Objectives

Machine learning is a vast field [47]. Various approaches are used for solving machine learning problems. These approaches comprise numerous algorithms that generally fall into one of four categories [46].

The first category is *supervised learning* [46]. One starts a supervised learning project with a data set that contains both the *input data* and the *target values*. For example, a data set used in supervised learning may be similar to the one presented in Table 4.

Input			Target
Color	**Shape**	**Weight**	**Result**
Yellow	Curved	100 grams	Banana
Red/yellow	Circular	65 grams	Apple
Yellow	Curved	80 grams	Banana
Red	Semi-round	15 grams	Strawberry
…	…	…	…

Table 4: *Example structure of a data set used in supervised learning.*

This mapping can be represented as $y: f(x)$, or, there exists some function f that maps input *x* (e.g. yellow, curved, 100 grams) to output *y* (e.g. banana). The goal of supervised learning is to identify some function $\hat{y}: f(x)$ that best matches the original function for new data. By consequence, in a supervised learning project, one aims to build models that derive patterns from examples [46]. It is the de facto standard approach today.

An approach that resembles supervised learning is *self-supervised learning* [46]. It is supervised in the sense that both input data and target labels are used, but they are not human-annotated. Rather, the annotation and thus labelling of target labels has been performed by a machine.

Self-supervised learning is however a fuzzy approach in between supervised learning and *unsupervised learning*, in which input data but no target data is used [46]. Rather, unsupervised algorithms are capable of finding "interesting transformations of the input data (…) for the purposes of data [visualization], data compression, or data denoising". Often, unsupervised learning and supervised learning go hand in hand, as a result of the complexity of real-world datasets [46]. For example, with *dimensionality reduction* presented in table 4, the dataset is first relieved of complexity, after which it is used in a supervised learning problem.

The fourth approach to machine learning is *reinforcement learning* [46]. Like self-supervised learning, it resides somewhere in between supervised and unsupervised learning. This approach introduces an intelligent agent which receives information about its environment. It then chooses actions of which it believes maximize the agent's objectives. Based on the outcome, the agent may adapt its internal wiring, in order to improve its understanding of the environment. Today, it is mostly a research area, but experts argue that commercial real-world applications are imminent [46]. Reinforcement learning techniques may be employed in fields such as education, self-driving cars and robotics.

2.2.1.3: Training Data, Validation Data and Testing Data

Machine learning models are often trained in an iterative fashion [46]. In the case of supervised learning, models learn by example, and consequently example data is required. The data set that serves as example data for training is also called the training set. However, we discussed before that models need to be optimized to best match the original function that maps inputs to outputs. This is known as the *predictive power* of the model. If during optimization we test the trained model with our training data, we will falsely find high accuracies. Another dataset must thus be used, which is called the *validation dataset* [46]. It contains similar data that the model has not seen before. Often, it is a subset of the original data set apart a priori.

Although a model must preferably acquire substantial predictive power, it must also be able to generalize this power to data it has never seen before. That is, a model needs *generalization power* as well. However, every time a small part of the model is adapted based on the validation data, some implicit knowledge about the validation data leaks into the model [46]. When using multiple iterations, the total information leak increases significantly, introducing the same problem as with testing performance on training data. Therefore, final model performance cannot be evaluated using the validation data. Another data set, named the *test set*, is used for this [46]. Similarly, this is often a subset of the original data set apart a priori. Once a model has finalized its training, it is evaluated against this set. Only then, the predictive *and* generalization power of a ML model can be evaluated objectively.

The split between training, validation and test data can be made manually. However, more advanced techniques can be used as well, such as simple hold-out validation and k-fold cross validation [46]. These are especially effective in situations when the amount of data is limited.

2.2.1.4: Data Pre-processing

Before data can be fed into a machine learning model, it must be prepared to fit the particular model. This preparation phase is also known as *data pre-processing* [46]. Techniques used in the pre-processing phase are highly domain specific. Nevertheless, it is possible to identify some techniques that overlap those problem domains. We discuss them next.

First, it is often necessary to transform the input dataset into vectors [46]. By mathematical definition, one-dimensional vectors can be represented as sets of scalar values, i.e. numbers. Today's most widely used machine learning frameworks, such as Tensorflow, require for efficiency reasons that data is input in vectorized format [46]. Consequently, one of the important techniques used within data pre-processing today is *data vectorization*.

Second, input data must often be normalized. That is, reducing variance present within the data. The lower the variance, the less models fluctuate during training and the better models can handle data sets [46].

Additionally, most values in the data should preferably be small. *Data normalization* is therefore also a widely used technique in the data pre-processing phase.

2.2.1.5: Underfitting and Overfitting

A common challenge in training machine learning models is the delicate balance between prediction power and generalization power [46]. The aim of any machine learning engineer is to produce a model that can make powerful predictions. On the other hand, the value of said model decreases if it cannot attain similar performance when tested against data it has never seen before. Consequently, it must also have the power to generalize to unseen data.

When there is still progress to be made with regards to both prediction and generalization power, the model is said to be *underfit* [46]. One can identify the opportunity for improvement by inspecting the training and validation accuracies; in case they both improve, the model can be improved. It is then wise to continue training.

However, when accuracies on training data improve while validation accuracies stall or degrade, the model is said to be *overfit* [46]. From that point onward, it will lose the power to generalize, after which it becomes less suitable for unseen data. It is therefore imperative for a machine learning engineer to identify the presence of underfitting and overfitting and to apply mitigation measures.

Today, various state-of-the-art tooling is available for developing machine learning models, such as Tensorflow and, possibly, Keras that can run on top of it [46]. These tools have built-in functionality that allows engineers to determine a priori when models should stop their training process. The best performing model with the finest balance between prediction and generalization can be saved automatically. Additionally, for the various model types that will be discussed next, various mitigation strategies are available that reduce their susceptibility to underfitting and overfitting [46].

2.2.1.6: Transfer Learning

For supervised learning projects, we argued, labelled training data is required for the identification of the truth hidden in the data. However, in many practical cases, it turns out to be the case that such training sets are unavailable. For example, it can be expensive or impossible to collect and label the data [12]. In those cases, the concept of transfer learning may be used.

In transfer learning, training data for one problem domain is used as training data in another [12]. It is inspired by the transfer of knowledge from one human actor to another. Multiple forms of transfer learning exist, based on whether no training data is available or whether training data has limited quantity. Those techniques have shown to be capable of improving machine learning models. Consequently, the notion of transfer learning can be useful to the application of the machine learning algorithms discussed next.

Algorithm	Application area
Artificial Neural Networks (ANN) and Deep Learning (DL) [48]	Wide-spectrum application of machine learning.
Autoregressive Integrated Moving Average (ARIMA) [48]	Classic time series, such as emission data.
Seasonal ARIMA [48]	Classic time series, such as emission data.
Support Vector Machines (SVM) [48, 46]	Wide-spectrum application of machine learning.
k-Nearest Neighbors (kNN) [48]	Wide-spectrum application of machine learning.
Fuzzy Logic (FL) [48]	Determining the position of a data point within a spectrum.
Simple Exponential Smoothing [48]	Classic time series, such as emission data.
Wavelet Transform [48]	Classic time series, such as emission data.
Holt-Winters models [48]	Classic time series dealing with seasonality.
Gaussian Process [48]	Data sets that contain stochastic processes.
Decision Trees and Random Forests [46]	Wide-spectrum application with data sets containing multiple parameters.
Gradient Boosting Machines [48]	Wide-spectrum application of machine learning.

Table 5: *Classes of machine learning and statistical models.*

2.2.2: Overview of Machine Learning Algorithms

Today's world considers the deep learning (DL) branch of machine learning as a highly promising area for future developments [12]. We will cover Artificial Neural Networks, partially composing this branch, in the following section. In this section, we aim to highlight the state-of-the-art in other ML areas. Practitioners often consider them to be traditional, yet they remain highly relevant for today's machine learning projects [46].

We identified various traditional algorithms by narratively reviewing the state-of-the-art in the literature. For example, Parmezan et al. [48] identified various classes of algorithms that are useful for time series applications. Additionally, Chollet provides a historical overview of the developments within the field of machine learning, including classes of algorithms that gained traction over the years [46]. In Table 5, we present some of these algorithms. From practical experience, we note that in today's machine learning world Support Vector Machines and Artificial Neural Networks (or DL) are most prominently used. We will therefore next proceed with a more detailed exploration of those two classes of algorithms.

2.2.3: Kernel Methods: Support Vector Machines

In the early days of machine learning and pattern analysis, linear models were considered to be the state-of-the-art [44]. Those models are linear functions of the input data, which allows them to distinguish between e.g. classes when data is linearly separable. They operate using some function $\hat{y}: f(x)$ that maps some input x to some output ŷ by using a parameter weights vector w [44]. The training process involves taking a subset of the training samples, inspecting them, and consolidating the ground truth hidden within them into w. Production usage of said models thus does not involve actual training data; rather, production data is compared with w, after which conclusions about the production sample(s) are drawn.

However, there exist machine learning techniques that do make use of training data during production usage. These are known as kernel methods [44]. Contrary to the linear models discussed before, kernel methods do make use of the training data during production usage, which can be mathematically represented as follows:

$$\hat{y}: f(x') = sign \sum_{i=1}^{n} w_i y_i k(x_i, x'),$$

where

- $\hat{y}$ is the predicted label and $\boldsymbol{x'}$ is the production-usage data sample;
- The **sign** function rounds the output O of the classifier to $O \in \{-1, 0, +1\}$;
- The sum iterates over *n* training samples, of which per iteration the learned weight $\boldsymbol{w_i}$ and target value $\boldsymbol{y_i}$ are used together with the output of the kernel function $\boldsymbol{k}$;
- The kernel function $\boldsymbol{k}$ computes the similarity between the training sample and the production-usage data sample.

In plain English, kernel methods thus work by continuously checking new samples against the set of training samples, of which their individual importance has been learnt in the training process. Kernel methods use so-called *kernel functions* for computing the similarity between a production sample and an individual training sample [44]. Such functions map the possibly non-linear input data into a higher-dimensional space that ensures the presence of a clear decision boundary, rendering e.g. classification possible.

A naïve way of using those kernel methods would be to physically map every training sample into a higher-dimensional space first, then computing similarities once the entire higher-dimensional space is computed. This is computationally expensive, giving rise to what is known as the *kernel trick*: by choosing kernel functions that allow a direct input-similarity mapping, the computationally expensive step can be avoided.

Applying kernel tricks in kernel methods has boosted their usage [44]. It is thus no surprise that various kernel methods have been developed over the years. However, they still face scalability issues given their mathematical representation. Upon closer inspection of the representation presented above, we see that similarity between input value and training sample is computed for every training sample. This is, once again, computationally expensive. However, reducing the number of training samples results in underfitting one's machine learning model, and is therefore impractical with respect to the rise of big datasets.

To face this challenge, a specific kernel method named *support vector machine* (SVM) has become popular in the late 1990s [44]. SVMs operate with a subset of the entire set of training samples and are therefore considered to be a *sparse* method.

However, in order to identify which training samples the SVM considers during operation, we must first discuss the decision boundary used by SVMs. If a training set is linearly separable (either directly in feature space or by using the kernel trick), by definition there exists at least one solution for doing so [44]. However, in case multiple solutions exist, which is best? This is what distinguishes SVMs from other types of kernel methods, besides the training subset: it is a maximum-margin method [44]. That is, when attempting to find the decision boundary between classes, it attempts to find the best boundary possible, by maximizing the margin between the so-called support vectors and the decision boundary.

Those support vectors are the training samples that we intended above. But which samples should we select as support vectors? Before training a SVM, it is possible to configure the distance from the vector that should be used for automatically selecting support vectors. By default, however, SVMs select vectors closest to the decision boundary.

By making non-linear data linearly separable via its mapping onto a higher-dimensional space, SVMs extend the strengths of the linear methods discussed above. It is therefore no surprise that they gained significant traction in the 1990s [12]. Today, they face competition from a class of machine learning models named Artificial Neural Networks, which we will discuss next. However, SVMs remain a popular tool in today's machine learning arsenal.

2.2.4: Artificial Neural Networks

A biologically inspired class of machine learning algorithms is the class of Artificial Neural Networks (ANNs) [45]. The algorithms are grounded in research into the human brain and have recently spiked in popularity with the rise of so-called Deep Neural Networks (DNNs), a subclass of ANNs [12]. In fact, DNNs have completely replaced SVMs and decision trees in various applications [46]. However, besides DNNs, there exist other classes of ANNs which have been studied for decades [45].

Although ANNs are a subclass of machine learning models, they are different than the more traditional machine learning approaches covered previously [45]. Three specific characteristics allow us to distinguish ANNs from these other approaches. First, ANNs are data-driven and self-adaptive. This means that few a priori assumptions are required in order to utilize ANNs in any data-related problem.

Second, contrary to more traditional methods, ANNs have the power to generalize. That means, they are more capable than traditional methods to identify the possibly invisible relationships between inputs and outputs within the data set, ignoring the noise present in real-world datasets [45]. As a consequence, they are capable of making predictions with significant accuracy on data they have never seen before. Why ANNs are capable of identifying these relationships is because they are universal function approximators [45], the third characteristic used for distinguishing ANNs from more traditional models. That is, they are capable of approximating any continuous mathematical function "to any desired accuracy" [45]. From this, it follows that, recognizing the quality of a dataset, ANNs have the power to generalize.

As a consequence, ANNs have been used for a wide variety of tasks [45]. Example application areas are health care, manufacturing, education, financial modelling, policing, and marketing [12]. We will next analyze the general components of ANNs and their training process. Subsequently, we will individually cover the salient aspects of various types of ANNs that we consider relevant to either the historical developments of ANNs or the research objectives for this work.

2.2.4.1: ANN Components and Training Process

Mathematically, any mapping between real-world input/output data can be represented as $y: f(x)$, i.e. there exists some function **f** that maps an input value **x** to an output value **y**. Globally, an ANN is thus conceptually a function f that maps **x** to an output value **ŷ:** $\hat{y}: f(x)$ [47]. Training such a network is thus equal to minimizing the difference between **y** and **ŷ** for any input value **x**. In plain English, the goal of an ANN is to numerically find a function that best represents the input/output mapping present in the original data.

But how to do this? ANNs attempt to mimic the structures present in human brains [45]. Those consist of neurons and pathways that interconnect them, named synapses. ANNs are sets of mathematical formulae that transform inputs into outputs, called neurons. A neuron can store a so-called weight vector (**w**) in an artificial memory that it, when triggered, multiplies with its input vector (**x**). The resulting scalar value is then propagated to the next neuron in line, if there exists any [47]. When starting the training process of an ANN, the weights of the neurons are often initialized randomly, and then adapted during training [47].

Following from this is the observation that neurons may be grouped into layers [47]. They are interconnected in the sense that layer output is used as input for the subsequent layer. These interconnections represent the synapses present in a human brain. However, since ANNs are known to be universal function approximators, it must be possible to compute both linear and non-linear continuous functions [45]. That is, the output values produced by the neurons must be capable of being non-linear. The scalar value produced by multiplying **x** with **w** is however the result of a linear operation, namely the computation of the dot product of both vectors. A missing piece is therefore how non-linearity is added to this linear computation.

This is where *activation functions* enter our discussion. Those are functions that map any real input to a real output before the data is passed on to the next layer. A wide array of activation functions exists. In fact, any continuous function can be used. However, as we will see next, the differentiability of an activation function determines its usability [45]. Additionally, it is preferably bounded somehow (i.e. have a maximum or minimum value) and should preferably increase monotonically. Consequently, only a small set of activation functions is relevant for practice. They have been outlined in Table 6. The linear function $f(x) = x$ can also be used, but its use is practically irrelevant because the model remains linear in that case.

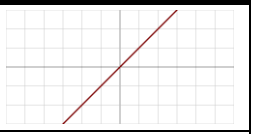
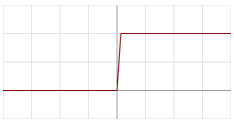
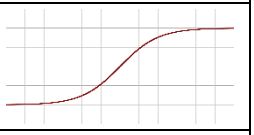
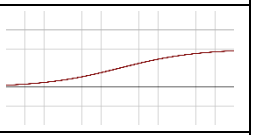
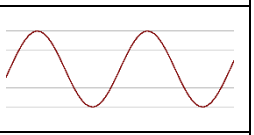
Activation function	Mathematical representation	Function plot
Identity / linear [45]	$f(x) = x$	
Heaviside step function [49]	$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$	
Tangens hyperbolicus (tanh) [45]	$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	
Sigmoid [45]	$f(x) = \frac{1}{1 + e^{-x}}$	
Sine [45]	$f(x) = \sin(x)$	
Rectified Linear Unit (ReLU)	$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$	

Table 6: *Most widely used Deep Learning activation functions.*
Function plot images courtesy of Laughsinthestocks, https://en.wikipedia.org/wiki/Activation_function. Licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/legalcode)

At any point in time, the output value **ŷ** is likely not equal to **y**. Consequently, there thus exists a *cost* of using the ANN [47]. The average cost for all training data points is also called a loss function in practice. The cost is especially large during the first training round, when the weight of every neuron is initialized e.g. randomly. Training the ANN, of which it now follows that it is a cyclical process, has the goal of minimizing the difference between **ŷ** and **y**, i.e. to minimize the cost function (or: loss function) of the model.

Although it looks simple at face value, choosing an appropriate cost function is often rather difficult [46]. It is however a key ingredient for achieving high performance: since the optimizer will attempt to find a shortcut to achieve lower cost in any way, the cost function used must be related to one's definition of success [46].

When we have a cost function, we can optimize our model. But how to optimize it efficiently? Mathematically, if a function is differentiable, it can be optimized by deriving its gradient (i.e., its speed of change) and equaling it to zero, finding the minimum or maximum value of the function. It then follows that optimizing an ANN is dependent on two new ingredients:

1. **An optimizer:** a means of numerically optimizing the gradient (i.e. a training algorithm);
2. **A gradient computer:** A means of computing the cost gradient in a computationally efficient way.

In the early days of neural network research into finding the minimum cost value, progress was "quite limited" as a result of the "lack of a training algorithm" [45]. Additionally, computing the gradient of an ANN was inefficient due to the methods used and the scarcity of computational resources [45]. However, in the past few decades, several developments have yielded significant progress in training ANNs. Of those, the introduction of the backpropagation algorithm (for efficiently chaining the gradient allowing one to compute the error backwards through the layers) and the introduction of optimization algorithms like stochastic gradient descent have been the most major ones [47] [45]. These have allowed ANNs to be utilized in various commercial applications. Today, the literature on neural networks is vast and growing [45]. Particularly, various sub classes of ANNs have been developed, which have strengths in particular problem domains. Next, we will review historical and recent classes, and discuss salient instantiations and their characteristics.

2.2.4.2: Single-layer Perceptron

In 1958, Rosenblatt invented a mathematical structure to mimic information storage and organization in human brains, named *perceptron* [49]. It can be defined as *one* neuron having the function

$$f(x) = \begin{cases} 1, & \mathbf{w} \cdot \mathbf{x} + \mathrm{b} > 0 \\ 0, & otherwise \end{cases}$$

Where $\mathbf{w} \cdot \mathbf{x}$ is the dot product $\sum_{i=1}^{m} w_i x_i$ of the weights vector **w** and the input vector **x**, and b is the bias value adopted by the network. The function itself is also known as the Heaviside step function, of which the output simply steps from 0 to 1 when the input exceeds a certain threshold value.

Later, in the era of Deep Neural Networks that we will cover later, it has become known as the *Single-layer perceptron* (SLP). Visually, it can be represented as follows. Its goal is binary classification, i.e. distinguishing between two classes of data. The SLP can only be used if classes are linearly separable [45].

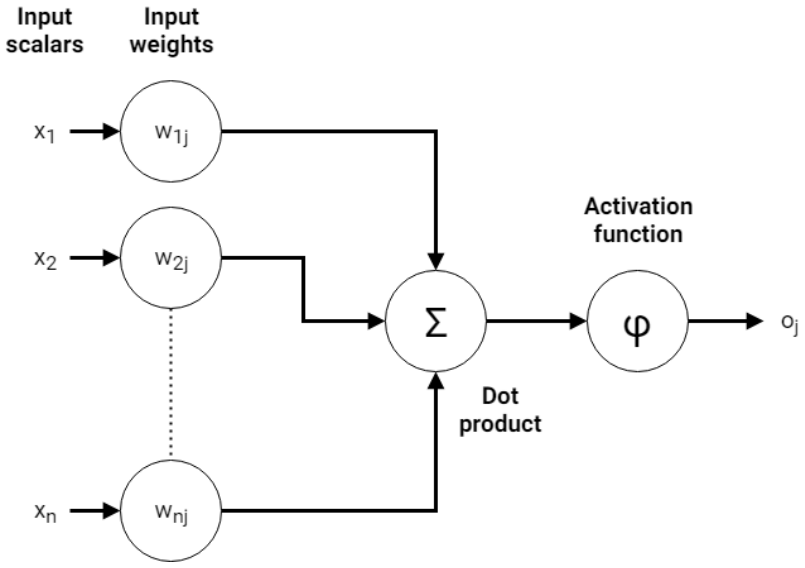


Image 2: *Rosenblatt's or Single-layer Perceptron.*

From this conceptual representation, it follows that input scalars $\mathbf{x_1}$ to $\mathbf{x_n}$ (that compose vector $\mathbf{x}$) are multiplied with their corresponding weights into the dot product highlighted below. The dot product is subsequently passed on to an activation function. For Rosenblatt's perceptron, this is traditionally the Heaviside step function, as we discussed before, producing output value $\mathbf{o_j}$. However, with modern technology, it is likewise possible to utilize advances in activation functions and to use the single-layer perceptron with e.g. one of the ones discussed previously.

Following from our analysis of ANN training, training a SLP thus yields the minimization of its cost function. At the time of Rosenblatt's, this would be done by using traditional methods and techniques resembling simple computation. Backpropagation and gradient descent can also be used, but come at a significant computational cost with respect to the traditional methods, and it is the question whether this is practically relevant.

2.2.4.3: Multi-layer Perceptrons

In order to allow for the propagation of more complex data through a neural network, multi-layer perceptrons (MLPs) were proposed in the 1980s [45]. Contrary to a SLP, MLPs are composed of multiple layers of neurons, three at minimum. The first layer is the so-called *input layer*, which receives input data [45]. Neural computations are eventually synthesized into one output value in the *output layer*. In between are one or more densely connected *hidden layers* that assist in capturing patterns in the data [45].

Finding the optimal number of such hidden layers and the number of neurons per hidden layer is problem-dependent and often a process of trial and error [45]. Training MLPs is performed using backpropagation and optimization algorithms like gradient descent. They have been the first class of ANNs that have successfully yielded commercial applications, such as forecasting in healthcare, education and marketing [45].

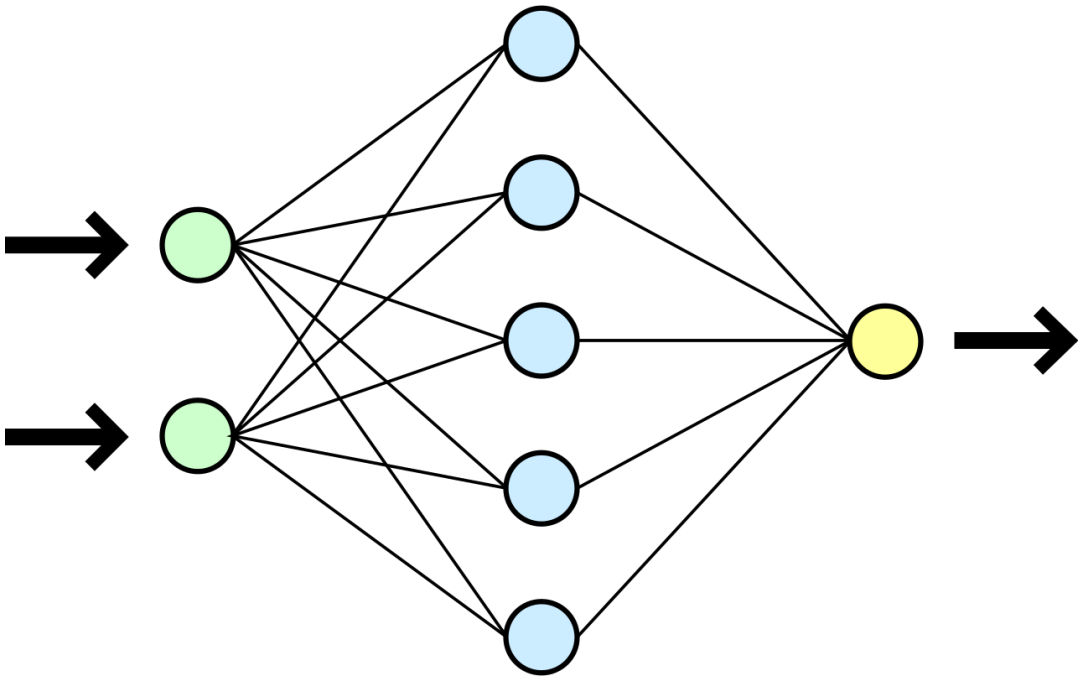


Image 3: *Multi-Layer Perceptron with forward data propagation.*
Image courtesy of Dake, Mysid, https://commons.wikimedia.org/wiki/Neural_network#/media/File:Neural_network.svg
Licensed under CC BY 1.0 (https://creativecommons.org/licenses/by/1.0/legalcode)

2.2.4.4: From MLPs to Deep Neural Networks

More recently, work on MLPs has expanded into a new class of algorithms named Deep Neural Networks (DNNs). DNNs "have had a major effect in recent years in computer vision and speech recognition", with "major improvements in performance over previous approaches" [12].

DNNs can be separated from the more traditional model-based approaches by applying several heuristics [47]. Firstly, DNNs are *networks of functions*. Every neuron represents a function that takes some input and produces some output; the functions are chained in a network [47]. This is similar to the neural networks discussed before. It does not distinguish DNNs from the MLPs proposed in the 1980s, since they are composed of layers of neurons as well. However, the *depth of the network* does distinguish them. Whereas MLPs often comprise three layers, DNNs often compose a significantly larger amount [47].

Contrary to the more traditional approaches such as SVMs and simpler ANNs, the high quantity of DNN layers allows them to interpret data in a *highly abstract way* [12]. Especially in layers that reside far downstream the layers, data is often very abstract. For example, images input can often no longer be classified by human beings after passing through approximately five layers [12].

Additionally, both DNNs and – by definition – MLPs use non-linear activation functions. This separates them from the more traditional model-based approaches [12]. Next, we will analyze several types of DNNs that are today considered to be standard or state-of-the-art.

2.2.4.5: Feedforward Networks

MLPs are part of a class of ANNs that are known by the name *feedforward networks*. They are called feedforward because information flowing through the function $\hat{y}: f(x)$ is evaluated in a forward fashion. That is, it starts as input **x**, then flows through the intermediate components that together define **f** and then emerges as output **ŷ** [47]. No feedback loops, feeding back intermediate outputs into the model, are present.

Typical MLPs and MLP-like feedforward networks have layers that are densely connected, also known as fully connected [45]. That is, nodes in adjacent layers are connected by acyclic graphs from a lower layer to a higher layer. Every node in the previous layer connects with all nodes in the subsequent one. Simple vectors can usually be processed by densely connected layers [46]. For example, Chollet makes use of these layers – and thus of a MLP-like structure – for binary classification of IMDB movie reviews [46].

However, when data becomes more complex, the peculiarities of the data set will often render processing by densely connected layers impractical or even infeasible. For example, sequence data (e.g. timeseries and texts) is typically processed by recurrent layers [46]. Those are layers with a feedback loop. For processing image and video data, convolutional layers are used. Their characteristics differ from those of densely connected feedforward networks. Next, we will therefore analyze those individually.

2.2.4.6: Convolutional Neural Networks

Densely connected layers of neurons, such as the ones present in MLPs, learn global patterns in their entire input space [46]. That is, they use the input as a whole to identify patterns between the input and the output data. This is an adequate approach if the input data can be considered relevant *as a whole*. However, this is not necessarily the case for all input data sets. For example, pictures can be represented as a spatial hierarchy of real-world objects and their components [46]. As another example, a picture of cats contains various individual cat objects. Those objects are itself composed of “modules” that are composed of e.g. noses, ears, et cetera. If one’s ML objective would be to identify objects (e.g. a cat) in pictures, it is no longer the *whole* picture that is of interest, but rather the presence of the *components* that together compose the cat.

For those tasks, MLPs and other densely connected networks have shown to operate substantially slower and with lower performance than so-called *convolutional neural networks* (CNNs) [46]. CNNs are a class of feedforward networks specialized in processing data that has a grid-like structure and employ two new components in order to process data more effectively [46]:

- The mathematical operation of **convolution**, and
- The subsequent down sampling operation of **pooling**.

The convolution operation entails a small filter sliding over the input data and converting it into a lower-size, higher-dimensional output called a *feature map*. A filter is often a block of 3x3 or 5x5 pixels converting the contents in one scalar value represented in the feature map. The pooling operation subsequently maps the feature map into an even lower-dimensional variant by once again sliding over the feature map with a small

filter-like structure, per slide producing one value (in the case of e.g. max pooling, the maximum value for every slice). This substantial dimensionality reduction lowers the processing power required for training CNNs.

The convolution operation has various benefits over the traditional operations present in densely connected layers [47]. First, traditional ANNs use different weight vectors for every neuron (cf. Rosenblatt's perceptron), which are used in computing the output value of a neuron. In CNNs, this is not the case. Rather, the weights are represented by a multidimensional *kernel,* or filter. This kernel is the small matrix (in computer vision problems e.g. 3x3 pixels [47]) discussed before. It is shared over the entire input space. Goodfellow et al. call this beneficial aspect *parameter sharing*. In CNNs, the number of trainable weights is thus drastically reduced.

Second, following from the observation that a kernel is a small matrix, we observe that CNNs perform *sparse interactions* [47]. That is, contrary to the densely connected layers in which one neuron is connected to all neurons in its subsequent layer, in CNNs this is not the case. When a kernel convolves over the input data (e.g. a picture), it passes its output to only a small set of neurons. This number is equal to the width of the kernel (e.g. 3 with a 3x3 pixel kernel). In CNNs, data is thus no longer propagated to the entire subsequent layer. In fact, the only learnt parameters are the kernel filters, reducing dimensionality upstream.

Third, the way parameters are shared results in *equivariance* to translation. That is, e.g. in the case of a picture, when objects in the picture are moved to other parts of the network, they can still be recognized by the network. In CNNs, this is also known as *translation invariance* [46].

As a consequence of these benefits, which are not shared by other classes of ML algorithms, CNNs have grown significantly in popularity. They have replaced traditional methods, such as SVMs, in computer vision competitions [46]. CNNs are widely applied: applications emerge in forecasting, computer vision, time series analysis, et cetera [46] [12]. A wide array of CNN architectures has been developed by researchers and practitioners. Practitioners consider some of them more relevant than others [50] [51].

We outline salient CNN architectures in Table 7. These architectures are either considered to be of historical significance (e.g. LeNet-5 [52] and AlexNet [53]) or to be state-of-the-art (e.g. YOLO versions 1 to 3 [54]). For every architecture, we include their main machine learning objective from the objectives identified previously.

Name	ML Objective	Year
LeNet-5 [52]	Image classification	1998
AlexNet [53]	Image classification	2012
VGG-16 [55]	Image classification	2014
ResNeXt [56]	Image classification	2016
R-CNN [57]	Object detection	2014
Faster R-CNN [58]	Object detection	2016
YOLO [54]	Object detection	2015
SSD [59]	Object detection	2015

Table 7: *Standard and State-of-the-Art Convolutional Neural Network Architectures*

2.2.4.7: Recurrent Neural Networks

In cases in which data must be represented as a sequence of vectors, i.e. $x^{(1)}$ ... $x^{(t)}$ where t is the number of time steps, another class of ANNs is preferred by practitioners [46], namely the class of Recurrent Neural Networks (RNNs). The reason as to why these are preferred is that inferring insights from such sequences at points in time, e.g. at ½t, requires the consideration of time steps processed prior to the time step under consideration. In short, sometimes it is necessary to include some kind of memory in an ANN. Feedforward networks like MLPs and CNNs do not have such memories, but RNNs do [46].

In fact, when propagating data through the network in a feedforward fashion, RNNs maintain a state that "[contains] information relative to what it has seen so far" [46]. RNNs can thus be considered to have internal loops. Especially data sets in which time is a critical element, e.g. time series used in forecasting or text analysis, can benefit from utilizing RNNs over other classes of ANNs [46].

Within RNNs, the Long-Short Term Memory (LSTM) layer is most prominently used today when creating RNNs [46]. Additionally, the so-called GRU layer is used in practice, alongside the SimpleRNN layer. Practitioners must use these layers in a fashion of trial-and-error [46].

2.2.5: Ensemble Learning

Generally speaking, training a RNN comes with higher computational cost than training architectures composed of other layers, such as MLPs (or densely-connected layered architectures) and CNNs [46]. This results from the addition of a memory element to RNN layers. For practitioners, it could therefore be more computationally pragmatic to train RNNs and augment them with the performance provided by e.g. CNNs, utilizing so-called **ensemble learning** techniques.

In ensemble learning, the predictions of different models are pooled together to produce better predictions [46]. Predictions can be better since ensemble models exploit the varying strengths and weaknesses possessed by every type of model. That is, each type captures a part of the truth, but not all of it. By combining the individual predictions into often large ensembles of models, performance often increases.

A critical success factor in ensemble learning is however that ensembled models should both be *as good as possible* and *as different as possible* [46]. When similar models are combined into ensembles, they do not cancel out each other's weaknesses. Practical success has been reported following from combining *deep and shallow* models, thus deep neural networks with shallow ones such as SVMs [46]. Contrary to ever-expanding deep models, increasing computational cost, ensemble learning can thus be a more efficient approach to obtaining similar performance.

2.2.5.1: Ensemble Learning Techniques

Multiple methods can be used to create ensemble models [44]. In today's literature, simple committees (also known as bagging) and weighted sequential committees (boosting) have gained some prominence with regards to creating model ensembles. More recently, those were augmented with stacked generalization (stacking). We briefly discuss them below.

Simple committees are also known by the name bootstrap aggregating, or *bagging* [44] [46]. It involves taking *m* samples from the training set uniformly, with replacement; a statistical technique called bootstrapping. With those samples, *m* predictive models are trained. For new data, the *m* outputs are averaged. This yields better results, since opposite errors introduced by the individual models are cancelled, converging towards the ground truth. In fact, when the errors of individual models are fully uncorrelated, it can be shown mathematically that the ensemble model's error is *m* times lower than those of the individual models [44]. In practice, however, errors are correlated to some extent. Nevertheless, the ensemble error rate is still lower.

Another ensemble technique is called *boosting*. It works by assigning weights to every data point in the entire training set, subsequently training a predictive model based on the data [44]. After calculating the error rate, the weights attached to the training samples are adapted, known as re-weighting. Another predictive model is then trained. This ensures that the new model is adapted based on errors identified previously. The process reiterates until *m* models are trained. The importance of the individual models is subsequently determined by assigning new weights to the models. For new data, a weighted average is then computed based on the individual predictions. Multiple boosting algorithms are common in today's machine learning practice, among them AdaBoost and Gradient Boosting, which is considered to be a generalization of the first [46] [44].

In *stacking*, different prediction models are trained using the available training samples [60]. Those are known as level-0 learners. Their outputs generate a new training set which is used to train another learner, known as a metalearner or level-1 learner. It learns from the data how to best combine the level-0 learners in creating a final prediction. It has been successfully applied besides bagging and boosting techniques.

Next, we will analyze which underground intelligence techniques are available for underground mapping. We specifically focus on the techniques that are used by TerraCarta B.V. and discuss the data that is produced by them.

2.3: Underground Mapping Techniques and their Data

In this section, we briefly analyze the underground mapping techniques used by TerraCarta B.V. We highlight their geophysical properties at a high level and, more importantly, identify what particular data they produce. We first start with Ground Penetrating Radar, which is primarily used for utility inspection. Radar imagery produced by this technique can be augmented with data produced by Buried-utility Locators, which we will highlight subsequently.

2.3.1: Ground Penetrating Radar

Ground Penetrating Radar (GPR) is a geophysical technique used for acquiring insights about the underground [11]. A GPR device is equipped with antennas that have a dual role, namely transmitting and receiving electromagnetic waves. At runtime, a GPR device is moved over the Earth's surface. By converting electric currents onto antenna elements into electromagnetic waves, it can transmit them into the ground.

Those waves may subsequently propagate into subsurface material, such as the medium, e.g. sand [61]. Objects buried in the medium have different electromagnetic properties than the medium itself. By consequence, a part of the waves is reflected back to the surface once they hit such buried objects. They are then called echoes. Those echoes make GPR an adequate device for non-destructive subsurface monitoring.

Since the GPR device moves over the ground, there exists a small but significant distance between the device and the subsurface, in which the propagation medium is the air. The waves emitted by the transmitter will therefore continuously experience a medium change during operation, namely the one from air to subsurface medium (e.g. sand). This change of medium, by definition, also produces echoes. This phenomenon is known by practitioners as a *ground bounce*.

During operation and concurrent to wave emission, another antenna converts the received echoes into electric currents, which can be stored onto device storage [61]. For any position (x, y), the window of opportunity in which echoes can be received is often a few nanoseconds (n x 10^{-9} seconds), before the GPR device arrives at a new position [62]. This closes the loop from transmission to reception.

2.3.1.1: Visualizing GPR Data: A-scans and B-scans

Signals received within the window of opportunity at any position (x, y) can be visualized in an A-scan [62]. In such a scan, the amplitudes of the echoes received by the antenna are plotted against time, often in nanoseconds. On A-scans, it is possible to determine the depth of underground objects by determining the wave propagation velocity v of the medium, i.e. the subsurface. By subsequently multiplying this velocity with time, one finds the distance from the antenna, thus depth [62].

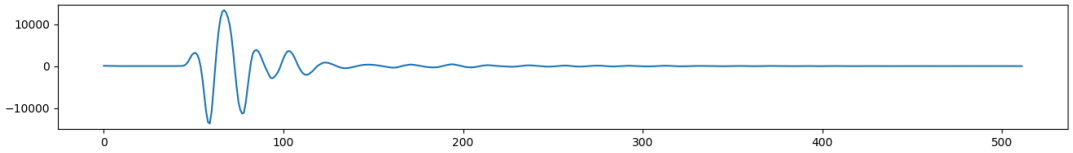


Image 4: *One A-scan without gain, with depth increasing towards the right.*

However, benefiting from the fact that a GPR device is moved over the Earth's surface horizontally, visualizing a set of A-scans instead of only one would provide much richer insights into the subsurface. Through a set of A-scans, it is possible to create rich 2D images, called B-scans [62]. In those, the amplitudes present within the individual A-scans are represented by some color scale or greyscale, as presented in image 5. On B-scans, underground objects are represented as hyperbolae. This signature occurs because radar waves are emitted spherically, capturing the object even when one does not directly stand on top of it. It is however unsurprising that its echo is visible at a later time, since the distance between object and antenna is larger. This results in a symmetrical pattern with the peak echo occurring when the device stands on top of the object, recognizable as the hyperbolae of which some are visible in image 5.

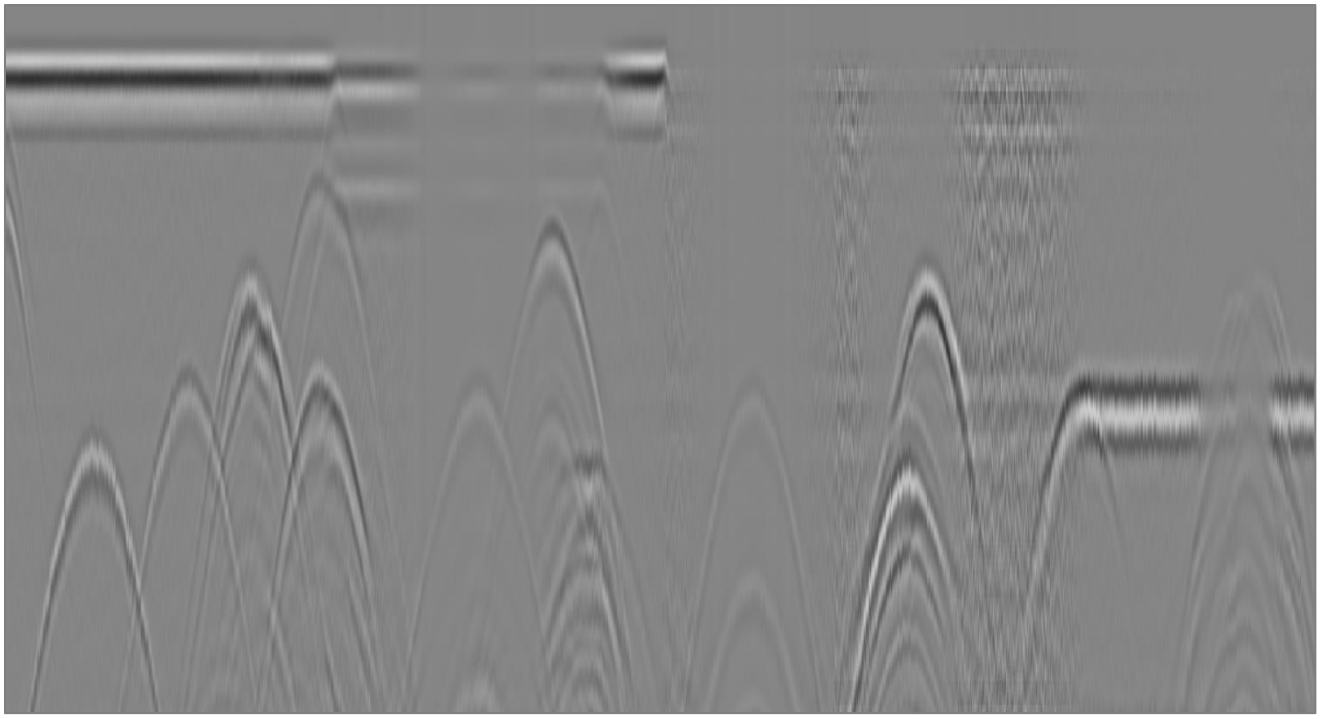


Image 5: *B-scan drawn from a subset of the A-scans available for an entire scan.*

Through significant training and a theoretical or practical understanding of (applied) geophysics, practitioners can derive certain insights from A-scans and B-scans. This is especially true when they can combine GPR data with contextual knowledge, such as information retrieved from the KLIC information repository that was discussed in section 1.1. Consequently, trained experts are capable of combining various data sources and geophysical insights into a thorough analysis of the subsurface, recognizing underground objects such as cables and pipelines.

2.3.1.2: Data Formats

Data retrieved with a GPR device must be exportable to PCs for analysis. A practitioner must also be able to share GPR data with fellow practitioners or clients. Consequently, data formats have been developed that can be used for exporting GPR data. These range from simple formats to more sophisticated ones and from open standards to proprietary ones [63] [64]. We present today's common data formats in Table 8. Based on anecdotal evidence from TerraCarta B.V., we note that choosing one is highly dependent on the analysis software being utilized by such firms. This, in return, is dependent on the manufacturer of the GPRs used.

Standard owner	Standard name	Proprietary
-	Comma-separated values (CSV)	No
Geophysical Survey Systems, Inc.	DZT (or GSSI)	Yes
Sensors & Software, Inc.	DT1/HD	Yes
Society of Exploration Geophysicists	SEG-Y	No
Malå GeoScience	RD3/RAD	Yes
-	Seismic Unix	No
3D-Radar	3DR	Yes

Table 8: *Common data formats for Ground Penetrating Radar imagery.*

2.3.1.3: Challenges Faced When Using GPR

Although GPR is a promising technique for subsurface analysis, it comes with certain challenges. Below, we list key issues faced by GPR practitioners [11]:

- The close proximity of receiver to transmitter slightly bows up or down the signal visible in A-scans, called *wow*. A so-called dewow filter can be used to suppress this effect. Today's digital systems often apply such a filter in real-time.
- The electromagnetic waves that travel through the subsurface are attenuated with distance. This yields a difference in amplitudes between waves from objects close to the surface and those that are present at greater depth. A *time-varying gain* filter is often applied to suppress this effect, normalizing the amplitudes visible in the A-scans.
- Sometimes, a *deconvolution* filter is used to boost GPR bandwidth. It is however not often applied in practice. It is sometimes augmented with a *migration* filter that removes directionality resulting from moving the GPR device through space.
- When one moves the GPR device over surfaces that have different topography (e.g. when a topographic boundary between sand and clay is crossed during measurement), it is imperative to apply *topographic correction* to the GPR data. This need results from the wave propagation velocities that vary based on the geophysical properties of the materials. Omitting such correction would yield anomalies between the A-scans from different topographic origin in one B-scan, possibly spawning human error during analysis.

Despite problem mitigation techniques and augmentation through contextual information, the origin of certain hyperbolae can in some cases remain unclear. Practitioners can then augment their analysis by combining GPR data with other measurements. This provides the analyst with more certainty. We next discuss a class of techniques often used by TerraCarta, named buried-utility locators.

2.3.2: Buried-utility Locators

The class of *buried-utility locators* can be used to identify electromagnetic disturbances caused by subsurface objects, particularly utility objects such as cables and pipelines [65]. Today, three instances of such locators are used at scale, namely (1) magnetic-induction locators, (2) power-cable locators and (3) radio-frequency tracking locators [65]. Commercial parties often name this class of techniques "radio detection".

Magnetic-induction locators are designed to detect subsurface metallic utilities, such as pipelines and cables [65]. They consist of separate transmitter and receiver units. The transmitter generates a magnetic field which propagates into the ground, encountering subsurface objects in the meantime. When those objects are metallic, an electrical current is caused to flow within the subsurface object. This generates another magnetic field, concentrically moving away from the object. This field is picked up by the receiver. Due to the concentric movement, during measurement one finds the maximum peak when one is closest to the object. That is, when one stands directly above it. Consequently, magnetic-induction locators can be used to detect subsurface objects if they are metallic.

Power-cable locators share the same detection principles with magnetic-induction locators, but do not use a transmitter [65]. Rather, their receiver is tuned to detect the magnetic field produced by the standard 50-60 Hz frequency range of the electric currents on the power line, as for them no transmitter is required. However, since no transmitter is included, they cannot be used for different purposes.

However, not every subsurface detection problem involves metallic objects. Sometimes, non-metallic cables and pipelines are equipped with metallic tape which enables identification using magnetic-induction locators. In other cases, pipelines are equipped with a metal wire within them, which renders the same opportunities. This is however not true for every cable and pipeline. In those cases, radio-frequency tracking locators can be used under the condition that the objects are accessible and, by definition, unpressurised [65].

Radio-frequency tracking locators are rather similar to magnetic-induction locators: they also work with a transmitter and a receiver. However, contrary to magnetic-induction locators, with radio-frequency tracking locators transmitters are positioned within the pipeline [65]. They subsequently emit a directional signal, which is detected by the receiver.

The similarities between the techniques outlined above result in fuzzy definitions in practice. Commercial locators are available, but are known under the umbrella terms *radio detection* and *electromagnetic locators*. Contrary to GPR, which produces A-scans that can be exported using e.g. open data forms, commercial

locators use proprietary data formats. Facilitating data export is therefore highly vendor-dependent and inclusion of them in this work is not worthwhile.

2.3.3: KLIC information repository

From chapter 1, we note that the Dutch land registry – Kadaster – maintains a registry of cables and pipelines, named KLIC [8]. It is an information repository in which the positions of subsurface cables and pipelines is registered. By law, accurate registration is now required [10]. It can be used by construction companies that aim to perform excavation activities; in fact, they are required to do so.

However, practitioners within subsurface analysis service firms can also use the KLIC information repository to augment their judgements based on e.g. GPR imagery. For example, a hyperbola in GPR may emerge as a result of various subsurface objects. In order to avoid deception into thinking that e.g. a metallic pipeline is present, a practitioner could use the KLIC information to augment their judgement. Obviously, when the KLIC information presents the presence of a nearby metallic pipeline, the certainty with which the practitioner can classify the GPR hyperbola as a metallic pipeline increases.

KLIC information is not available on an ad-hoc basis. Rather, it must be requested on a perimeter basis from the Dutch land registry. Given this overhead, it is not worthwhile to include the KLIC in our work, since the goal of the IA scenario is to allow for a quick analysis and does not benefit from added time delay.

2.4: Machine Learning Approaches for Mapping the Underground

As we discussed previously, human beings can infer the presence, location and characteristics of underground utilities. However, this requires substantial training efforts and insight into the Earth's geophysical properties. Given the advances in machine learning techniques, it is therefore no surprise that they have been used for this application. We will next systematically review the developments in this domain.

2.4.1: General State of Machine Learning for Underground Mapping

Over the last decade, many studies have been performed to identify which machine learning approach works best when analyzing the underground. According to Pasolli et al., these studies can be grouped into four groups by their objectives [66]. Primarily, studies have been performed on (1) object detection and localization, assuming invariance to object characteristics. More recent work has also been performed into (2) object material recognition, (3) object dimension estimation and (4) object shape estimation.

It is interesting to note that most work has been performed on combining machine learning techniques with GPR imagery and/or GPR waveforms [66]. Few works are available on electromagnetic field locators (or buried-utility locators) which we discussed in section 2.3. No recent works are available which combine the KLIC information repository with other underground intelligence techniques. Given this, and since this work focuses on object material recognition, we will next focus on machine learning being applied to GPR imagery.

2.4.2: Machine Learning for Object Detection and Localization

A literature review performed by Travassos et al. clearly shows how the works on machine learning and GPR have evolved over the years in terms of both statistical and machine learning models used [13]. We highlight the most important ones here and refer to their work for a more detailed discussion. Their review starts in the 2000s, in which the GPR became popular as a device for non-destructive evaluation of the underground. As a consequence, the number of works which attempted to automate the analysis of GPR imagery grew rapidly.

In this era, basic neural networks were introduced into research [13]. The preference for those models introduces substantial data pre-processing and feature extraction activities. This results from the observation that geometrically, higher-dimensional feature spaces require an exponentially increasing amount of training data for predictions to be significant [44]. This effect is known as the *curse of dimensionality*. Raw GPR imagery is everything but low-dimensional (for example, a 200x200 pixels RGB B-scan image would yield 200x200x3 = 120.000 features). It therefore cannot be fit to machine learning methods that do not exploit dimensionality reduction techniques. By consequence, studies from this era propose a large and inhomogeneous variety of pre-processing and feature extraction approaches.

For example, Singh and Nene apply a two-layer feedforward network with 1200 input neurons and one output neuron to GPR data [67]. Data pre-processing included applying filters against clutter, noise and the air-ground reflection. About 75% of the hyperbolae identified by a human operator is detected using their approach. Those were promising results at the time for automated object detection, still leaving open room for improvement. Similarly, Núñez-Nieto et al. use neural networks for object detection and localization, but perform their work in the context of automated landmine and unexploded ordnance detection [68].

In a different study, Dou et al. propose an end-to-end solution for automated object detection using a clustering algorithm and a neural network [69]. They first employ filters to reduce noise and the air-ground reflection. Subsequently, a thresholding technique is used to extract regions of interest. Using a novel clustering algorithm, they subsequently identify the most prominent ones for further analysis. Those are fed into a densely-connected (i.e. basic) neural network, which classifies them as a hyperbola or non-hyperbola. The areas of interest classified as hyperbolae are finally visualized by initializing a hyperbola function using an orthogonal-distance fitting algorithm [69].

Given the adequate performance of classic ANNs, research interest dropped until the late 2000s, when scholars started applying different algorithms to the object detection and localization problem. For example, Pasolli et al. propose a genetic algorithm for this purpose. They first reduce random signal noise and eliminate the air-ground reflection [66]. They then apply a time-varying gain filter to reduce the impact of signal attenuation and, consequently, allow for depth-invariant object detection. Using a thresholding technique, they finally segment between *object* and *background* classes, before they fit the data to the genetic algorithm, which identifies hidden patterns, allowing them to classify new data.

However, the breakthrough in artificial neural networks (ANNs) for computer vision in 2012 has caused a shift in prominence back to applying modern ANNs within the entire machine learning community [12] [53].

Specifically, Chollet notes that most SVM-driven developments have been replaced by ANNs, in the area of computer vision especially convolutional ones (CNNs) [46]. It is therefore no surprise that this shift can also be observed in the literature about applying machine learning to GPR imagery for object detection and localization.

In the context of the detection of buried explosive hazards, Besaw and Stimac propose a CNN-based approach for classifying B-scans [70]. Initially, using various waveform phase alignment techniques, the B-scans are pre-processed. They subsequently use an anomaly detector which is capable of detecting areas of interest. Windows centered around these areas of interested are used as features for the CNN, which classifies areas as being an explosive hazard or noise. We observe that windows from the pre-processed B-scans are used as input to the CNN, and note that no dimensionality reduction techniques are used, contrary to the approaches discussed earlier. Experimental results demonstrate significant performance improvements over traditional approaches [70]. Another type of CNN, a convolutional autoencoder, is used as an anomaly detector for landmine detection by Picetti et al., proposing another promising approach for object detection and localization [71].

Recently, Pham and Lefèvre note that "[most works so far] have focused on classification or patch-based detection using [a] sliding window over the whole image" [16]. We have discussed such an approach above from Besaw and Stimac's work. Pham and Lefèvre, however, propose an end-to-end pipeline using the Faster-RCNN architecture. This architecture serves the purpose of object detection and is therefore a good fit for this purpose. They first pretrain the CNN embedded in the architecture on the generic Cifar-10 dataset and then fine-tune the network by continuing the training process with real and simulated GPR data. Validating the architecture on simulated and real data, they qualitatively demonstrate that the architecture performs considerably well.

An important observation arising from Pham and Lefèvre's work is that they pretrained the CNN used in their architecture using the Cifar-10 dataset. Pretraining is a form of transfer learning, which we discussed in section 2.2.1, and can substantially improve the performance of a CNN. It is especially interesting for object detection using GPR imagery, given the costs associated with obtaining an expert-labeled training set [72]. Reichman et al. validate this generalized claim for the particular application to GPR-based object detection for buried threats [72]. In their work, they also pretrain a CNN on the Cifar-10 dataset to empirically investigate its applicability to GPR imagery. Additionally, they empirically validate another form of transfer learning for CNNs known as data augmentation, in which features are visually transformed to expand the size of the training set. Their work shows that both pretraining and data augmentation improves the accuracies of CNNs used for object detection within GPR imagery.

Building upon those results, Bralich et al. expand the pretraining to include a dataset containing patches extracted from aerial images of the city of Fresno, which is publicly available [73]. They investigate whether locking the so-called convolutional base [46], i.e. not allowing the pretrained neural weights in some layers to change when it is trained on GPR imagery, affects the performance of the model. They qualitatively show that pre-training can lead to improved detection performance, especially when using the aerial data without locking the convolutional base. They do however note that this observation may not be universally true.

Together, we can thus consider the state-of-the-art within machine learning for object recognition to be a mix of various machine learning approaches. Generally, recent works favor an approach using CNNs over more traditional approaches with SVMs and densely-connected ANNs. At the same time, traditional approaches are still improved incrementally and are not replaced entirely.

2.4.3: Machine Learning for Object Material Recognition and Other Characteristics

Given those developments, however, the scientific community considers object detection and localization for GPR to be a trivial task [66]. As a result, research efforts are transitioning towards object material recognition, for which the number of works is substantially scarcer. Nevertheless, with regards to the ML algorithms applied, a similar trajectory as for underground object detection becomes apparent from the literature.

Given the popularity of SVMs, they were the primary choice for studies during the early 2010s. Once again, however, scholars experienced the curse of dimensionality, with a large variety of pre-processing and feature extraction approaches applied in their works. For example, El-Mahallawy and Hashim first reduce A-scan signal noise by applying median and Wiener filters [74]. They then extract feature vectors by using discrete cosine transform (DCT), which is a signal compression technique, and fit them to an SVM. To allow for comparison, they repeat this process with different feature extraction techniques, comparing DCT-based

features with timeseries-based and statistical-based feature vectors. They show that DCT-based features are superior to the others, even when the degree of noise in the GPR imagery increases.

In a different study, Shao et al. first sparsely represent an A-scan, which involves “[expressing] a signal as a linear combination of elementary waves”, called atoms [75]. From this representation, they subsequently extract features which they fit to an SVM. In the comparison of their approach with DCT and a dictionary learning method, they show that sparse representation improves model performance.

Pasolli et al., who first use genetic algorithms for the localization of objects, subsequently attempt to identify material characteristics for the detected hyperbolae [66]. They do however propose a completely different approach. Instead of using the signal waveform provided by the individual or averaged A-scans, B-scans are used for feature extraction. For every hyperbola apex detected by the genetic algorithm, the authors crop the B-scan image into a small-size window centered around the apex. They subsequently average and normalize the pixel values in its columns, which represent signal intensities, into a one-column window. This feature vector is, given the small size of the cropped window, low-dimensional yet highly information-rich, avoiding the curse of dimensionality. Testing their approach by training a SVM on three classes (limestone, metal and air) yields average accuracies of about 80%. The authors do however manually select the kernel function that is used in their approach, which introduces bias. In another study, Pasolli et al. demonstrate that another approach, namely a gaussian one, produces promising results for the estimation of buried object size [76].

The shift from SVMs to modern ANNs after the 2012 CNN breakthrough is also apparent in the literature discussing object material recognition. For example, Zhang et al. propose and validate an architecture of three neural networks which allows them to recognize the shape, material and size of an underground object [77]. Additionally, their network is capable of computing the depth of the object as well as the dielectric constant of the medium. Validation yields success rates of 90 to 100%, but its functional scope is limited. For example, the architecture can only distinguish between air, limestone and metal when classifying the material of the object. Additionally, a signal processing technique called Hilbert transform is used to normalize the A-scans by extracting the signal envelope, which is used as a feature vector. This technique increases computational requirements for real-time deployment. Nevertheless, it is an interesting step forward.

Another ANN-based approach is outlined by Almaimani [14]. By designing and validating four CNN architectures, she demonstrates that they can be used to classify and regress the material, depth and size of underground objects, as well as the dielectric constant of the medium. In the validated architectures, no feature engineering is performed; rather, a full B-scan slice is used as a feature vector. This is in line with the strengths of CNNs, which are capable of learning salient features in images themselves. Most experiments yield success rates of 50 to 60%. Only one model, which is deeper than the others, shows above-average performance with 90 to 100% classification success rates and relatively low error percentages for the regression tasks. The depth of the network could however be problematic for practice given its computational requirements when deployed in real-time. Consequently, Almaimani welcomes more research that demonstrates the applicability of CNNs to object material recognition [14].

Recently, Scymczyk and Szymczyk proposed a different approach to estimating characteristics based on GPR A-scans [15]. They use regular ANNs with feature extraction efforts preceding neural network utilization. In their work, they reduce A-scan dimensionality by approximating a polynomial function by using a linear equation solution with a Vandermonde matrix [15]. Empirically, they show that they only need 21 coefficients to accurately approximate the polynomial. This technique substantially reduces the dimensionality of the feature vector with which the neural network is trained while retaining information richness. Validation yields accuracies of almost 99% in a soil classification task. Utility classification is yet to be performed and is suggested as future work.

3: Research Methodology

The problem statement and main research question outlined in chapter 1 together describe the goal of this work: to build a Proof-of-Concept web application that amplifies knowledge workers' analyses of GPR imagery using machine learning. This is a typical design problem in which the research goal is to design and develop an artefact that aims to improve a problem context [17]. To attain a certain level of scientific rigor in the design and validation of such artefacts, a design science methodology can be used. Such methodologies ensure that research outputs – the artefacts – are both theoretically sound and practically relevant. In line with our research goals, we have to choose a suitable design science methodology before proceeding with any work.

Several design science methodologies can be used in design science research. Choosing one partially relies on the practicality of the research at hand, since certain methods are more theoretically robust than others, whereas others often allow researchers to align their research with practice more easily.

For example, the design science methodology developed by Wieringa is a very thorough method [17]. It provides a strict blueprint for doing design science research that molds a design science problem into a fixed set of questions and steps. It assumes a strong link between knowledge questions – and thus literature that answers these questions – and the subsequent design stages.

Alternatively, the Design Science Research Methodology (DSRM) developed by Peffers et al. provides a less strict blueprint [78]. Rather, it provides an iterative process model that "would suggest a good way to do it". It guides a researcher from problem identification and motivation to artefact evaluation through design and development activities. The DSRM encourages researchers to ground their design decisions in relevant bodies of knowledge. However, the link assumed between theory and the resulting artefact is not as strict as the blueprint discussed in [17]. Instead, researchers utilizing the DSRM can use their "knowledge of theory (…) [for] (…) a solution".

Additionally, it differs from Wieringa's methodology in that it allows researchers to initiate their design science project from one of four starting points. Whereas Wieringa's methodology requires that researchers start in a problem-centered fashion, the DSRM does not require this. Projects may also start from artefact solution objectives, theoretically unsound artefacts and demand from practice based on theoretically sound but practically irrelevant artefacts [78].

Another popular design science framework was created by Hevner et al. [79]. It allows researchers to perform a cyclical process of development and evaluation. The activities in this process must be grounded in relevant literature yet need to be tested against the environment in which the artefact will be used. This ensures that researchers maintain a proper balance between scientific rigor and practical relevance. Contrary to the DSRM, but similar to Wieringa's methodology, it does not allow researchers to initiate the design process from multiple starting points.

According to Sein et al., the methodologies and the framework discussed before have one common shortcoming: they "fail to recognize that the [artefact] emerges from interaction with the organizational context even when its initial design is guided by the researchers' intent" [80]. They argue that the methodologies mentioned before allow for insufficient agility when research teams develop artefacts together with industry partners, which often favor pragmatism over scientific rigor. Borrowing from the organization-oriented Action Research methodology, Sein et al. propose a new method named Action Design Research (ADR). It allows researchers to iteratively interweave building (parts of) the artefact, organizational intervention and evaluation; more iteratively than with the other methodologies, according to the authors. It is especially relevant for researchers who design artefacts in collaboration with industry partners and aim to solve a business problem.

The research carried out in this work has in fact been triggered by the particular business problem discussed in the problem statement. However, it is a business problem to which solution objectives were attached prior to the start of this work. Consequently, in terms of Peffers et al., it is not problem-centered, but solution objective-centered, since it is "triggered by an industry or research need that can be addressed by developing an [artefact]" [78]. Additionally, it primarily attempts to answer a well-scoped business problem, while advancing the related academic fields by consequence. Intuitively, one would thus preferably use the DSRM for developing such a solution-oriented artefact. However, given the practicality of the work, it would be preferable to maximize the degree of flexibility, allowing us to work closely together with TerraCarta B.V. while retaining scientific rigor. While the DSRM is thus a suitable method, it may still be too linear, separating the development and evaluation stages. Consequently, we choose the ADR methodology to guide our work.

Prior to the design phase, a literature review was performed. The results of this review are discussed in chapter 2 and serve as the theoretical basis for the subsequent work. The ADR methodology was followed with respect to how such theoretical bases should emerge. That is, the Problem Formulation Stage was completed, which prescribes the following activities before starting any design work. First, we identified and conceptualized the research opportunity in chapter 1 and provided an initial research question with sub questions. In the same chapter, the problem was casted as an instance of a class of problems, albeit implicitly. Second, chapter 2 synthesizes the results of our literature review and therefore provides the theoretical basis, discussing prior technology advances. Third, long-term organizational commitment was secured using an agreement approved by both the author and TerraCarta B.V. This agreement also implicitly includes the roles and responsibilities for both partners. Consolidated, those activities justify the work performed next.

Using the ADR methodology, one would subsequently move towards the iterative stages of Building, Intervention and Evaluation (BIE). The researcher would utilize their theoretical basis to create an initial version of the artefact. For this work, that would be an initial machine learning model, which is subsequently validated, varied, and finally deployed in the IA tool proposed in the research objectives. The instance of the ADR used in this work is illustrated in Image 6.

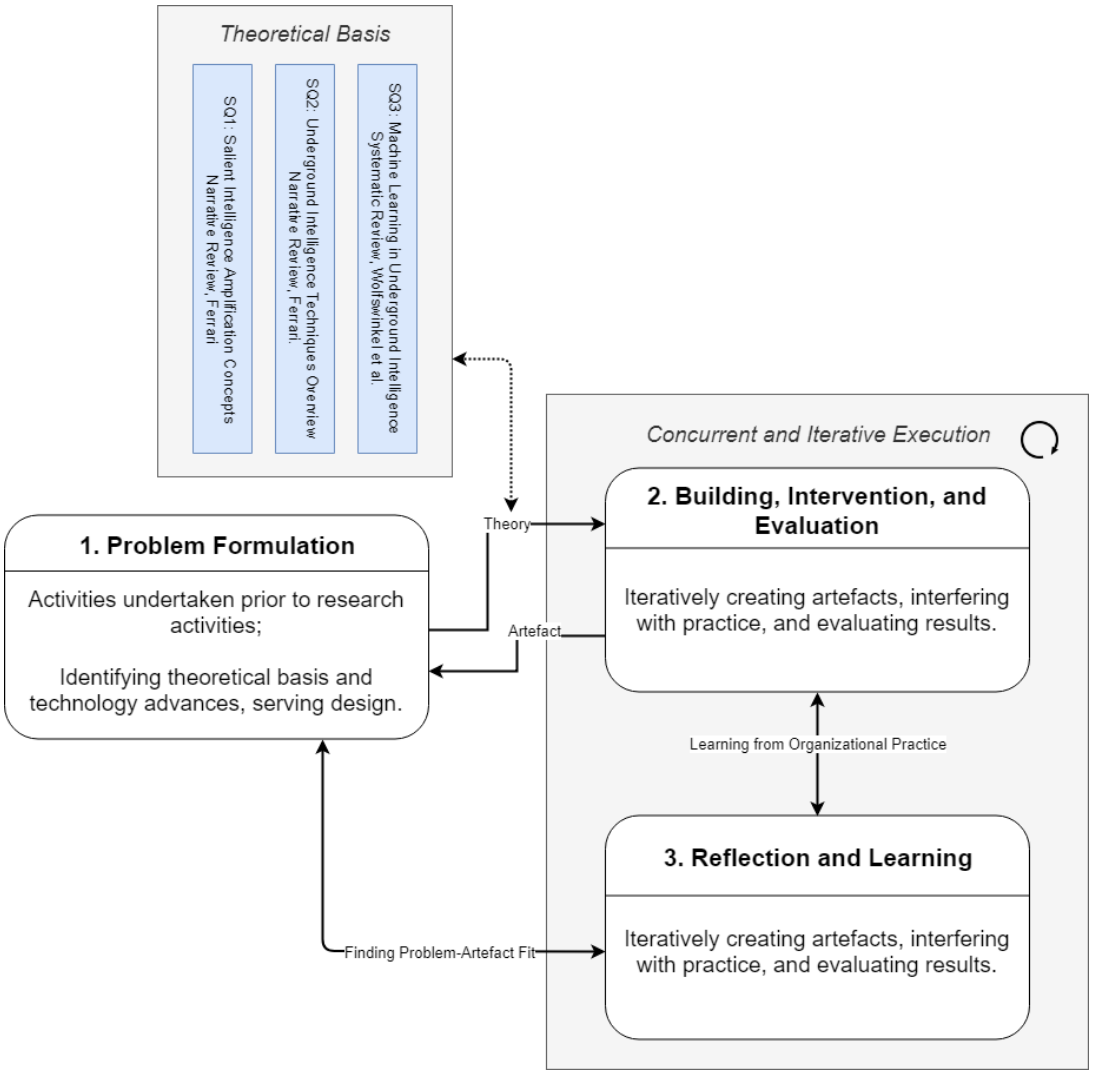


Image 6: *ADR based Design Research Methodology for this work.*

4: Novel Algorithms for Utility Material Characterization

In this chapter, we propose three novel algorithms for utility material characterization. In section 4.1, we address why we consider them suitable candidates for our study. Our discussion proceeds with the general machine learning data pipeline, which is present in all three algorithms, in section 4.2. Subsequently, sections 4.3-4.5 provide a discussion on the idiosyncratic characteristics of each algorithm. Section 4.3 specifically discusses an algorithm that utilizes A-scan histograms as features. Section 4.4 elaborates on utilizing the Discrete Cosine Transform to extract suitable features from A-scans. In section 4.5, we move on towards B-scans, and discuss a Window-based approach. Together, they discuss the main contribution of this work.

4.1: Algorithm Proposal Rationale

Our starting point for the algorithm design phase, specifically the idiosyncratic aspects of the algorithms proposed in sections 4.3-4.5, was the literature review on applying machine learning to GPR data. In this review, we found that machine learning approaches for underground object detection provide state-of-the-art results. However, with respect to the characterization of those objects – i.e., predicting the target materials – much progress is still to be made. This section discusses the rationale for the novel algorithms in more detail.

4.1.1: Representing Signals with Histograms

One of the methods with which signal variance can be captured is a histogram. This technique was inspired by Jaw and Hashim, who use a histogram technique to characterize various underground objects based on the patterns hidden in an individual A-scan [81]. However, they do not automate the process by applying machine learning, but instead leave interpretation to a human being. Given their promising results with respect to feature extraction and subsequent discrimination, it is a prime candidate for extending it with machine learning techniques, and thus for our work. Given the recent advances of DL and specifically CNNs, this work therefore proposes a histogram based CNN for material classification. We discuss its idiosyncrasies in section 4.3.

4.1.2: Signal Compression for Discrimination

In another study, El-Mahallawy and Hashim propose a multi-stage data pipeline which is also based on A-scans [74]. This data pipeline includes noise filtering followed by feature extraction using the Discrete Cosine Transform (DCT). Based on the extracted features, a SVM is trained for classification of utility material types.

Although the validation results in their work are promising, we can argue that recent technology advances suggest that limitations are present in their design. This argument is threefold. First, as we have seen in chapter 2, training an SVM requires the choice for a kernel function – which is a design decision. Up front, it cannot be known whether the kernel function maximizes the decision boundary, especially when we can choose many kernel functions from the set of alternatives. Second, SVMs cannot be used in a multiclass classification problem (e.g. predicting material type where the number of material types > 2) unless many binary ones are trained and aggregated into a framework. Third, SVMs do not scale well with larger data sets. Our literature analysis shows that by consequence, many authors use a limited data set with often a limited amount of classes, which is not scalable with respect to practical applications of such algorithms.

Before we continue, we must explicitly note that the authors are not to blame for these design limitations, given the state of technology advances at the time of performing their study.

What is interesting about their work, however, is their approach towards feature extraction. The DCT is a signal processing technique and is based on the assumption that a waveform can be considered to be a highly transformed cosine, and can thus be broken down into its atomic elements (i.e. cosines with different frequencies, which together re-compose the original waveform). It thus allows one to transform a time waveform (i.e. wave amplitudes plotted against time) into its frequency spectrum. That is, the spatial and time aspects of the wave are removed altogether, and what remains is the set of coefficients of atomic cosine frequencies which together re-compose the wave.

This approach is promising for multiple reasons:

- Since the A-scan waveform is transformed into cosine frequencies, the feature vector becomes depth-invariant with respect to the underground object. For this to be true, it must be noted that the A-scan waveform must be pre-processed to take attenuation into account, e.g. by applying a time-varying gain, which in this work is considered to be a global data pre-processing technique.
- The DCT is known for its energy compaction, which means that the distribution of *signal energy* over the individual DCT coefficients follows a power law. For example, El-Mahallawy and Hashim used only

seven DCT components as a feature. Specifically, the performance of their model deteriorated when using more coefficients, presumably due to overfitting.
- For every training sample containing a similar object, the same frequencies should emerge in the DCT transformed feature vector. With a relatively large training set, one should be able to successfully predict the material type of an object based on shared frequency elements.
- Since for new values the DCT should only be computed for one A-scan per prediction, its computational requirements are limited, making the algorithm suitable for practice.

Recent technology advances, especially the rise of CNNs, could potentially reduce the SVM design limitations discussed above. First, CNNs require no kernel function to be included up front, but come with implicit kernels which update during run-time: they can thus learn the optimal decision boundary themselves. Second, by including an appropriate activation function in the output layer (e.g. a *softmax* activation function), CNNs can generate a multiclass probability distribution for new input values. Third, since CNN kernels are sparse and their parameters are shared over the entire input space, CNNs scale significantly better with larger data sets.

It could therefore be worthwhile to build upon the work by El-Mahallawy and Hashim by taking theirs as inspiration for a novel algorithm. In this chapter, we therefore propose a DCT-based CNN algorithm for identifying the characteristics of underground utility objects. It comprises a data pipeline with pre-processing steps to maximize uniformity of the A-scans, after which a DCT-based feature extraction approach is used, followed by a CNN classifier. We discuss its idiosyncrasies in section 4.4.

4.1.3: B-scans Extending A-scans

Instead of focusing on individual A-scans and signal processing techniques, Almaimani focuses on B-scans in her work [14]. It can be argued that B-scans, which reflect the underground object more thoroughly, can prevent an algorithm from detecting noise. This is especially the case when such noise occurs in the specific A-scan provided as input to the trained machine learning algorithm. On a B-scan, this would become visible as the occurrence of noise in one trace, but also as the absence of a hyperbola. Human beings can then immediately classify the noise as noise, whereas an A-scan based machine learning algorithm might become confused. Note that since our algorithms will be controlled by a human being, who selects a trace for classification, this will very likely not occur. However, it is still interesting to compare a B-scan based algorithm with the A-scan based ones discussed before. We therefore also propose a B-scan based algorithm. For similar reasons as with the DCT based algorithm, we propose this algorithm to be driven by a CNN.

We propose to combine findings from state-of-the-art studies which together compose the design of our algorithm. For this, we start with one of the most novel works on B-scan based machine learning algorithms for utility characterization. In this work, Almaimani normalizes B-scan windows to a fixed size which are then piped into a CNN for classification [14]. With the network type that we use in our work, she has achieved substantial results. Yet, an argument for improvement can be made that is manifold.

First, we assume that the CNN was trained on data that was not pre-processed. By consequence, no time-varying gain was applied to the input data. This could distort the results, because the target wave attenuates with time. Second, the information available about the object is most clearly available near the hyperbola apex. Consequently, we argue that it is not necessary to use the entire hyperbola as one's feature space, especially when the machine learning objectives are to *amplify* rather than to *replace* human beings.

For example, Pasolli et al. use a small window around the hyperbola apex to estimate object size [76]. Their approach yields promising results. Therefore, we aim to combine those two approaches and add novel aspects, such as a CNN-based classifier and some data pre-processing considered to be imperative by geophysicists, noted in conversations with TerraCarta. Since it works with B-scan windows, we coin it to be a *B-scan Window Based* CNN algorithm. We discuss its idiosyncrasies in section 4.5.

4.2: Machine Learning Data Pipeline

In this section, we discuss the generic machine learning data pipeline used in our work. It is the foundation of training and operating all three algorithms discussed in sections 4.3-4.5. We first discuss the machine learning objective of our algorithms, since it impacts the design of our data pipeline. We then proceed with a high-level design of our data pipeline and a more detailed discussion per component.

However, as stated, we must first discuss the machine learning objective of our algorithms. In our work, we attempt to solve a multiclass classification problem: given a predetermined set of classes (i.e., the material types for various underground objects), we train our machine learning models to distinguish between them. Through training, the models thus become capable of generating a multiclass probability distribution for new samples. That is, for a new sample, they compute the probability that it belongs to one of the classes from our predetermined set. Together, the probabilities summate to 1. This, given the notion that the sum of all probabilities in a probability scenario always yields 1, renders the conclusion that model output for this machine learning objective is said probability distribution.

The generic ML data pipeline itself is composed of the elements that are shared by all three novel algorithms. In the discussions in sections 4.3-4.5 on their idiosyncratic characteristics, the generic pipeline is extended and algorithm-specific features are added. Nevertheless, any data pipeline for classification consists of multiple sequential steps [46]:

1. Data selection;
2. Data pre-processing;
3. Feature extraction;
4. Classification.

We will next discuss the generic pipeline's individual steps in more detail.

4.2.1: Data Selection Dilemma

The first phase in our data pipeline is *data selection*. In various fruitful discussions with TerraCarta B.V.'s geophysicist and other project champions, we weighted the types of data that can be used in our work. Eventually, we decided that this work will use simulated GPR imagery. This argument is manifold. First, by using simulated GPR imagery, variations in soil type (and consequently dielectric constant), object depth, object diameter and object material can be pre-programmed. This yields improved and accurate labelling, or even labelling *by design*.

Second, utilizing simulated GPR imagery is the de facto standard for works from the recent literature [74]. Third, simulated GPR imagery is relatively noise-free, and could thus help demonstrating the strength of our approaches without interference from e.g. signal loss. Fourth, pre-programming in simulated GPR imagery can ensure that only one object is present within the B-scans. Fifth, simulated GPR imagery results in B-scans (and by consequence A-scans) that are of equal width and height. This makes the work of the machine learning engineer easier. Sixth, although TerraCarta B.V. possesses a database of labelled GPR echoes, for early validation of the machine learning approaches it is less worthwhile to first structure the database for inclusion as a training set compared to simulating GPR imagery.

However, the choice for utilizing simulated GPR imagery comes with certain drawbacks. For example, simulated imagery presents idealistic representations of buried objects. This could result in overly optimistic model performance. Specifically, models could lose their generalization power for real GPR imagery. However, for the early validation activities for which the algorithms are proposed, the downsides of using simulated GPR imagery do not exceed the possible benefits. Consequently, it was decided that simulated GPR images are used for this work.

4.2.2: Simulating GPR Imagery with gprMax

For simulating GPR imagery, we use the open-source software program gprMax [82]. GprMax implements the finite-difference time-domain (FDTD) method for solving Maxwell's equations, which are a set of equations that are the foundation of classical electromagnetism. Since a GPR device emits waveforms and receives those, the FDTD method can be used to simulate GPR behavior.

When simulating GPR imagery, it is mandatory to choose a specific GPR antenna with which the simulation is performed. Those antennas, which are software implementations of the specific real-life antennae, simulate emitting and receiving waveforms, returning the GPR A-scan(s) and, possibly, B-scan. Since TerraCarta B.V.

utilizes GPRs produced by Geophysical Survey Systems, Inc. (GSSI), we use a GSSI antenna in our simulations. Specifically, we use a 400 MHz antenna instead of the 1.5 GHz antenna which is also available. The reason for this is that it produces imagery that is more granular and therefore more detailed.

GprMax comes in two flavors, namely, the 2D and 3D versions. In the 3D version, Maxwell's equations are solved in three dimensions, which is how GPR devices operate in real life. However, the 2D version is significantly faster (using a machine equipped with a Nvidia Titan RTX, one of the fastest GPUs on the market today, 2D modelling of 770 B-scans with 100 A-scan traces takes approximately 2 days, whereas 3D modelling takes approximately 12 weeks). This occurs because in gprMax 2D the third axis of the emitted signal is flattened. An underground object is then modeled as if it is detected in a two-dimensional way. For underground utilities, given the insights from TerraCarta B.V.'s geophysicist, this could possibly yield deviations from signals in 3D models, rendering our work only indirectly applicable to practice. However, this is not too problematic: if our machine learning algorithms show sufficient discriminative power when applied to 2D imagery, we expect it to work with 3D imagery as well, creating demand for expensive 3D simulations. Additionally, previous studies have used gprMax 2D with promising results and realistic hyperbolae [74]. Consequently, we choose to benefit from the increased speed of simulating the models and perform our work with GPR imagery simulated with gprMax 2D, recognizing it as a possible limitation.

In the gprMax simulation, several parameters must be configured before the simulation can be started. For the signal emitted by the emulated antenna, we used an air-shot generated with a true GPR device, using the custom wavelet features provided by gprMax. By using this custom waveform, our aim is to resemble the true 3D waveform as much as possible. This design decision was made after deliberation with TerraCarta's geophysicist, who suggested that it maximizes resemblance with the waveforms emitted by TerraCarta's GPRs. This way, the impact of the limitation recognized before is minimized to the largest extent possible.

Simulations were made for various materials which either represent cables and pipelines, or noise. Primarily, high density polyethylene and similar plastics (HDPE), stoneware, cast iron, a perfect electric conductor (PEC) such as copper or steel, concrete (i.e., real objects) and tree/plant root objects (i.e., common noisy objects) are simulated. Consequently, the targets used in our work are as follows:

$$Target\ object\ classes = \{\, Concrete, HDPE, Iron, PEC, Root, Stoneware \,\}$$

The number of targets, therefore **NUM_TARGETS**, is 6. Besides target materials, target contents also influence the signal received by the GPR antennas. For example, a concrete sewage filled with water or with gas produces a different signal. Per target, we thus next also specify its possible contents used in gprMax. Note that the specification PEC in $Contents_{HDPE}$ indicates a copper or steel wire isolated with HDPE. Additionally, the set of contents for a plant/tree root is empty, since such objects cannot have any contents.

$$Contents_{Concrete} = \{Sewage\ Air\ and\ Water, Sewage\ Gas\}$$

$$Contents_{HDPE} = \{\, Gas, PEC, Water\}$$

$$Contents_{Iron} = \{Gas, Water\}$$

$$Contents_{PEC} = \{\, Water \,\}$$

$$Contents_{Root} = \{\, \emptyset \,\}$$

$$Contents_{Stoneware} = \{Gas, Water\}$$

In our simulations, the target objects have various diameters. Those diameters are different with respect to the target class. Next, we provide the diameters used in our work to generate GPR imagery for the particular object. Those diameters are either provided in mm, or as the *inside diameter size value* (ID; Dutch: DN, binnendiameter). Those values approximate the outer diameter of the object. Together with TerraCarta's geophysicist and engineers, we tailored those diameters to objects commonly found in the Dutch underground. This way, we maximize the relevance of our simulations for practice scenarios.

$$Diameters_{Concrete} = \{\, DN300 \,\}$$

$$Diameters_{HDPE} = \{\, 10, 15, 20, 25, 30, 32, 63, 110, 250 \,\}$$

$$Diameters_{Iron} = \{\, DN50, DN100, DN200 \,\}$$

$$Diameters_{PEC} = \{\, DN100, DN200 \,\}$$

$$Diameters_{Root} = \{\, 50, 100 \,\}$$

$$Diameters_{Stoneware} = \{\, DN100, DN150, DN200, DN250 \,\}$$

Additionally, the target objects are buried at various depths. For choosing depths per target class, a common sense approach is used based on the presence of certain objects in Dutch infrastructure. This means that not every object is available for every depth. For example, sewage pipelines are not present at the 30cm depth levels, because common sense dictates that they are buried deeper in the underground. They were consequently not added for such depths. For plant/tree roots, different depths were used as well, given common locations. Those are listed separately. Depths are provided in centimeters.

$$Depths = \{\, 30, 60, 90, 120 \,\}$$

$$Depths_{Root} = \{\, 5, 10, 20, 40, 60 \,\}$$

The objects are simulated in various soil types because soil type influences wave velocity and therefore how objects are represented on A-scans. The *homogeneous* soil type is our baseline soil type and has a dielectric constant of $\varepsilon_r = 9$. Other soil types differ from our baseline soil type, but given the fact that soil is often a random variation of different mediums, no exact ε_r values can be provided.

$$Soil\ types = \{\, Homogeneous, Clay, Sandy\ Clay, Sand \,\}$$

Our simulation design produced 770 B-scans in total. Each simulation contains 150 traces (i.e. A-scans) per B-scan. When visualized, the hyperbola representing the simulated object is visible prominently. The air-ground interface is relatively undistorted. The B-scan is also relatively noise free. Those two design choices could possibly be limitations of this work, since real-world GPR data has a varying air-ground interface and is often relatively noisy. Recognizing them, we do nevertheless proceed with our simulations, since our aim is to demonstrate the initial feasibility of the machine learning approaches proposed next.

4.2.3: Loading Data

The next step of our generic ML data pipeline is loading the simulated GPR images for the purpose of feature extraction. To make this possible, we perform various steps sequentially:

1. GprMax simulations are output in their own format (i.e., .out files) which are converted into GSSI-compatible files (.dzt files) first using MATLAB conversion scripts provided by gprMax.
2. Part of this conversion is a resampling operation which resamples the A-scan contents, which may be of various lengths, into a vector of 1024 scalars. This length was chosen because it lies between common sample formats (512) and the easily larger raw waveforms.
3. Then, using the **READGSSI** software tool, the .dzt files are converted into .csv files. Those are stored together in a single folder.
4. Subsequently, by using Python and specifically the **PANDAS DATA FRAME** library, the .csv files are read into a data frame. A data frame allows an engineer to manipulate the data with great flexibility. The data is now ready for applying global data pre-processing techniques.

This process is started 770 times, for processing the individual GPR images. Note that when completed once, steps 1 and 2 do not need to be repeated if data needs to be reloaded. Rather, only step 3 – which is relatively inexpensive – needs to be repeated.

4.2.4: Global Data Pre-Processing Techniques

The second step in the data pipeline is *pre-processing* of the data used in the machine learning models. Depending on one's viewpoint, it can also be considered to be *post-processing* of the GPR output (and thus for this work, the simulation output). A large number of pre-processing techniques exists which can improve target detection [83].

Many of them are application dependent. That is, they are only useful in some machine learning context. For example, the SVMs that were discussed in the previous chapter required extensive data pre-processing, whereas the CNNs used by Almaimani did not. Therefore, most of the pre-processing discussion is provided at the algorithmic level, in the next sections. Globally, however, we apply those pre-processing techniques in sequential order:

1. Ground bounce removal;
2. Time-varying gain;
3. Feature-wise normalization.

We will next discuss the individual components.

4.2.4.1: Time-varying gain

However, one pre-processing technique that is used globally throughout the algorithms is time-varying gain. The waves transmitted into the ground lose energy as a function of time, which becomes visible as signal attenuation [11]. This explains why in raw GPR images the air-ground reflection (or, the ground bounce) is clearly visible whereas hyperbolae are visible in a limited way.

Time-varying rain reduces such attenuation. After discussion with TerraCarta B.V.'s geophysicist, we choose the SEC time-varying gain function, also known as the energy decay gain. It applies a gain exponentially. It is a temporal gain function, which means that it operates at the level of individual A-scans [11]. It is chosen because it is a relatively fool-proof gain function compared to other available gain functions. We implement it as follows based on tests with our data:

$$Gain = \min(GAIN_{MAX},\ 0.05 * SAMPLE * (GAIN\_EXPONENT^{T/20}))$$

Where **SAMPLE** is the current sample (i.e, one amplitude scalar in the A-scan) which is exponentially increased using the **GAIN_EXPONENT**, which in this case is 1.2. This exponential increase is used to gain the initial **VECTOR** which is then multiplied with 0.05 to reduce absolute values. Values can be gained up to a maximum of **GAIN_MAX**, which is 3 x 10^5. The numeric values were determined experimentally by inspecting how the GPR simulated images change with respect to the applied gain.

4.2.4.2: Feature-wise Normalization

Neural networks generally tend to work better when numeric variance within the input data set is small [46]. For this reason, feature-wise normalization must be applied to the input data. That is, for every feature (i.e. A-scan), we subtract the mean of the feature vector and divide by the standard deviation. This ensures that "the feature is centered around 0 and has a unit standard deviation" [46].

4.2.4.3: Ground Bounce Removal

In section 4.2.2, we outlined that we use simulations of underground objects created with gprMax. Those simulations represent B-scans of those objects and are composed of 150 individual A-scans. Since a simulated signal starts in air and relatively quickly moves into the ground, the change of dielectric constant of the medium is represented by what is known as *ground bounce*: a highly characteristic echo, available within every A-scan, relatively stable over the entire radargram [61].

The ground bounce does not provide any information with respect to the object. It is therefore considered to be good practice if an analyst attempts to remove it [61]. There exist many methods for ground bounce removal. Usually, choosing the best performing method is dependent on the ground bounce visible in the radargram. However, generally speaking, more complex methods such as Principal Component Analysis (PCA) often yield better results than simple methods, such as mean-based and median-based ground bounce removal [84]. Complex algorithms such as PCA-based ground bounce removal are however more complex algorithmically and computationally.

In this work, we choose to remove the ground bounce using a median-based approach. The reasons for this choice are twofold. First, PCA-based ground bounce removal is especially more successful than mean- and

median-based ground bounce removal when the ground bounce is noisy. That is, when disturbances cause the ground bounce to vary between A-scans in the B-scan. This is not the case with our simulations. Rather, the ground bounce is relatively free of disturbances. This removes the need for computationally complex algorithms such as PCA and make mean and median-based algorithms prime candidates. We choose the median-based algorithm since it can also be made adaptive to smaller intervals at the B-scan level [84].

4.2.5: Feature Extraction and Classification

The third step in the data pipeline is *feature extraction*. This step is entirely dependent on the individual algorithms that will be proposed next. Consequently, we will elaborate on our approach towards feature extraction at the individual level.

However, we must note that beyond feature extraction another shared step can be presented. Data which is pre-processed and from which features are extracted is stored in the HDF5 format. This format allows one to store data uniformly and in a standardized way. It benefits (1) later utilization and (2) removes the need to perform pre-processing every time the models are run. We use **H5PY**, a Pythonic interface to HDF5.

Given the dependencies of feature extraction on the individual algorithms, we will next elaborate on our approaches towards classification of GPR imagery at the individual level. As illustrated by the rationale provided in section 4.1, we will first discuss the signal histogram based CNN. It is followed by a discussion on the DCT and B-scan window based ones. Together, the three classes of algorithms allow for comparison between performance using individual A-scans and B-scans, the latter of which composes multiple A-scans.

4.3: Histogram Based CNN Algorithm

An A-scan is a representation of both the spatial and time domains of the received echo. That is, one axis shows the 'delay' with which the echo was received and the other shows the amplitudes. This means that if we can strongly represent the target (i.e., the underground object) and reduce the echoes from noise and e.g. the air-ground interface, most of the variance within the A-scan is caused by the target. This would make features suitable for feeding them as input into machine learning models. This section presents a novel algorithm based on echo signal histograms, used for predicting the target materials introduced in section 4.2.

4.3.1: Pre-processing

Pre-processing involved applying the general ML data pipeline. It provides a sufficiently normalized result: most of the ground bounce is eliminated, the true signal is gained with respect to the time domain and has been normalized feature-wise to make it most suitable for neural networks.

4.3.2: Generating Backscatter Histograms

As a second pre-processing step, we extract signal histograms from the normalized signal backscatters. For doing this, we use NumPy's *histogram* function, which allows the conversion of an 1D array of arbitrary size into a histogram array. The first design decision with respect to histogram the number of histogram bins, i.e. the shape of the feature vector used as input for training the ML model. Since by applying global pre-processing techniques our features are standardized, and i.e. are centred around 0 and have the number of standard deviations (σ) as their unit, we choose 101 bins of size σ/10, with a bin interval of [-5σ, 5σ] (note that the bin size is not 100 since 0σ is also included). Setting the number of bins at 100 allows us to obtain a relatively thorough feature vector (with bins sized 0.10 the standard deviation) that is sparse as well. Consequently, our feature vector will be one-dimensional and be of length 101 with a shape of (101,).

4.3.3: Convolutional Neural Network Based Classifier

The histogram feature vector can be used to train a CNN for classification. In her work, Almaimani shows that a relatively generic yet thorough CNN architecture performs best [14]. It comprises three convolutional blocks followed by one densely-connected hidden layer that provides a multiclass probability distribution using a softmax activation function. We take inspiration from her work and take it as starting point and propose a similar architecture, presented in image 7.

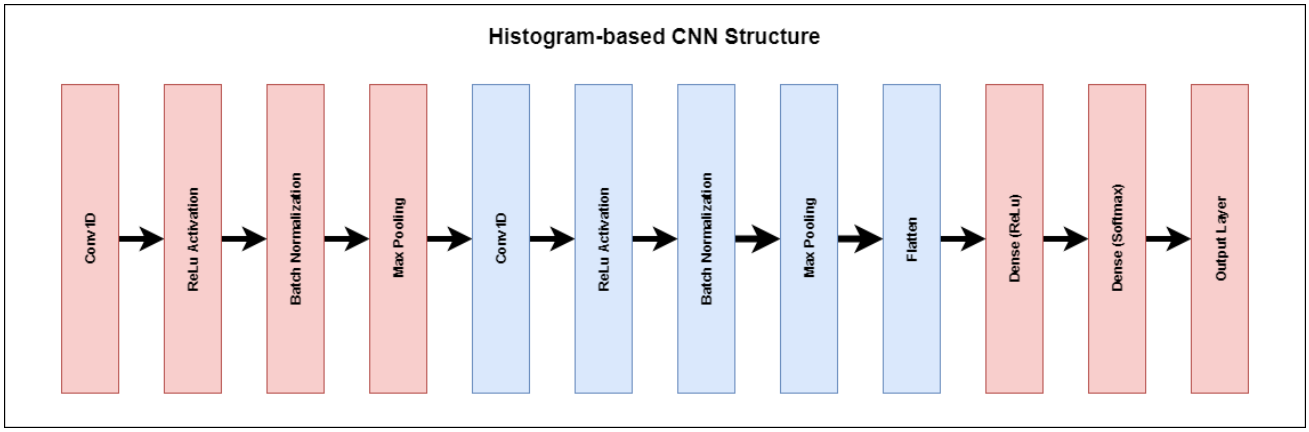


Image 7: *Proposed histogram-based CNN.*

It is composed of two convolutional blocks and one densely-connected block. We use two rather than three convolutional blocks given the relatively small size of the feature vector. Whereas Almaimani used B-scans and thus her networks required the capability to handle a substantial feature space, our histogram based feature extractor generates feature vectors of shape (101,). This is substantially sparser. Three full-blown convolutional blocks might then even yield a performance reduction because data is compressed too tightly.

Every convolutional block comprises a **CONV1D** layer. This layer allows the network to learn patterns in the data itself by allowing kernels to slide over the input vector. Conv1D layers are designed specifically for one-

dimensional temporal or spatial structures. Since our data is of this format, we use this layer type. The first Conv1D layer is implicitly linked to the input layer through the **INPUT_SHAPE** parameter.

The first Conv1D layer learns a number of 10 filters with size 3 and stride 1. This number is chosen relatively intuitively, but lies between 4 and 16, as suggested by default. As our activation function, we use Rectified Linear Unit or ReLU. This is the de facto standard activation function in DNNs. Subsequently, we apply Batch Normalization and then use max pooling to down-sample our data and to reduce the impact of large swings during optimization. Specifically, for max pooling, we use a pool size of 2, effectively halving the input.

We repeat this process in the second convolutional block, but instead of 10 it now learns 20 filters of size 3 with stride 1. The assumption here is that whereas the first convolutional block learns global patterns, more detailed patterns are learnt in downstream layers, requiring a larger number of filters to be learnt. We then halve the input again using max pooling. By applying a **FLATTEN** layer, we ensure that the convolutional output can be used by densely-connected layers, which are added to allow the model to generate predictions.

The densely-connected layers allow the model to structure the patterns learnt by the convolutional blocks. The neurons in the first **DENSE** layer use ReLu as their activation function, which is the common activation function that we used before. Given the relatively large output size of the Flatten layer, the number of outputs of the first Dense layer is set at 60. For the second layer, however, **SOFTMAX** is used as an activation function. Softmax allows the model to generate the multiclass probability distribution we need to predict target probabilities. It does however require that the number of output nodes for this layer is equal to **NUM_TARGETS**, which in our work is 6. Effectively, this architecture is expected to allow the model to characterize the A-scan through its histogram.

4.3.4: Full Design

Consolidating the discussion above into the final data pipeline with respect to this algorithm yields the final design presented in Image 8.

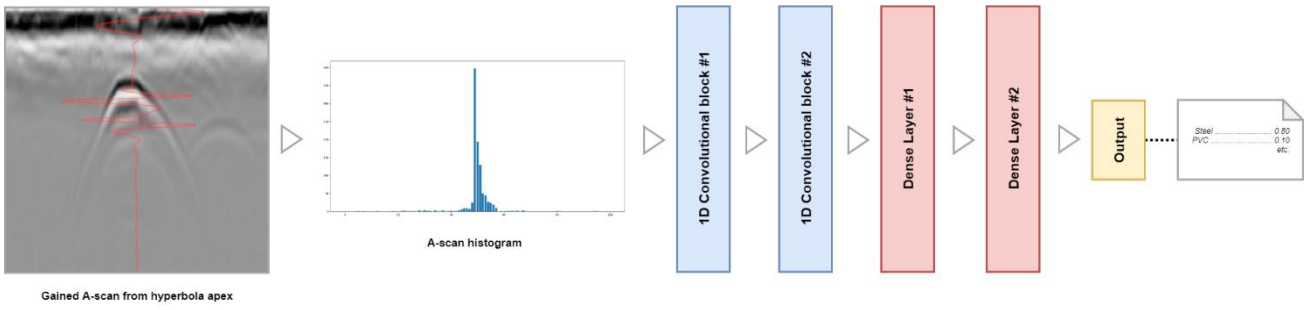


Image 8: *Full histogram-based data pipeline design*

4.4: Discrete Cosine Transform Based CNN Algorithm

Another way with which one can reduce the dimensionality of the A-scan waveform is by applying the Discrete Cosine Transform (DCT). With this mathematical operation, one transforms a waveform expressed as a finite array into a set of cosine functions oscillating at different frequencies. That is, by merging those cosines again, one finds the original waveform.

The DCT benefits from a property called *energy compaction*, which means that it represents the majority of the signal energy in its first cosine coefficients [74]. That is, if a signal is converted into e.g. 100 coefficients, if subsequently the first 5 are drawn and input into the inverse DCT, the output is likely to be similar to the original waveform. Differences between the original waveform and the inverted DCT are extremely small due to energy compaction. The property thus allows for a substantial reduction of feature space dimensionality. It is therefore not surprising to find the DCT being applied in compression, e.g. the JPG image format.

Since previous applications of the DCT for dimensionality reduction have yielded very promising results, this section presents a novel combination of DCT based feature extraction with CNN based classification. We will first discuss pre-processing techniques extending the general pre-processing techniques discussed in section 4.2. We then introduce the mathematical basis of the DCT used for feature extraction. It is followed by a discussion on the individual components of the DCT-based CNN classifier. We conclude with its global design.

4.4.1: Pre-processing

During discussions with TerraCarta B.V.'s geophysicist, we identified a set of minimum requirements which together make this approach worthwhile for validation. First, it is critical that the model is trained with A-scans taken from directly above the underground object. This is a necessary requirement since signal strength with respect to the underground object is strongest at this point. Consequently, the A-scan representing the peak of the B-scan hyperbola is used as the unprocessed feature input to the feature extractor.

Pre-processing involved applying the general ML data pipeline. It provides a sufficiently normalized result: most of the ground bounce is eliminated, the true signal is gained with respect to the time domain and has been normalized feature-wise to make it most suitable for neural networks.

4.4.2: Discrete Cosine Transform

For the DCT, we use the DCT type 2, which is considered to be the *de facto standard* by the signal processing community. We specifically use the definition and implementation provided by the **SCIPY** Python package for signal processing. SciPy's default settings are used to compute the DCT as follows:

$$y(k) = 2 * \sum_{n=0}^{N-1} x_n \cos\left(\frac{(\pi * k * (2n+1)}{2 * N}\right), \quad 0 \leq k < N.$$

Applying the DCT to the entire array yields DCT coefficients for the entire input array. That is, if our input array is of length 1024, the DCT array will be of length 1024 as well. However, El-Mahallawy and Hashim show in their work that the performance of their classifier is greatly reduced when the number of coefficients is increased. We expect that this behavior occurs because the most global signal properties are present in the first coefficients. Adding additional ones introduces more granular details to the training set, increasing the odds of overfitting. In their study, El-Mahallawy and Hashim used only seven coefficients with A-scans of length 512. Therefore, we initially align our work with theirs, using 14 DCT coefficients as a feature vector.

4.4.3: Convolutional Neural Network Based Classifier

The relatively low number of features in our feature space introduces the need for a different design compared to the histogram based approach. Specifically, it should benefit from the power of CNNs while not minimizing the feature space too much, effectively creating a bottleneck. We therefore propose the following design.

At a high level, it consists of two convolutional blocks and one block of densely-connected layers. The convolutional blocks are expected to be capable of learning filters for the idiosyncratic patterns present in the DCT coefficients. The densely-connected block is expected to be capable of subsequently structuring convolutional outputs, generating a multiclass probability distribution over the targets, i.e. material types.

The first convolutional block is composed of a one-dimensional convolutional layer (**CONV1D**). Using the **INPUT_SHAPE** parameter in Keras, we link the implicit input layer of shape (14,) to the Conv1D layer, allowing us to propagate data into the network. Given the relatively small size of our feature space, we learn 4 filters in

the first convolutional block, use kernels of 1 and use a stride of 1. We choose **ReLu** based activation since it is a commonly used function and in practice often yields improved results when compared to the other two commonly used functions, **tanh** and **sigmoid**.

In order to normalize weight swings during optimization, we apply **Batch Normalization** here as well. Subsequently, we perform **max pooling** for down-sampling the data. Our pool size is 2.

We repeat this process in the second convolutional block, although we remove the max pooling operation (since further down-sampling could reduce dimensionality too strongly) and introduce a **Flatten** layer, as with the histogram algorithm. Since it is common practice to multiply the previous number of learnt filters by two for the number of filters learnt in a particular layer, this Conv1D layer will learn 8 filters.

The densely-connected block consists of two **Dense** layers. Those are very similar to the ones used in the histogram based algorithm. The first Dense layer uses ReLU activation; the second utilizes Softmax for generating the multiclass probability distribution. Given the output of the Flatten layer, we set the number of outputs of the first Dense layer to 40. The second layer number of outputs must be equal to **NUM_TARGETS**, which in our work is 6. The entire DCT-based CNN design is presented in Image 9.

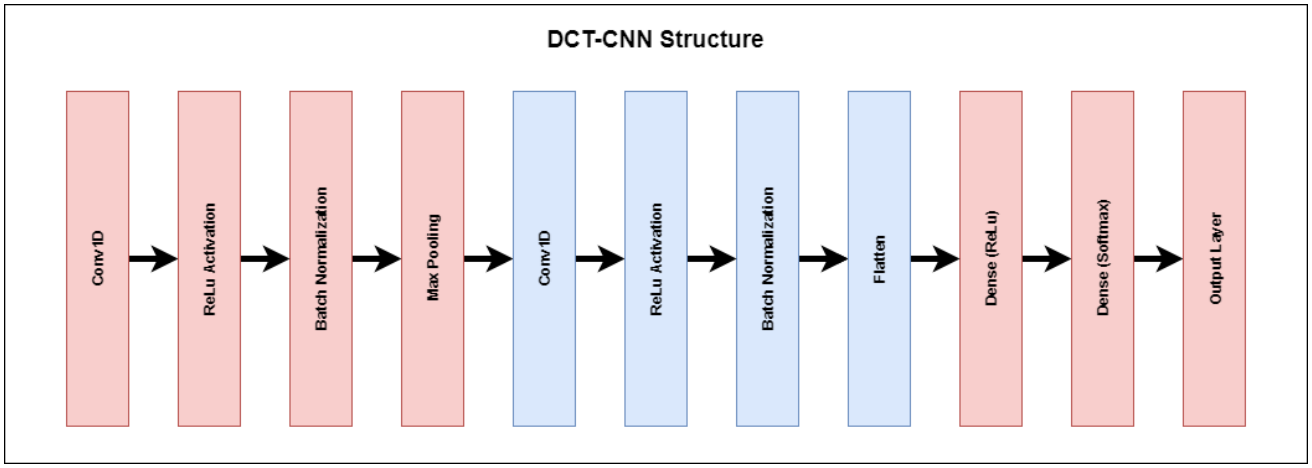


Image 9: *Proposed DCT-based CNN*

4.4.4: Full Design

Consolidating the discussion above into the final data pipeline with respect to this algorithm yields the final design presented in Image 10.

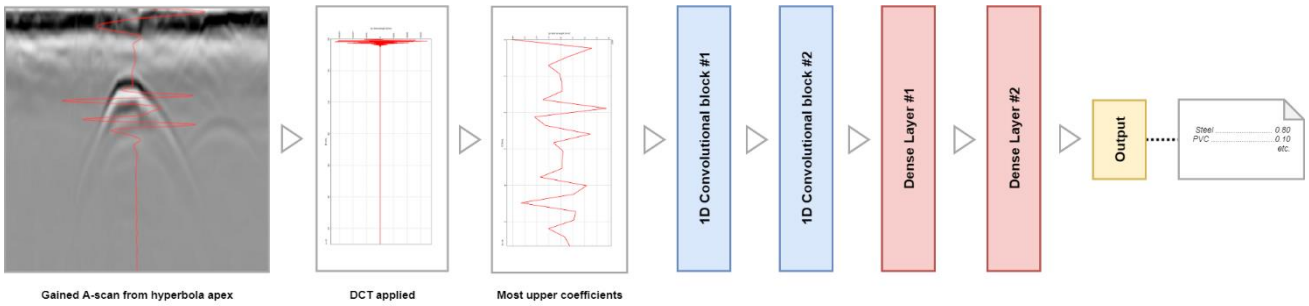


Image 10: *Full DCT-based data pipeline design*

4.5: B-scan Window Based CNN Algorithm

In this section, we discuss the idiosyncrasies of the B-scan window based CNN algorithm. Specifically, we discuss pre-processing for this algorithm, followed by our approach to extracting features from the B-scans. Subsequently, we discuss the CNN classifier and its components. We conclude with its full design.

4.5.1: Pre-processing

Pre-processing involved applying the general ML data pipeline. It provides a sufficiently normalized result: most of the ground bounce is eliminated, the true signal is gained with respect to the time domain and has been normalized feature-wise to make it most suitable for neural networks.

4.5.2: Extracting Features from B-scans

Contrary to the previous two approaches, which yields *one* feature vector that is piped to the convolutional layers, the B-scan based approach yields *multiple* vectors that are together piped to the convolutional layers. These together are the B-scan window image.

To extract these features from the B-scan, our starting point is the hyperbola apex and, by consequence, the A-scan representing this apex. We subsequently include all A-scans 25 traces to the left of the apex A-scan as well as 25 traces to the right. Initial inspection of the data yields that the entire vertical scan is of relevance for classifying the material type, given the contents of the material. For example, continued echoes of a pipeline with water contents propagate towards the end of the scan with repeated hyperbolae. Since our simulations have 1024 samples (i.e., amplitudes) in an A-scan, the feature vector that is input into the CNN classifier proposed next has a shape of (51,1024).

4.5.3: Convolutional Neural Network Based Classifier

The B-scan based CNN algorithm, given its significantly larger **INPUT_SHAPE** of (51,1024), requires a more extensive architecture. Contrary to the A-scan based algorithms, we propose to use three convolutional blocks. Its main difference with respect to the A-scan based algorithms is that the convolutional layers used are **CONV2D** instead of Conv1D, since the input window is two-dimensional.

The first convolutional block is thus composed of a Conv2D layer, which implicitly links to the output of the input layer of (51,1024) through the **INPUT_SHAPE** parameter. Given the relatively larger input shape, we learn 12 filters of size 3x3. The stride used is 1. Similar to the A-scan based algorithms, a **RELU** activation function is used to introduce non-linearity to the linear combinations computed within every neuron. **BATCH NORMALIZATION** is applied to reduce variance in the model and **MAX POOLING** is used to down-sample the input. The pool size we use is 2, effectively cutting the input shape in half.

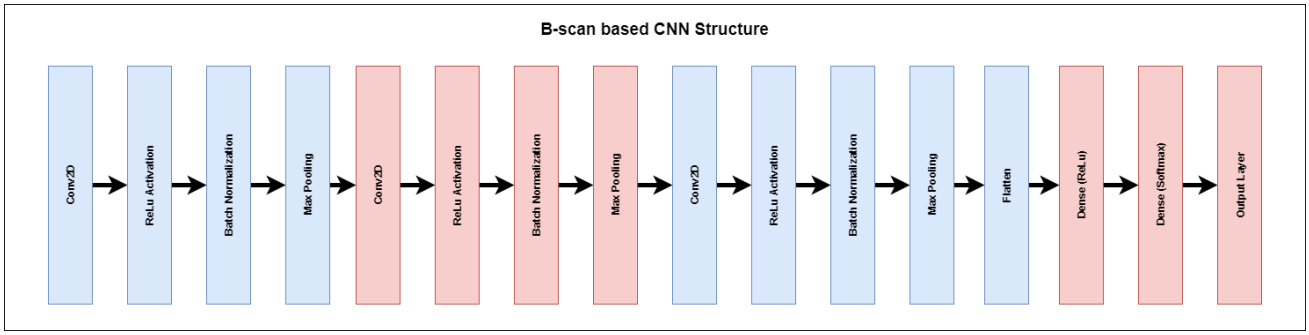


Image 11: *Proposed B-scan based CNN*

In the second convolutional block, we repeat this process, but instead learn 24 filters of size 3x3. Once again, we make this design decision because more granular patterns are learnt downstream the layers. The max pooling layer then similarly cuts the shape of the data in half.

In the third convolutional block, we repeat this process, but given the relatively small shape of the data, decrease the size of the filters learnt to 1x1. We do however learn 48 filters instead of 24, using the multiplication factor of 2 used before. Once more, max pooling cuts the input shape in half. Since this is our

final convolutional layer, from which the output must be propagated into densely-connected layers, we use a **FLATTEN** layer. It is followed by two densely-connected layers, which structure the convolutional filters and provide a multiclass probability distribution over the target classes to the output layer. The Flatten layer does however provide a relatively large output, since it merges the filters before passing them into the Dense layers. For the first Dense layer, we therefore set the number of outputs to 128, which is common. For the second Dense layer, given the Softmax activation function, the number of outputs must be equal to **NUM_TARGETS**, which in our work is 6. The full design of the B-scan based CNN is presented in Image 11.

4.5.4: Full Design

Consolidating the discussion above into the final data pipeline with respect to this algorithm yields the final design presented in Image 12.

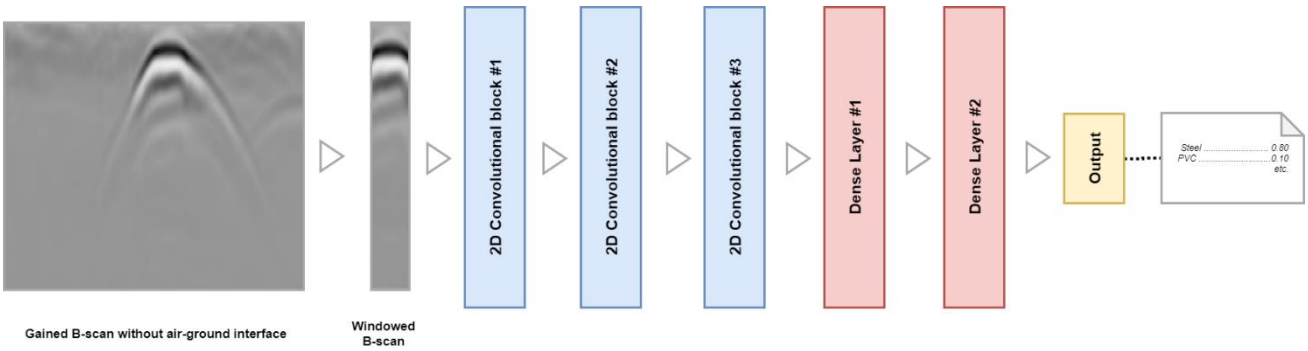


Image 12: *Full B-scan based data pipeline design*

5: Validation Strategy

In the previous chapter, we discussed three novel machine learning algorithms for underground utility material characterization. It is of academic and practical relevance to validate them, i.e. to convert our designs into code; to subsequently train the models with the gprMax simulated imagery and to validate their performance with respect to the research goals.

Training neural networks is perhaps more of an art than a true science. However, the ML engineer makes certain decisions up front about how he or she intends to train their models. This chapter presents this validation strategy. Section 5.1 discusses a brief recap on how neural networks are trained as well as the frameworks we will use for training. Section 5.2 provides a discussion on how we intend to split our data set into training, validation and testing data. Section 5.3 proceeds with global network configuration and discusses network parameters and hyperparameters. We then proceed with a discussion on validation metrics in section 5.4. Subsequently, in section 5.5, we discuss the shape of the individual CNNs we intend to use. Section 5.6, finally, provides a high-level overview of our validation design.

Python ML Framework	Use case
Spark MLLib / Deep Learning on Spark	Various frameworks can be used for training machine learning models benefiting from Apache Spark parallelism when handling big data sets. Utilization does however require that data is available from within Spark, and therefore that data is stored on e.g. Hadoop.
Tensorflow	Google-developed framework for processing of numerical values, often represented in multidimensional vectors named tensors. Can be used to create machine learning and deep learning models, but has a highly technical API and thus a steep learning curve. Is capable of running on Nvidia CUDA capable GPUs to speed up the training process.
PyTorch	Python framework for deep learning. Is said to be efficient, but we do not have substantial experience with implementing algorithms in PyTorch.
Scikit-learn	Python framework for traditional machine learning algorithms and relatively simple neural networks (perceptron and multi-layer perceptron). Cannot be used for deep learning. Has relatively simple APIs for creating models, but is in our experience not memory optimized for handling large data sets.
Keras	Abstraction layer for Python-based deep learning frameworks. Allows an engineer to create deep neural networks with a relatively simple and straight-forward API. Subsequently allows those models to run on either Tensorflow, Theano or Microsoft CNTK, enabling interoperability between models trained previously. We have substantial experience training models with Keras on top of TensorFlow.
Theano	Python framework for deep learning. We do not have substantial experience with Theano.
Microsoft CNTK	Framework for deep learning with Python APIs. We do not have substantial experience with Microsoft CNTK.

Table 9: *Non-exhaustive list of Python frameworks for machine learning.*

5.1: Training the Machine Learning Models

In this section, we discuss the high-level aspects of training our machine learning models. First, we provide a brief recap on neural network training at a high level, which allows the reader to better comprehend the more granular discussions in the subsequent sections. Second, before discussing the configuration of our models in detail, we constrain ourselves to a particular framework for training the models – Keras, running on top of TensorFlow. This constraint is required since parts of the configuration aspects discussed later, for example the Learning Rate Range Test for determining an optimum default learning rate, require the application of framework-specific tools.

5.1.1: Recap on Neural Network Training

Training a neural network is an iterative process. In every iteration, data is first propagated forward, and for every sample a prediction is made. Those predictions are aggregated, after which the error is computed backwards through the network. During this computation, changes with respect to the network's trainable parameters are computed as well, which can be multiplied with the so-called learning rate (how much of the change the network learns) and are then processed. This completes an iteration, which is also known as an

epoch. Training a neural network yields many epochs before a model reaches its optimum value. In some cases, tens of epochs can be used, while in other cases, tens of thousands of epochs are required. One considers the training process to be finished when during validation, the validation loss starts to increase [46].

5.1.2: Deep Learning Frameworks of Choice: Keras and TensorFlow

Years ago, it was extremely difficult to train neural networks, since they had to be built from scratch. Fortunately, there exist many frameworks and languages today specifically targeted at data science and machine learning. With respect to languages, R and Python are the de facto standard ones for handling large data sets. Given our substantial experience with Python, it is the language of choice for this work.

Subsequently, there is a wide array of frameworks that can be used in Python for training machine learning models. We list them in Table 9 and give our viewpoint based on practical experience. Please note that this list is non-exhaustive.

Our experience with Python frameworks for machine learning is primarily with Scikit-learn and Keras on top of TensorFlow. Since Scikit-learn is unable to handle deep neural networks, both in terms of available APIs and the data set sizes it can handle based on our experience, we do not use this framework. Keras, on the other hand, allows one to define neural networks using relatively straight-forward APIs. Additionally, it allows one to map those models to TensorFlow, which can handle large datasets. Furthermore, Keras is equipped with callbacks such as **EARLYSTOPPING** and **MODELCHECKPOINT** which allow one to stop the training process automatically based on their optimum loss value, saving the best performing instance of the model. Combined with our experience, we decide to use Keras on top of TensorFlow for training the machine learning models embedded in our algorithms.

5.2: Data Set Splits

As we discussed in the background chapter, successfully applying machine learning requires an engineer to find a balance between the predictive power of a model and its power to generalize. This is often a fine balance; for example, it is relatively easy to overfit a model, resulting in high predictive but low generalization power [46].

Successful identification of this balance requires one to use their data set in a multi-purpose way [46]. First, the model must be trained. Second, during training, the model must be validated continuously for improving its predictive performance. Third, after training, it must be tested to guarantee its generalization power.

Those objectives put different requirements on the input data set, which results in the input data set being split into three distinct sets: a training set, a validation set and a testing set [46]. During the training phase, statistical patterns are extracted from the input data. To find patterns that are statistically significant to some extent, one must use relatively large quantities of training data. Consequently, the training set must be relatively large, and thus substantially larger than the validation and testing set.

However, the model must also be optimized. This cannot be done with the training data, for the simple reason that it has identified patterns from this data set. Since validating the training result with its own data would resemble butchers inspecting their own meat, this is considered bad practice. Instead, a validation set is used, which was not used in training. It allows the engineer to (often automatically) validate the identified patterns against new data. The validation set is used iteratively, i.e. after every epoch, the model is validated against the same validation set.

This engineering choice, however, provides another challenge with respect to the eventual performance of the model. While validation sets can increase predictive power, they eventually bring about a loss of generalization power. This is the result of re-using the same validation set after every epoch. Since the validation set is used to optimize the model, it becomes intrinsically linked to model output. In statistical terms, the validation set samples are thus no longer independent from the samples used to train the model; rather, some unspecified correlation will be introduced that becomes stronger with every epoch. Worse, this information leak impacts how the model is optimized, rendering even larger overfitting on the validation set. Eventually, the effect is that the validation process is no longer sufficient to guarantee generalization power.

This is why another relatively small set of data, namely the testing set, is used after training machine learning models. Since its samples are not used for training, nor for validation, it contains data which the machine learning model has never seen. By consequence, it cannot be intrinsically linked to the previous phases and

can thus be considered to be relatively independent. Relatively, in this sense, because engineers should still be cautious about the contents of a training set, since e.g. time-series data also has a temporal dimension [46]. This testing set is used to test the final model, to identify whether it can generalize to new data. If it successfully passes testing, the model can often be applied in the real world.

Different methods for splitting the data set into three distinct parts can be applied [46] [85]. In this section, we discuss simple hold-out validation as well as various cross-validation techniques, which are more computationally expensive but are substantially less naïve. We finally discuss how we will split the data sets in this work.

5.2.1: Simple Hold-out Split

A relatively naïve but computationally inexpensive technique is simple hold-out validation [46]. It is nothing more than "[setting] apart some fraction of your data as your test set". It can be extended into holding out a validation set as well. Usually, the split is made with a predetermined ratio (e.g. 70%, 15% and 15%), and a random shuffle determines which sample belongs to which set. This is acceptable in many cases, but there exist a few edge cases that must be kept in mind [46]:

1. **Data representativeness:** all sets must be representative for the patterns that underly the total data set. For example, if the first part of an unshuffled data set consists of training samples of giraffes, and the latter part of samples of rhinos, simply holding out the latter part will guarantee problems. Random shuffling, as we indicated, can often be used to mitigate this challenge.
2. **The arrow of time:** with time series data, i.e. "[predicting] the future given the past" [46], random shuffling will yield a *temporal leak*: the "model will effectively be trained on data from the future".
3. **Data redundancy:** when some samples appear more than once, random shuffling can introduce redundancy between the training, validation and test sets. By consequence, their samples are no longer statistically independent. This is particularly problematic for the validation set and even more for the test set. One should therefore ensure that training, validation and test sets are disjoint [46].

5.2.2: Cross-validation

In the case when relatively little data is available, cross-validation techniques can be used for splitting the input data set [46]. With cross-validation techniques, one repeats the training process for a number of different splits, generally averaging the results. This reduces the variance introduced by simple hold-out validation between population and sample, which can be large when the data set is of limited quantity.

There exist multiple cross-validation techniques. One of the most widely used techniques is K-fold cross-validation, in which the input data is split into K partitions of equal size [46]. For every partition i, a model is trained on K-1 partitions and subsequently tested against i. For all partitions, the results are averaged into a final performance score. This does however not exempt one from using a distinct validation set [46]. Although the variance of splitting one's data is reduced through averaging and thus performance is generally better, K-fold cross-validation is a relatively expensive algorithm, since model training has to be re-done K times [85].

A specific instance of K-fold cross-validation is Leave out one cross-validation (LOOCV), where K=N and N is the number of samples in one's input data set. This greatly reduces bias and increases the power of the model to maximize pattern extraction from the training split (given its increased size). It is however extremely expensive (since the partition size is 1) and has increased susceptibility to outliers (for the same reason) [85].

Above, we discussed three challenges with respect to splitting one's input data set. Data representativeness, i.e. the giraffe-rhino problem, can be a challenge with K-fold cross-validation and LOOCV, since an equal balance of targets over the partitions must be ensured manually. Stratified cross-validation, however, extends K-fold cross-validation by equally distributing the presence of targets (i.e. classes) over the partitions [85]. It reduces bias and variance between the partitions, but is more expensive than regular K-fold cross-validation. It unfortunately cannot be used for multiclass classification given its method of distributing targets.

If one's input data set is a time series, time series cross-validation can be used [85]. By using forward chaining, the model is trained with an input set of increasing length. It is however less relevant for applications that have no time series as input data.

5.2.3: Data Split for GPR Imagery

This latter is the case for our algorithms. Although every individual sample is a time series, the input data set is not. Rather, for the A-scan based algorithms, the input data consists of A-scans taken from hyperbola

apexes. The B-scan based algorithm crops a window from the B-scan, which can be conceptualized as being an image. Consequently, since our input data set is not a time series, time-series cross-validation is no suitable validation technique for our work.

Neither is simple hold-out validation given the relatively small amount of input data; we assume that too much variance would be introduced by applying a random split. With respect to our input data set, the classes are relatively equally distributed over the input data set. By consequence, stratified cross-validation is too expensive for the requirements put forward by our work. We expect that we also do not need K=N based cross validation, thus LOOCV. Rather, we use K-fold cross validation. Typically, based on empirical results, the value of K ranges between 5 and 10. In our work, we use K=10. Subsequently, per fold, we split the training data into real training data and validation data. We use a 80/20 split which is common in the machine learning community: 80% of the training data will be used for training whereas 20% will be used for validation.

The design decisions made above ensure that training data is used for improving the model, validation data is used to determine its performance after each epoch prior to model optimization and that testing data is used to determine generalization power after training.

5.3: Network Configuration: On Parameters and Hyperparameters

Beyond neural network architecture, data pre-processing and applying a method for splitting one's data set lies the configuration of the neural networks themselves. We note that *network configuration* in this sense is different than the network architectures, which were discussed in the previous chapter. Those architectures can be compared to generic drawings, describing the building blocks that together compose the network. The network configuration, however, describes the actual instantiation of one: it discusses *one when assembled*, and specifically how it is assembled.

We discuss network configuration in this section. We first discuss the network parameters with respect to weight initialization. We also discuss the hyperparameters used globally, namely the loss function, optimizer, learning rate and regularization techniques. For validation purposes, we also discuss the validation metrics we use. Until now, the discussion is global, i.e., all three networks are configured that way. However, the final part of this section includes a discussion on the shape of the CNNs, which is provided for the individual algorithms. We finally provide a complete overview of our validation strategy.

5.3.1: Weight Initialization

In a neural network, every neuron is trainable. It is therefore also called a trainable parameter [46]. A parameter consists of a *kernel* and a *bias* attribute. The kernel contains the patterns learnt, whereas the bias is a threshold that adapts how the particular neuron fires towards its activation function.

Prior to the first training epoch, those weights must be initialized. Proper initialization is critical towards network convergence, i.e. finding a mathematical optimum under which the model performs best [86]. Fortunately, there exist multiple ways to initialize one's trainable parameters, some of which are more naïve than others. In this section, we discuss how we will initialize our parameters; specifically, the neurons of the neural networks embedded in our algorithms.

To do this, we will first briefly look into commonly used initialization methods, both with historical and current significance in the machine learning community. This also includes a discussion on their flaws and consequential challenges for the machine learning engineer. We finally discuss the choice of an initializer to be used in our work.

5.3.1.1: Zero Initialization

A machine learning engineer can decide to initialize their trainable parameters to zero. That is, initializing either kernel or bias to zero, or both. This is acceptable for the biases, since non-zero weights can be adapted during learning to mimic a missing bias, should it have been a necessary addition. The model will thus still be capable of learning at full strength.

However, initializing the kernels to zero is considered to be (very) bad practice by the machine learning community [87]. This results from the inner workings of a deep neural network. We recall from chapter 2 that an individual neuron works with vector multiplications and the addition of a bias term (i.e., $\mathbf{w} \cdot \mathbf{x} + \mathrm{b}$). If we initialize the kernels ($\mathbf{w}$) to a zero vector, the entire dot product ($\mathbf{w} \cdot \mathbf{x}$) will be zero as well. In this scenario, only the bias attribute would be propagated to the subsequent layer. Since the bias is static, and learning

adapts it globally, one would effectively deprive the model of its capability to learn from highly dynamic data. In fact, zero initialization results in a model that does not perform better than traditional linear models [87].

5.3.1.2: Random Initialization

To overcome the challenges of zero initialization, the machine learning community has applied random initialization instead [46]. Random initialization, which in deep learning frameworks can be applied at the level of individual layers, does what its name suggests: it initializes the trainable parameters randomly.

It must be noted with respect to random initialization that random number generators in common machines are not random; rather, they are pseudo-random. From experience with practice, we note that it is therefore good practice to seed the random number generator used for training a deep neural network prior to starting the training process. This fixes the pseudo-random number generator to perform in a certain and static way. By fixing the number generator, one safeguards themselves against peculiarities with respect to model performance resulting from the generator's pseudo-randomness. Randomly initializing the network weights has produced substantially well-performing models, but comes with additional challenges [87].

5.3.1.3: Vanishing and Exploding Gradients

When one trains a deep neural network, adding additional layers to the architecture would intuitively correspond with better model performance. Note that for convenience, here we ignore the increased odds of overfitting to one's data, but generally deeper models can better capture the complexities in the data than models that are more shallow.

Experiments do however suggest otherwise in the case of random initialization [88]. The reasons as to why this phenomenon occurs were unknown for a long time, but were eventually traced to two distinct but related problems, namely the *vanishing* and *exploding* gradients problems.

An epoch, of which many are performed during neural network training, consists of propagating the data forward and computing the error backwards [46]. Computing the error backwards, Chollet argues, means that "the network [computes] the gradients of the weights with regard to the loss (…) and [updates them] accordingly". This is performed using the backpropagation algorithm, which is the de facto standard for chaining the gradients of the neurons in all subsequent layers. Backpropagation allows one to compute the error backwards to the first layer, and allows the weights to update, in line with Chollet's statement.

However, mathematically, randomly initialized weights produce an exponentially smaller product of gradients with more layers. Consequently, the backpropagation algorithm – which is in effect a product containing gradients from the loss function to a layer upstream – will by consequence only produce small gradients, and thus small changes, for upstream layers. This is known as the *vanishing gradients* problem and occurs with random initialization: the more layers are added, the slower the initial ones will learn. This has become problematic since deep neural networks have become increasingly deeper over time [46].

Another challenge that can emerge during training is the *exploding gradients* problem. It is a distinct problem, not necessarily caused by random initialization, but is strongly related to vanishing gradients since it also emerges from utilizing backpropagation. Specifically, it also results from backpropagation's characteristic of computing the gradients in a chained way, with products of downstream gradients. Suppose that one initializes their network parameters randomly, but has done little data pre-processing. An exemplary case would be omitting the application of normalization to the input data. In that case, the gradients found for updating the network parameters could become extremely large. For the most downstream layer, this is perhaps not problematic, but for upstream layers, those large gradients are included in the backpropagation products. At some point, number overflow occurs, and the network can no longer adapt those particular weights. By consequence, the performance of a network deteriorates significantly when gradients explode.

5.3.1.4: Extensions to Random Initialization

Random initialization of the trainable network parameters can thus make the network unstable in two different ways. However, in both cases, the model no longer learns in its best possible way. Naïve engineers would suggest that backpropagation must no longer be used, since it is very troublesome with respect to network initialization. However, it is still an extremely efficient way to compute weight updates in a chained way, especially when comparing it with traditional, numeric approaches.

Fortunately, serious approaches that attempt to avoid the occurrence of vanishing and exploding gradients have also emerged. With respect to exploding gradients, the answer primarily lies in proper data pre-processing and the application of regularization techniques, which we will discuss later. For vanishing

gradients, however, new initialization techniques have emerged that extend traditional random initialization. Today, the de facto standard initializers used are Xavier initialization and He initialization [86].

We recall from chapter 2 that the result from the computation of a neuron (i.e., $\mathbf{w} \cdot \mathbf{x} + \mathrm{b}$) is first input into an activation function before it is propagated to the next layer. Various functions can be used for this purpose, which has a consequence for the computation process discussed above [86]. That is, some activation functions are differentiable at 0, whereas others are not. Specifically, the ReLU activation function that is widely used by the machine learning community today, is not differentiable. Other de facto standard activation functions, namely tanh and sigmoid, are differentiable at 0.

This notion is important for understanding why certain extensions to random initialization are useful whereas others are not. The initial extension to random initialization was proposed by Glorot and Bengio, whose study proposed an initializer called the *Xavier initializer* [86]. It still utilizes the same probability distribution for generating the initializations, but makes use of a different variance, showing effectively that the challenges with respect to random initialization can be reduced. Since it is highly effective for two of the three widely used activation functions (tanh and sigmoid), it is one of the standard initializers used in today's machine learning community. Based on Glorot and Bengio's work, Kumar derives a general weight initialization strategy for "any arbitrary differentiable activation function" [86]

He et al. however show that the Xavier initializer cannot be used for the other widely used activation function, namely ReLU. To resolve this problem, they propose a different strategy instead, which is called the *He initializer*. It uses a different variance, of which *why it works* was not understood until recently [86]. Kumar argues that the conclusions put forward by He et al. are correct and provides mathematical proof as to why the Xavier initializer is ineffective for ReLU activation functions, which originates in the fact that ReLU is not differentiable [86]. Utilizing the proof, Kumar shows that the optimal variance for ReLU is indeed as suggested by He et al. in their original work, namely *He initialization*.

5.3.1.5: Choice of an Initializer for Our Work

The algorithms we proposed in chapter 4 all utilize the ReLU activation function for the hidden layers, and all share the Softmax activation function which eventually generates a multiclass probability distribution. Based on the discussion in the previous sections, we first argue against utilizing a **ZERO** initializer. Second, **RANDOM** initialization can result in vanishing and exploding gradients and is therefore neither the preferred choice. The **XAVIER** initializer does not work properly when using ReLU activations, which is how our algorithms are designed. By consequence, we choose **HE** initialization for our work, specifically He uniform initialization.

5.3.2: Loss Function

The discussion on the training process of a neural network in the background section demonstrates that a neural network is trained iteratively. That is, the entire set of training data is fit once to the neural network, after which its performance is calculated, i.e., an epoch [46]. Training a neural network consists of a large number of epochs, ranging between dozens up to many thousands, before optimum performance is reached.

However, when we aim to optimize a model, we first need to have a uniform method of comparing the performance per epoch. Loss functions serve this purpose. They are defined as "the quantity that will be minimized during training (…) [representing] a measure of success for the task at hand" [46]. Depending on the neural architecture, one or multiple loss functions are used. The latter case occurs when a neural network is divided into multiple branches, and consequently shares multiple output layers with multiple output values. Since our algorithms have only one output layer, we use only one loss function.

A machine learning engineer can choose between various loss functions. This choice is, fortunately, relatively straight-forward. In his book, Chollet proposes a mapping between machine learning objectives and loss functions that are most prominently used for this type of problem, presented in Table 10 [46].

Machine learning objective	Loss function
Binary (i.e., two-class) classification	Binary crossentropy
Multiclass classification	Categorical crossentropy
Regression	Mean squared error

Table 10: *Mapping between machine learning objectives and appropriate loss functions.*

The algorithms proposed for our work all have an output layer that activates using a *softmax activation function*. This activation function is known to produce a multiclass probability distribution [46]. Hence, our machine learning objective is classification for many classes, and consequently, **CATEGORICAL CROSSENTROPY** is the loss function of our choice. It must however be noted that it might be worthwhile to train the algorithms that we proposed with a wide range of loss functions, which allows experimenting with everything controlled but the loss function (and, strictly speaking, random initialization of the neurons before training).

5.3.3: Optimizer

In the previous section, we discussed the loss function to be used in our experiments, or the quantity that will be minimized. Next, we must choose a *minimizer*, which can be used to actually decrease the loss computed with the loss function. In the machine learning community, such a minimizer is known as an optimizer [46].

Since the machine learning parts of our algorithms are neural networks, our focus for finding an optimizer must be on the deep learning community. Gradient-based optimization methods are most commonly used in this community [46]. To visualize a gradient-based optimization method, one can take by analogy the process of descending a mountain. Suppose that one has reached the top and wants to descend downwards. While doing so, a rational human being always takes the safest yet most efficient path to the valley. This ensures that the human reaches the valley in good health (avoiding an increase in loss).

If we take this analogy back to gradient-based optimization, we can further illustrate our point. During the first epoch of training, the loss is presumably very large due to random initialization of the neurons. At this point, the neural network stands at the top of the *loss mountain*. Next, it should find a way to move downwards most efficiently. This is the task of the optimizer. Gradient-based optimizers compute the gradient, i.e. the path downwards, by computing the minimum of the loss function at hand, which is theoretically possible [46]. The valley, in this case, is the minimum and thus optimum of the initialized loss function used.

5.3.3.1: Stochastic Gradient Descent

One of the most widely optimizers is stochastic gradient descent (SGD) [46]. It works as follows. First, a batch of training samples is drawn randomly – hence *stochastic* – together with the corresponding targets. During training, the initialized network is run on the batch to obtain predictions. For the batch of predictions, the loss value with respect to the original targets can be computed subsequently, followed by a computation of the gradient. Moving the network parameters a little bit into the direction of the gradient, subsequently, is expected to improve network performance. How little is determined by the *learning rate*, which is preconfigured and is a fraction of the gradient. We will discuss learning rates in a subsequent section.

There exist multiple variants of stochastic gradient descent. For example, Chollet discusses mini-batch, true and batch SGD. In mini-batch SGD, the random batch of samples used is small. With true SGD, the batch is of size 1. With batch SGD, the size of the batch equals the size of the training set. Choosing a variant of SGD then correlates with one's tolerance for computational expensiveness, and its related trade-off with accuracy. Batch SGD is more accurate than true SGD, but far more expensive [46]. Choosing a SGD optimizer therefore always introduces a check against, e.g., a business case made for the machine learning project.

Although SGD is one of the widely used optimizers, it comes with various challenges [89]:

1. It is difficult to choose a correct learning rate. If it is too small, the network converges to the optimum loss very slowly – wasting computational resources. If it is too large, the network may not be capable of finding the optimum, overshooting the minimum loss value continuously.
2. Although it is possible to define a scheme with which learning rates can be adjusted, they must be defined in advance, thus, at design-time. At this point, it is unknown how the network will behave during operation. Therefore, its effectiveness is limited by definition.
3. The learning rate defined applies to all neurons, which means that they update their weights in the same way. Mathematically, one can improve convergence speed if one allows every neuron to update its weights all slightly differently.
4. Many initializations of loss functions are non-convex. That is, they do not have one minimum; rather, they have various suboptimal local minima, where the gradients are close to zero. By definition, if SGD gets trapped into such a local minimum, it cannot exit the minimum, which yields a suboptimal final model.

5.3.3.2: Extensions of SGD to Overcome Challenges

With SGD, one preconfigures their learning rate. This is also known as the *default learning rate*. From the previous section, various challenges have emerged with respect to a static learning rate. Fortunately, various extensions to SGD have been proposed over the years to overcome these challenges, summarized in a work by Ruder [89]:

- **Momentum:** Ruder argues that when a ball is pushed down a hill, it gains momentum, and when gradient changes, it will keep moving into the original direction for a while. By applying a similar principle to SGD, oscillation occurring during descent into various directions purely based on the computed gradient can be reduced. This improves the speed with which the network converges.
- **Nesterov accelerated gradient:** if the hill of the momentum scenario is very steep, the ball may overshoot the minimum, oscillating back and forth until gravity finally keeps it at the minimum. Since we know the loss function *and* the momentum-based gradient function, an engineer can effectively look ahead to see whether the steepness of the hill decreases for the computed gradient. This possibly indicates convergence to the minimum. Nesterov accelerated gradient extends simple momentum based gradient in this sense, making the ball smarter, as Ruder argues.
- **Adagrad:** the previous two extensions do not adapt the learning rate with respect to the parameters, i.e. the individual neurons. Since frequently adapted neurons are more likely to change again than less frequently adapted ones, Adagrad takes note of those changes. Frequently changed neurons are assigned a larger learning rate than less frequently ones. It is found that Adagrad greatly improves SGD robustness, and it eliminates the need to manually tune the learning rate.
- **Adadelta:** Adagrad's weakness is that after some time, all learning rates shrink regardless of change frequency. This emerges from Adagrad's mathematical formula, which computes the gradient using a denominator which, among others, contains the sum of squared gradients over time. Since this accumulation and by consequence the denominator grows, the learning rates shrink. At this point, Adagrad can no longer improve network performance. Adadelta extends Adagrad by restricting the sum of gradients to some time window w. Adadelta does not require a default learning rate.
- **RMSprop:** at the same time of Adadelta development, Geoff Hinton published another approach to reduce the vanishing learning rates, named RMSprop. It works similar to Adadelta.
- **Adam:** momentum-based optimization algorithms extend SGD by reducing oscillation around valleys and ravines; adaptive learning rate based algorithms accelerate network improvement at more granularity. So far, those have been two different paths, but Adaptive Moment Estimation (Adam) combines them to take the best of both worlds. Specifically, it combines the best of the Adagrad and RMSprop adaptive learning rate algorithms with a momentum-like operation [90]. Results demonstrate substantial speed improvements with respect to network convergence.
- **AdaMax:** after some time, the gradients computed by Adam can become unstable given its mathematical formulation. This reduces the ability of the network to capture more knowledge, and produces a final network with sub-optimal performance. The authors therefore propose AdaMax which improves gradient stability [90]. It is similar to Adam, but replaces parts of Adam mechanics.
- **Nadam:** although basic momentum improves model performance, the Nesterov accelerated gradient algorithm was proposed to anticipate arriving at optimum loss. Adam and AdaMax utilize a momentum-like operation, but it is very basic. The Nadam optimizer combines Nesterov acceleration with Adam.

5.3.3.3: Choice of an Optimizer for Our Work

From this overview, it becomes apparent that optimization algorithms have improved over time, rendering better performance [89]. However, the wide array of optimization algorithms makes it challenging to choose one for this project, especially since we cannot experiment and visually inspect the loss function with respect to the GPR data beforehand.

Fortunately, Ruder provides heuristics for which optimizer to use in which scenario [89]. He suggests that one likely achieves good results with one of the adaptive learning rate based methods. He suggests that given all the incremental improvements with respect to previous methods, Adam might be the "best overall choice" [89]. Given the trade-off between practical relevance of our work and the number of works on Nadam available at face value, we specifically do not choose Nadam, but it can be an interesting choice in future studies.

Some years have passed since Adam and related optimizers were introduced. Since then, studies have been performed into the performance of optimizers. These yield the insight that although Adam-like optimizers can

greatly improve speed of convergence while still converging the optimum, they also have challenges [91]. Given the time window used by Adam as a result of RMSprop adoption, neural networks fail to converge to their optimum in some cases. The authors of [91] propose AMSGrad, which is an optimizer that also works in those cases, and extend it to AdamNC, another optimization algorithm. The algorithms have not been widely applied in scientific research and practice, but seem to yield promising results for situations of non-convergence [91]. AMSGrad and AdamNC are not available in frameworks such as Keras, but manual implementations injectable into Keras can be used.

Since AMSGrad and AdamNC improve the optimization problem only in a small set of cases, and since Keras implementations are not available by default, the choice for an optimizer with respect to this work is the **ADAM** optimizer. Nevertheless, we keep an open eye towards AMSGrad and AdamNC should ill-performance occur.

5.3.4: Learning Rate

During optimization, after computing the gradient, one needs to decide *how much of the gradient* must be used to change the weights of the parameters. This is the case independent of the method with which the parameters are altered, i.e. all similarly or based on change frequency, thus dependent on the used optimizer. How much the gradient must be changed is known as the learning rate.

From the discussion on a suitable optimizer, we noticed that traditional SGD comes with various challenges [89]. Work so far has focused on speed of convergence and overcoming local minima: whereas momentum-based optimizers reduce oscillation during descent, adaptive parameter-based optimizers focus on changing the learning rate per parameter. By consequence, optimizers are capable of handling non-convex loss functions, can assign individual learning rates and decide on learning rates at run-time rather than design-time.

However, the main question is not resolved by the optimizers themselves: which learning rate is correct? We argue from experience with practice that this is dependent on the training stage. That is, during the early stages of training, the network learns very rapidly: after its random initialization, it is capable of capturing new knowledge with an impressive speed. For this stage, it is therefore worthwhile to choose a high learning rate, i.e. to change the individual parameters substantially based on the computed gradient. However, this is no longer relevant when one enters the final stages of training, since the loss function converges to its (preferably global) minimum. When we maintain a large learning rate, we risk overshooting the minimum time after time. We must therefore choose a low learning rate in the final stages of training our model. Since we cannot have both, it becomes clear that it is impractical to pick a fixed learning rate at design-time when training a deep neural network.

5.3.4.1: Learning Rate Decay

There exist multiple methods to face this challenge. Naïvely, one could choose a learning rate that lies somewhere in between a large and a small learning rate. However, better methods exist, which ensure that the learning rate *decays* with time.

Some of them are embedded into the model at design-time: for example, by dividing the learning rate by the iteration (i.e., epoch) at hand, one can decrease the learning rate in a stepped way. Those decay-based learning rates are built into most optimizers with modern deep learning frameworks; this is also the case for the optimizer of our choice, i.e. the Adam optimizer.

5.3.4.2: Learning Rate Range Test

Even when the learning rate decays with time, it is uncertain when one should stop the training process. In order to make it more straight-forward to find the correct default learning rate, Smith has proposed the so-called Learning Rate Range Test (LR Range Test) [92]. It is relatively straight-forward: in a test run, one starts with a very small learning rate, for which one runs the model and computes the loss on the validation data. One does this iteratively, while increasing the learning rate exponentially in parallel. One can then plot their findings into a diagram representing loss at the y axis and the learning rate at the x axis. The x value representing the lowest y value, i.e. the lowest loss, represents the optimal learning rate for the training data.

Especially when this test is used with a subset of the training data, it can be a relatively quick way to accurately assess the optimum learning rate for the data at hand. It must be noted that this subset must be drawn randomly. For Keras, an extension named **KERAS_LR_FINDER** is available. It is open source software based on the LR Range Test available within the fast.ai deep learning framework.

5.3.4.3: Cyclical Learning Rates

In the same work, Smith also proposed a radically different way of improving the learning rate, named Cyclical Learning Rates (CLR) [92]. Whereas traditional techniques aim to improve model convergence by applying learning rate decay, a CLR does not do this. Rather, the learning rate oscillates back and forth between higher and lower learning rates. Experiments suggest that CLR achieves similar results while substantially reducing the time to convergence. However, today, the extent to which those results can be generalized to any deep learning scenario is unclear, whereas decaying learning rates are commonly applied.

5.3.4.4: Choice of a Learning Rate for Our Work

Choosing a fixed learning rate at design-time would be a very naïve approach. Therefore, we aim to use one of the improvements discussed before to inform our choice for a learning rate. With respect to cyclical learning rates, we argue that its generalizability is too uncertain for inclusion in our work. Specifically, our work is to identify novel algorithms for GPR-based utility characterization, rather than a study into the applicability of CLRs for GPR imagery. We do note, however, that this application spawns interesting hypotheses for additional studies.

In our work, before training the models with maximum computational resources and with the full data set, we use **KERAS_LR_FINDER** to perform a LR Range Test. This allows us to find the maximum learning rate with which the model does not overfit. We perform this test for all three algorithms and per algorithm choose this learning rate as the base learning rate.

We subsequently apply a linear decay rate, which is defined as follows:

$$Learning\ rate\ decay = \frac{Default\ learning\ rate}{Number\ of\ epochs}$$

5.3.5: Regularization

In his textbook on deep learning, Chollet argues that there often exist multiple ways to capture the underlying complexities in a data set [46]. This can be interpreted in terms of tweaking the *hyperparameters*, i.e. options such as the learning rate, but also in terms of letting the model find *parameters* (i.e. neural weights) through optimization. With respect to the latter, Chollet argues, simpler models are often preferred over complex ones, due to the fact that those result in increased odds for overfitting. This is specifically the case when one makes use of relatively small datasets. With simpler models, Chollet means models with “a smaller entropy” in their distribution of parameter values. By consequence, this means that parameter values (i.e. neural weights) must be relatively small for a model to be relatively simple.

If the aim of our work is to reduce the odds of overfitting, we too must reduce the complexity of our models. Fortunately, regularization techniques are designed precisely for this purpose. Many have been proposed over the years, some of which are de facto standard ones today [46]. Specifically, in this section, we cover L1 regularization, L2 regularization and – given the fact that the machine learning aspect of our models makes use of CNNs – dropout regularization and data augmentation. We also include a discussion on the regularization effect of Batch Normalization. We subsequently argue about which regularization techniques are used when training our algorithms.

5.3.5.1: L1, L2 and Elastic Net Regularization

A class of regularization techniques borrowed from statistics includes L1 and L2 Regularization [46]. They are similar in the sense that they penalize large weights, and by consequence must thus work in relation to optimizing the model by adapting weights. As we recall from the previous sections, optimization is done with respect to the cost or loss function. Consequently, those regularization techniques must be added to this optimization process. Indeed, when one adds L1 or L2 regularization techniques, one attempts to minify the *combination of the loss function and the regularizer* rather than the cost function alone. The regularizer, here, attempts to reduce large weights to be generated through optimizing the cost function.

However, L1 and L2 regularization have differences. When computing the new weight vectors, L1 Regularization penalizes the absolute value of the weight, i.e. $|weight|$ [93]. This ensures that extreme values, both positive and negative, are penalized. By consequence, its derivative is a constant scalar value, which yields that L1 Regularization subtracts this value every time during optimization. In practice, since this process

eventually drives weights to zero – removing corresponding features from the model – L1 Regularization induces sparsity [93]. It therefore effectively serves as an internal feature selector and can be used as such.

L2 Regularization, on the other hand, penalizes the square of the weight, i.e. $weight^2$ [93]. It is also called weight decay, since it impacts less extreme weights substantially less than ones that share large values [46]. Its derivative is therefore equal to $2 * weight$. L2 Regularization can therefore be intuitively interpreted as reducing the weights with some percentage every time [93]. Since, however, percentages are relative with respect to the value to be reduced, this latter value can never reach zero. L2 Regularization is therefore not capable of making its outputs sparse and thus serving as a feature selector, but it is generally more capable of interpreting complex data patterns [94]. It can be used when models need to be regularized, but when e.g. all input values are relevant.

It is also possible to combine L1 and L2 Regularization. When they are applied at the same time, one is said to perform Elastic Net Regularization [95].

5.3.5.2: Dropout Regularization

According to Chollet, Dropout Regularization is "one of the most effective and most commonly used regularization techniques for neural networks" [46]. It works as follows. When it is applied to the outputs of a layer, it randomly drops out a number of outputs by setting them to zero. Since the model becomes partially blind to the data, it is therefore highly effective in reducing the odds of overfitting. By consequence, it is one of the most substantially used regularization techniques used today.

Unfortunately, combining Batch Normalization with Dropout Regularization is found to introduce instability in the performance of a model [96]. Therefore, it is questionable whether they should be combined in practice.

5.3.5.3: The Impact of Batch Normalization on Regularization

Besides its impact on Dropout Regularization, Batch Normalization also impacts regularization itself. Strictly speaking, Batch Normalization is no regularization technique – rather, it attempts to normalize the trainable parameters of a neural network in order to increase its speed of convergence. However, since the result of this normalization is a real number which is continuous and by consequence has a possibly infinite decimal representation, a *very small* difference with respect to the original parameters is introduced when applying Batch Normalization. This difference, in effect, can serve as a reducer for overfitting, and by consequence share similar behaviour with true regularization techniques.

5.3.5.4: Data Augmentation

In the case of CNNs, when the input data set is small, the odds of overfitting increase. One can however increase the size of the data set by applying Data Augmentation [46]. It works as follows. For every input (e.g., an image), various random transformations are performed on the input, and subsequently added to the input data set. For example, in the case of this image, it can be stretched, rotated, cropped, et cetera, after which those new images are added to the training set. Data Augmentation has shown to reduce the odds of overfitting by increasing the input data set.

5.3.5.5: Regularization for GPR Imagery

The algorithms that we proposed utilize Batch Normalization to ensure faster convergence. Given the challenges of combining this technique with Dropout, we therefore do not apply Dropout in our work. We do neither apply data augmentation, which is due to the unique nature of the waveforms received by the antennas. We must discuss this with respect to our algorithms: the two A-scan based ones and the B-scan based one.

For example, for the two A-scan based algorithms (i.e. working with histograms and DCT as feature selectors) there is no benefit to gain from rotating the waveforms, since their *numerical* differences are of primary interest, not their direction. Similarly, cutting off the waveform in various ways reduces essential aspects of the underground object being represented by the received wave. Data augmentation is therefore expected to reduce model performance in our idiosyncratic case, instead of improving it.

Similarly, it cannot be applied to the B-scan based algorithm, which works with cropping the heart of the B-scan. Due to the geophysical characteristics of a GPR device, hyperbolae will always be represented with their apex towards the north. We therefore expect Data Augmentation, specifically rotation, to reduce performance. We therefore also omit data augmentation techniques from the B-scan based algorithm.

With respect to L1 and L2 Regularization, practice shows that their effects are often highly situation-dependent. L1 Regularization itself is not required for the A-scan algorithms, since the feature extractors already substantially reduce the dimensionality of the feature space. Additionally, L2 Regularization is not expected to benefit the *convolutional* layers of our A-scan algorithms, since the kernels already result in an average representation of the input space and batch normalization is applied. We are however interested in applying L2 Regularization on the densely-connected layers which yield the eventual multiclass probability distribution, since those do not represent an average whatsoever. Consequently, we initially train our algorithms with L2 Regularization. If the performance of our training is insufficient, we will subsequently attempt to train without L2 Regularization. In a third possible training attempt, Elastic Net Regularization can be attempted. Primarily, however, we expect that L2 Regularization will reduce the odds of overfitting.

5.3.6: Other Parameters

There exist a number of other parameters which do not require a substantial discussion, but which need to be highlighted regardless. Below, we provide a discussion for these parameters, for the sake of completeness of our validation design. The parameters are as follows:

- **Batch size:** in every epoch, data is propagated forward in batches. Preferably, the batch size equals the size of the training data, but this is very memory intensive. To reduce the memory footprint of a neural network during training, smaller batches are used. The consequence of the reduced memory requirements is a less accurate estimation of the gradient during optimization, for the reason that fewer samples are used for estimating the loss in the particular batch. It is therefore wise to balance between gradient accuracy and memory requirements. We therefore choose a batch size of 70 in our work. This batch size splits our training set in 11 batches. We expect this choice to meet both criteria.
- The **number of epochs** describes how many times the training process should propagate forward the batches of training data, subsequently computing the error in a backwards fashion. This number can approximate infinity, but in that case inevitably yields a model that is overfit to the training data. A machine learning engineer however cannot determine a priori what number of epochs fits their data best. By consequence, this number must be determined experimentally. Fortunately, Keras is equipped with callbacks that allow the training process to be manipulated automatically should certain conditions occur. Two of those are `EarlyStopping` and `ModelCheckpoint`, which remove the necessity of determining the number of epochs a priori. By configuring an extremely large number of epochs and combining those with the two callbacks, the training process stops early when certain conditions are met. The best performing model is saved at that point. Specifically, we choose to set the number of epochs at 200000, stopping early when the validation loss no longer improves for 30 epochs: `EarlyStopping(monitor='val_loss', mode='min', patience=30)`. The model checkpoint callback saves the best performing epoch in terms of validation loss: `ModelCheckpoint(filepath, monitor='val_loss', save_best_only=True, mode='min')`.

5.4: Validation Metrics

Besides the loss, one can also use other metrics during validation [46]. Compared to loss values, those metrics generally give a more intuitive view of the performance of a model. Metrics that are commonly used are accuracy, the area under the receiver operating characteristic curve (ROC AUC), precision and recall [46]. It is even possible to use custom validation metrics. Chollet argues that for well-balanced classification problems, accuracy and ROC are common metrics [46]. The GPR imagery in our work is relatively well-balanced, since the number of classes is distributed relatively equally. In our work, we therefore use two measures of performance. First, we use categorical crossentropy for computing loss, both for validation and testing. In parallel, we use accuracy in order to also utilize a more intuitive validation metric.

5.5: Complete Overview of Validation Strategy

In the sections above, we discussed our validation strategy in a rather detailed way. If we wish to understand it as a whole, we must take a holistic approach to this strategy. Table 11 provides this overview. It specifies the design choices altogether for targets $T = \{ Concrete, HDPE, Iron, PEC, Root, Stoneware \}$.

	Histogram-based CNN	DCT-based CNN	B-scan based CNN
Feature extractor	A-scan based histogram with bins of size σ/10, interval [-5σ, 5σ].	A-scan based Discrete Cosine Transform with only 14 first coefficients for predictive and generalization power.	Windows extracted from the B-scan, centred around the hyperbola apex both horizontally.
Feature vector shape	(101,)	(14,)	(51,1024)
CNN Design	Conv1D > Conv1D > Dense > Dense > (Softmax).	Conv1D > Conv1D > Dense > Dense > (Softmax).	Conv2D > Conv2D > Conv2D > Dense > Dense > (Softmax).
CNN Output	Multiclass probability distribution over target classes.	Multiclass probability distribution over target classes.	Multiclass probability distribution over target classes.
Train/Test Split	K-fold cross-validation with K=10, given the relatively small size of our input data.	K-fold cross-validation with K=10, given the relatively small size of our input data.	K-fold cross-validation with K=10, given the relatively small size of our input data.
Train/Validation Split	80/20	80/20	80/20
Total data set size	770	770	770
Batch size	70	70	70
Number of epochs	Until deterioration of validation loss determined with EarlyStopping callback, patience = 30.	Until deterioration of validation loss determined with EarlyStopping callback, patience = 30.	Until deterioration of validation loss determined with EarlyStopping callback, patience = 30.
Weight Initialization	He uniform initialization, since ReLu activation functions are used and He then performs better than Xavier.	He uniform initialization, since ReLu activation functions are used and He then performs better than Xavier.	He uniform initialization, since ReLu activation functions are used and He then performs better than Xavier.
Loss Function and Validation Metrics	Categorical crossentropy, since we generate a multiclass probability distribution. Accuracy is used in parallel as a more intuitive validation metric.	Categorical crossentropy, since we generate a multiclass probability distribution. Accuracy is used in parallel as a more intuitive validation metric.	Categorical crossentropy, since we generate a multiclass probability distribution. Accuracy is used in parallel as a more intuitive validation metric.
Optimizer	Adam, since it resides at the cutting edge between reducing optimization challenges and being thoroughly tested in practice.	Adam, since it resides at the cutting edge between reducing optimization challenges and being thoroughly tested in practice.	Adam, since it resides at the cutting edge between reducing optimization challenges and being thoroughly tested in practice.
Learning Rate	Linear (epoch-based) learning rate decay starting at default value predetermined through Learning Rate Range Test.	Linear (epoch-based) learning rate decay starting at default value predetermined through Learning Rate Range Test.	Linear (epoch-based) learning rate decay starting at default value predetermined through Learning Rate Range Test.
Regularization	L2 Regularization on Dense layers, since they are not protected from normalization whatsoever.	L2 Regularization on Dense layers, since they are not protected from normalization whatsoever.	L2 Regularization on Dense layers, since they are not protected from normalization whatsoever.
Final model	Model saved and evaluated is the best performing epoch in terms of validation loss, determined with ModelCheckpoint based on EarlyStopping / number of epochs driven training process.	Model saved and evaluated is the best performing epoch in terms of validation loss, determined with ModelCheckpoint based on EarlyStopping / number of epochs driven training process.	Model saved and evaluated is the best performing epoch in terms of validation loss, determined with ModelCheckpoint based on EarlyStopping / number of epochs driven training process.

Table 11: *Validation strategy for our novel algorithms.*

6: Validation Results

In the previous chapters, we outlined three new machine learning driven algorithms for utility material classification and a strategy to validate them. This chapter presents the results of performing the validation activities and the results of minor modifications to the algorithms based on those initial results. Section 6.1 presents the results of our global ML data pipeline. For example, it allows the reader to visualize the effects of e.g. time-varying gain. Section 6.2 focuses on the histogram based algorithm. Section 6.3 presents the results of the DCT based algorithm. Finally, section 6.4 illustrates how the B-scan window based algorithm performs.

6.1: Global Data Pre-processing Results

This section presents the results of global data pre-processing. In section 6.1.1, we present the results of median trace-based ground bounce removal. Section 6.1.2 presents how time-varying gain impacts the signal. Finally, in section 6.1.3, feature-wise normalization results are presented.

6.1.1: Ground Bounce Removal

In section 4.2 we discussed how ground bounce is removed from our radar imagery, namely using median based filtering. Image 13 presents the results for our approach. It contains three waveforms. The upper waveform is a randomly selected simulated waveform from our entire set of simulations that was received by the GPR's emulated antenna. Clearly, one observes a strong reflection in the early part of the time domain of the waveform, representing the ground bounce. One can additionally see that it is much stronger than the signals that are relevant from an information point of view. This is due to the strength of the ground bounce, combined with the fact that the signal had not yet been gained.

The middle waveform is the same waveform as the upper one, except for the fact that median-based ground bounce removal was applied. Note that this ground bounce removal is highly effective in our case given the relatively disturbance-free ground bounce. The main signal is visible clearly and ground bounce is removed.

One exception to the success with ground bounce removal is the scenario in which the object is buried very shallowly or in soil types which result in very large wave frequencies. In those cases, the ground bounce and signal overlap to some degree. This is visible in the lower image, which was taken from a different A-scan. It cannot be stated with certainty that all ground bounce is removed. Nevertheless, it seems that the majority is no longer present. It remains unclear to which extent this impacts machine learning performance.

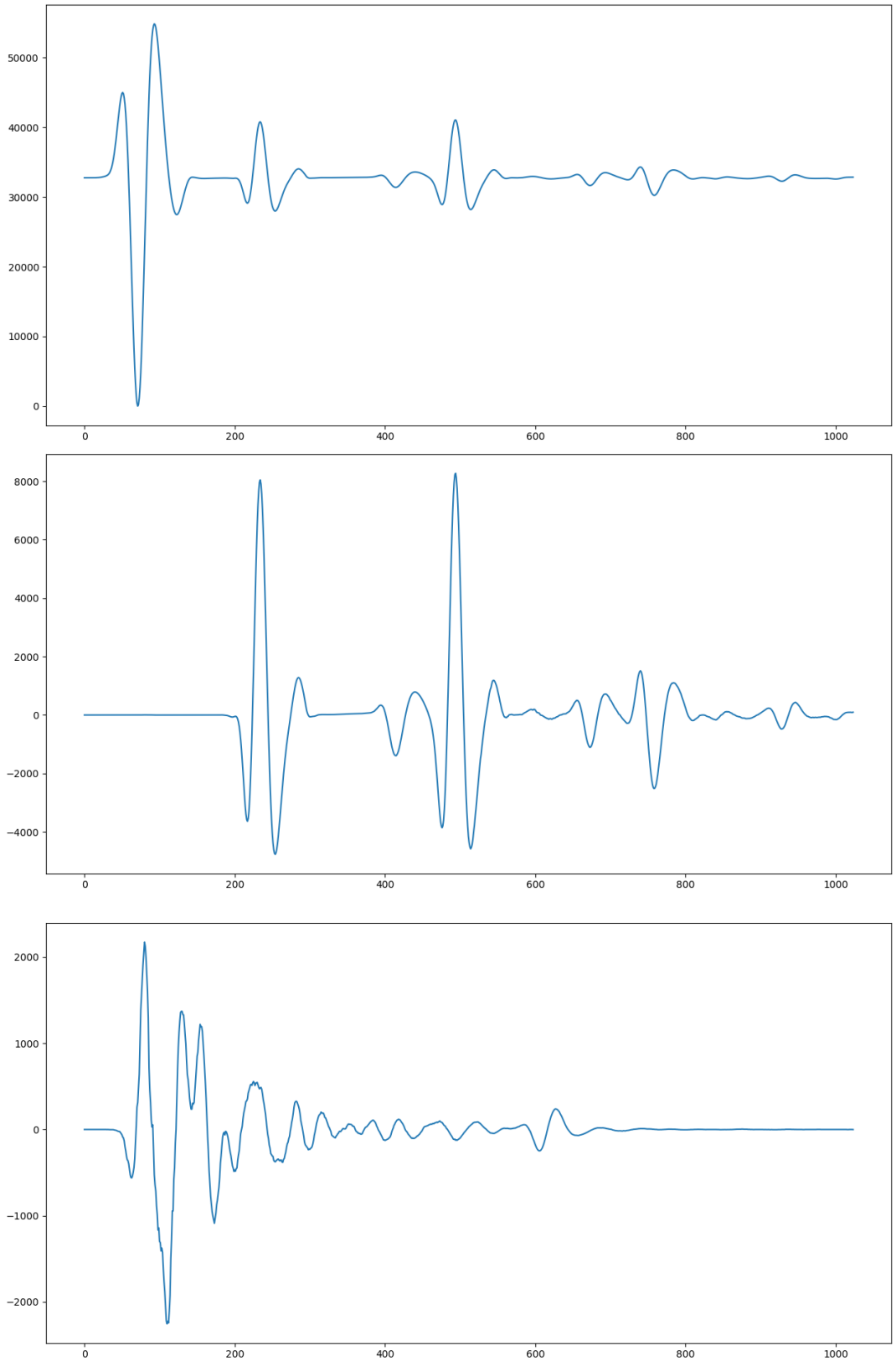


Image 13: *Unprocessed A-scan (above), A-scan with ground bounce removed without gain applied (middle), A-scan with ground bounce from noisy early signal (down).*

6.1.2: Time-varying Gain

A signal which propagates into the underground attenuates with respect to the time domain. That is, it becomes weaker over time, and by consequence so do the echoes. As a result, a GPR analyst must attempt to normalize this effect. Normalization can be attempted using time-varying gain [61]. In this work, exponential gain is applied to the A-scan in order to reduce signal attenuation. Image 14 presents the results, in which time-varying gain was applied to the bottom A-scan.

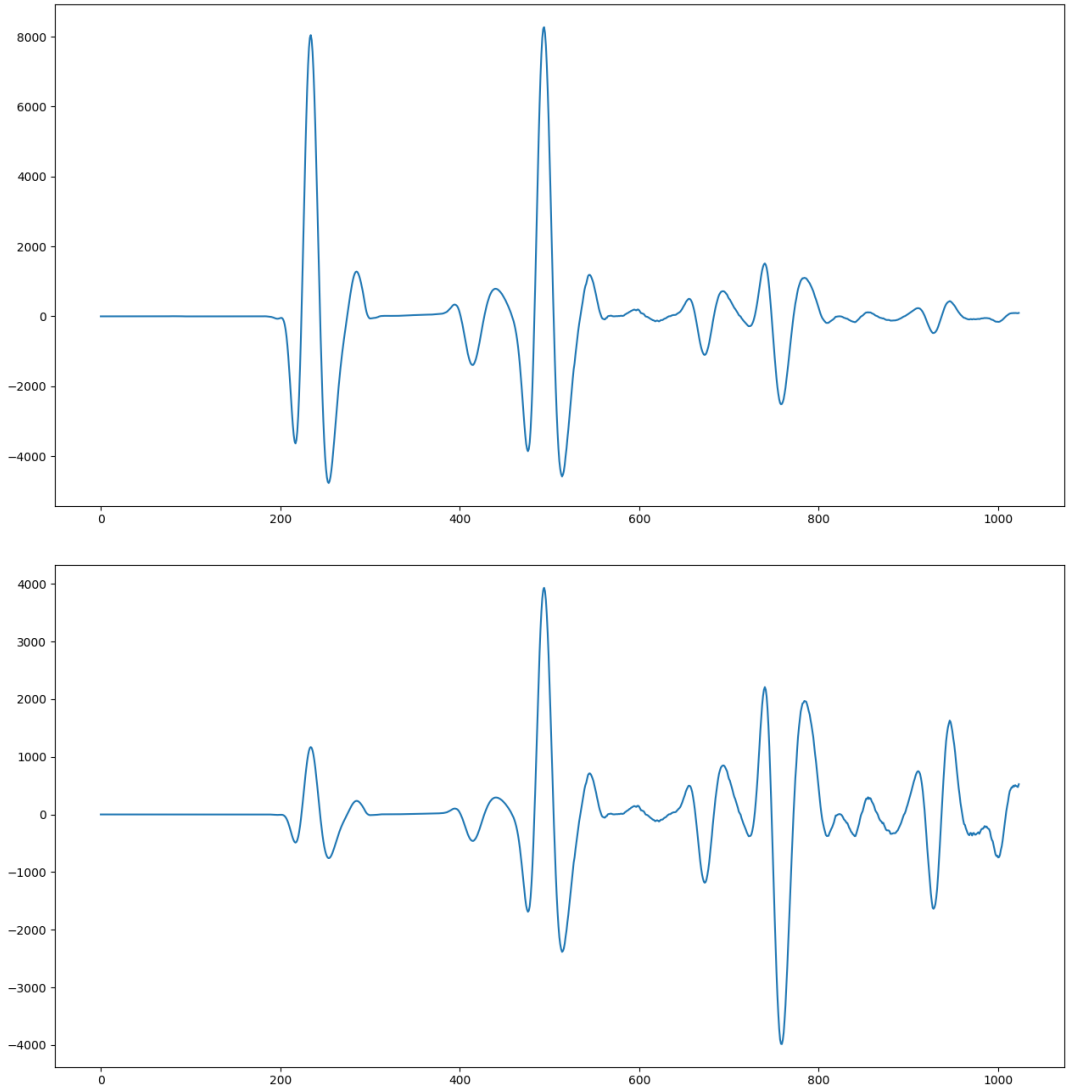


Image 14: *A-scan signal with ground bounce removed without gain (above); with gain (below).*

6.1.3: Feature-wise Normalization

Neural networks are generally better capable of handling feature vectors with relatively little variance over ones in which variance between features is high. We included feature-wise normalization in our validation strategy. That is, for every feature, we subtract the mean of the feature vector and divide by the standard deviation. This preserves the shape of the signal, but changes the unit of measurement into the number of standard deviations. Image 15 presents the results of normalization for the gained A-scan from Image 14.

Numerically, the need for such normalization becomes clear too. From the 1024 samples in the gained A-scan from Image 14, many reside in the [-100,100] range, but others go up and down to -2000 and +3000. More precisely, the variance for the A-scan above is 137618.14; the standard deviation is 370.96.

In the normalized A-scan presented in Image 15, variance was reduced to 0.999. What is best is that the shape itself has not changed. Instead, the numerical values of the y-axis are reduced, as is visible in Image 15. This way, the neural networks can still accurately detect the shape of the waveform whereas they are not hampered by poorly presented feature vectors (i.e., extremely large ones in terms of variance).

Next, we will discuss feature extraction, further pre-processing, model training and model performance per algorithm.

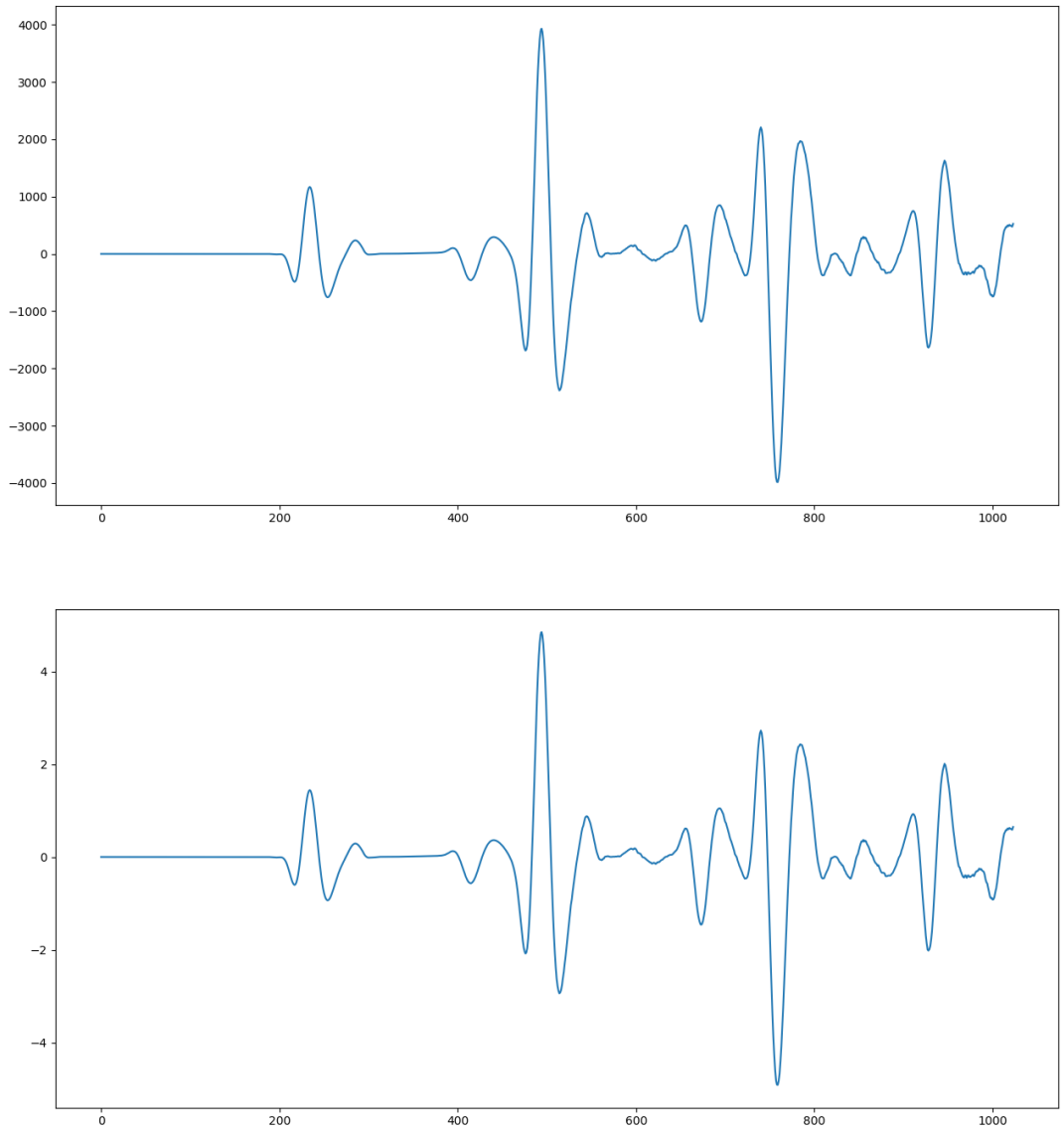


Image 15: *Gained A-scan without feature-wise normalization (above); with normalization (below).*

6.2: Histogram Based CNN Algorithm

In the previous section, using a raw A-scan, we presented the results of removing ground bounce, applying time-varying gain and reducing variance using feature-wise normalization. The pre-processed A-scan is now ready for further processing. For the histogram based CNN algorithm, this means extracting a feature histogram. We exploited the new unit (degrees of standard deviation) which is the result of feature-wise normalization and used histogram bins of size σ/10 on the interval [-5σ, 5σ], resulting in a total number of 101 bins.

Histograms were extracted for every simulation. Image 16 presents an exemplary visualization of the histogram feature vector extracted from the normalized A-scan presented in Image 15.

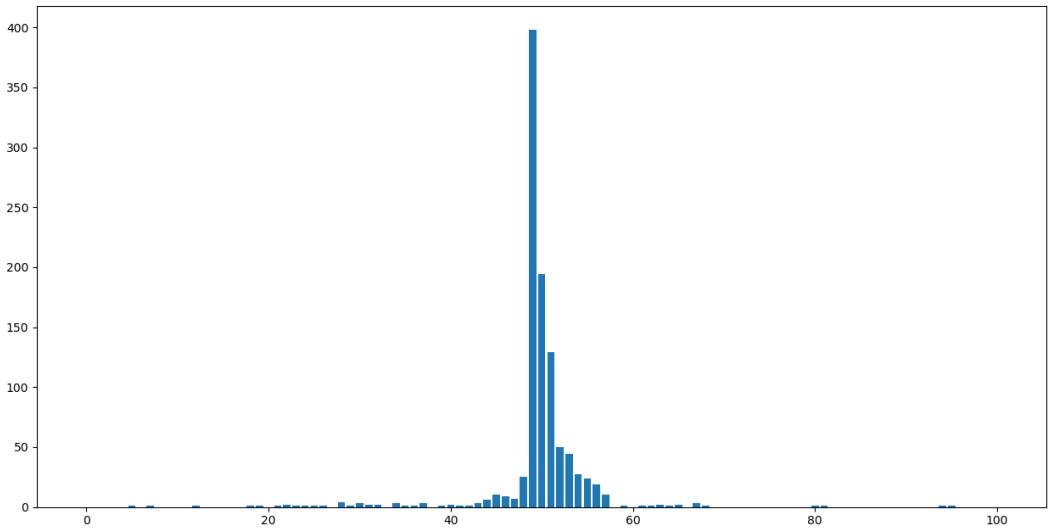


Image 16: *Visualization of extracted histogram feature vector.*

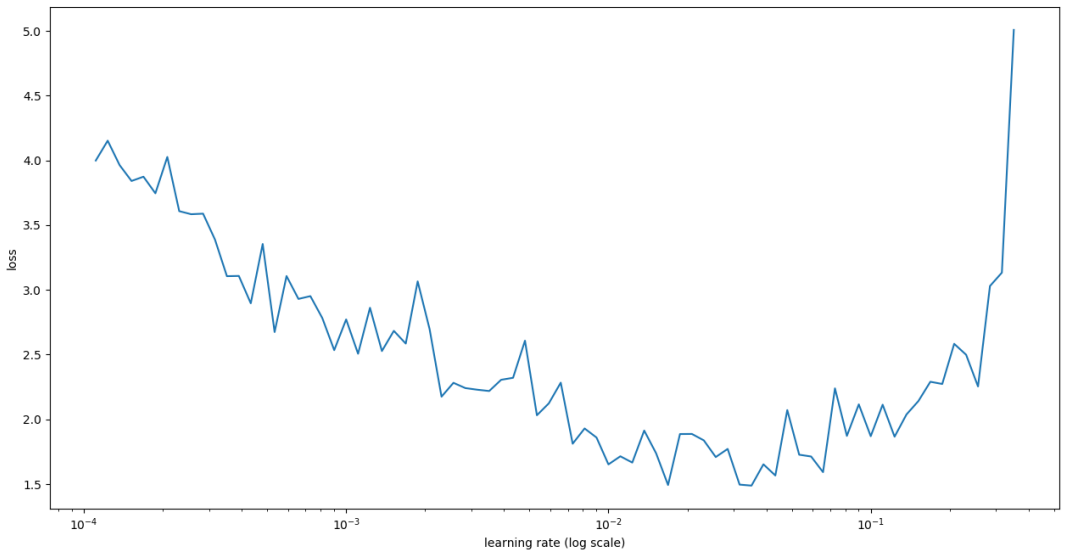


Image 17: *Plot of the losses found using Learning Range Rate test for histogram algorithm.*

6.2.1: Learning Rate Range Test

The next step we performed, in line with our validation strategy, is empirically finding the most optimal learning rate given our data. By using the **KERAS_LR_FINDER** library, we found that for the histogram features extracted from our simulations the most optimal learning rate is approximately $10^{-1.93}$. This result was confirmed by running the test multiple times. Image 17 presents the results of the Learning Rate Range Test for the histogram based CNN algorithm.

6.2.2: Algorithm Performance

The next step involved training the model based on the validation strategy outlined before. Given this strategy, 554 samples were used for training, 139 for validation and 77 for testing. Results are presented in Table 12.

Fold	Categorical crossentropy test loss	Test accuracy
1	1.0713	70.13%
2	1.2497	59.74%
3	1.3461	59.74%
4	1.6429	50.65%
5	1.2184	64.94%
6	1.1783	64.94%
7	1.4970	62.34%
8	1.2853	63.64%
9	1.2793	58.44%
10	1.3935	58.44%
Average	**1.3162**	**61.30%**

Table 12: *Initial validation results for histogram based CNN algorithm.*

6.2.3: Variations to the Initial Algorithm

The average accuracy of training with the initially proposed algorithm yields an average accuracy of approximately 61% over ten folds. It is a no-brainer to state that the model must be improved further. We do primarily aim to discuss our findings in the next chapter, but wish to expand the set of findings based on early hypotheses conjectured immediately after training, before we proceed with our actual discussion. Those hypotheses are as follows:

1. The amount of data used is insufficient for the algorithms to generalize successfully. This hypothesis primarily emerges from the current data pipeline set-up: for every soil type, every unique object only appears once. That is, once the train/validation/test split is made, an object can be present in only one of the three categories. This may disallow the model to generalize.
2. In the GPR community, soil type is known to introduce substantial variance to A-scans of similar objects, due to their differing dielectric constant and, consequently, the speed with which a wave is propagated. We hypothesize that when keeping everything else constant, soil type may result in different histograms for the same object. This might negatively impact model generalization.
3. Our model currently only regards ‘material type’ as a target value and ignores contents. Consequently, a power cable isolated with HDPE is considered to be ‘HDPE’, but the same is true for a pipeline made of HDPE while containing water. We hypothesize that the contents of an underground object yield different histograms.
4. Visual inspection of the speed of convergence of the ML model yields the observation that it converges relatively quickly (in between 75-250 epochs out of 100.000 epochs allowed in our validation design). Convergence of the model is dependent on the default learning rate and its decay. As discussed in section 5.3.4, if the learning rate decays too slowly, we risk overshooting the global minimum time after time; if it decays too quickly, model convergence is slow. With respect to convergence observed, we are concerned and consequently hypothesize that there remains insufficient balance between speed of convergence and finding the optimum loss.

5. We hypothesize that various parameters of our model (e.g., gain used in pre-processing, activation functions used, number of layers, optimizer used, regularization used, and so on) impact the model negatively.
6. We hypothesize based on visual inspection of some pre-processed A-scans that the feature-wise normalized feature vector is clipped, i.e., that the amplitudes of the waveform recorded sometimes exceed -5σ and 5σ. We are concerned that this effect may impact model performance.

We converted those hypotheses into a workflow for validating variations to our initial model. This workflow is visualized and is presented in Image 18. First, since our largest concern is that the number of simulations used for our training procedure is insufficient, we expanded the training set and retrained the model. It was expanded by regenerating the same objects as simulated before, with random variations in soil type and random variations in the degree of noise. In total, 1656 new simulations are included, expanding the total dataset from 770 to 2426 simulations.

Under the condition that increasing the data set improves the model, our workflow branches in order to validate the remaining hypotheses. First, the effects of soil type were studied by training the histogram based algorithm using one particular soil type, namely the homogeneous one (with Epsilon = 9). Second, the effects of separating the material of the object (e.g., HDPE) together with its contents (e.g., water) into separate target values were studied. Third, we substantially increased learning rate decay (our initial findings yield that a balance occurs when we increase the decay by about 175.000 times) and studied its effects. Fourth, we studied model variations by altering model structure and hyperparameters. Fifth, we expanded our feature extractor to include an interval of [-10σ, 10σ], effectively doubling the shape of our feature vector to (201,). Sixth, we trained various variations to the gain applied to the signals during pre-processing.

Subsequently, we combined well-performing individual alterations and studied model performance. Based on those observations, we interpret our findings in chapter 8, and subsequently present the elements that yield a maximum performing model based on histogram features. The next section presents the performance of the proposed variations.

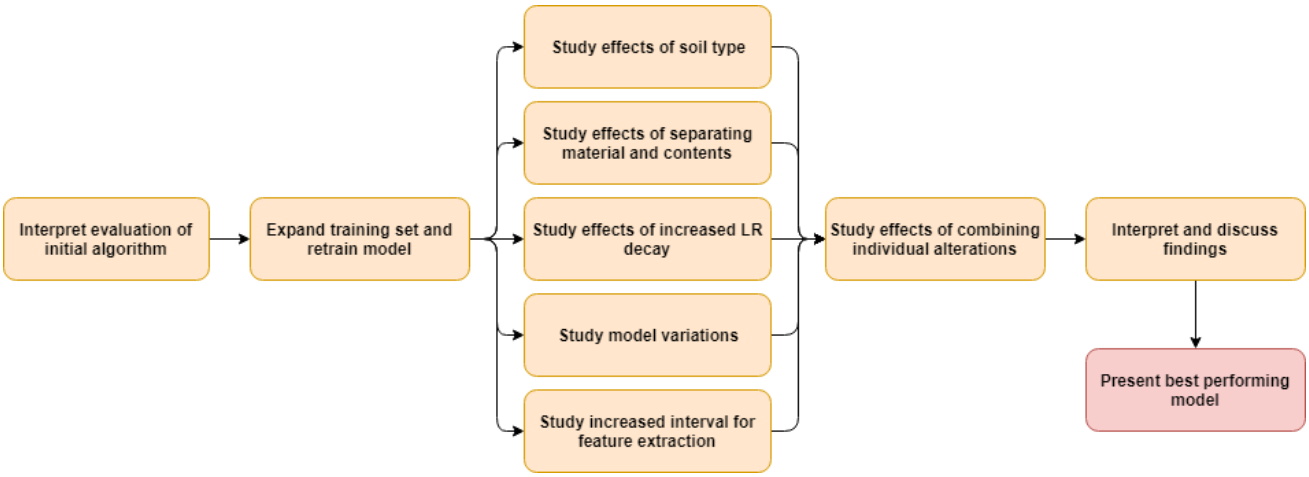


Image 18: *Variations to the histogram based algorithm.*

6.2.4: Performance of Variations

Table 13 presents the performance of the variations proposed in the previous section. Generally, the variations do not improve model performance. Rather, the reported accuracies and loss values are similar to the results reported during initial model training. Since based on those results it is reasonable to assume that its combinations do neither improve the model, we did not proceed with combining individual variations. We will explain this observation in section 8.1.

One interesting observation can be made about training the model for one soil type. Rather unfortunately, this experiment turned out to be infeasible given available training data. More specifically, the metadata available

for the data with which the original dataset was extended did not include the soil type. Rather, it only contained the raw soil parameters, which are varied randomly – and by consequence, no single epsilon value can be derived for those rows. Since training the model for epsilon = 9 on the original data set will by definition yield extreme overfitting (since only a subset of the 770 would be usable), the experiment was not performed.

Variation	LR Range Test Result	Averages	Extremes
Individual variations			
Expanding training set to 2426 simulations.	$10^{-1.95}$	Loss: 1.3830 Accuracy: 54.54%	Max: 1.5077 / 60.08% Min: 1.3041 / 48.14%
Training model on one soil type (homogeneous soil, epsilon = 9).	Could not be performed (explanation above).		
Separated materials and contents in targets.	$10^{-1.91}$	Loss: 2.0650 Accuracy: 32.89%	Max: 2.2012 / 39.91% Min: 1.9369 / 29.33%
Increased LR decay by 175.000.	$10^{-1.65}$	Loss: 1.3401 Accuracy: 55.35%	Max: 1.4483 / 62.14% Min: 1.2289 / 49.17%
Increased feature extraction interval to [-10σ, 10σ].	$10^{-2.00}$	Loss: 1.3741 Accuracy: 52.92%	Max: 1.4779 / 56.79% Min: 1.3020 / 49.58%
Studying effects of varying batch size			
Swish instead of ReLU activation function on expanded data set.	$10^{-1.60}$	Loss: 1.414 Accuracy: 52.63%	Max: 1.4839 / 58.44% Min: 1.3191 / 47.11%
Leaky ReLU instead of ReLU activation function on expanded data set; α = 0.1.	$10^{-1.70}$	Loss: 1.3175 Accuracy: 54.12%	Max: 1.3885 / 60.08% Min: 1.2617 / 50.62%
Tanh instead of ReLU activation function; Glorot uniform instead of He uniform initialization, on expanded set.	$10^{-1.96}$	Loss: 1.573 Accuracy: 51.69%	Max: 1.6402 / 54.73% Min: 1.4793 / 47.93%
Studying effects of varying batch size			
Batch size = 5	$10^{-2.69}$	Loss: 1.3740 Accuracy: 53.75%	Max: 1.4867 / 58.44% Min: 1.3264 / 49.59%
Batch size = 15	$10^{-1.20}$	Loss: 1.3812 Accuracy: 48.43%	Max: 1.4459 / 53.91% Min: 1.3064 / 48.43%
Batch size = 25	$10^{-1.00}$	Loss: 1.3907 Accuracy: 48.23%	Max: 1.4434 / 53.91% Min: 1.3409 / 45.45%
Batch size = 35	$10^{-1.25}$	Loss: 1.3493 Accuracy: 49.63%	Max: 1.3955 / 53.91% Min: 1.2967 / 46.28%
Batch size = 50	$10^{-1.95}$	Loss: 1.3690 Accuracy: 53.50%	Max: 1.5308 / 57.20% Min: 1.2719 / 47.33%
Batch size = 90	$10^{-1.25}$	Loss: 1.3318 Accuracy: 51.40%	Max: 1.3865 / 56.38% Min: 1.2762 / 45.04%
Batch size = 115	$10^{-1.85}$	Loss: 1.3799 Accuracy: 54.49%	Max: 1.4708 / 58.02% Min: 1.2887 / 50.83%
Batch size = 140	$10^{-1.90}$	Loss: 1.3986 Accuracy: 52.6%	Max: 1.5177 / 58.85% Min: 1.2863 / 48.76%
Studying effects of varying gain			
No gain applied in pre-processing, re-introducing signal attenuation.	$10^{-2.25}$	Loss: 1.3697 Accuracy: 51.03%	Max: 1.5092 / 53.91% Min: 1.2837 / 47.93%
Strong exponential gain applied, introducing inverted signal attenuation.	$10^{-1.73}$	Loss: 1.3191 Accuracy: 54.45%	Max: 1.434 / 60.49% Min: 1.2095 / 46.69%
Linear gain applied instead of energy / exponential gain.	$10^{-1.72}$	Loss: 1.3212 Accuracy: 54.82%	Max: 1.3915 / 58.02% Min: 1.2158 / 49.59%

Table 13: *Validation results for variations applied to histogram based CNN algorithm.*

In section 6.3.3, we proposed to “study model variations”, that is, investigating the change in model performance when its structure and/or hyperparameters are changed. Those variations were conceived during the application of variations reported in the previous section, and are by consequence discussed next.

6.2.4.1: Swish instead of ReLU for activation

Although the ReLU activation function is one of the de facto standard activation functions, it comes with certain drawbacks [97]. Its method of activating, in which the activation is 0 for all inputs <= 0, benefits sparsity but may also result in the death of the model. This may occur when it is initialized wrongly and optimization in the early phases of training yields large weight swings. Additionally, small values <= 0 might still contain relevant information with respect to the data, which are filtered out with ReLU, possibly decreasing the predictive power of a model.

Swish is a relatively new activation function created by a team from Google Brain, and attempts to overcome ReLU’s bottlenecks [97]. It activates as $f(x) = x * sigmoid(x)$ and is visualized in Image 19. As one can observe, it resembles the ReLU function: for larger negative values, its output is 0; for larger positive values, it activates as $f(x) = x$. It does however activate differently when its input $x \approx 0$. Around this point, it activates with small values close to but not zero. This way, the sparsity benefit of ReLU is retained whereas a model can possibly better capture information. Empirical tests have demonstrated significant performance improvements compared to ReLU on large-scale datasets [97]. However, for the histogram based algorithm, Swish results in slightly deteriorated model performance, with a loss of 1.414 and accuracy of 52.63%. We will attempt to explain this performance and draw conclusions for machine learning projects in section 8.1.2.

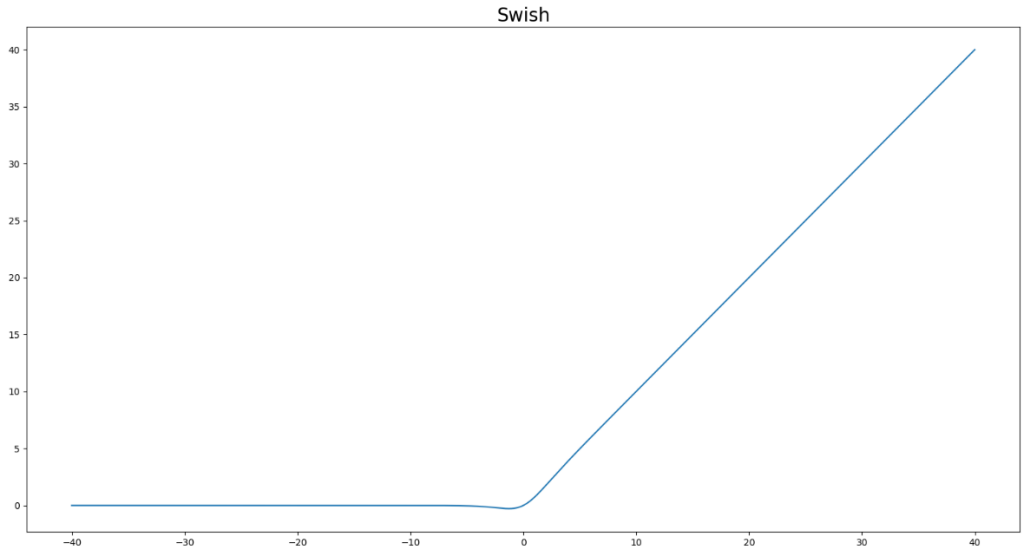


Image 19: *Swish visualized.*

6.2.4.2: Leaky ReLU instead of ReLU for activation

Swish is not the only activation function proposed for this purpose. Rather, there is a multitude of them, of which Leaky ReLU is also commonly used in the deep learning community [98]. Leaky ReLU also resembles traditional ReLU, but activates slightly differently:

$$Leaky\ ReLU: f(x) = \begin{cases} x\ if\ x\ \geq 0 \\ \frac{x}{\propto}\ if\ x < 0 \end{cases}$$

where α can be configured a priori in the range (0, ∞). Leaky ReLU with α = 30 is visualized in image 20. As one can see, its activations for $x\ < 0$ are negative values. By consequence, the death of a neural network can be avoided. Model performance supports this: the loss decreases to 1.3175 and accuracy remains 54.12%.

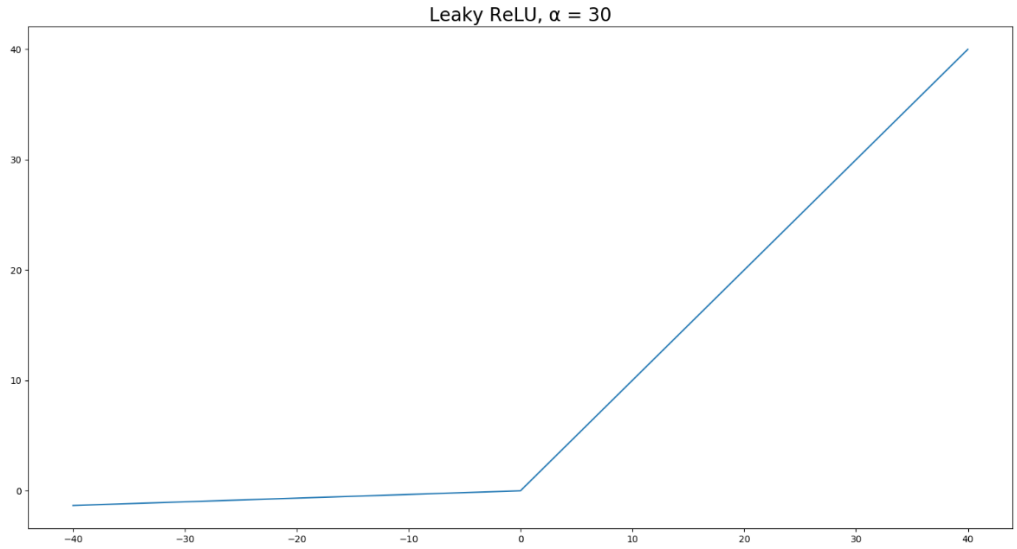


Image 20: *LeakyReLU visualized, α = 30.*

6.2.4.3: Tanh instead of ReLU activation

Although ReLU is commonly used in the deep learning community, another highly used activation function is Tanh. It is visualized in image 21 and activates in the range [-1, 1], with the highest fluctuations around $x \approx 0$. The deep learning community suggests to "try tanh [in one's deep learning projects], but expect it to work worse than ReLU(…)" [99]. For this reason, we also trained a variation with tanh.

It must however be noted that the mathematical proof rendering our choice for He initialization with ReLu activation functions dictates that Xavier initialization must be used when using Tanh [86]. By consequence, in this variation, we replace He (uniform) initialization with Xavier uniform initialization, also known as Glorot uniform initialization.

With an average accuracy across ten folds of 51.69% and a loss value of 1.573, the model performs slightly worse than the one trained with the expanded data set.

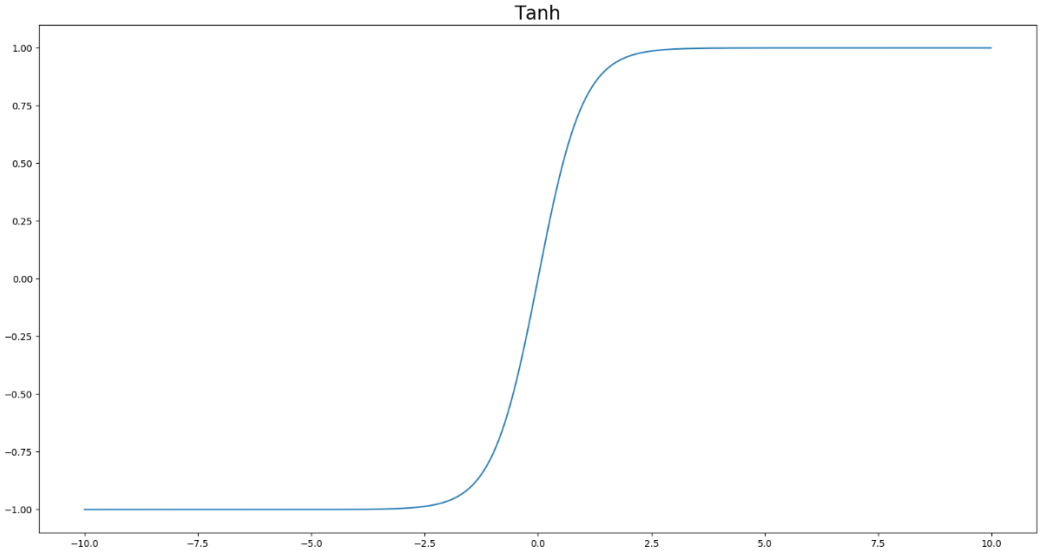


Image 21: *Tanh visualized.*

6.2.4.4: Studying the effects of batch size

As we discussed in section 5.3, batch size impacts training performance given the trade-off between memory requirements and gradient estimation accuracy. For this reason, we studied the effects of batch size on predicting the material types present in GPR imagery by retraining the expanded data set model. We used $batch\ sizes = \{5, 15, 25, 35, 50, 90, 115, 140\}$. For the histogram algorithm, varying batch size does not substantially improve the model. Accuracies range from 48% to 55%. However, the lowest loss values are found when batch sizes are configured in the range of [35, 90].

6.2.4.5: Studying the effects of signal gain

In our data pipeline, during data pre-processing, we apply gain to reduce signal attenuation, i.e. the decrease in signal strength with respect to depth. It might be the case that the gain we applied effectively decreases model performance, and by consequence it is worthwhile to study the effects of different applications of gain on model performance.

We therefore retrained the variation with the expanded data set for various gain settings. Image 22 presents the A-scan of the hyperbola apex and the B-scan window directly around this apex for our baseline scenario, in which energy gain is applied with a coefficient of 1.2. Performance of the histogram algorithm for this gain setting is equal to the expanded data set variation, i.e. with an average loss value of 1.3830 and average accuracy of 54.54%. In all scenarios, maximum gain before feature-wise normalization is $3.0 * 10^5$.

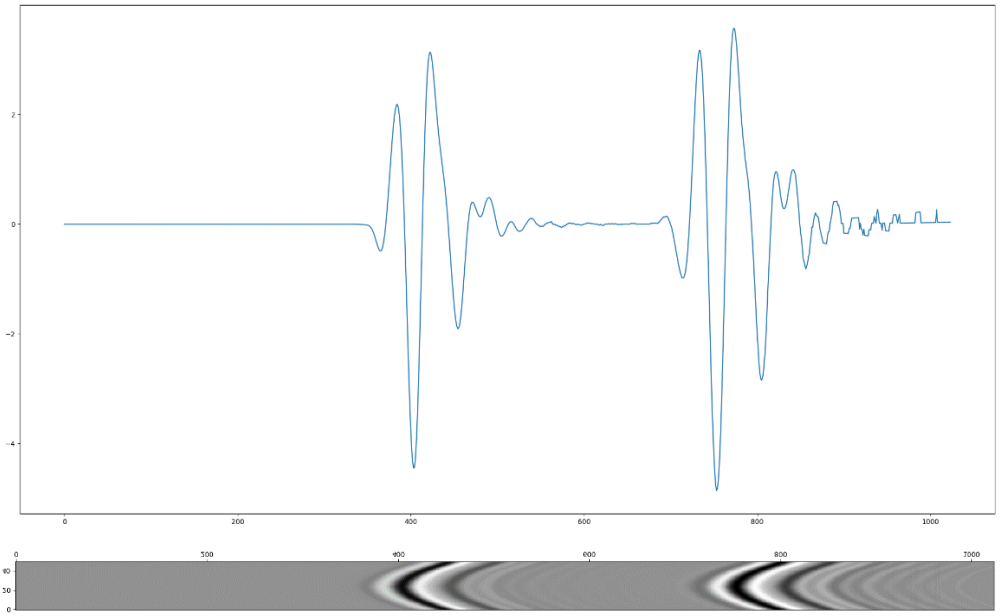


Image 22: *Baseline scenario, energy gain with coefficient 1.2, presents relatively equal signal strengths across the time domain.*

The first variation, presented in Image 23, removes the gain altogether by setting the coefficient to 1.0, effectively removing the exponential character of the gain. Clearly, the backscatters beyond the end of the time domain are weaker than before. Model accuracy slightly deteriorated to 51.03%, but loss slightly improved to 1.3697. The significance of those results is uncertain, but they were obtained over 10 folds.

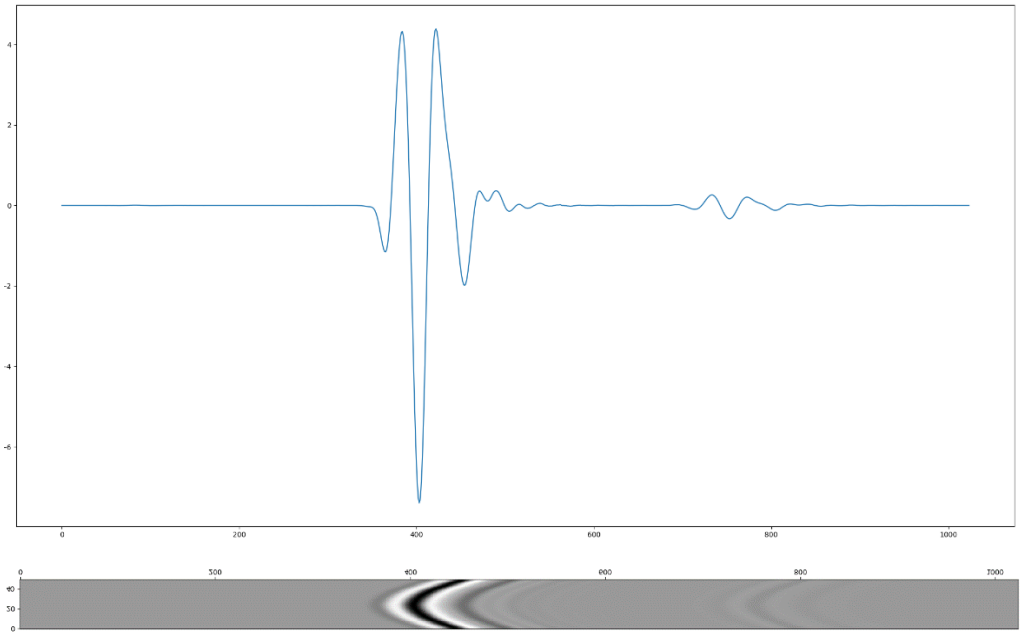


Image 23: *Scenario in which no gain is applied.*

In the third scenario, the gain coefficient was increased from 1.2 to 1.3, effectively inverting signal attenuation. This scenario is presented in Image 24. It becomes clear that signals towards the end of the time domain, i.e. signals deeper down in the ground, are stronger with respect to signals earlier in the time domain. This effectively introduces inverted signal attenuation. It must also be noted that an issue is introduced with this kind of domain related to loss of smoothness of the signal function, which is visible to the right in Image 23. This issue is further analyzed and explained in section 8.2.3. With an accuracy of 54.45%, the model performs relatively equal in terms of accuracy, but average loss improves to 1.3191.

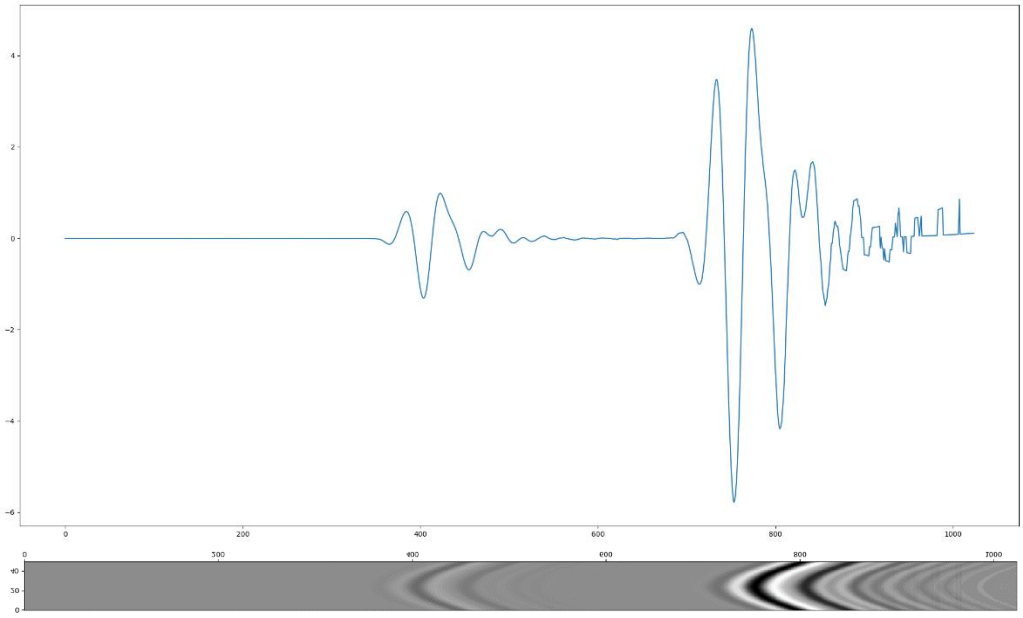


Image 24: *Scenario in which energy gain is increased.*

In the fourth and final gain variation scenario, linear gain is applied instead of energy gain. This type of gain can be expressed as $f(x) = i * x$ where $i = \{0, 1, \ldots 1023\}$, since our A-scans contain 1024 samples (i.e., wave amplitudes). It is a common gain applied in GPR settings and it is therefore worthwhile to investigate the effects of replacing energy with linear gain in our machine learning setting [61].

The application of linear gain and the effects on both A-scan and B-scan for hyperbola apex are visualized in Image 25. It must be noted that the effect of linear gain resembles our no gain scenario. It is therefore not surprising that model accuracy is relatively equal with 54.82%. Model loss, however, is lower, with 1.3212.

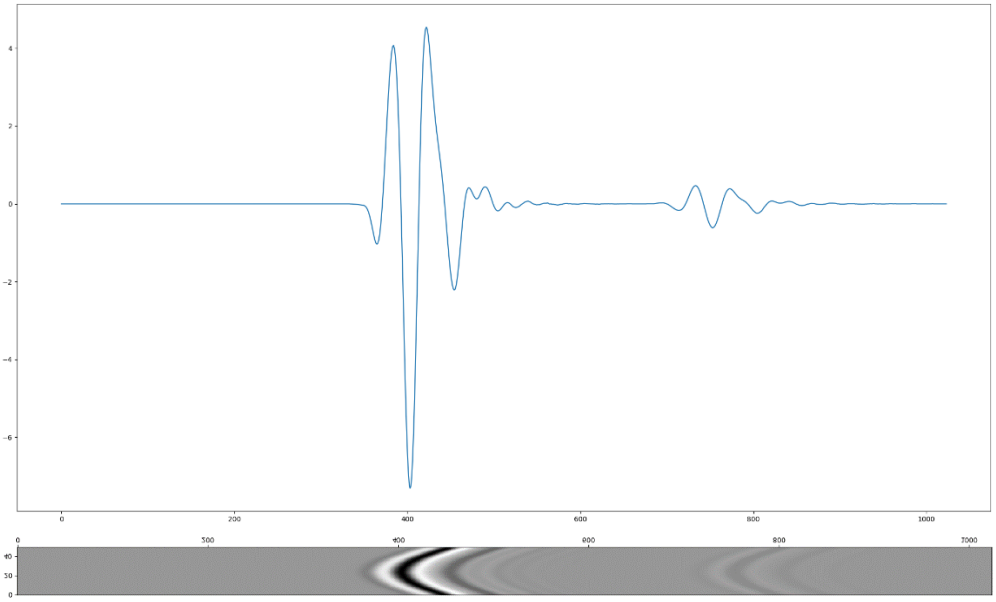


Image 25: *Scenario in which linear gain is applied.*

6.3: Discrete Cosine Transform Based CNN Algorithm

For the DCT based CNN algorithm, we used the DCT-II generator provided by the NUMPY Python package. In line with the validation strategy, we extract only the 14 first coefficients, since those contain most of the signal energy due to the DCT's energy compaction feature. Image 26 presents visualizations of the 14 DCT components for various underground objects, buried at various depths and in various soil types. Many initial observations can be made from those visualizations.

First, soil type seems to influence the DCT substantially. For example, plots 26C and 26F are visualizations of a perfect electricity conductor (PEC), in this case a steel pipeline with a diameter of 11cm, containing water, buried in dry sand. Despite the fact that the depth apparently introduces some variance, the shapes resemble each other. The soil type for this object was altered in plot 26I; it was changed into wet sand. As becomes visible, this relatively simple change substantially alters the computed DCT. Interestingly, this behavior seems to occur not all the time – for example, with a HDPE PEC object (i.e. a metallic cable isolated with polyethylene) presented in plots 26B and 26E, changing the soil type does not alter the DCT. This dissimilarity must be taken into account when analyzing the effects of the DCT based ML model.

Second, although variance between DCTs for similar objects is present, the shapes for DCT coefficients seem to be similar. Given that soil type is constant, this could potentially mean that the DCT can be used to discriminate between material types successfully. Next, we will therefore continue with training the ML models.

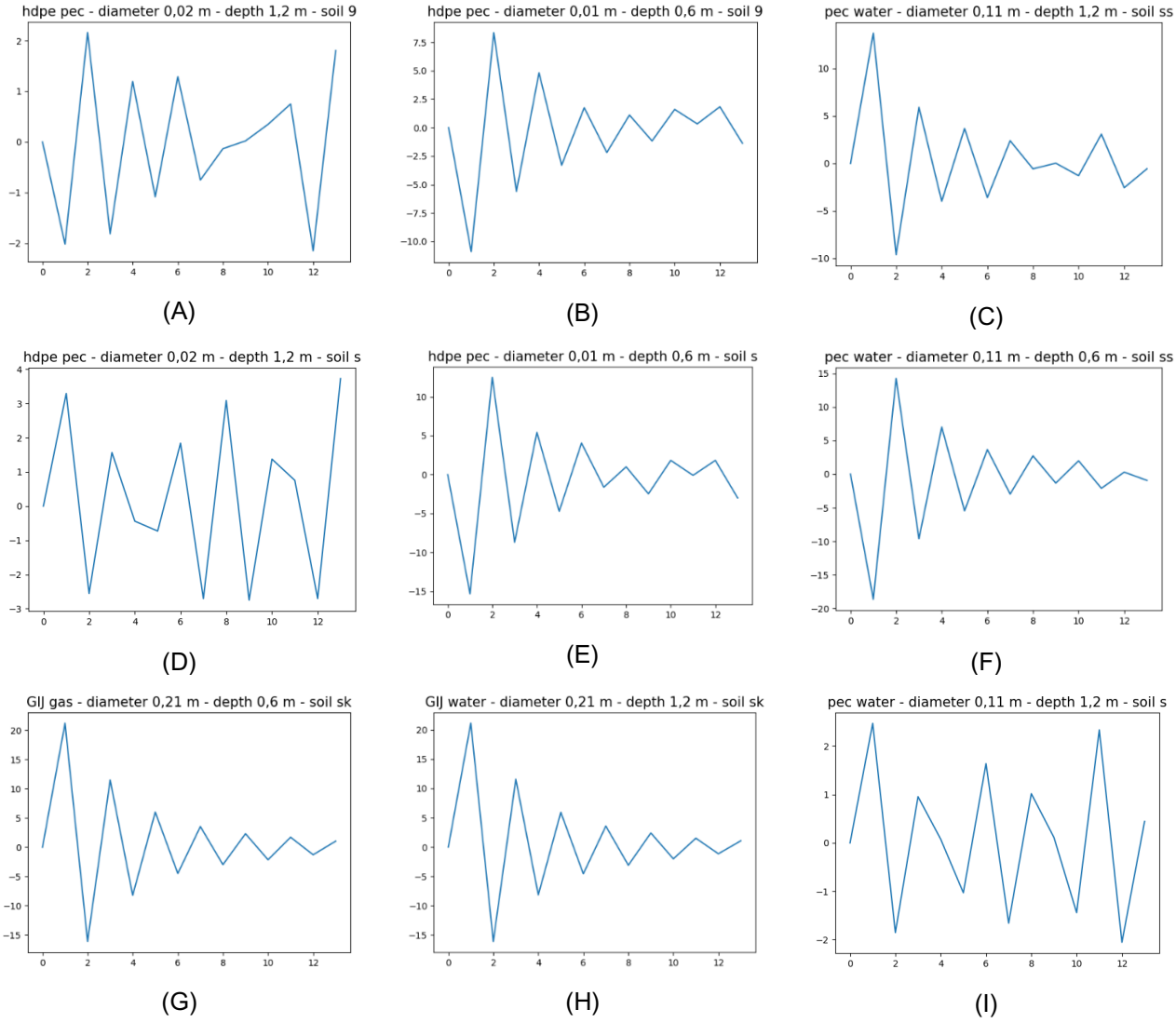


Image 26: *Visualizations of the first 14 DCT-II components for different underground objects, buried at various depths and in various soil types.*

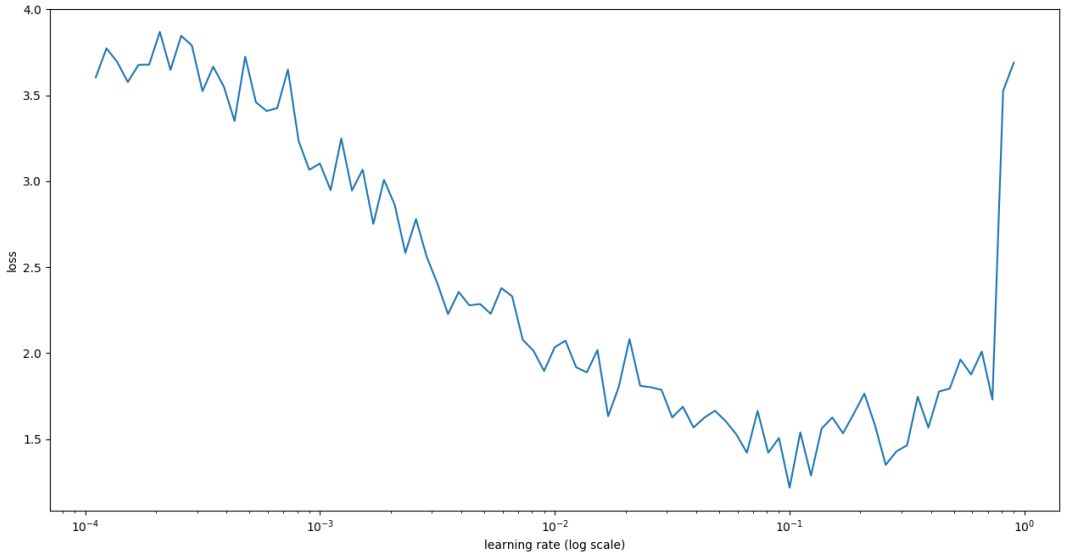


Image 27: *Plot of the losses empirically determined using Learning Rate Range Test for the DCT algorithm.*

6.3.1: Learning Rate Range Test

Similar as for the histogram based algorithm, we also performed a Learning Rate Range Test for the DCT based algorithm. This yields that approximately 10^{-1} is a suitable learning rate. We therefore use this learning rate for training. Image 27 presents the results of the Learning Rate Range Test for the DCT based algorithm.

6.3.2: Algorithm Performance

The training process was performed in line with our validation strategy. Table 14 presents the results.

Fold	Categorical crossentropy test loss	Test accuracy
1	1.0237	64.94%
2	1.2048	58.44%
3	1.2251	58.44%
4	1.2936	62.34%
5	1.1960	64.94%
6	1.2676	66.23%
7	1.3204	61.04%
8	1.0495	63.64%
9	1.2255	62.34%
10	1.3699	55.84%
Average	**1.2176**	**61.82%**

Table 14: *Initial validation results for DCT based CNN algorithm.*

6.3.3: Variations to the Initial Algorithm

In section 6.2.3, we provided hypotheses which may explain the relatively mediocre performance of the histogram based CNN. For the DCT based CNN, a similar mediocre performance can be observed from Table 15. We expect that many of the hypotheses proposed in section 6.2.3 are also valid for the DCT based CNN algorithm. The reasons for this are manifold. First, besides some parameters, the architecture used is similar. Second, the same amount of data is used. Third, for each object, the model attempts to derive an average picture of 14 DCT coefficients. Consequently, both soil type and the material/contents split might influence model performance. Fourth, the balance between learning rate decay and speed of convergence observed might here indicate the need for improving this balance as well. Fifth, hyperparameters used during the training process may negatively impact the model.

The only hypothesis put forward in section 6.2.3 that is not applicable to the DCT based algorithm is the clipped histogram used as a feature vector. Nevertheless, a similar hypothesis can be made with respect to the DCT. That is, our new hypothesis for this algorithm is that the number of DCT coefficients used is insufficient to allow for generalization.

Similar to the histogram based algorithm, we therefore tested various variations to the initial algorithm. First, we expanded our training set with the same amount of simulations as the histogram based algorithm, namely 2426. Similarly, for the DCT based algorithm, we studied the effects of a constant soil type, the effects of separating material and contents and the effects of increased LR decay. The effects of varying the model architecture and hyperparameters were also studied together with, not present previously, the effects of variations in number of DCT coefficients used. Finally, we studied the effects of combining individual alterations with respect to model performance. Image 28 presents the variations as a workflow diagram.

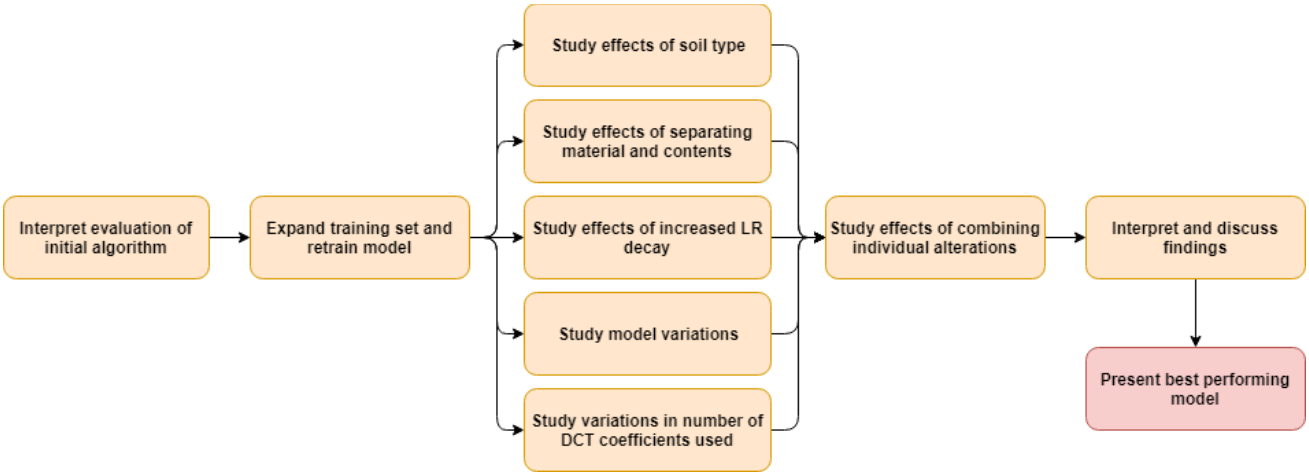


Image 28: *Variations to the DCT based algorithm.*

6.3.4: Performance of Variations

Table 15 presents the performance of the variations proposed in the previous section. Similar to the performance of variations applied to the histogram based CNN algorithm, the variations do not result in improved model performance. Rather, they often result in worse loss and accuracies with respect to the variant trained initially. Additionally, training the model on one soil type could not be done given the lack of data and possible overfitting reported previously.

Similar to the histogram algorithm, the DCT algorithm was also trained with variations in activation functions (Swish, Leaky ReLU and Tanh/Glorot), batch size and the applied gain. For all of them, no substantial improvements in model performance can be reported.

Variation	LR Test Result	Averages	Extremes
Individual variations			
Expanding training set to 2426 simulations.	$10^{-1.5}$	Loss: 1.3103 Accuracy: 52.46%	Max: 1.3822 / 55.97% Min: 1.256 / 45.04%
Training model on one soil type (homogeneous soil, epsilon = 9).	Could not be performed (explanation in section 6.2.4).		
Separated materials and contents in targets.	$10^{-0.99}$	Loss: 2.0797 Accuracy: 23.41%	Max: 2.1094 / 28.40% Min: 2.0302 / 20.66%
Increased LR decay by 175.000.	$10^{-0.77}$	Loss: 1.3103 Accuracy: 51.93%	Max: 1.3551 / 57.61% Min: 1.2703 / 48.34%
Reducing number of DCT coefficients to 7.	$10^{-0.1}$	Loss: 5.6085 Accuracy: 48.22%	Max: 9.3181 / 53.90% Min: 1.3817 / 45.45%
Increasing number of DCT coefficients to 28.	$10^{-1.72}$	Loss: 1.2554 Accuracy: 57.74%	Max: 1.3815 / 62.55% Min: 1.1772 / 52.89%
Increasing number of DCT coefficients to 56.	$10^{-1.72}$	Loss: 1.2984 Accuracy: 60.55%	Max: 1.4000 / 65.43% Min: 1.2126 / 54.54%
Increasing number of DCT coefficients to 112.	$10^{-1.72}$	Loss: 1.3178 Accuracy: 60.51%	Max: 1.4004 / 64.60% Min: 1.2236 / 56.61%
Studying effects of model variations			
Swish instead of ReLU activation function on expanded data set.	$10^{-1.15}$	Loss: 1.3497 Accuracy: 49.09%	Max: 1.3944 / 53.91% Min: 1.2991 / 43.80%
Leaky ReLU instead of ReLU activation function on expanded data set; α = 0.1.	$10^{-1.70}$	Loss: 1.3069 Accuracy: 52.06%	Max: 1.3696 / 57.61% Min: 1.245 / 45.87%
Tanh instead of ReLU activation function; Glorot uniform instead of He uniform initialization, on expanded set.	$10^{-1.10}$	Loss: 1.3622 Accuracy: 49.63%	Max: 1.4274 / 54.32% Min: 1.3116 / 43.39%
Studying effects of varying batch size			

Batch size = 5	$10^{-2.00}$	Loss: 1.3240 Accuracy: 49.50%	Max: 1.3624 / 55.14% Min: 1.2715 / 45.46%
Batch size = 15	$10^{-0.50}$	Loss: 1.3884 Accuracy: 48.23%	Max: 1.4402 / 53.91% Min: 1.3443 / 46.28%
Batch size = 25	$10^{-1.20}$	Loss: 1.3475 Accuracy: 49.25%	Max: 1.4030 / 54.32% Min: 1.2843 / 44.63%
Batch size = 35	$10^{-0.95}$	Loss: 1.3453 Accuracy: 49.67%	Max: 1.4187 / 53.91% Min: 1.2878 / 45.45%
Batch size = 50	$10^{-1.20}$	Loss: 1.3422 Accuracy: 50.20%	Max: 1.3970 / 53.91% Min: 1.2863 / 43.80%
Batch size = 90	$10^{-1.00}$	Loss: 1.3412 Accuracy: 49.67%	Max: 1.3970 / 53.91% Min: 1.2863 / 43.80%
Batch size = 115	$10^{-1.00}$	Loss: 1.3390 Accuracy: 49.17%	Max: 1.3961 / 54.32% Min: 1.2804 / 44.21%
Batch size = 140	$10^{-0.80}$	Loss: 1.3444 Accuracy: 49.38%	Max: 1.3970 / 53.91% Min: 1.2857 / 45.45%
Studying effects of varying gain			
No gain applied in pre-processing, re-introducing signal attenuation.	$10^{-1.59}$	Loss: 1.2873 Accuracy: 52.64%	Max: 1.3541 / 57.02% Min: 1.2375 / 49.17%
Strong exponential gain applied, introducing inverted signal attenuation.	$10^{-0.93}$	Loss: 1.3408 Accuracy: 50.90%	Max: 1.3894 / 55.14% Min: 1.2935 / 45.45%
Linear gain applied instead of energy / exponential gain.	$10^{-1.00}$	Loss: 1.3182 Accuracy: 50.66%	Max: 1.3691 / 52.67% Min: 1.2595 / 46.69%

Table 15: *Validation results for variations applied to DCT based CNN algorithm.*

6.4: B-scan Window Based CNN Algorithm

In section 6.4, we report on the results found when training the B-scan window based CNN algorithm.

6.4.1: Learning Rate Range Test

Similar to the other algorithms, we performed a Learning Rate Range Test for the B-scan window based CNN algorithm. The results of this test are visualized in Image 29. From this visualization, it results that a learning rate of approximately $10^{-3.6}$ is most suitable.

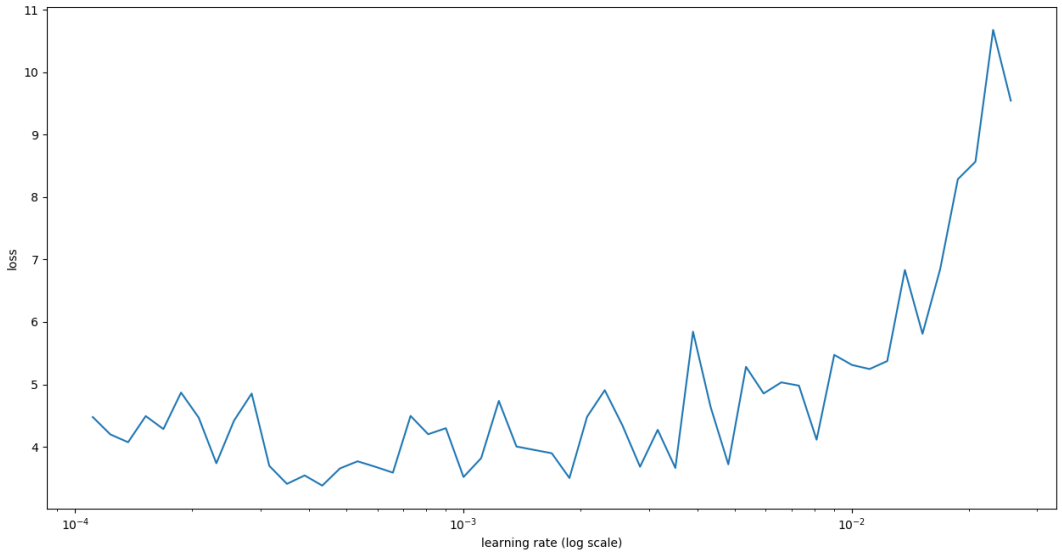


Image 29: *B-scan window based CNN LR test plot.*

6.4.2: Algorithm Performance

The training process was performed in line with our validation strategy. Table 16 presents the results.

Fold	Categorical crossentropy test loss	Test accuracy
1	1.2268	76.62%
2	2.0274	67.53%
3	1.3072	66.23%
4	2.4696	62.34%
5	1.4646	75.32%
6	1.7543	66.23%
7	2.2381	63.64%
8	1.7750	71.43%
9	1.6268	67.53%
10	3.6089	58.44%
Average	**1.361339**	**67.53%**

Table 16: *Initial validation results for B-scan based CNN algorithm.*

6.4.3: Variations to the Initial Algorithm

In sections 6.2.3 and 6.3.3, we observed that many hypotheses are applicable to both algorithms. Although the B-scan CNN performs approximately 6 percentage points better, its performance is still mediocre to fair, and it is worthwhile to attempt improving it. Similar to the histogram and DCT based algorithms, the global hypotheses are applicable to the B-scan algorithm as well. That is, we hypothesize there is insufficient data; soil type and material/content split impact model performance negatively, and so on. Additionally, we

hypothesize that the B-scan window is too narrow. Therefore, we also tested various variations to the B-scan based algorithm. Image 30 presents these variations.

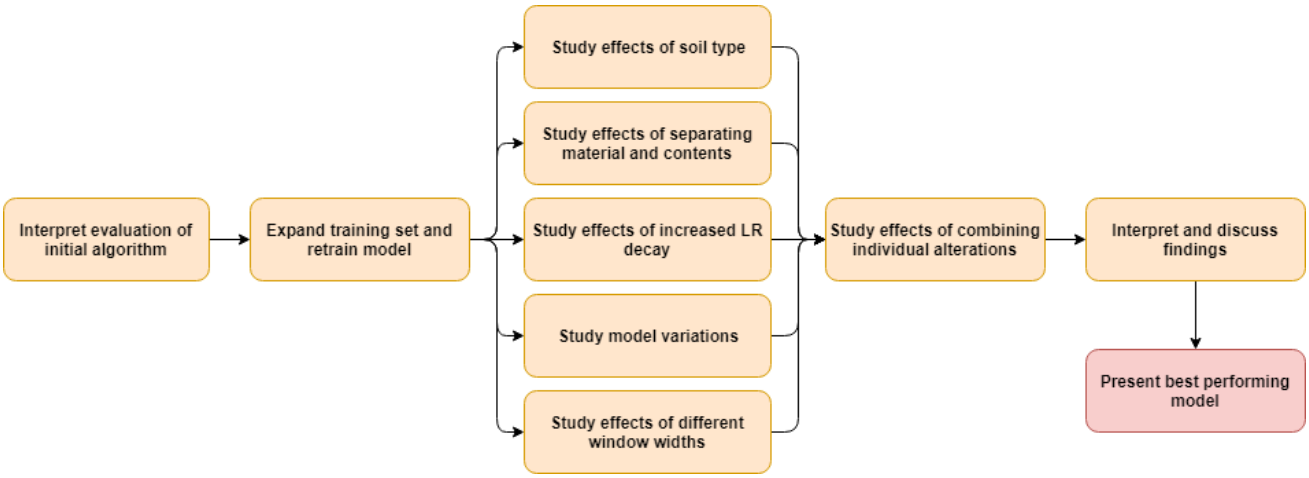


Image 30: *Variations to the B-scan based algorithm.*

6.4.4: Performance of Variations

Table 17 presents the performance of the variations proposed in the previous section. Contrary to the other algorithms, the variations with which the B-scan window-based CNN was retrained yield substantial performance improvements. Generally, accuracy improves with 10 to 15 percentage points. On average across the folds, maximum accuracy achieved is 81.54%. Inspecting the folds individually, maximum accuracy of 86.36% is found. The loss value is also found to improve substantially. Whereas in the initial attempt loss approximated 1.36, variations often yield losses < 1. The best loss for an individual fold is found to be approximately 0.54. Rather unfortunately, retraining the model on one soil type could also not be formed given the lack of training data and resulting overfitting reported previously.

Contrary to the deteriorative effect of applying the other model variations for the histogram and DCT based algorithms, applying them with B-scan data yields higher performance. Specifically, and counterintuitively, Tanh and Glorot initialization yield substantial improvements. With respect to batch size variations, the interval [35, 50] works well, with better loss values compared to the initial variation, i.e. the expanded dataset. Surprisingly, applying linear gain yields model improvements. Combined, those improvements make it worthwhile to combine multiple variations for new validation rounds, which was performed subsequently.

Namely, three combinations of variations were performed. Table 18 presents those combinations. Although accuracies do not improve, loss values are substantially lower compared to the initial variation. This indicates that the models improved nevertheless. In particular, combining the initial variation with the extended (101, 1024) shape, ReLU and He initialization, batch size = 50, linear gain and an increase of LR decay with a factor 175.000 yield the highest improvements. Beyond the regular combinations, we also attempted to train our model on a so-called Inception-V3 block, but it proved to overfit strongly, so this approach was discarded.

Variation	LR Range Test Result	Averages	Extremes
Individual variations			
Expanding training set to 2426 simulations.	$10^{-2.86}$	Loss: 0.8834 Accuracy: 77.57%	Max: 1.0377 / 83.95% Min: 0.6694 / 74.89%
Training model on one soil type (homogeneous soil, epsilon = 9).	Could not be performed (explanation in section 6.2.4).		
Separated materials and contents in targets.	$10^{-3.45}$	Loss: 1.1695 Accuracy: 70.20%	Max: 1.3978 / 75.72% Min: 1.0193 / 65.43%
Increased LR decay by 175.000.	$10^{-3.00}$	Loss: 0.8665 Accuracy: 77.57%	Max: 1.1991 / 84.36% Min: 0.6564 / 71.48%
Reducing shape to (25, 1024)	$10^{-1.87}$	Loss: 2.0215 Accuracy: 65.74%	Max: 8.5609 / 75.30% Min: 1.0745 / 47.93%
Expanding shape to (75, 1024) **Note:** batch size reduced to 35.	$10^{-2.96}$	Loss: 0.9112 Accuracy: 79.18%	Max: 1.0379 / 83.12% Min: 0.7806 / 75.20%
Retrained “expanding training set to 2426 simulations” with input scaled to 33%.	$10^{-1.90}$	Loss: 0.9706 Accuracy: 78.23%	Max: 1.1149 / 82.71% Min: 0.8083 / 73.96%
Expanding shape to (101, 1024) **Note:** input scaled to 33%.	$10^{-2.10}$	Loss: 0.9421 Accuracy: 79.80%	Max: 1.0864 / 83.95% Min: 0.7708 / 75.20%
Studying effects of model variations			
Swish instead of ReLU activation function with rescaled (101, 1024) shape.	$10^{-2.75}$	Loss: 0.8798 Accuracy: 78.81%	Max: 1.0079 / 81.89% Min: 0.6617 / 76.03%
Leaky ReLU instead of ReLU activation function with rescaled (101, 1024) shape; $\alpha = 0.1$.	$10^{-1.84}$	Loss: 0.9805 Accuracy: 77.62%	Max: 1.1309 / 80.58% Min: 0.8418 / 70.66%
Tanh instead of ReLU activation function; Glorot uniform instead of He uniform initialization, with rescaled (101, 1024) shape.	$10^{-2.48}$	Loss: 0.8783 Accuracy: 81.16%	Max: 0.9701 / 85.19% Min: 0.7179 / 78.19%
Trained (101, 1024) variation on an Inception-V3 block.	Immediate overfitting on validation loss; aborted; max acc. 70%.		
Studying effects of varying batch size on (101, 1024) variant			
Batch size = 5	$10^{-1.50}$	Loss: 2.8527 Accuracy: 48.23%	Max: 8.7393 / 53.91% Min: 1.3449 / 45.45%
Batch size = 15	$10^{-1.90}$	Loss: 1.1110 Accuracy: 66.16%	Max: 1.3824 / 78.51% Min: 0.8767 / 46.91%
Batch size = 25	$10^{-1.99}$	Loss: 0.9294 Accuracy: 77.87%	Max: 1.1087 / 81.48% Min: 0.8056 / 71.60%
Batch size = 35	$10^{-2.50}$	Loss: 0.8599 Accuracy: 80.96%	Max: 1.017 / 84.77% Min: 0.7274 / 76.54%
Batch size = 50	$10^{-2.40}$	Loss: 0.8108 Accuracy: 81.54%	Max: 1.000 / 86.36% Min: 0.6311 / 76.13%
Batch size = 90	$10^{-2.00}$	Loss: 0.9900 Accuracy: 79.39%	Max: 1.2734 / 83.88% Min: 0.8302 / 73.97%
Batch size = 115	$10^{-2.75}$	Loss: 0.8369 Accuracy: 79.72%	Max: 1.0426 / 84.36% Min: 0.6001 / 76.03%
Batch size = 140	$10^{-2.35}$	Loss: 0.7931 Accuracy: 80.63%	Max: 0.9070 / 85.19% Min: 0.5494 / 74.49%
Studying effects of varying gain			
No gain applied in pre-processing, re-introducing signal attenuation.	$10^{-2.25}$	Loss: 1.3299 Accuracy: 67.65%	Max: 1.5853 / 72.84% Min: 1.1667 / 59.26%
Strong exponential gain applied, introducing inverted signal attenuation.	$10^{-1.75}$	Loss: 1.2134 Accuracy: 67.35%	Max: 1.3344 / 76.54% Min: 1.055 / 60.33%
Linear gain applied instead of energy / exponential gain.	$10^{-2.25}$	Loss: 0.8517 Accuracy: 79.93%	Max: 1.0964 / 83.13% Min: 0.6262 / 74.49%
Combined variations			
Combined best-performing variations: • (101, 1024) shape	$10^{-2.48}$	Loss: 0.8390 Accuracy: 78.44%	Max: 0.9533 / 82.30% Min: 0.6836 / 76.13%

• Tanh/Glorot • Batch size = 50 • Linear gain • LR decay factor 175.000x of original			
Combining well-performing variations with an eye to the original design: • (101, 1024) shape • ReLu/He • Batch size = 50 • Linear gain • LR decay factor 175.000x of original	$10^{-2.48}$	Loss: 0.7419 Accuracy: 79.31%	Max: 0.8314 / 83.13% Min: 0.6600 / 76.54%
Combining well-performing variations but mimicking original design: • (101, 1024) shape • ReLU/He • Batch size = 50 • Original gain • LR decay factor 175.000x of original	$10^{-2.88}$	Loss: 0.7758 Accuracy: 79.76%	Max: 0.9007 / 85.54% Min: 0.6085 / 75.62%

Table 17: *Validation results for variations applied to B-scan based CNN algorithm.*

7: Machine Learning Driven IA Tool for GPR Analysts

In this chapter, we discuss the design and development of an Intelligence Amplification tool for GPR analysts. The tool, which is explicitly a Proof of Concept and thus prototypical in nature, allows practitioners to upload real GPR data generated by GSSI GPR devices (in so-called .dzt files) into their web browser. They can subsequently tune data processing from their browser and, finally, analyze material type by clicking on the apex of a hyperbola. Section 7.1 outlines the software design by means of its architecture and how it can be embedded in a generic GPR data analysis process. Section 7.2 discusses the instantiation of our tool.

7.1: Software and Process Design

In this section, we cover the software and process design for the IA tool. Specifically, in section 7.1.1, we discuss the software architecture and discuss the role of a so-called Riverflow Rock which serves as the container for our machine learning model. In section 7.1.2, we investigate how it can be embedded in the analysis process of a GPR analyst.

7.1.1: Software Architecture

Image 31 presents the software architecture for the IA tool developed in this work. Specifically, it contains a frontend as well as a Riverflow Rock instance. The frontend was developed using Angular, a framework for developing single-page applications developed and maintained by Google as well as individual contributors. It allows a developer to develop applications in a component driven way and is, together with other single-page application frameworks such as React and VueJS, the de facto standard for today's modern web application development practices.

We specifically use Angular over React and VueJS since it allows for direct manipulation of the DOM, or Document Object Model, which means that native HTML elements can be directly manipulated by TypeScript code transpiled into JavaScript. This is highly beneficial for our intended goal, since the radargram data is loaded into a HTML Canvas element on screen, to which event listeners must be attached.

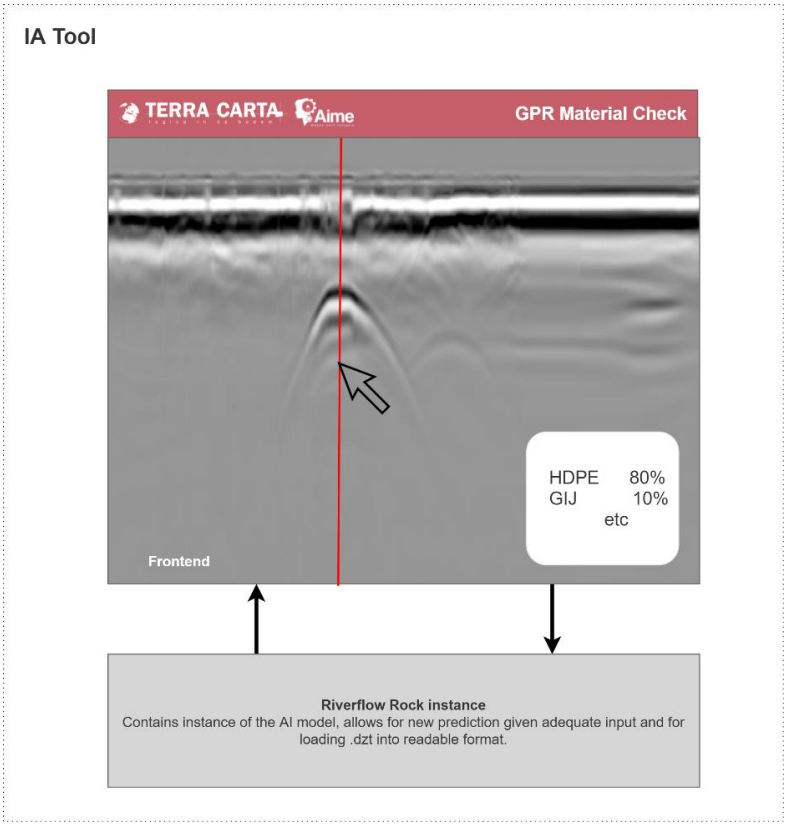


Image 31: *Software architecture for the IA Tool.*

The primary event listener attached to the HTML Canvas listens to the ONCLICK event. That is, when the user clicks the radargram image, the event listener is triggered and the callback function is executed. This specific callback function is programmed to crop a (101, 1024) slice around the specific trace selected by the user, visualized in red in Image 31. This slice is subsequently passed onto the Riverflow Rock instance running in the background. This Rock forwards the slice into the machine learning model embedded into it for prediction and subsequently returns the prediction to the front-end, which is then designed to display it using a popup.

A Riverflow Rock is open source AI container technology developed and maintained by de Gasfabriek in the Netherlands[1]. It is part of the Riverflow AI platform, which was developed specifically for managing technical debt usually emerging in ML model lifecycles. Technical debt is a metaphor introduced "to help reason about the long term costs incurred by moving quickly in software engineering" [100]. Whereas the concept of technical debt is relatively known to regular software engineering practice, it has so far been less known to the machine learning community. Nevertheless, the authors of [100] argue that ML systems have a "special capacity for (…) [such] debt, because they have all of the [traditional problems] plus (...) additional (...) ML-specific issues". Specifically, with respect to machine learning deployment, problems range from the fact that:

- Models can have *undeclared consumers*. That is, the model can be used by an audience for which it was not intended. If, for example, the original authors of the model upgrade it by e.g. retraining it, it can become useless to those undeclared customers. This performance deterioration might even occur without them knowing it, because they were undeclared in the first place. By consequence, without an uniform abstraction layer, *ML version management* becomes highly ad hoc or even impossible.
- There exist dozens of frameworks and programming languages for creating machine learning models. Every engineer prefers one framework over another. By consequence, many frameworks will be used in an arbitrary organization. As a result, interfacing to such models is not performed uniformly, increasing the risk for *glue code*, various interdependencies and possibly inaccessible models.
- So-called *pipeline jungles* may occur when data pre-processing is not performed in an uniform way and in an isolated fashion, i.e., on a per-model basis. In this case, elements used for data pre-processing are stored in a rather scattered way, possibly introducing hidden interdependencies.
- Models may become redundant and may eventually be killed from production use. As a result, interfaces will die too, which can be unknown to the original engineer and/or the software engineer who depends on the model. Those are so-called *dead code paths* that must be avoided.
- Model run-time *configuration* becomes ad hoc.
- Organizations often wish to obtain statistics about their models. For example, it may be interesting to know what the performance requirements for a particular model are, in order to up- or downscale the hardware requirements currently configured for the model. Measuring model performance and/or utilization is impossible if no standardized means of doing so is used.

A Rock provides this standard means of deploying machine learning models into the cloud. It provides a collection of Python code into which one's machine learning model can be embedded. Currently, it is under active development, and an increasingly amount of Python based machine learning frameworks is supported.

By subsequently embedding the Python code and one's machine learning model into a Docker container, a Riverflow Rock can run on any host system that runs Docker. By consequence, our ML solution is highly scalable, is easily deployable and migratable across various cloud service providers and runs independently of the operating system running on the host machine. Additionally, by deploying the ML models used in our IA tool using a Rock, we can approach them uniformly using REST over HTTP. In the future, machine learning utilization can be measured as well as its performance requirements, and it can be deployed with one click. This altogether reduces the technical debt usually introduced with machine learning deployments in one's infrastructure and provides TerraCarta with tooling that is highly robust against external factors.

[1] https://github.com/gswrx/riverflow-rock

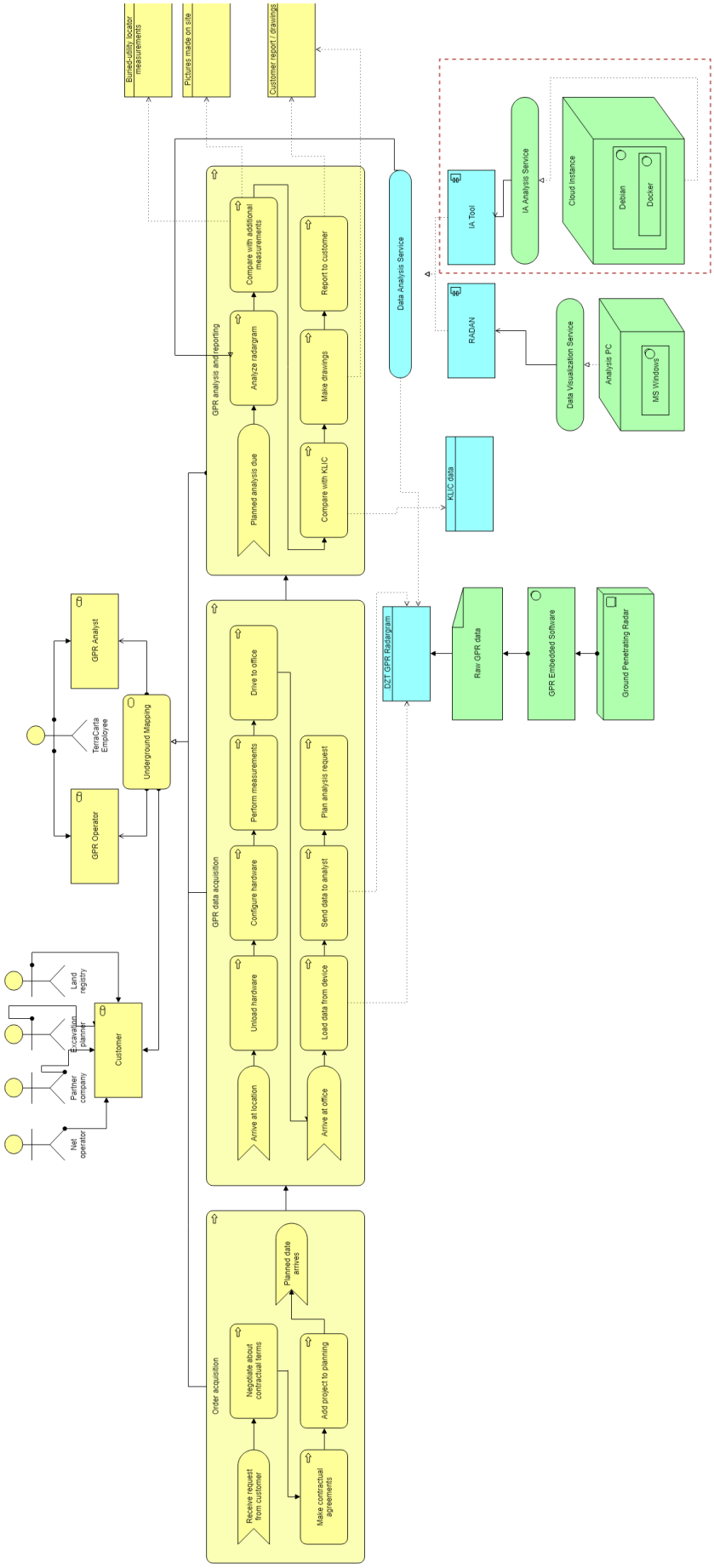


Image 32: *ArchiMate model for a generic GPR analysis process.*

7.1.2: Embedding in GPR Analysis Process

Beyond the software design, it is also of interest to identify how an IA tool can be embedded into practitioners' analysis processes. For this reason, we propose a conceptualization of a generic GPR analysis process inspired by the process used by TerraCarta. It is presented in Image 32. The process starts when a customer requests a quote for certain analysis activities from the analysis company. Customers can be net operators, partner companies which have too much work on their hands, excavation planners and land registries. As is usual with business requests, a process of negotiations and contractual arrangements is then started. When finalized, it results in a contractual agreement. The project is then added to the project planning and the process halts until the date planned for undertaking activities arrives.

At that date, an employee of the analysis company having the role of *GPR operator* drives to and arrives at the site where data has to be recorded. The operator unloads the hardware (i.e., the GPR device) brought along to the site, configures it (i.e., applies hardware settings such as gain and filters to be applied on-device) and performs measurements. Once completed, the operator drives to the office, where he or she loads data from the device and sends it to the analyst. This can be done through a cloud-based environment, by using a USB stick or email, depending on the organization. A request for analyzing the data is then planned.

When the analysis is due, a GPR analyst (possibly being the same actor as the GPR operator, but this is not necessarily the case) opens the relevant files on his computer for analysis. First, given the contextual and geophysical knowledge already present with the analyst, the GPR radargram is analyzed. Specifically, the hyperbolae are classified according to one's background knowledge.

Various software tools can be used for this analysis. Which tool is used is highly dependent on the GPR's manufacturer, which often combines proprietary hardware with proprietary software. Fortunately, open source software tools are available which allow one to read proprietary files as well. With respect to GSSI based GPR imagery, the proprietary software tool "GSSI RADAN" is often used for interpretation and analysis.

In Image 32, our IA tool – a web application running as a Docker container in a cloud instance – is modeled to be a part of the "data analysis service" application service composition. It is highlighted in a red dotted box. It can be used in parallel with tools like RADAN for interpretation. Currently, those tools do not integrate given the proprietary nature of the latter. However, where possible, the IA tool – and specifically the backend discussed in the previous section, which handles data interpretation and generating machine learning predictions – can be integrated relatively easily. Altogether, the diagram demonstrates how the IA tool amplifies the analyst's background knowledge during analysis.

These early insights are then compared with additional measurements and the KLIC registry, if necessary. Additional measurements may be signals retrieved with buried-utility locators or pictures of trenches created alongside cables and pipelines in the field. Such measurements allow the analyst to increase their certainty about the material types of certain cables and pipelines. Since digging trenches is a labor-intensive job, and applying buried-utility locators also induces more work, the benefit of our IA tool for practice becomes evident.

Based on all background knowledge, a drawing is made of the perimeter with all relevant cables and pipelines. This drawing is then consolidated, possibly with other drawings, into a report, which is sent to the customer. This way, the customer is informed of the current position of cables and pipelines and the analysis company successfully completed its analysis task.

7.2: Tool Instantiation

The software architecture for our tool was instantiated in order to visually demonstrate the technical feasibility of our machine learning models in a human-in-the-loop setting. Image 33 provides a screenshot of the tool when it is being used. Note that the radargram displayed on-screen

When the tool is started, the analyst can upload a .dzt file, which is sent to the Riverflow AI container running in the background for numerical processing (i.e., loading the numerical data and returning it in a standardized format).

The radargram embedded in the .dzt file is then loaded into the browser, using the HTML Canvas mentioned earlier, which is capable of dynamically rendering numerical data. On the right, the user can subsequently change the application of gain and the ground bounce removal applied to the radargram. Those values are by default configured to energy (exponential) gain with a coefficient of 1.2 and median ground bounce removal; those are also the values used for training the machine learning model running in the Riverflow AI container. The analyst can however change applied gain to *linear gain* and *no gain* and can also use *mean-based ground bounce removal* or *no ground bounce removal*. The manipulated radargram then immediately changes in the web browser.

By clicking on an arbitrary position in the radargram, a sliced window in line with the shape of our radargram (i.e. (101, 1024)) is input into the machine learning model running in the Riverflow AI container. The prediction is subsequently displayed on screen, in a popup message. With a red line, the user is notified which area of the radargram is analyzed. In Image 33, this is visible slightly to the left of the analysis popup.

Since it only makes sense to analyze the apex of a hyperbola, the user is instructed on the right of the application to click on this apex when analyzing the materials of underground objects. Currently, the system does however not forbid the user to analyze non-hyperbolic areas. Given the multiclass probability distribution over a set of *target objects*, the system will then produce a prediction nevertheless. However, any human being using common sense will understand the non-usability of this analysis result. In future work, this behavior can be avoided by adding an additional interpretation layer which automatically identifies the hyperbolae a priori. The user can then be warned when he or she clicks on a non-hyperbolic signature.

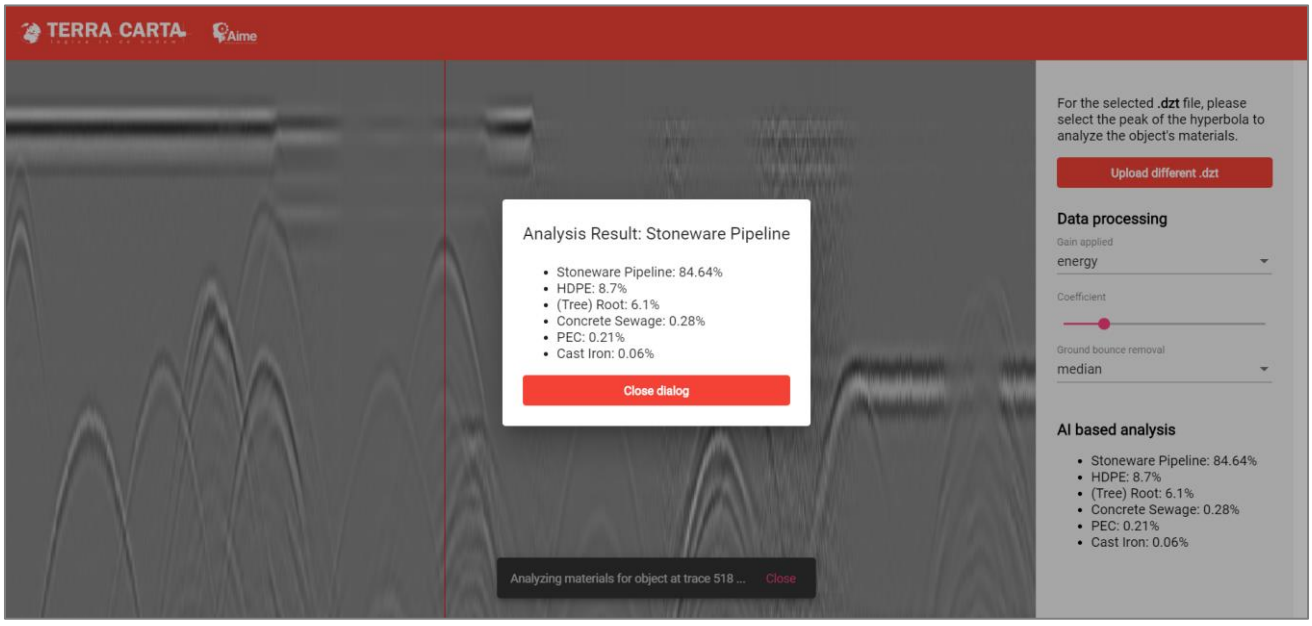


Image 33: *Screenshot of the tool in action.*

7.3: Tool Performance with Real Data

No extensive testing was performed with real radargrams, which was out of scope for this study. However, out of curiosity, some real life GPR data was uploaded into the IA tool and analyzed. The early conclusions that can be attached to those tests are that the variance between test class outcomes increases substantially with real GPR data. It is unknown why this is the case: it could be due to the differences between GprMax and real life data, the substantially larger amount of noise, the fact that multiple (overlapping) hyperbolae are present in the radargram, and so on. The next chapter, specifically the section on study limitations, will present possible explanations as to why the IA tool should still be used with caution, despite its early success.

7.4: Early Validation Comments

The tool was not validated thoroughly in a work setting. This was out of scope for the research objectives set out for this study, which were to demonstrate technical feasibility for this ML-based IA scenario and its embedding in the generic analysis process. However, in line with ADR methodology principles, the tool was developed in a spirit of co-creation. That is, sketches of the end result were made together with TerraCarta prior to artefact development. Feedback on those was subsequently requested from TerraCarta's upper management and a GPR analyst, who acknowledged the possible business value of the artefact. Additionally, the instantiated tool itself was also demonstrated to those actors, who responded positively. One remark stood out: "this tool could potentially change entirely the way I do my work".

8: Discussion

In this chapter, we will discuss our findings presented previously. We focus on explaining the differences in performance between the three machine learning based algorithms in section 8.1. We discuss whether the possibility of feature extraction applied twice results in relatively poor performance in section 8.1.1, discuss the effectiveness of the various variations applied in section 8.1.2 and subsequently discuss the impact of electric characteristics of buried materials in section 8.1.3. In section 8.2, we present the limitations of our study.

8.1: Explaining Model Performance Differences

In this section, we analyze the results reported in chapter 6 with respect to the difference in performance between the classes of machine learning models and the variations performed for every class. In section 8.1.1, we provide a rationale for why the histogram-based and DCT-based algorithms might not work as intended. Section 8.1.2 discusses the effectiveness of the applied variations. Finally, in section 8.1.3, we present arguments for why model performance in the best-performing class plateaus around 80% on average.

8.1.1: Feature Extraction Applied Twice?

The histogram based CNN algorithm and the DCT based CNN algorithm were inspired by previous work in which Support Vector Machines were used for GPR hyperbolae material type classification [74]. Support Vector Machines were a very popular machine learning technique many years ago, but face the drawback that they can only handle relatively sparse data. As we noted before, GPR radargram data is anything but sparse: for example, a (101, 1024) B-scan feature vector yields $101\ x\ 1024 = 103.424$ features. By consequence, ten years ago, scholars were required to reduce the dimensionality of their input data. The Discrete Cosine Transform, applied in [74], is a prime example of this reduction: only 14 DCT coefficients were input to a SVM. This is a feature vector of 14 features – effectively reducing the feature vector size by almost 7.400! The energy compaction principle of the DCT allowed the authors to use this transform.

The same argument can be made for a signal histogram: the histogram used in our work reduced the feature vector to 101 features, which makes it approximately a thousand times sparser.

However, exactly this feature vector sparsity might result in poor performance when CNNs instead of SVMs are used for classification. Our results demonstrate that those sparse feature vectors yield relatively poor model performance. Accuracies as if one is randomly picking a class are reported. Non-sparse feature vectors, however, yield promising results with accuracies boosted to approximately 81.5%.

We therefore argue that applying dimensionality reduction techniques to GPR input data for CNNs deteriorates model performance. We suggest that this behavior occurs because feature extraction is then effectively applied twice. This can be explained through the inner workings of a CNN: the convolution operation applied to the input data effectively allows the network to learn a preconfigured amount of filters itself. Applying one convolutional layer already makes the input data substantially sparser. Additionally, a CNN is often composed of multiple convolutional layers augmented with other layers enhancing sparsity, such as Max Pooling. This way, we suspect that applying feature extraction techniques to reduce input data dimensionality blinds CNN convolution operations to idiosyncrasies in the data. We expect that this results in the relatively poor performance reported in our results. By consequence, the general argument in the deep learning community that minimum feature extraction is required with CNNs (e.g. as outlined in [46]) seems to hold true for GPR data and might even be imperative to follow.

8.1.2: Effectiveness of Variations

Next, we will discuss the effectiveness of the individual variations applied to our CNN algorithms. In doing so, we primarily focus on the variations applied to the B-scan algorithm. For the other two algorithms, no comparisons can be made, since they do not show any substantial performance throughout initial training and subsequent variations, as explained in the previous section.

8.1.2.1: Effectiveness of Expanding the Data Set

In section 6.4, we observed that training the B-scan model initially yielded an accuracy of 67.53% and a loss of 1.3613. We argued that the relatively poor performance could be related to the size and variety of our data set. First, CNNs generally require a substantial amount of training data for improving the model. Second, our data set consisted of distinct objects; that is, a unique object (represented by object material, depth, soil type and contents) present in the training set was never present in the testing set. By consequence, we argued, the model was underfit. This point of view was confirmed by expanding the data set from approximately 800 to approximately 2425 simulated objects, in which redundancy with respect to objects was introduced. In those,

the characteristics of the soil (e.g. noise) are varied randomly to avoid overfitting. As a result, the model trained with 2425 objects improved with approximately 10 percentage points to ≈ 77.5%, and subsequent variations yielded average accuracies > 80%. Despite this success, it remains unknown whether the model is still underfit. We discuss this further in section 8.1.3.

8.1.2.2: Effectiveness of Varied Activation Functions

In section 6.2, we argued that although ReLU is the de facto standard activation function used by the deep learning community today, it has certain drawbacks. We also proposed variations which use state-of-the-art activation functions, in particular Leaky ReLU and Swish, which both incorporate the positive aspects of traditional ReLU while reducing the drawbacks.

The results presented in chapter 6.4 suggest that the B-scan based models trained with Leaky ReLU and Swish produce approximately equal performance compared to traditional ReLU.

A peculiar observation for the application of Leaky ReLU is that it shows a substantially higher loss compared to the algorithm trained with the expanded dataset. This behavior may be explained by the α parameter, which was set to 0.1 in our study. Particularly, the creators of Leaky ReLU suggest to "to set [α] to a large number like 100" [98]. Relatively unfortunately, the Keras documentation provides a default α of 0.3, which inspired us to use 0.1, before noticing the statement on the relatively large α in the original work.

How this difference between αs impacts the activation of Leaky ReLU is presented in Image 34, in which the behavior of Leaky ReLU is visualized for α = 0.1 (our choice) and α = 30 ("a [relatively] large number"). In particular, our choice for α results in an aggressively decreasing activation for inputs < 0. The relatively large number, α = 30 in this case, results in a slow decrease instead (although the image suggests that the outputs are 0, they are in fact small but non-zero). Our choice for α when applying Leaky ReLU may thus have resulted in large model swings and may have been countered by our regularization activities (i.e., the effect of BatchNormalization applied in the Conv layers and L2 regularization in the Dense layers). Those weight swings can impact model performance and possibly explain the substantially higher loss for Leaky ReLU. This may yield the conclusion that Leaky ReLU is still useful for this machine learning problem, but that α should be applied more carefully. Investigating the effectiveness of Leaky ReLU is however future work.

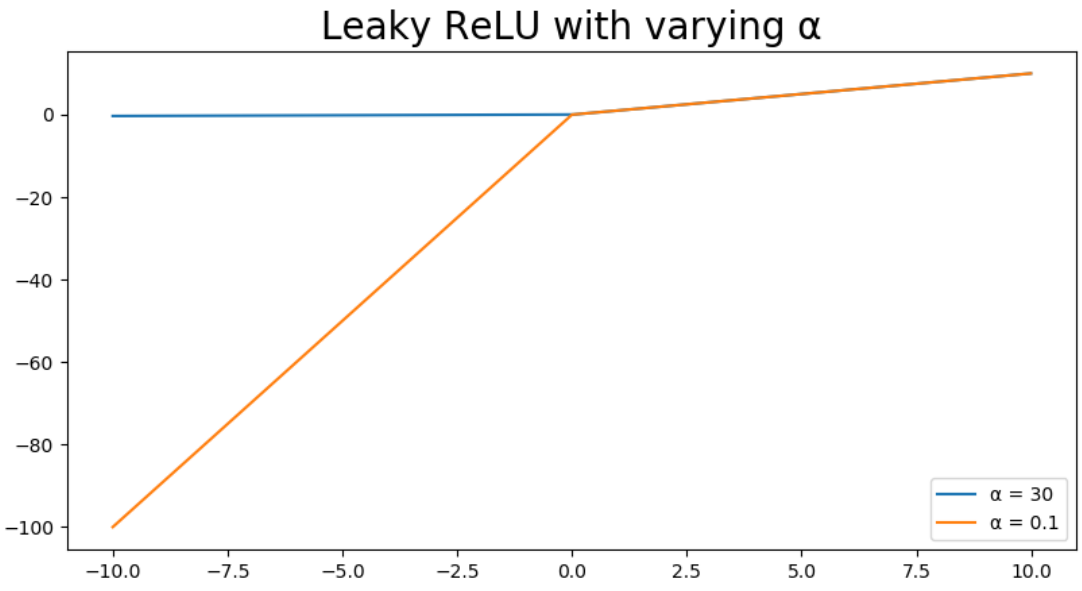


Image 34: *Activation of Leaky ReLU for different α.*

Generally, however, we can argue that the application of state-of-the-art activation functions does not yield improved performance over traditional ReLU. Why this occurs might be explained in multiple ways. First, the death of ReLU powered neural networks is often caused by extremely large negative weight values from which the neurons cannot recover. In many cases, this occurs due to poor initialization of the network. In our

case, we made a deliberate choice for He initialization, for which mathematical evidence is available of its suitability for ReLU activated networks. Second, the occurrence of such large weight swings in our network is unlikely regardless of initialization, since we applied regularization techniques (i.e., BatchNormalization and L2 Regularization, as mentioned before). Third, the positive effects of e.g. Swish have so far primarily been reported for very large machine learning problems [97]. Those are machine learning problems where either the data set is very large, the model is very deep, or both, which is often the case. This can be explained by the observation that such networks are generally more sensitive to the vanishing gradients problem discussed in section 5.3.1.3, since the impact of this problem increases with the number of layers present in a model. The death of a model caused by zeroed ReLU activations is in a way an instance of this problem, since gradients for layers upstream the model get closer and closer to zero. However, our model is relatively compact, in the case of the B-scan algorithm only applying 3 Conv2D blocks and 2 Dense blocks. Extremities with respect to model depth today include variants with hundreds of layers; for example, the network for which Swish has shown improvements has dozens of layers. By consequence, we empirically argue that generally ReLU works as well as Swish and Leaky ReLU for relatively compact neural networks.

Another peculiar observation with respect to activation functions can be made with respect to the improved performance when using the Tanh activation function. This improvement is surprising, since we noted before that generally Tanh performs worse than ReLU. In our work, Tanh improved both the loss and the accuracy, the latter with approximately 3.5 percentage points. It is unclear why this behavior occurs. One direction for future research could be investigating the impact of BatchNormalization and/or L2 Regularization on ReLU activated networks, which could potentially reduce the large values produced by this activation function. Tanh, instead, activates on an interval of [-1, +1], possibly rendering to be a more native fit to regularized networks.

8.1.2.3: Effectiveness of Varied Batch Size

We studied the effectiveness of varying with the batch size for the (101, 1024) variant. This clearly shows that the larger the batch size, the better the model performs. For example, when $batch\ size = 5$, the accuracy is ≈ 48.0% with a loss of 2.85. When the batch size is increased step-wise to $batch\ size = 140$, performance fluctuates but generally improves to 80.6% and 0.7931, respectively. This provides support for the general notion that larger batches allow for a better estimate of the gradient during optimization. However, since the improvement levels off around $batch\ size = 140$, we did not further increase batch size to illustrate our point.

What we also encountered during training is that large batch sizes indeed require a large memory footprint; sometimes, our model crashed during training for it requesting too much memory. We overcame those limitations by scaling our input data to a smaller size. This ensures that the patterns are kept but that the input data is more compressed. Altogether, the behavior with respect to variations in batch size is not surprising.

8.1.2.4: Effectiveness of Varied Signal Gain

For studying the effects of varying signal gain, we trained and compared three variations across our algorithms:

- No gain applied in pre-processing, by setting the energy gain coefficient to 1.0;
- Applying strongly exponential gain by setting the energy gain coefficient to 1.3, introducing inverted signal attenuation;
- Applying linear gain instead of energy gain.

For the B-scan window based CNN, we used the (101, 1024) extended algorithm as our baseline algorithm. This algorithm performed relatively well, with an average accuracy of 79.80% across ten folds and a loss value of 0.9421.

Applying no gain and applying stronger gain both reduce the performance of the model, yielding accuracies of approximately 67-68% and loss values between 1.20 and 1.30. Linear gain, however, improves model performance, with an average accuracy across ten folds of 79.93% and a loss value of 0.8517.

Image 35: *Linear gain applied to a B-scan.*

This behavior can possibly be explained by the effect of linear gain on the signal, as visualized in Image 35. Apparently, the main reflection of the object is considered to be most important for classification of the material. Sub reflections (the lighter reflection down the time domain in Image 35 is the same object; the effect is introduced by water mirroring the reflection back and forth inside the pipeline) seem not to be the main discriminators, but effectively support the model in making a decision. This hypothesis is further supported by the fact that stronger gain – effectively making the sub reflections stronger than the main reflection – introduces worse performance, whereas no gain – making the sub reflections much weaker than the main reflection – does the same. Although this is an interesting explanation, it must be studied in future work.

8.1.2.5: Effectiveness of Combined Variations

For the B-scan algorithm, we created three groups of individual variations which performed well to investigate whether their sum results in better performance than their individual applications. The composition of the three groups is as follows:

1. Combining the best-performing variations
    a. 2426 simulations for training;
    b. (101, 1024) shape;
    c. Tanh/Glorot;
    d. Batch size = 50;
    e. Linear gain;
    f. LR decay factor 175.000 times original value;
2. Combining well-performing variations with an eye to the original design:
    a. 2426 simulations for training;
    b. (101, 1024) shape;
    c. ReLU/He;
    d. Batch size = 50;
    e. Linear gain;
    f. LR decay factor 175.000 times original value;
3. Combining well-performing variations but mimicking the original design:
    a. 2426 simulations for training;
    b. (101, 1024) shape;
    c. ReLU/He;
    d. Batch size = 50;
    e. Original gain;
    f. LR decay factor 175.000 times original value.

The latter performs best, followed by the second and the first. Since the number of simulations for training, the shape, batch size and learning rate decay factor are equal across the groups, the variation might possibly be explained by the combination between the applied gain and the activation function used. However, this does not follow from the fact that ReLU/He is the activation function used in the first two out of three best performing models. Perhaps, using Tanh individual yields a relatively large boost in performance, whereas it cannot keep up its strength when combined with other variations. By consequence, we must thus argue that further research is required to acquire more insights into *why* certain combinations work better than others.

8.1.3: Explaining Plateauing Model Performance

The improvement of model performance for the B-scan algorithm plateaus around 80% on average across ten folds. Although a deep neural network never achieves 100% - analogous to the fact that human beings make errors when inferencing in real life – CNNs can achieve accuracies in the range between 95 and 98% [46]. We will now attempt to explain the plateau observed during training.

8.1.3.1: Underfitting

As we observed in section 2.2.1.5, machine learning models can be underfit and overfit. In this section, we argued that the best-performing model provides the best balance between *predictive power* (i.e., how well it performs when predicting) and *generalization power* (i.e., how well it maintains this predictive power when it is used on data it has never seen before). Overfit models maximize predictive power while deteriorating on generalization power. Underfit models did not yet attain maximum predictive power.

For the B-scan based algorithm, training our model initially yielded accuracies of approximately 67%. Expanding our dataset from approximately 800 to approximately 2425 samples resulted in substantial

improvement, after which the variations applied to this extended dataset yielded small improvements but then plateaued again. It might therefore still be the case that our model is underfit, possibly explaining the plateau.

It is therefore worthwhile to retrain our models with a data set that has been expanded even further. There is however no one size fits all answer to the question how large data sets used in deep learning scenarios must be [101]. The size instead seems to be dependent on various considerations:

- How different the classes are in terms of input data;
- Whether techniques such as image augmentation can be applied to the training process;
- Whether training can be performed using a pretrained CNN;
- Whether BatchNormalization can be applied;
- …and more.

8.1.3.2: Electric Characteristics of Buried Materials

Applying image augmentation to GPR imagery is difficult, as we noted before. Rotating the image, for example, is not worthwhile, since hyperbola representing underground cables and pipelines are always south-opened. However, “how different the classes are in terms of input data” might enhance our explanation with respect to the plateau. It might be that the plateau is caused because the electric characteristics of buried materials overlap to some extent. For example, the conductivity of metallic objects such as perfect electricity conductors (PECs, e.g. steel) and less perfect ones (e.g. cast iron) might result in correlated signal backscatters. This correlation might be an additional explanation for the plateau and may be an additional motivation for expanding the data set even further. This can be done in multiple ways:

- Adding more of the same objects with additional random variations in soil type;
- Adding more of the same object with additional variations in noise available within the radargram;
- Adding additional objects where other layers, such as a change between soil types, are also present;
- Adding additional objects which are disturbed by the presence of other objects.

This way, we expect that our hypothesis with respect to possibly remaining underfitting can be validated. As a result, model performance might be improved.

8.2: Study Limitations

In this section we discuss the limitations of our study. Section 8.2.1 specifically covers the differences between our GprMax 2D simulations versus 3D radar signal emissions from GPR devices in the real world. The effects of signal noise on the performance of our models versus a real-world scenario is discussed in section 8.2.2. We present an issue discovered when applying time-varying gain in section 8.2.3 and discuss CNN hyperparameter tuning in section 8.2.4. Section 8.2.5 transforms the explanation with respect to possibly remaining underfitting into a study limitation and 8.2.6 discusses the extent to which the IA tool was validated.

8.2.1: GprMax 2D vs 3D in Real World

One of the primary limitations of our work is the fact that GprMax 2D was used for generating the simulations. This limitation was already introduced in section 4.2, as well as its rationale for moving forward nevertheless (that is, generating a sufficient amount of simulations with GprMax 3D would take approximately 12 to 36 weeks on a highly powered GPU).

Antennas simulated with GprMax 2D emit the output wave onto an (x, y) plane, which looks like a funnel. However, in real life, a wave is not emitted in two directions, but in three instead. The waveform in this (x, y, z) cube can thus be visualized as a flashlight.

We expect that this design choice did not if only scantily impacted the performance of our machine learning models. This argument is supported by the observation that the training, validation and test sets all contain simulations generated with GprMax 2D. By consequence, the idiosyncrasies with respect to one particular object are identical across the total data set, and patterns should be captured properly.

Another argument for this statement is that we used GprMax's built-in facilities for emitting a custom signal, which allowed us to mimic the 3D signal to a maximum extent. This was achieved by generating a so-called air shot with a real GSSI GPR at TerraCarta's premises in which the setup contained two antennas (an emitter and receiver) located closely together. The signal amplitudes, which together constitute a discrete set of points representing the signal, were converted into a continuous one by first applying the Discrete Cosine Transform to the signal amplitudes, and then applying the inverse DCT to find the same function, which is then continuous. Although very small, there is an offset between inverse DCT and the original signal. We do however expect that this does not impact signal quality.

This air shot based signal was used in GprMax 2D for signal emission. This way, we assume that 2D signals mimic real world signals to a maximum extent, minimizing the deviation between GprMax 2D simulations and real-world 3D signals. Early tests with real radargrams however demonstrate that the predictions generated by the IA tool become less accurate than with simulated radargrams. Why this is the case, and whether it occurs due to GprMax 2D vs 3D differences or other limitations discussed next, must be analyzed in future work.

By consequence, with respect to real-world IA Tool utilization, analysts must currently apply it with caution.

8.2.2: Noise Level of Simulations

Practitioners with TerraCarta B.V. regularly noted during our work that GPR imagery is sensitive to various types of signal interference. For example, reflections from objects nearby, tree roots present in the underground and electromagnetic signals such as cell phone signals can interfere with the signal.

This interference becomes visible on the radargram as noise. GPR imagery is by consequence often rather noisy when acquired in practice. Our simulations, on the other hand, did contain noise, but might still be too *standardized* with respect to real-world imagery. This is particularly the case with simulated ground bounce, which is relatively smooth rather than noisy. This might reduce IA tool effectiveness for real-life scenarios.

Fortunately, the convolution operation used when training our models is – given a sufficient amount of data – increasingly insensitive to noise, since it attempts to derive a generic image of an underground object. More traditional classes of machine learning models, such as SVMs, are more sensitive to noise than CNNs, due to their internal structure. By consequence, we can argue that our models are likely to be less sensitive to noise than the ones used in previous approaches.

Additionally, our data pipeline contained various preprocessing steps related to noise removal; particularly the ground bounce removal is a prime example of this. During data preprocessing, the ground bounce was removed using a median filter, which increasingly allows *distinct* objects to become more visible. Nevertheless, for real data, this might be insufficient, given non-smoothness of the ground bounce. Using

more advanced ground bounce removal filters might reduce the impact of this limitation in future work. Additionally, shallowly buried objects sometimes interfere with ground bounce and by consequence it cannot be ensured that the bounce is remove in its entirety. Those are pointers for future work.

8.2.3: Gain Issue: Non-Smooth Signal Ends

During the process of training the machine learning models, we noted for some A-scans that the visualized signal is no longer smooth for the end of the time domain. Image 36 presents such a signal which was gained with strong exponential gain, introducing inverted signal attenuation. We note small but significant malformations on the A-scan for samples >= 875. This effect is visible on the B-scan as well.

After analysis together with the geophysical expert at TerraCarta's, we found that a possible cause for this matter is the conversion process from GprMax simulation output files into GSSI-readable radar files, for which a third-party conversion script was used which we adapted. Specifically, GprMax's signal output is of the 32-bit floating point data type, which the conversion script converts into integers. By consequence, small amplitude changes (e.g. from 1.4999999(…) to 1.5) in GprMax will yield large swings in the conversion script (in this case, from 1 to 2). Particularly towards the end of the time domain, where substantially small amplitudes occur due to signal attenuation, this effect becomes visible when gain is applied.

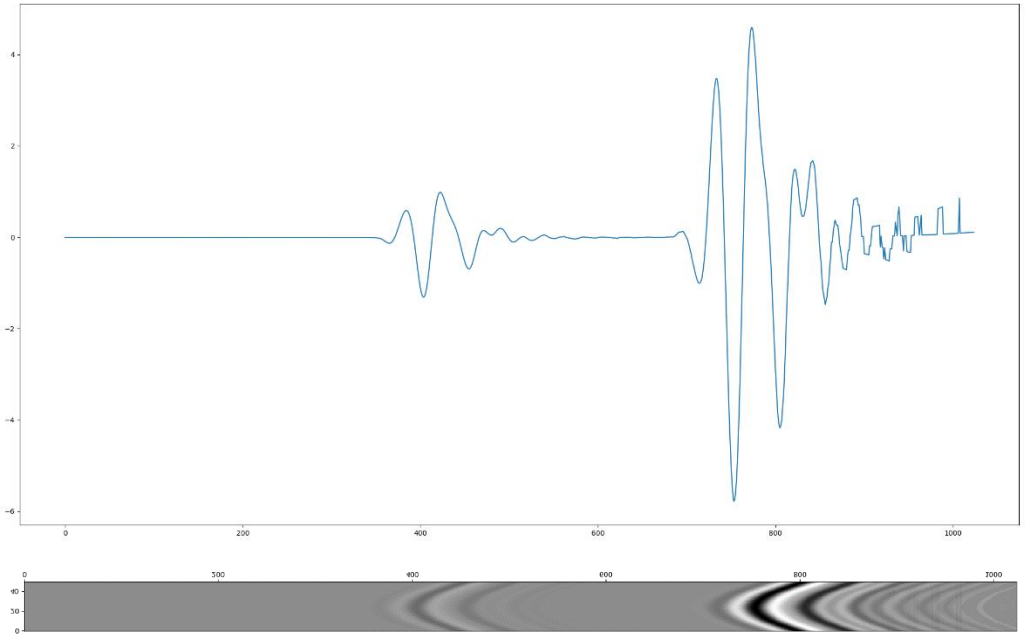


Image 36: *Signal issue visible near the end of the time domain, for A-scan and B-scan.*

Fortunately, since this issue occurs during global pre-processing, this deviation is *absolute* in the sense that all signals used for training our machine learning models are affected. By consequence, we can argue that although the likelihood that our training process was impacted because of this matter is relatively high, the impact on ML performance is likely low. This argument is further supported by the fact that the main objects are often buried higher in the ground and consequently become visible earlier in the time domain. They are then not impacted. Additionally, CNNs provide the benefit of location invariance; that is, the exact position of an object in the radar image is no longer of extreme importance. This way, we argue, we expect that these non-smooth signal ends do not substantially impact the performance of our machine learning models.

8.2.4: Manual vs Automated Hyperparameter Tuning

In this work, we noted that neural networks can be configured in multiple ways. One can for example change the *model architecture*, being the number of layers as well as their type and structure. Another approach influences *which parameters (i.e. neurons) are trained*. Finally, one can also choose to influence how the model performs its training and optimization process (through so-called *hyperparameters*).

With respect to the latter, we must bring forward another limitation of our work. Finding the optimum set of hyperparameters is essentially a search problem that is largely dependent on the data set used for training. In our work, we trained our models with manually selected hyperparameters, based on experience present with the author and findings discussed in chapter 5. In subsequent variations, we manually varied our hyperparameters based on intermittent results. Although this practice is acceptable to the machine learning community, so-called *AutoML* systems have also risen in popularity [46]. Those systems attempt to try and solve the search problem by suggesting a suitable architecture and related set of hyperparameters. By consequence, our results could likely be improved slightly further if such a training process was followed. However, given the research goals outlined in chapter 1, doing so was out of scope for our study. It is however a very interesting path for future work.

8.2.5: Is More Data Required?

In section 8.1, we already briefly highlighted the improvements observed when our data set was expanded from approximately 800 to 2425 samples. Additionally, we attempted to explain the plateau observed around 80% accuracies through the argument that overlapping electrical characteristics might correlate and by consequence confuse the model. The relatively low size of our data set – approximately 2500 samples – may be an explanation for this confusion, providing a limitation for our work.

Deep neural networks are however known to scale very well with increasingly large data sets [46]. In section 8.1.3, we provided various approaches to expanding our data set even further. Those render very interesting experiments for future work.

8.2.6: Limited Organizational Validation of IA Tool

Intelligence amplification by definition needs a human being whose knowledge must be amplified. In this work, we trained machine learning models which can automatically recognize the material types of underground objects. On top of those models, a web application was built which allows GPR analysts to analyze imagery generated in the field. This successfully demonstrates the technical feasibility of such an intelligence amplification scenario.

Beyond technical feasibility, we also proposed a business process embedding for the IA tool. This embedding clearly demonstrated the value an IA tool like ours can deliver to the GPR analysis business process. However, this work did only validate the tool with GPR analysts in a limited way, which is a research limitation. Through ADR principles, feedback was acquired from TerraCarta's upper management and a GPR analyst.

Although their feedback was positive, the work must be validated more thoroughly with more analysts. Adoption criteria for IA tools like ours could namely be sought after in future work, generating more insights in the *human* and *organizational aspects* of IA for geoscientists in general and GPR analysts in particular. Our prototypical IA tool can then be adapted to organizational needs and possibly be commercialized together with an industry partner.

9: Conclusion and Future Work

In the Netherlands, cables and pipelines are struck during excavation work every 3 to 4 minutes. Ensuing consequences yield 25 million Euros in annual material damages, with even higher immaterial costs. Strikes still happen because the KLIC, the Dutch registry for cables and pipelines, is not up-to-date or accurate, despite legislation that is ten years old. The reasons for those inaccuracies are e.g. georeferencing limitations for very old objects given technology developments back then. Organizational tension between the stakeholders involved with excavation work is also reported as a key cause for excavation damage.

As a result of the legislation mentioned, which requires parties performing excavation work to have a proper understanding of the underground infrastructure at a construction site, a business model has emerged for organizations which specialize in mapping such infrastructure. The primary partner for our work, TerraCarta B.V., is such an organization. It maps the underground through applying geophysical tools and techniques such as Ground Penetrating Radar (GPR) and Radio Detection (or, buried-utility locators). Given the fact that analysis of such measurements is labor-intensive and therefore costly and risk-inducing, TerraCarta spawned the question whether parts of the analysis process could be automated. Specifically, they inquired whether human beings can be spared from repetitive work to be fully deployed to complex tasks requiring creativity.

Techniques related to Artificial Intelligence, specifically machine learning and its sub branch deep learning, have achieved major breakthroughs over the past decade. The underlying technologies are hyped today but can still provide value to organizations if approached with common sense. That is, ML techniques are specifically useful in situations in which repetitive yet relatively straight-forward work occurs but often fail to deliver on their promise when their prediction task becomes more complex.

In this work, we integrated those statements and findings and set out to identify whether it is possible to employ those new technologies to *amplify the intelligence of a GPR analyst*, rather than to replace one. Specifically, we were interested whether it is possible to unburden analysts from the repetitive, easy analysis tasks, while still challenging them with tasks requiring creativity. Those research goals were consolidated into the following main research question (MQ):

> ***MQ.*** *How can machine learning techniques theoretically and practically amplify human intelligence in characterizing buried cables and pipelines?*

Before we can answer the MQ, we must answer the sub questions SQ1-SQ5 composing the MQ first.

> ***SQ1.*** *What are the most salient concepts in the field of Intelligence Amplification?*

Since World War 2, the nascence of the transistor has resulted in the era of computing technology. In the early days, being the 1940s, technologists considered automation to be a binary choice: either the path towards solving a problem was fully automated or it was fully left to human beings. This viewpoint was expanded in the early 1950s, when researchers provided heuristics to help make the decision between full or no automation.

Those heuristics were however criticized as they did not include scenarios in which one's tasks can be augmented by machines and vice-versa. As a consequence, so-called Levels of Automation emerged, where level 1 means 'no automation' and 10 'full automation'. The levels in between provide a weighted balance between human beings and machines.

In the 1950s and 1960s, the field of cybernetics thrived, or hyped, even. In this field, scholars discover and predict the future interrelationships between human beings and technology. Their dream was that the world would become an exponentially better place by means of machines. In this period, thought leaders such as William Rosh Ashby proposed ideas in which a strong relationship between men and machines is presented. By allowing machines to amplify human intelligence, Ashby argued that "[g]iven a little bit of [human intelligence], [the machine] will emit a lot of it". This way, machines can help human beings in solving their problems, which according to Ashby is mostly equal to picking one best solution out of a set of alternatives.

Licklider, another thought leader, argued in line with Ashby that human beings can be amplified by machines, but suggests that it is a bilateral relationship instead of an unilateral one. Additionally, Licklider argued that men have particular strengths whereas machines have other ones, and that they are perpendicular. By taking

an analogy with the biological process of symbiosis – “living together in intimate association (…) of two dissimilar organisms” – Licklider presented the vision that by coupling “human brains and computing machines (…) together very tightly, (…) the resulting partnership [may] think as no human brain has ever thought [before] (…)”. That is, when human beings and machines amplify each other, overall performance may be greater than the sum of its individual parts. The field of Intelligence Amplification was born.

Over the years, frameworks have emerged to conceptualize the components of Intelligence Amplification and their interrelationships. One particular framework was published in 1962 by Doug Engelbart as a result of U.S. government research efforts. The framework approached IA in a systems engineering fashion in which “humans do not exist singularly but rather as a part of a larger system consisting of a Human using Language, [Artefacts], and Methodology in which he is Trained”, or the H-LAM/T system.

The framework, which focused on problem solving, assumes that human actors use processes (“little steps of action”) when solving a problem, breaking them apart in smaller subprocesses. This allows humans to make processes reusable or to pick processes that have been defined by other people or artefacts. This way, one’s overarching process can be performed entirely by the human actor themselves, entirely by artefacts (i.e., by picking artefact-only subprocesses to execute the main process) or by a composite process, or IA process, which combines the strengths of men and machines.

Such a system of processes and subprocesses is impacted by time. That is, subprocesses change over time, human actors can learn about new processes, and so on. By consequence, Engelbart argues, the system continuously *co-evolves* over time. An interesting observation made is that given those interrelationships any action made by humans is reflected in technological developments. The future of technology developments may thus be more about the past than we think.

More recently, Engelbart’s IA framework was extended by Xia et Maas because technology has never been more personalized before. The principle of co-evolution remains valid, but becomes more personal. Today, rather than “how can IA change mankind?”, the question “how can IA change myself?” becomes increasingly important. They expand the range of IA artefacts from problem solving alone to augmenting one’s memory, increasing one’s motivation, assisting in decision making activities and improving one’s mood. This range of possible IA applications has never been larger before.

In another framework, Dobrkovic et al. take a rather pragmatic stance towards IA. They argue that “every problem can be decomposed in two or more [subproblems]”. The complexity and workload of those subproblems, taking into account the strengths of men and machines, subsequently determine whether an actor, an artefact or both should solve the subproblem. The authors call such artefacts the “AI system”, which today is often a machine learning driven system that automatically discovers regularities in data. Amplifying one’s intelligence rather than full automation has thus come a long way and given technology developments seems to be increasingly important in today’s automation dilemmas.

> ***SQ2.*** *How does underground intelligence through wave emission and reception work; especially, what data is produced?*

For answering this question, an inquiry was made with TerraCarta B.V. about which equipment is used for mapping the underground infrastructure. Primarily, Ground Penetrating Radar is used, augmented with buried-utility locators and the Dutch KLIC repository should difficulties arise during analysis. We therefore focused primarily on GPR, but also briefly covered buried-utility locators and the KLIC.

Ground Penetrating Radar is a geophysical technique that can be used for identifying underground objects. The device is equipped with antennas, at minimum one that transmits electromagnetic waves and one that receives them. Since buried objects have different electromagnetic properties than the medium in which they reside, echoes are produced when waves encounter those objects. In order to identify underground objects, a practitioner moves the device over the surface after which waves are transmitted and echoes received. For any position (x, y), those echoes are registered using the *time of arrival* method, often in nanoseconds.

Echoes received at one position can be visualized in a so-called A-scan. In this scan, the amplitudes of the echoes are plotted against the time of arrival. This provides insight in disturbances in the underground, possibly indicating the presence of underground objects.

A GPR device is moved over the surface horizontally. By consequence, a trail of A-scans can be visualized by plotting the A-scans vertically, with the trail itself on the horizontal axis. This visualization is known as a B-scan and presents a more detailed overview of the measurements performed, because the entire scan is now visualized together. Specifically, underground objects become visible as a hyperbola, whereas regular noise does not. This way, the analyst can visually distinguish where underground objects are located.

Although GPR is a promising technique for mapping the underground, it presents certain challenges. In fact, analyzing a GPR radargram is not as easy as it sounds. Among others, the close proximity of receiver antenna to transmitter antenna slightly bows up or down the signal visible in A-scans, called wow. Since a change in medium produces echoes, the small but present interface between air and the underground produces so-called ground bounce. This is often a burden for the analyst, especially when objects are shallow.

Additionally, the electromagnetic signal traveling through the underground attenuates with time, significantly reducing the strength of echoes from deeper down the underground. Medium conductivity also significantly impacts the GPR signal. In scenarios where two different topographies are present in one B-scan, this produces substantial offsets too, which must be accounted for. By consequence, analyzing a GPR radargram is more complex than one may initially realize.

Fortunately, various filters and other data processing techniques have been developed to reduce the impact of these challenges. Dewow filters can reduce the effect of wow, whereas a multitude of ground bounce removal filters has been developed as well. Signal attenuation can be reduced by applying gains such as linear gain and energy gain, which is exponential in nature. Topographic correction might be used when different topographies are present in the B-scan. This illustrates that although the challenges introducing complexity can be overcome, human creativity and insights need to remain part of the equation when producing tooling that automatically analyzes the material types of underground objects.

Another class of tools used for underground mapping are so-called buried-utility locators. They work by identifying electromagnetic disturbances caused by subsurface objects, particularly utilities. Magnetic-induction based locators generate a magnetic field which propagates into the ground, generating another concentric magnetic field around metallic underground objects. This field is subsequently received by a receiver, and signal strength is strongest directly on top of the object.

Power-cable locators are a subset of those locators, but use the magnetic field already produced by the 50-60 Hz frequency range of the electric currents on the power line. No transmitter is required in that case. By consequence, they cannot be used for other purposes.

Radio-frequency based tracking locators are similar to magnetic-induction based locators, but differ in that the transmitters are located within the pipeline. The signals emitted by the transmitters are intercepted by receivers, based upon which the object can be located.

Although the business case for underground utility mapping is evident, the market for GPR devices and buried-utility locators is relatively closed. Manufacturers all produce proprietary equipment, yielding a wide range of GPR devices and utility locators, all with different properties (e.g. frequencies used for wave emission) and proprietary data formats. This introduces additional complexity for the GPR analyst and, by consequence, our IA research goal.

> ***SQ3.*** *What are the state-of-the-art machine learning architectures used in characterizing underground objects using underground intelligence techniques; especially, GPR?*

Today's conceptualizations of IA prominently place an AI based system in the spotlight of amplifying human intelligence. Such AI systems, which are primarily machine learning systems today, are capable of automatically generating predictions for new data, providing an analysis of the data at hand. Whereas traditional AI systems were more symbolic in nature, those recent machine learning approaches favor a pattern recognition approach using large data sets. One particular class of machine learning approaches that has gained much popularity over the last 5 to 10 years is deep learning, in which – among others – deep neural networks are used for performing machine learning tasks. Given technology developments, those models can handle non-linear data natively (whereas this required the so-called kernel trick with SVMs) and scale well with extremely large datasets (a challenging task with SVMs).

It is therefore no surprise that machine learning approaches have been validated for analyzing the underground based on GPR imagery. Pasolli et al. group them into four groups, namely (1) object detection and localization, (2) object material recognition, (3) object dimension estimation and (4) object shape estimation. Most works have attempted to find methods for object detection and localization, today considered to be a trivial task given state-of-the-art object detectors available within the deep learning community.

The research goals of this work revolve around object material recognition, or characterization of the material type of a buried utility. The body of knowledge on this topic is substantially less extensive. Specifically, using a structured literature review, we noted that most works use Support Vector Machines for classifying the material types. Support Vector Machines are a somewhat older class of machine learning techniques and were popular approximately 10 years ago. However, results for relatively small datasets were promising, with studies showing accuracies of approximately 80%. The downside of SVMs is that they cannot scale with high data volumes and high data dimensionality. By consequence, various data pre-processing techniques were used to make the feature vectors (i.e., the inputs to the SVMs for training) sparser. Various statistical and signal processing approaches were used for this, with prime examples being signal histograms, the Discrete Cosine Transform, sparse representation of A-scans and normalized cropped hearts of the hyperbola apex.

With the nascence of deep learning, SVMs have lost their popularity in practice in favor of deep neural networks. However, the number of works investigating the effectiveness of this new class of models on object material recognition lags behind somewhat. Effectively turning object material recognition into an image recognition problem, convolutional neural networks have been the way forward in studies so far. Although results are promising, studies so far were limited by either the number of objects included, the computational requirements put forward by the networks trained or the necessity of substantial data transformation before CNNs could be applied. This provided an opportunity for our work for training and validating the effectiveness of simpler CNNs for object material recognition.

> ***SQ4.*** *Which machine learning model architectures are suitable for identifying and segmenting underground objects with ground penetrating radar?*

In this work, we proposed three new classes of algorithms for underground object material recognition. Two of the three classes were inspired by previous research and used signal histograms and Discrete Cosine Transform coefficients as feature vectors. The third used a window sliced around the hyperbolic signature as feature vectors. For all three algorithms, CNNs were used instead of SVMs for training the models. The models inspired by previous research used two 1-dimensional convolutional blocks; the latter one used three 2-dimensional blocks. All convolutional blocks were followed by densely-connected layers. Each block consisted of a convolutional layer (with respect to the dimensionality either 1D or 2D), max pooling and batch normalization. This configuration was chosen to maximize the model's pattern recognition capability while maintaining sparsity and reducing large variance during optimization.

Approximately 800 simulated objects were used as a training set. They were generated using GprMax 2D, which is simulation software for GPR imagery. The simulations were made using a custom wavelet derived from a real GSSI GPR due to differences between signal emission in GprMax 2D and real GPRs, which emit signals in 3D. Global pre-processing included ground bounce removal using a median based ground bounce removal filter, applying energy gain and feature-wise normalization. Per algorithm, this was followed by individual preprocessing, such as extracting a histogram, generating DCT coefficients and slicing a window around the hyperbola. This yielded three distinct data sets for training. K-fold cross validation with K = 10 was used for creating a more average view of model performance.

Beyond the model architecture and data set, the models were configured with respect to its hyperparameters. For this configuration, state-of-the-art insights from theory and practice were used to configure the initialization of neuron weights (He uniform initialization), loss function (categorical cross entropy), optimizer (Adam), activation functions (ReLU), learning rate (LR range test for determining optimum default LR a priori followed by LR decay during training), regularization (batch normalization and L2 regularization) and other parameters, such as batch size (70, minibatches of approximately 10%) and the number of epochs (200.000 with early stopping and model checkpointing callbacks attached).

Initially, model performance for all three algorithms was mediocre, with accuracies ranging between 60-70%. Various variations were performed to identify paths to improve performance without opening oneself to

overfitting. The first variation we performed was to expand the dataset to approximately 2425 simulations, since we observed that one simulation was unique with respect to the object's unique characteristics and by consequence could only be present in one set (i.e., training/validation or test set). This yielded substantial improvements for the B-scan window based algorithm while the other two did not improve.

Performing various additional variations subsequently did not produce new insights: the histogram and DCT based algorithms did not improve, whereas the B-scan based algorithm improved to accuracies of approximately 81.5% on average across ten folds, with outliers to 86%. Those variations included studying the effect of separating the material and its contents when generating target classes, increasing the decay of the learning rate, and studying the effects of varying input (i.e., applying different gains). The effects of varying the number of DCT coefficients, the number of histogram bins and the width of B-scan windows was also studied. Similarly, various model variations were studied, with variations in batch size across the three algorithms as well as differences in hyperparameters (with e.g. Swish and Leaky ReLU used instead of traditional ReLU to study sensitivity to the vanishing gradients problem and Tanh with Glorot uniform activation functions to study the effect of a more traditional activation function). Combining those variations into one yielded the most promising results in terms of loss; especially the B-scan based algorithm with (101, 1024) shape, ReLu/He, batch size = 50, linear gain and 175.000 times increased LR decay provided superior results.

Subsequent to the training stage, we were able to explain the differences in model performance by logical reasoning. With respect to the histogram based CNN algorithm and the DCT based CNN algorithm, we argued that mediocre model performance emerges because feature extraction is effectively applied twice: first by the histogram or DCT feature extractor and subsequently by the filters learnt by the CNN. Our assumption is that CNN filters become blind to idiosyncrasies present in the data when feature extractors are applied. This argument is strengthened by the increase in model performance observed when performing similar variations to the B-scan based algorithm, where the filters are the only feature extractors utilized. By consequence, we recommend with respect to applying CNNs to GPR imagery not to use any feature extraction techniques besides the convolutional filters themselves.

With respect to the state-of-the-art activation functions used, we noted that they did not substantially improve model performance. This effect partially emerges because substantial regularization is already employed by means of batch normalization and L2 regularization. On the other hand, the death of neural networks with ReLU as activation function prevented with Swish and Leaky ReLU merely occurs when neural networks are relatively deep. Our model architectures are not. Our recommendation is that Swish and Leaky ReLU must be used especially in large scale model and/or data scenarios, but that one could expect model performance to remain similar when more compact model architectures are used. For Tanh, improved model performance remained unexplainable, but could be related to strong regularization techniques applied.

Not surprisingly, model performance increased when the batch size did as well. This behavior is also logically sound: when the size of a minibatch is larger, a more accurate gradient can be computed, rendering more effective model optimization. A downside of large minibatches is a large memory footprint, which requires a machine learning engineer to balance between batch size and memory requirements. With respect to CNNs applied to GPR data, particularly raw B-scan windows, we recommend that an engineer experiments a priori with batch size, subsequently utilizing the best balance between batch size and hardware capabilities.

Model performance deteriorated when varied gains were applied. Specifically, no gain and strong exponential gain (introducing inverted signal attenuation) provided to be the worst-performing variations with respect to gain variations. The application of linear gain yielded similar performance. Visually, this could be explained because initial gain (light energy gain) produced B-scans that resembled linearly gained B-scans. Apparently, the CNNs derive the majority of their predictive power through the reflections of the objects represented in the GPR imagery. Secondary reflections, e.g. double ones down the time domain when pipelines are filled with water, seem to enhance the discriminative effect but are incapable of providing full discriminative power.

Generally, we can thus note that the class of B-scan window-based algorithms, which represents a pipeline of pre-processed but still semi-raw data before training the CNN, performs best, especially when the variations presented above are applied.

SQ5. *How can a PoC improve the GPR analysis processes at TerraCarta B.V. for underground cables and pipelines through amplifying human intelligence?*

We created a prototypical web application which allows GPR analysts to upload GSSI radar imagery into their web browsers. They can subsequently analyze hyperbolae with respect to their material type. By doing so, we demonstrated the technical feasibility of the IA scenario; i.e., that amplifying human intelligence with machine learning is possible and that it can be used by non-technical end users.

Our tool supports employees of underground mapping companies during their work. Specifically, by creating an architectural diagram of the generic process from customer inquiry to delivery of charts and customer reports, we demonstrated where our tool can benefit organizational processes. Primarily, value is created during the analysis subprocess, in which GPR data is loaded into analysis software and visually inspected. During this subprocess, the analyst initially classifies the radargrams using their background knowledge, using relatively limited contextual information. This information is added later, especially for hyperbolae that remain uncertain. Our tool can help in adding additional certainty for the analyst. Although it now runs in a different interface (i.e., the web browser), its backend was created to allow for relatively easy integration with existing tools (e.g., GSSI RADAN).

Systematically analyzing the effectiveness of this tool was out of scope for this study. Nevertheless, we performed some tests with real-world GPR data. Those tests demonstrated that the machine learning model underlying the IA tool becomes sensitive to the differences between real world data and simulations. Specifically, the model is less certain about the specific target, which introduces complexity for the end user. Future work should investigate the causes for those differences. We do however assume that the limitations of our work may play a role in explaining those differences – them being the differences between GprMax 2D simulations and real-world 3D signal emission, the level of noise, an issue with gain during conversion from GprMax output files into GSSI readable files, and the relatively small data set used for training. Reducing the impact of these limitations may even further enhance the business value created by our AI tool.

The tool was neither validated systematically in a work setting. However, our industry partner provided early validation comments, especially because artefact design and development was performed in an agile fashion using the ADR methodology. Those comments acknowledged the business value a tool like ours can provide for underground mapping organizations.

MQ. *How can machine learning techniques theoretically and practically amplify human intelligence in characterizing buried cables and pipelines?*

Altogether, we can conclude that theoretical developments since the 1940s have demonstrated that conceptually at first but realistically today, human intelligence can be amplified by machines, and vice versa. By taking a symbiotic path towards solving a problem or performing a task one's performance is suggested to be better together. Today, AI systems, specifically systems driven by machine learning, are the key driver for such intelligence amplification efforts, since they can perform tasks which are not appreciated by humans. Interestingly, they show inadequate capabilities for tasks where humans are in fact strongest. As such, a promising situation for symbiosis emerges in many settings today.

To investigate the feasibility of applying IA in a GPR analysis setting, we successfully trained convolutional neural networks for object material recognition, validating the effectiveness of many variations in the process. We designed and validated three classes of CNNs, inspired by works performed previously that used SVMs. Specifically, we trained CNNs based on signal histograms, based on Discrete Cosine Transform coefficients and based on slices of semi-raw (i.e. slightly preprocessed) data.

From these, the class of B-scan window based algorithms performed best, with average accuracies across ten folds per trained variation plateauing around 81.5%. The other classes of algorithms lagged behind in performance with accuracy plateaus around 60%. We argued this behavior emerged because essentially, feature extraction is applied twice.

Given the performance of our models, we created a web application for intelligence amplification for demonstrating feasibility of the IA scenario. The GPR analyst can upload their radar imagery into the web

browser after which it becomes visible on-screen. By clicking any of the hyperbolic signatures visible to the analyst, a prediction is generated by the machine learning model. This way, we demonstrated that an analyst's intelligence can successfully be amplified through machine learning models using knowledge acquired previously. The tool performs well when simulations are used for analysis, but performance slightly deteriorates when real-world GPR imagery is used. Early validation comments by our industry partner acknowledged the possible business value of such tools. This concludes the full cycle from early ideation to prototype development and (early) validation.

9.1: Contributions

Our work contributes to the scientific community and practice in multiple ways:

- First, we demonstrated that it is possible to apply deep convolutional neural networks to GPR based object material recognition. Although this was already achieved in previous studies, the targets trained for then were both low in volume (often, approximately 3 classes) and correlating electromagnetic properties (e.g., air, limestone and metal, which differ substantially). In our work, we included more classes and trained with correlating properties. Still, the promise for machine learning based object recognition remains in place, but we recognize that more work is required.
- Second, we introduced an entirely new approach to object material recognition automation into the scientific fields related to GPR and GPR analysis. Specifically, we take the stance that human beings should not be replaced during GPR analysis, but rather amplified with knowledge acquired in the past.
- Third, we demonstrated the feasibility of this approach by designing and developing an intelligence amplification tool. This tool, which can be cloud based or installed on-premise and runs in one's web browser, allows the analyst to generate material predictions in real time, based on one of the machine learning models embedded into the software.
- Fourth, we successfully demonstrate through our application of Riverflow AI that challenges with respect to technical debt in machine learning systems can be successfully managed. Specifically, using Riverflow AI reduces interface heterogeneity by wrapping the models in a HTTP REST based container, reduces the risk for glue code by providing a model embedding framework inside the container and allows TerraCarta to analyze model utilization and model performance in a later stage.
- Fifth, our work has allowed TerraCarta – our industry partner – to work cooperatively on creating and validating new innovations that may optimize their business processes.

9.2: Suggestions for Future Work

We propose multiple paths for future research activities to be undertaken:

- First, we suggest that the data set is expanded even further. This might further increase the predictive power of the machine learning models. This is especially evident because deep neural networks scale well with increasing data sets and our results suggest plateaus occurring around 80% accuracies. Therefore, under the condition that the classes remain well-balanced for reasons of generalization power, the more data is available, the better. Specifically, it could be expanded as follows:
    - Adding more of the same objects with additional random variations in soil type, and radargrams where noise is stronger;
    - Adding more of the same objects with additional variations in noise available within the radargram;
    - Adding additional objects where also other layers, such as a change between soil types, are present;
    - Adding additional objects which are disturbed by the presence of other objects.
- Second, we suggest that the effect of varying hyperparameters is studied with an even wider set of hyperparameters used. That is, it is worthwhile to investigate whether more state-of-the-art hyperparameters such as newer optimizers can further improve model performance. Additionally, the relationships between using Tanh and Batch Normalization must be analyzed in more detail. Next to tuning the hyperparameters manually, future research activities might also focus on applying automated hyperparameter tuning and automated selection of model architecture, i.e. AutoML tooling, to the problem of GPR based object material recognition.

- Third, although we argue that applying exponential gain with a normal coefficient yields best performance because all objects are reflected equally, the validity of this claim must be numerically analyzed in future work. This can provide new insights to the GPR community, which now often already applies gain on the GPR device.
- Fourth, the combinations of variations yielding better model performance should be investigated in future research activities, both in terms of the number of variations and the reasons as to why they perform better. Our results have demonstrated that combining individual variations and retraining different models improves the models substantially based on decreasing entropy. More research can attempt to retrieve variations which perform even better.
- Fifth, we suggest that the models are retrained with real-world data to pinpoint whether slightly deteriorated model performance in the IA tool is caused by the differences between GprMax 2D data and real-world data. This provides new insights as to whether GprMax 2D is a suitable tool to replace real-world imagery, possibly allowing underground mapping companies to expand their data sets substantially.
- Sixth, we suggest that the models are retrained with simulated data where the issue with respect to signal gain is removed. This can be done by applying the gain *before* converting the data from GprMax output files into GSSI readable files, but this requires modifications to the conversion scripts, which is relatively complex.
- Seventh, if the models can be improved even further, we suggest that future research activities focus on more thoroughly validating the IA tool built during this work. Although we demonstrated technical feasibility and pinpointed where the tool can be embedded in an analysis process, it was not validated with practitioners. Future research activities can generate design insights relevant to the GPR and geoscience communities with respect to the tooling they use.

References

[1] Central Intelligence Agency, "CIA World Factbook," [Online]. Available: https://www.cia.gov/library/publications/the-world-factbook/geos/nl.html. [Accessed 17 January 2019].

[2] United Nations Development Programme, "Human Development Indices and Indicators: 2018 Statistical Update," 2018.

[3] NOS, “1 op de 8 Vlaamse huizen niet aangesloten op riool, hoe zit dat in Nederland?,” 11 January 2019. [Online]. Available: https://nos.nl/artikel/2267049-1-op-de-8-vlaamse-huizen-niet-aangesloten-op-riool-hoe-zit-dat-in-nederland.html.

[4] Dutch National Government, [Online]. Available: https://www.rijksoverheid.nl/onderwerpen/bodem-en-ondergrond/graafschade. [Accessed 17 January 2019].

[5] Central Statistics Bureau of the Netherlands, "Bedrijfsleven; arbeids- en financiële gegevens, per branche, SBI 2008," 29 March 2018. [Online]. Available: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/81156NED/table?dl=F052. [Accessed 17 January 2019].

[6] J. v. Zanden, The Economic History of the Netherlands 1914-1995, Routledge, 1998.

[7] P. v. d. Eng, De Marshall-hulp, Een Perspectief voor Nederland, 1947-1953, Houten: De Haan, 1987.

[8] Kadaster, "Klic-wab: het registreren van uw belang via Internet," 2008.

[9] Dutch National Government, "Wet informatie-uitwisseling ondergrondse netten," 7 February 2008. [Online]. Available: https://wetten.overheid.nl/BWBR0023775/2014-01-25. [Accessed 17 January 2019].

[10] Dutch National Government, "Wet informatie-uitwisseling bovengrondse en ondergrondse netten en netwerken," 1 January 2019. [Online]. Available: https://wetten.overheid.nl/BWBR0040728/2019-01-01. [Accessed 17 January 2019].

[11] H. M. Jol, Ground Penetrating Radar: Theory and Applications, Elsevier, 2009.

[12] "Machine learning: Trends, perspectives, and prospects," *Science,* no. 6245, pp. 255-260, 2015.

[13] X. L. Travassos, S. L. Avila and N. Ida, "Artificial Neural Networks and Machine Learning techniques applied to Ground Penetrating Radar: A review," *Applied Computing and Informatics,* 2018.

[14] M. Almaimani, "Classifying GPR images using convolutional neural networks," The University of Tennessee at Chattanooga, Chattanooga, Tennessee, 2018.

[15] M. Szymczyk and P. Szymczyk, "Neural networks based method for automatic classification of GPR data," in *AIP Conference Proceedings*, 2019.

[16] M.-T. Pham and S. Lefèvre, "Buried Object Detection from B-Scan Ground Penetrating Radar Data Using Faster-RCNN," in *IEEE International Geoscience and Remote Sensing Symposium*, 2018.

[17] R. J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Berlin Heidelberg: Springer-Verlag, 2014.

[18] T. S. Kuhn, The Structure of Scientific Revolutions, Chicago: The University of Chicago, 1962.

[19] W. Isaacsson, The Innovators: How a Group of Hackers, Geniuses and Geeks Created the Digital Revolution, Simon & Schuster, 2014.

[20] H. Lasi, P. Fettke, T. Feld and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering,* vol. 2014, no. 4, pp. 239-242, 2014.

[21] M. Xu, J. M. David and S. H. Kim, "The Fourth Industrial Revolution: Opportunities and Challenges," *International Journal of Financial Research,* vol. 9, no. 2, pp. 90-95, 2018.

[22] A. Diez-Olivan, J. Del Ser, D. Galar and B. Sierra, "Data fusion and machine learning for industrial prognosis: Trends and perspectives towards Industry 4.0," *Information Fusion,* vol. 50, pp. 92-111, 2019.

[23] H. Kagermann, W. Wahlster and J. Helbig, "Recommendations for implementing the strategic initiative INDUSTRIE 4.0," acatech - National Academy of Science and Engineering, 2013.

[24] S. Jeschke, C. Brecher, H. Song and D. B. Rawat, Industrial Internet of Things and Cyber Manufacturing Systems, Cham: Springer, 2018.

[25] X. Wang, W. Feng, L. Hou, W. Wang, Y. Li and B. Liu, "Low power consumption wide area system design based on lora," *Technical Bulletin,* vol. 55, no. 17, pp. 315-320, 2017.

[26] I. Lee, "Big data: Dimensions, evolution, impacts, and challenges," *Business Horizons,* vol. 60, pp. 293-303, 2017.

[27] R. L. Brandt, The Google Guys: Inside the Brilliant Minds of Google Founders Larry Page and Sergey Brin, Random House, 2011.

[28] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST, 2011.

[29] H. Chen, R. H. L. Chiang and V. C. Storey, "Business Intelligence and Analytics: From Big Data to Big Impact," *MIS Quarterly,* vol. 36, no. 4, pp. 1-24, 2012.

[30] A. G. Frank, L. S. Dalenogare and N. F. Ayala, "Industry 4.0 technologies: Implementation patterns in manufacturing," *International Journal of Production Economics,* no. 210, pp. 15-26, 2019.

[31] C. Britton and P. Bye, IT Architectures and Middleware, Boston: Addison-Wesley Educational Publishers Inc, 2004.

[32] M. Cummings, "Man versus Machine or Man + Machine?," *IEEE Intelligent Systems,* vol. 2014, no. 5, pp. 62-69, 2014.

[33] P. Fitts, "Human engineering for an effective air-navigation and traffic-control system," Division of the National Research Council, Oxford, England, 1951.

[34] R. Parasuraman, T. B. Sheridan and C. D. Wickens, "A Model for Types and Levels of Human Interaction with Automation," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,* vol. 30, no. 3, pp. 286-297, 03 May 2000.

[35] F. W. Geels, "The dynamics of transitions in socio-technical systems: A multi-level analysis of the transition pathway from horse-drawn carriages to automobiles (1860–1930)," *Technology Analysis & Strategic Management,* vol. 17, no. 4, p. 445–476, 2005.

[36] D. A. Döppner, R. W. Gregory, D. Schoder and H. Siejka, "Exploring Design Principles for HumanMachine Symbiosis: Insights from Constructing an Air Transportation Logistics Artifact," in *Thirty-Seventh International Conference on Information Systems*, Dublin, 2016.

[37] W. Ashby, An Introduction to Cybernetics, London: Chapman & Hall Ltd., 1957.

[38] J. Licklider, "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics,* pp. 4-11, 1960.

[39] D. Griffith and F. L. Greitzer, "Neo-Symbiosis: The Next Stage in the Evolution of Human Information Interaction," in *Selected Readings on the Human Side of Information Technology*, IGI Global, 2009, pp. 410-424.

[40] D. Williams, M. Couillard and S. Dugelay, "On human perception and automatic target recognition: strategies for human-computer cooperation," in *International Conference on Pattern Recognition*, 2014.

[41] D. Engelbart, "Augmenting Human Intellect: A Conceptual Framework," Washington, D.C., 1962.

[42] C. Xia and P. Maes, "The Design of Artifacts for Augmenting Intellect," in *Augmented Human International Conference*, Stuttgart, 2913.

[43] A. Dobrkovic, L. Liu, M.-E. Iacob and J. Van Hillegersberg, "Intelligence Amplification Framework for Enhancing Scheduling Processes," in *Ibero-American Conference on Artificial Intelligence*, San Jose, 2016.

[44] C. M. Bishop, Pattern Recognition and Machine Learning, New York: Springer Science+Business Media, LLC, 2006.

[45] G. Zhang, E. Patuwo and M. Y. Hu, "Forecasting with artificial neural networks: The state of the art," *International Journal of Forecasting,* vol. 14, pp. 35-62, 1998.

[46] F. Chollet, Deep Learning with Python, New York: Manning Publications Co., 2018.

[47] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[48] A. R. S. Parmezan, V. M. Souza and G. E. Batista, "Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model," *Information Sciences,* vol. 484, pp. 302-337, 2019.

[49] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review,* vol. 65, no. 6, pp. 386-408, 1968.

[50] J. Jordan, "Common architectures in convolutional neural networks," 19 April 2018. [Online]. Available: https://www.jeremyjordan.me/convnet-architectures.

[51] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO—Object Detection Algorithms," 9 July 2018. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[52] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," in *Proceedings of the IEEE*, 1998.

[53] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, Lake Tahoe, Nevada, 2012.

[54] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2015.

[55] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.

[56] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," 2016.

[57] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[58] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 2016.

[59] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector," 2015.

[60] M. Graczyk, T. Lasota, B. Trawiński and K. Trawiński, "Comparison of Bagging, Boosting and Stacking," 2010.

[61] J. J. Daniels, "Ground Penetrating Radar Fundamentals," 2000.

[62] B. Scheers, Ultra-Wideband Ground Penetrating Radar, with Application to the Detection of Anti Personnel Landmines, 2001.

[63] S. Kandul, M. Ladkat, R. Chitnis, R. Sane and D. Varpe, "Developing Scans from Ground Penetrating Radar Data for Detecting Underground Target," in *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, 2018.

[64] J. E. Lucius and M. H. Powers, "GPR Data Processing Computer Software for the PC," U.S. Geological Survey, Denver, 2002.

[65] S. R. Bigl, "Locating buried utilities," US Army Corps of Engineers, Hanover, 1985.

[66] E. Pasolli, F. Melgani and M. Donelli, "Automatic Analysis of GPR Images: A Pattern-Recognition Approach," in *IEE Transactions on Geoscience and Remote Sensing*, 2009.

[67] N. P. Singh and M. J. Nene, "Buried Object Detection and Analysis of GPR Images: Using Neural Network and Curve Fitting," in *International Conference on Microelectronics, Communication and Renewable Energy*, 2013.

[68] X. Núñez-Nieto, M. Solla, P. Gómez-Pérez and H. Lorenzo, "GPR Signal Characterization for Automated Landmine and UXO Detection Based on Machine Learning Techniques," *Remote Sensing,* vol. 6, pp. 9729-9748, 2014.

[69] Q. Dou, L. Wei, D. R. Magee and A. G. Cohn, "Real-Time Hyperbola Recognition and Fitting in GPR Data," in *IEEE Transactions on Geoscience and Remote Sensing*, 2017.

[70] L. E. Besaw and P. J. Stimac, "Deep Convolutional Neural Networks for Classifying GPR B-Scans," in *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets*, 2015.

[71] F. Picetti, G. Testa, F. Lombardi, P. Bestagini, M. Lualdi and S. Tubaro, "Convolutional Autoencoder for Landmine Detection on GPR Scans," in *International Conference on Telecommunications and Signal Processing*, 2018.

[72] D. Reichman, L. M. Collins and J. M. Malof, "Some Good Practices for Applying Convolutional Neural Networks to Buried Threat Detection in Ground Penetrating Radar," in *9th International Workshop on Advanced Ground Penetrating Radar*, 2017.

[73] J. Bralich, D. Reichman, L. M. Collins and J. M. Malof, "Improving Convolutional Neural Networks for Buried Target Detection in Ground Penetrating Radar Using Transfer Learning Via Pre-training," in *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets*, 2017.

[74] M. S. El-Mahallawy and M. Hashim, "Material Classification of Underground Utilities From GPR Images Using DCT-Based SVM Approach," in *IEEE Geoscience and Remote Sensing Letters*, 2013.

[75] W. Shao, A. Bouzerdoum and S. L. Phung, "Sparse Representation of GPR Traces With Application to Signal Classification".

[76] E. Pasolli, F. Melgani and M. Donelli, "Gaussian Process Approach to Buried Object Size Estimation in GPR Images," *IEEE Geoscience and Remote Sensing Letters,* vol. 7, no. 1, pp. 141-145, 2009.

[77] Y. Zhang, D. Huston and T. Xia, "Underground Object Characterization based on Neural Networks for Ground Penetrating Radar Data," in *Nondestructive Characterization and Monitoring of Advanced Materials, Aerospace, and Civil Infrastructure*, 2016.

[78] K. Peffers, T. Tuunanen, M. A. Rothenberger and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems,* Vols. 2007-8, no. 3, pp. 45-77, 2007.

[79] A. R. Hevner, S. T. March, J. Park and S. Ram, "Design Science in Information Systems Research," *Management Information Systems Quarterly,* vol. 2004, no. 28, 2004.

[80] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi and R. Lindgren, "Action design research," *MIS Quarterly,* vol. 2011, no. 35, pp. 37-56, 2011.

[81] S. W. Jaw and M. Hashim, "Ground Penetrating Radar Backscatter for Underground Utility Assets Material Recognition," 2012.

[82] A. Giannopoulos, "Modelling ground penetrating radar by GprMax," *Journal of Construction and Building Materials,* vol. 19, no. 10, pp. 755-762, 2005.

[83] A. Strange, J. C. Ralston and V. Chandran, "Signal Processing to Improve Target Detection Using Ground Penetrating," 2002.

[84] S. Tjora, E. Eide and L. Lundheim, "Evaluation of Methods for Ground Bounce Removal in GPR Utility Mapping," 2004.

[85] R. Khandelwal, "K fold and other cross-validation techniques," Data Driven Investor, [Online]. Available: https://medium.com/datadriveninvestor/k-fold-and-other-cross-validation-techniques-6c03a2563f1e.

[86] S. K. Kumar, "On weight initialization in deep neural networks," 2017.

[87] N. Doshi, "Medium.com," 26 March 2018. [Online]. Available: https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94. [Accessed 27 April 2019].

[88] Neural Networks and Deep Learning, "Why are deep neural networks hard to train?," [Online]. Available: http://neuralnetworksanddeeplearning.com/chap5.html.

[89] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[90] D. P. Kingma and J. L. Ba, "Adam: A Method For Stochastic Optimization," in *ICLR*, 2015.

[91] S. J. Reddi, S. Kale and S. Kumar, "On the convergence of Adam and beyond," in *ICLR*, 2018.

[92] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 - Learning rate, batch size, momentum, and weight decay," US Naval Research Laboratory, 2018.

[93] Google, "Regularization for Sparsity: L_1 Regularization," [Online]. Available: https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization.

[94] R. Khandelwal, "L1 and L2 Regularization," Data Driven Investor, [Online]. Available: https://medium.com/datadriveninvestor/l1-l2-regularization-7f1b4fe948f2.

[95] Enhance Data Science, "Machine Learning Explained: Regularization," [Online]. Available: http://enhancedatascience.com/2017/07/04/machine-learning-explained-regularization/.

[96] X. Li, S. Chen, X. Hu and J. Yang, "Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift," 2018.

[97] P. Ramachandran, B. Zoph and Q. V. Le, "Swish: a Self-Gated Activation Function," 2017.

[98] B. Xu, N. Wang, T. Chen and M. Li, "Empirical Evaluation of Rectified Activations in Convolution Network," 2015.

[99] "Convolutional Neural Networks for Visual Recognition," CS231n, [Online]. Available: http://cs231n.github.io/neural-networks-1/.

[100] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," in *Conference on Neural Information Processing Systems*, 2015.

[101] "What is the minimum sample size required to train a Deep Learning model - CNN?," ResearchGate, [Online]. Available: https://www.researchgate.net/post/What_is_the_minimum_sample_size_required_to_train_a_Deep_Learning_model-CNN.

[102] itdxer, "What is batch size in neural network?," [Online]. Available: https://stats.stackexchange.com/q/153535.

UNIVERSITY OF TWENTE.