Technische Universität München
TUM School of Engineering and Design

Aspects of Algorithmic Information Theory in Spatial Machine Learning

Gabriel Dax

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 24.10.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 05.03.2024 angenommen.

Abstract

In spatial computing, data-driven systems are used to process a large amount of data and to make an assumption about its context. The complexity of the machine learning algorithms and the increasing size of the datasets leads to the need to scale the computational hardware. While this enables time-efficient processing or even the possibility of computing it, this is also connected with economic consequences in terms of having higher operational costs. Additionally, the real-time availability, accessibility of data, and reproducibility of experiments is becoming a non-trivial challenge. The use of compression is able to optimize memory and computational efficiency for a data-driven system without the need to scale the hardware. Moreover, especially the compression of algorithms is necessary for the deployment to edge computing hardware. While the domain of compression is highly researched for every single element within the data-driven system, limited research focuses on the domain of spatial computing: the compression of all aspects within the system or the combination of different technologies. Therefore, it is from importance to investigate the role of compression in spatial computing. For this purpose, a data-driven system and its main components in the context of machine learning are defined. While the theoretical perspective provides an in-depth analysis of the compression possibilities, the combination, and the impact on the entire system, the application examples do give a foundation for using compression in spatial computing. Furthermore, the results showed that compression could be used to optimize the time and energy efficiency by sacrificing a comparatively small amount of precision on the computed outcome. Moreover, this enables the deployment of edge computing hardware which is able to compute in constrained environments, for example onboard processing units in spacecrafts.

Zusammenfassung

Um große Datenmengen zu verarbeiten und Annahmen über deren Kontext zu treffen, werden im Spatial Computing datengesteuerte Systeme eingesetzt. Bedingt durch die Komplexität der Algorithmen des maschinellen Lernens und die Größe der Datensätze wird die Skalierung der Rechenhardware notwendig. Dadurch wird zwar eine (zeiteffiziente) Verarbeitung möglich, jedoch steigen dadurch die operativen Kosten. Die Echtzeitverfügbarkeit, Zugänglichkeit und Reproduzierbarkeit von Daten werden zu zusätzlichen komplexen Herausforderungen. Durch den Einsatz von Kompression können die Speicher- und Recheneffizienz von datengesteuerten Systemen optimiert werden, ohne die Hardware zu skalieren. Außerdem ist die Komprimierung der Algorithmen notwendig, um auf Edge-Hardware eingesetzt werden zu können. Die Forschung widmet sich intensiv der Komprimierung für jedes einzelne Element, jedoch nur begrenzt im Bezug zu Spatial Computing, insbesondere die Komprimierung aller Aspekte innerhalb des Systems, oder der Kombination verschiedener Technologien. Daher ist es wichtig die Rolle der Kompression im Spatial Computing zu untersuchen. Im Kontext des maschinellen Lernens wird zu diesem Zweck ein datengetriebenes System und seine Hauptkomponenten definiert. Die theoretische Perspektive liefert eine eingehende Analyse der Kompressionsmöglichkeiten, der Kombination und der Auswirkungen auf das Gesamtsystem, während die Anwendungsbeispiele eine Grundlage für den Einsatz der Kompression im Spatial Computing geben. Des Weiteren zeigen die Ergebnisse, dass die Komprimierung zur Optimierung der Zeit- und Energieeffizienz ohne große Verluste der Genauigkeit bei den berechneten Ergebnissen genutzt werden kann. Zusätzlich wird dadurch der Einsatz von Edge-Hardware ermöglicht, die in der Lage ist, in extremen Umgebungen, wie beispielsweise im Weltraum, zu rechnen.

Contents

Aknowledgement

First of all, my special thanks to Prof. Dr. Martin Werner, who did not only give me a chance to do my doctorate but also supported me through all the years and always had an open ear for me. Additionally, I would like to thank my colleagues at the professorship for Big Geospatial Data and all over the department Aerospace, both scientific and administrative. Primarily, Dr. Hao Li was a significant support for me during the time we worked together.

Last but not least, I want to thank my family, friends, and all other loved ones who have supported me through the years with their love and understanding.

1 Introduction

Artificial intelligence (AI) technologies have increased in importance and is used in a rising number of domains, such as environmental research like chemistry. Furthermore, this technology permeates the workflows of the industry, where processes are automated, supply chains optimized, or customer services are done with chatbots. Moreover, during the last few years, the publicity of AI has increased dramatically because AI reached society in the form of, for example, chatbots that are able to provide (comparably accurate) domain knowledge in a wide range of fields. Nevertheless, experts within this domain said that there are also certain risks regarding the employ and the development of especially artificial general intelligence that needs to be considered, like other societal-scale risks[1]. Furthermore, next to these consequences that may lay in the future, the rapid development of current state-of-the-art algorithms faces the issue of requiring a large amount of hardware and economic resources. Especially when training a deep learning model, large computational resources are required, for example, when training a machine learning model that segments the content of images. The increasing complexity of the designed architectures and sizes of the datasets led to a need to scale the computational components, either vertically to horizontally. An economic as well as ecological problem that occurs is that the operation of especially non-consumer grade hardware can be an investment itself. For example, a recently published chatbot that uses a large language model, that is globally applied by a large number of queries each day, has estimated operational costs of a low to medium three-digit million euro amount each month. All this, leads to a lack of resources in the future.

The number of fields that apply machine learning to extract knowledge from data is too large to be covered in general. Therefore it makes sense to narrow the focus to a single research field, namely spatial computing. One

[1]Center For AI safety: Statement on AI Risk (https://www.safe.ai/statement-on-ai-risk) [Accessed on 08.06.2023], The New York Times: What Exactly Are the Dangers Posed by AI? (https://www.nytimes.com/2023/05/01/technology/ai-problems-danger-chatgpt.html) [Accessed on 08.06.2023]

representative of this community is the domain Earth observation, where machine learning algorithms are used to investigate the content of the data that has been produced by remote devices. Furthermore, satellites are orbiting the Earth and are constantly producing data that need to be transferred. Each time a satellite crosses the visibility window of a base station, data can be up- and downloaded over the communication channel, which is the bottleneck in the system due to the limited bandwidth, that is provided as well as the physical time delay of the transmission. The amount of data that is produced is massive– in 2018 the German Aerospace Center estimated to reach 180 petabytes of remote sensing data in the year 2023 [1]. This task of transmitting the data to Earth and storing them is not only an economic problem but also a challenge to keep the data permanently and eminently accessible. In this particular problem, the utilization of compression could optimize the workflow of this data-driven system. For example a satellite that monitors the Earth's surface to detect wildfires (where this particular event is very rare, due to the large quantity of scenarios where a fire is unlikely, such as in winter or non-forest areas). Therefore, the largest part of the data that is transmitted does contain only a small portion of relevant information that is related. The amount of data that is transmitted down to Earth could be limited by deploying a machine learning model on board this satellite that determines whether the lastly sensed scene is suspicious. If this is the case, the product is selected to be transmitted to the ground. As a consequence, the deployment of machine learning algorithms in space is able to reduce the communication amount besides saves resources on Earth.

While this particular problem focuses on a single aspect of such a data-driven system, the use of compression applied to each element is able to reduce the computational effort that needs to be invested to reach a specific target goal. Additionally, the consequences are that the required hardware– and economic resources are minimized. Therefore the objective of this thesis is ***to investigate the role of compression in spatial computing***.

When considering the entire data-driven system's processing pipeline in Earth observation, there is limited to no research investigating the impact of compression. While the training and inference of deep learning models on edge computing hardware increase in interest [2], [3] and often require to compress the model itself. A large number of research projects use the full complexity of the data [4] including the model [5]. It is questionable which portion of information of each element in the dataset is meaningful and relevant to achieve a particular program's output. A data-driven system can be divided into its main components, namely the input, the operation, and the output, where each one represents an individual discipline that is not necessarily related to Earth observation. Moreover, the impact of compression is well-researched for each individual discipline.

The first part of a data-driven system is the input data itself, which can have various shapes and structures. While trajectories can be employed to predict the mobility type, another discipline is to investigate the content of images. In previous work [6], our group provided an approach to reduce the memory footprint of Twitter data by embedding them into a probabilistic data structure. In contrast, the compression of especially computer vision images is widely established, for example, is the file format JPEG commonly employed for smartphone cameras. Furthermore, [7] uses this file format to compress images that serve as a basis to train a machine learning model and investigate the impact. On the other side, the authors of [8] applied quantization as well as pruning on images before classification, where aggressive compression rates have a significant impact on the prediction performance. While there are projects that focus on the use of compressive file formats, only a portion do consider the utilization of spatial data. A representative for the latter one is the Sentinel-2 imagery, which is provided as a ZIP archive including meta data and the bands as individual JPEG2000 files with 13 bands and a depth of 16 bits per channel [9]. A wide range of remote sensing datasets [10], [4] do store the bands/scenes using TIFF files.

The second part of a data-driven system is the processing, which can be, for example, a deep learning model where the weights are the parameters. The whole complexity of such a model uses a 32-bit floating point number. Compression is categorized into two main categories, quantization, and pruning. The authors of [11] furnish an algorithm that falls in the first category and is called Distance-Aware-Quantization, which addresses the weights as well as the activation. In contrast, [12] states that the quantization error relates to the accuracy after training. Therefore, a method is proposed that applies a permutation of the weights to find optimal combinations. While those methods reduce the complexity that is available to represent a single weight, pruning removes parts of the model that are of less importance for the final classification result. The authors of [13] provide a method that combines both strategies: channel-wise pruning selects areas within filters that are then quantized.

The third part of a data-driven system is the output that is generated by the processing unit. While there are lots of different output types, such as a generated text or the estimated class of an image, it needs to be determined whether the outcome is trustworthy or not. The authors of [14] faced the issue of having a large amount of noise within their label. Therefore they used abstaining to overcome this issue and provide a model that is more robust than before with the costs of having less training data.

Considering all the achievements and active research that is currently done in this domain, spatial computing faces three major challenges: (1) compression is traditionally applied only on a small number of components within the system, leading to ineffective computing and unnecessary scaling of the hardware; (2) the consideration of the full complexity of the given data can lead to the problem of taking information into account that is not effectively supporting a machine learning operation, some information even can have a negative impact on the outcome itself; (3) the computation in a constrained environment such as space has limitations in terms of energy consumption and hardware characteristics. Therefore, it is required to scale the model instead of maximizing the hardware aspects.

Based on these considerations, this thesis provides an investigation of how compression can be used to optimize the performance of a data-driven system. Therefore the potential is analyzed from its theoretical perspective to enable efficient deep learning without the need for hardware scaling. The theoretic foundations are then proven by real-world application examples to the essential components. While a data-driven system consists of a large number of aspect that can be compressed, this thesis focuses on the input, the algorithm, and the output. Nevertheless, it needs to be mentioned that the compression of the hardware, the program, and the inter-program communication itself is essential to investigate.

1.1 Research Questions

To be able to investigate the role of compression in spatial computing according to the challenges that have been stated before, it is necessary to define a data-driven system's pipeline, its components, and requirements. Beforehand, this dissertation builds the following research questions:

RQ-1: *What is the minimum of the complexity and informational content of heterogeneous data required for a data-driven system?*

RQ-2: *What is the potential and limitation of (aggressive) algorithm compression and randomized data structures in an edge computing scenario?*

RQ-3: *How can abstaining and model ensembles facilitate learning of a data-driven system?*

Next to those questions, some research aspects and directions are important to mention but are not covered within this thesis because they cross the boundaries of this domain. While the first aspect is the compression of the program itself, it would be interesting to see if the compression of inter-program communication optimizes the entire system's performance. Additionally, another field is if the hardware components can be compressed to optimize the performance characteristics of a data-driven system, such as the runtime and the energy consumption, without losing precision on the outcome.

1.2 Structure of the Thesis

This thesis provides an in-depth investigation of the compression of data-driven systems and the consequences that arise. More specifically, the compression of the component of such a system, that are namely the input data, the operation/algorithm, and the output, is analyzed in detail from its theoretic perspective. This includes the basic mechanisms as well as the impact that is caused by the more compact representation of the different elements. Furthermore, besides the theoretic investigation, the thesis provides some application samples for the essential components.

The rest of the thesis is structured as follows: Chapter 2 provides the theoretical fundamental that is required, which includes an introduction to Shannon's information theory, followed by the principles of algorithmic information theory and the mechanisms for image classification. Additionally, a short overview of the available hardware accelerators is given. A theoretical analysis of the research questions is provided in Chapter 3. Furthermore, the theoretic main principles and investigation that have been created in this part are proven using application samples in Chapter 4. The thesis is finished by the discussion in Chapter 5, which summarizes the main findings as well as the conclusions that can be stated. Figure 1.1 provides a conceptual overview of the structure of the thesis.

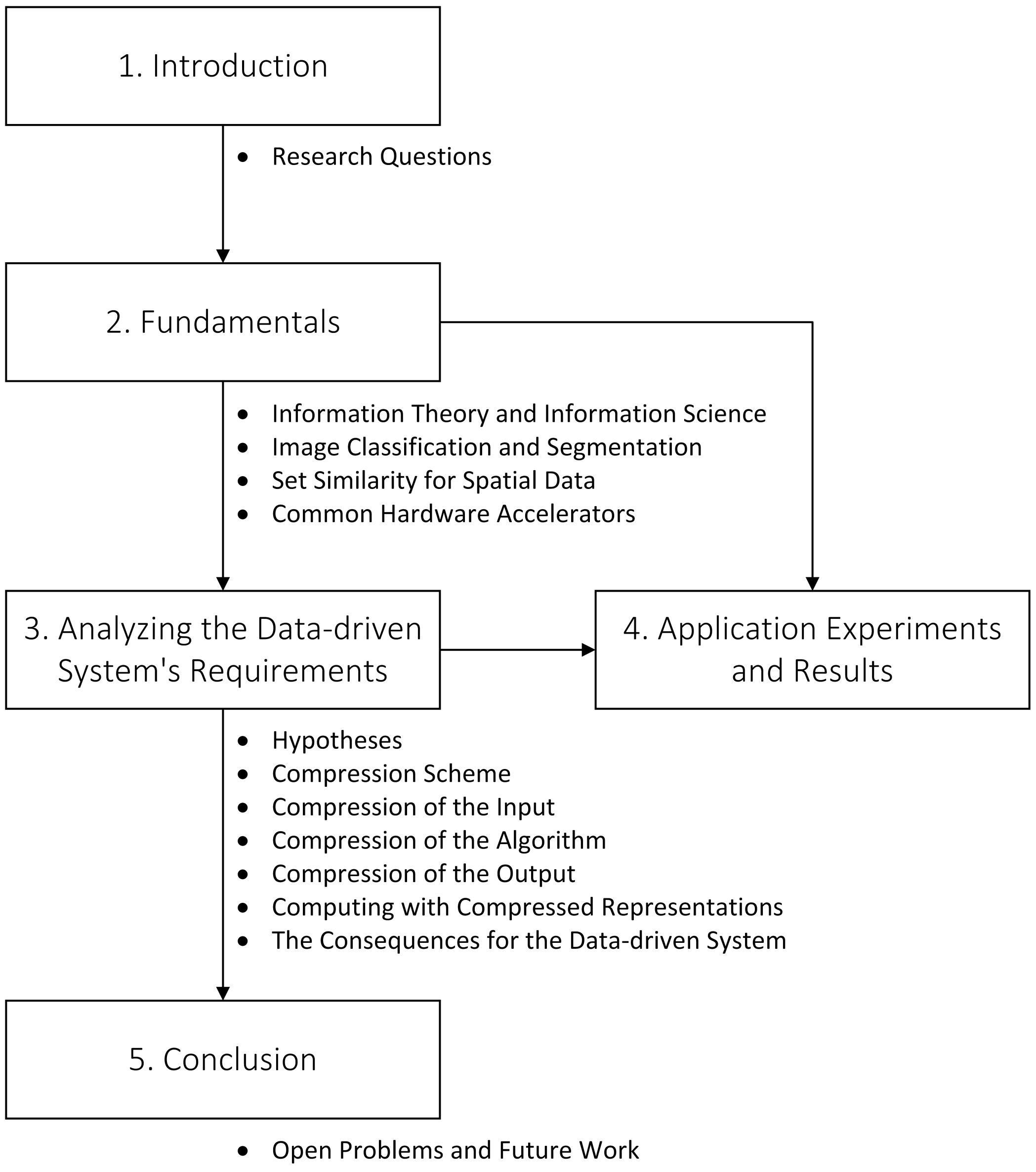

1. Introduction
• Research Questions
2. Fundamentals
• Information Theory and Information Science
• Image Classification and Segmentation
• Set Similarity for Spatial Data
• Common Hardware Accelerators
3. Analyzing the Data-driven System's Requirements
4. Application Experiments and Results
• Hypotheses
• Compression Scheme
• Compression of the Input
• Compression of the Algorithm
• Compression of the Output
• Computing with Compressed Representations
• The Consequences for the Data-driven System
5. Conclusion
• Open Problems and Future Work

Figure 1.1: Conceptual overview of the structure of the thesis.

2 Fundamentals

This chapter discusses the fundamental principles that are necessary for the following theoretical investigation and practical applications; therefore, three domains are covered within this part of the thesis. First is the information theory, covering Shannon's as well as Kolmogorov's approach. The second domain covers the advanced basics in machine learning with a focus on deep learning especially using convolutional neural networks. Those methods are the algorithms of interest for most experiments in the application section. The third domain is the field of computing similarities between spatial data items, such as trajectories. Additionally, the fundamental principles are followed by a list of the most commonly used hardware accelerators.

2.1 Information Theory and Information Science

Shannon's information theory is a central component in modern computing and is of importance since it has been investigated. Next to this approach, another versions has been introduced, such as the algorithmic information theory. Based on those, the compression of data can be described and used for tasks such as maximizing the throughput of a channel or the computation of similarity between two objects. Nevertheless, firstly we describe a classical notation of information as developed by Shannon based on this analyses of communication. This includes common compression methods for objects, such as strings. Secondly, we describe a different approach to information content following the algorithmic perspective of Kolmogorov.

2.1.1 Communication Theory and Compression

The information theory was first introduced in 1939 by Shannon in [15], extended in [16]. As [17] states, the theory answers significant problems of modern information technology. The first one is to determine a valid an precise definition of an compressed version of an given data element. The entropy is able to provide a proper measure of information that is able to serve an theoretic bound for the compression. Second problem is the transition are within communication systems, which is formulated as the capacity of a transition channel. Those two questions and answers build the core concepts of Shannon's information theory besides are a basis for intersections to other fields like computer science or philosophy. For example, the theorem Ocamm's razor has another viewpoint to some information theory aspects. The theorem states that if there are multiple hypotheses about the same theory, the preferred one should be the simplest, which is the strategy with the least amount of parameters [18, p. 580]. Other than that, the algorithmic information theory (see Section 2.1.2) builds the intersection to computer science. The theory's core is the Kolmogorov complexity which is defined as the shortest program that outputs a defined binary string based on an input string [19].

2.1.1.1 Shannon's Communication Model

The information theory is based on a communication model that sends a message from one point to another. The medium called channel that transmits the data has a specific amount of noise and therefore causes errors in the message. The authors of [20] describe that an example of such a channel is an (analogous) telephone connection, where the message is encoded and decoded by a digital modem. The transmission using an analog connection can increase the nose in the receiver system. An equivalent example is an internet connection of a computer. The message is encoded utilizing a particular layer of the OSI model and is then transmitted to another device. The physical layer is nowadays a wireless connection which might add some inferences and nous to the (send) message. Nevertheless, the goal is to minimize the error possibility caused by the connection. Among others, this can be solved by changing the physical characteristics of hardware components or by system design.

The communication model described in [15],[16] consists of several components. The first one is the *information source* and represents a physical device producing message [21]. As described above, the output can have various forms and shapes, such as an analogous signal, like our voice. Another example would be raster-shaped data such as a three-channel image taken by a random digital camera. The second component is the *sender/encoder* and according to [20], [17] it describes the part unit which transfers the (digital) data item into a form (analogous) that is able to be transmitted

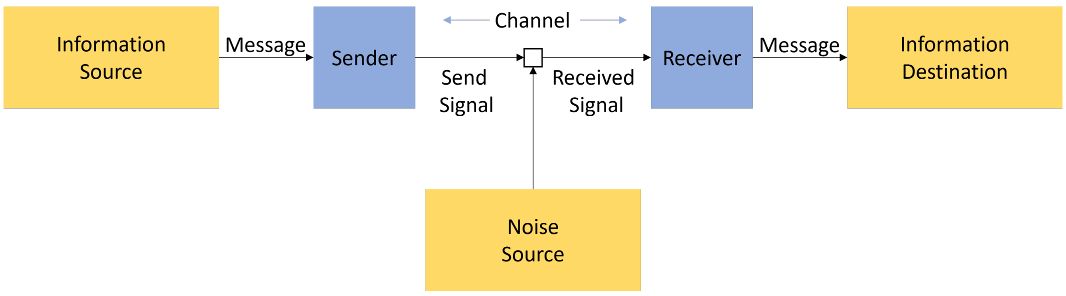


Figure 2.1: Schematic representation of Shannon's communication model, where an information source creates a massage which is then encrypted and sent through a channel to its destination where the message gets decoded. The channels have a specific error probability caused by the noise source. This figure has been adapted from [15].

over the channel. A form used by, e.g., computers is a binary representation, which will be discussed in detail below. The third component is the *channel*, which is the medium that transfers the signal from the sender/encoder to the receiver/decoder [15]. An example would be, for example, a wireless or wired connection. The channel has some amount of noise caused by external influences. The fourth part is the *decoder*, which reverses the encoder, so the received signal is translated into its source format and reaches the *destination*. Figure 2.1 provides an schematic visualization of the described communication system.

The entropy is a measure of information for Shannon's communication system, that transmits a discrete variable X with a given probability $P(x) = P(X = x)$. In addition the information can be defined as $-\log(\frac{1}{p(x)})$. Furthermore, as stated above, the encoder translates, for example, a code word into a binary string $\{0,1\}^*$. In this particular case, each letter within the word is mapped to a specific binary string. Lets say the character $a \in S$, where $S = \{a, b, c, d, e, f, g, h\}$, is mapped to 0101, formally, $f : S \rightarrow \{0,1\}^*$. The optimum length of the binary string depends on the number of elements in S and the probability of how often an element occurs. The entropy can also be interpreted as the optimum average count of bits needed to encode a random element from set S. [21],[20]

Definition 2.1. *As provided by [21], let the entropy H be the average amount of information contained in a transmitted signal x, which is a discrete random* *variable.*

$$H(x) = \sum_{x \in X} p(x) \log \left(\frac{1}{p(x)} \right)$$

If the log with base e is used, the entropy produce to natural units (nat). While the natural logarithm tends to be more often employed in mathematics, the base two logarithm leads to the unit bits for the entropy. Consequently, the latter measures how many bits are needed on average to encode a sequence of elements from a defined set.

A random variable X can be considered as an element from a set S that has a length of eight items, where the probability of these values are uniformly distributed. Based on the Definition 2.1, the (binary) entropy of X is

$$H(X) = \sum_{x \in X} \frac{1}{8} \log_2 \left(\frac{1}{1/8} \right) = 3 \text{ bits}, \tag{2.1}$$

which means that 3 bits are needed to encode an event from X. Similar is the case where the outcome of X is unevenly distributed. The authors of [17] state the example where the probabilities of the elements of X are $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}\}$. This leads to an entropy of

$$H(X) = \sum_{i=1}^{X} p(x_i) \log_2 \left(\frac{1}{p(x_i)} \right) = 2 \text{ bits}, \tag{2.2}$$

which means that, on average two bits are needed to encode a possible outcome from X. Furthermore, one option to encode the elements of X would be 0, 1, 01, 10, 11, 001, 010, 011, which has an average description length of 2.125 where a uniformed distribution would need at least 3 bit.

Definition 2.2. *As provided by [20], the conditional entropy $H(X \mid Y)$ is the average amount of information of X when Y is given. This is formally defined as*

$$H(X \mid Y) = \sum_{x,y \in X,Y} p(x, y) \log \left(\frac{1}{p(x \mid y)} \right),$$

where the log *can be any logarithm. As it is the case for Definition 2.1 the use of a base two logarithm leads to the unit bit for the entropy.*

Definition 2.3. *As provided by [21] let I be the mutual information*

$$I(X;Y) = H(X) - H(X \mid Y),$$

where the x, y are two joint events. The I can be interpreted as the amount of information of Y, which is contained in X.

Moreover, one needs to mention that the entropy and the conditional entropy do consider only one single probability distribution. Compared to that, [22] states that the Kullback-Leibler divergence considers multiple distributions. Defined as

$$D_{KL}(p \mid\mid q) = \sum_{x \in X} p(x) \log \left(\frac{p(x)}{q(x)} \right), \tag{2.3}$$

where p, q are two probability functions. Note that the Kullback-Leibler divergence is not symmetrical, therefore $D_{KL}(p \mid\mid q) \neq D_{KL}(q \mid\mid p)$, except if $p = q$.

Furthermore, in Shannon's communication system the encoded message is transmitted from a source to a destination over the channel. According to [17], the capacity depends on the conditional entropy (see Definition 2.2) and the mutual information (see Definition 2.3). The channel's capacity considers the input message X on the source and the message Y on the destination.

Definition 2.4. *As provided by [17], [21], the data transition supremum is known as the capacity C of a channel and is defined by*

$$C = \sup_{p(X)} I(X;Y),$$

where X is the input message at the source and Y is the message received and encoded by the receiving destination. Furthermore, the supremum is over all possibilities of $p(x)$.

2.1.1.2 Shannon-Fano Code

Shannon created the proposed information theory, more specifically, the definition of the communication system as the basis of digital communication, which heavily influenced the development of digital computers. The data compression part of Shannon's is the main interest of this work. With the definition of entropy, $H(X)$ can be interpreted as the average code length needed to encode an element from X. With the definition of entropy, $H(X)$ provides us a 1-bit range for the average length of the best code that encodes an element from X. While this is a Shannon-defined theoretical approach in [15], Fano proposed

Table 2.1: An example of a Shannon-Fano code using a given probability for each element.

Element No.	Probability	Shannon-Fano Code
1	1/2	0
2	1/4	10
3	1/8	110
4	1/16	111
5	1/64	11100
6	1/64	11101
7	1/64	11110
8	1/64	11111

from [23] an implementation of the theoretical approach, which is nowadays known as Shannon-Fano encoding.

The coding encrypts an element from a random variable with n possible outcomes $\{x_1, x_2, ..., x_n\}$ with the related probabilities $\{p_1, p_2, ...p_n\}$, where each outcome is encoded into a binary sequence $\{0, 1\}^*$. Moreover, according to [17], the expected length L of the code word is $H(X) \leq L \leq H(X) + 1$. Considering the previous definition of variable X with its probabilities $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}\}$ (sorted in descending order) produce $H(X) = 2\,bit$. The set of probabilities is split into two parts such that the sum of probabilities is approximately the same, and each side is signed to a single-digit binary number. When one has only a single element, it is not further split. This generates the code represented in Table 2.1.

2.1.1.3 Huffman Code

Another method to encode words is the Huffman code introduced by [24], that tries to assign shorter code words to outcomes of a random variable X, which has a higher probability of occurring, and longer code words to other outcomes. While the idea and motivation are similar to the Shannon-Fano code, this encoding method constructs the code word backward by constructing a tree.

According to [20], two steps are required to construct the Huffman tree. The first step is to combine the two symbols with the smallest sum of probabilities – those will have a code word with the same length. The second step is to create a sub-tree by combining those elements and repeating the process. In this way, a tree is constructed step-by-step. Let us take again the example from above where a random variable X has the possible outcomes $\{A, B, C, D, E, F, G, H\}$

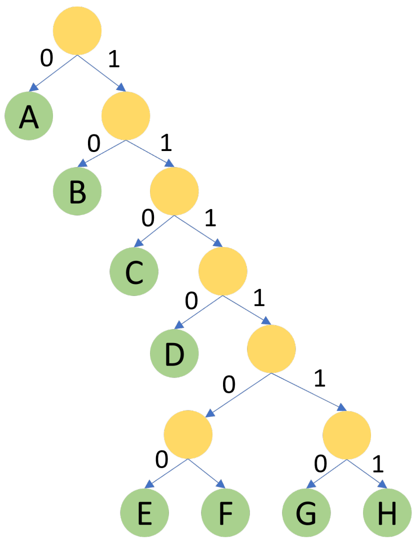


Figure 2.2: The Huffman code generates a binary tree by recursively combining the two elements with the lowest probability and the edges of the tree representing either zero or one. The code word is then the combination of all edges to the target.

with the related probabilities $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}\}$ leading to $H(X) = 2bit$. While the tree used to create the Huffman code is visualized in Figure 2.2, Table 2.2 shows the resulting code-words.

2.1.1.4 Quantization of Numbers

The communication channel between components can have various forms where one example would be the transfer between devices that are not directly connected. Another form of channels are those that are within a single device,

Table 2.2: example of a Huffman code using a given probability for each element.

Element	Probability	Huffman Code
A	1/2	0
B	1/4	10
C	1/8	110
D	1/16	1110
E	1/64	11100
F	1/64	11101
G	1/64	11110
H	1/64	11111

such as a bus between the central processing unit (CPU) and the graphical processing unit (GPU). Especially when working on use cases that are either time-critical or have constraints related to power consumption, a limiting factor is usually the capacity of the channel. As a consequence there is a need to reduce (or compress) the recitation of numerical computation. Quantization of numbers enables the possibility to reduce the numerical space. Moreover, another application is used in the electrical engineering field, where an analogous signal is converted into a digital one. Both types of quantization – among other applications and definitions in different domains – have in common that a continuous input is transferred to a discrete output. Overall, this method comes with the downside of being a lossy compression method. For instance, when processing a continuous analog signal to a digital one, there is always an error rate caused by factors such as the sampling rate, resulting in losing information between the taken samples. A similar problem occurs when reducing the numerical precision, such as when converting an actual number into a natural number, $f : A \in \mathbb{R}^n \to A \in \mathbb{N}^n$, a rounding error occurs. According to [25], another problem is the possible presence of underfitting and overfitting. While the first one represents, for example, a division by zero, leading to errors in some programming languages. Therefore it is preferred to have a division by a small number is preferred, even if it causes some rounding errors. Overfitting, on the other hand, would be the division with infinite or a buffer overflow. Therefore it is necessary to consider the size of the data for all computational problems.

As mentioned above, the analog-digital-converter is able to transfer a continuous signal, for example, $X = sin(x)$, where $x \in \mathbb{R}$, into a discrete form. However, a digital device is not able to process this signal, therefore, it gets sampled using a specific rate, for instance, with 50 Hz. This results in a stream with 50 values per second. Figure 2.3 shows this process, where the blue line represents the continuous signal and red markers the sampled points and how long the value is valid. The principle of reducing the precision of numerical computation works similarly to the analog-digital converter. An example is the gradient descent algorithm, which represents the central element of the training phase in deep learning. Figure 2.4 visualizes the structure of a signed integer with 32-bit and a floating point number (FP) IEEE 754 [26], one can see the latter one has a reserved area for the digits. When transferring this number (18.234) to an integer, everything behind the comma gets cut off, which is caused by the missing space for the fractional part.

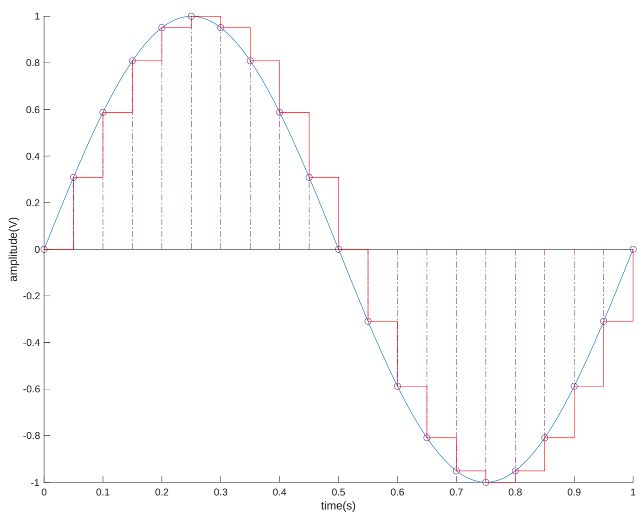


Figure 2.3: Quantization is a process where the values of a continuous, discrete variable are converted to discrete values. The blue line represents an analogous continuous input signal and gets sampled with a specific rate (purple, dash-dotted line). The new sampled digital discrete signal is represented with the solid red line.

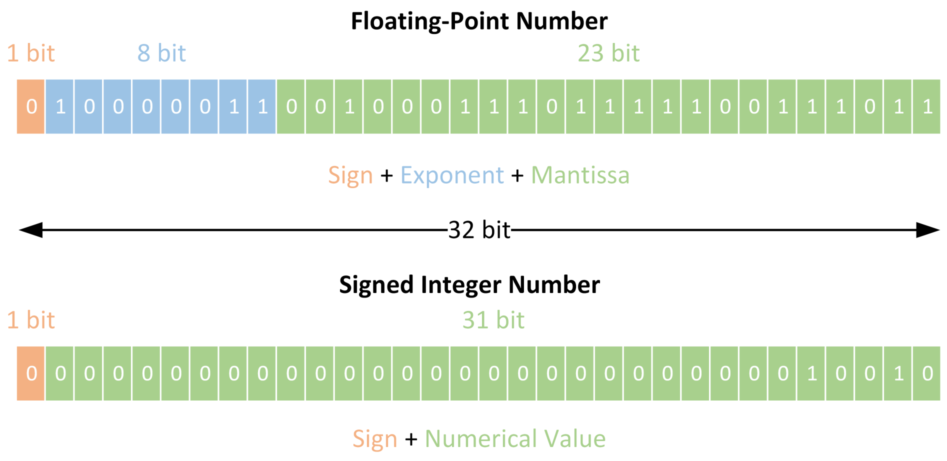


Figure 2.4: One numerical type is a floating-point number (FP) using the norm IEEE 754 [26]. The displayed single precision has a total size of 32 bit, where 1 bit is reserved for the sign, 8 bit for the exponent, and 23 for its mantissa. The bits in the array build the number 18.234. In contrast to this, the bottom numerical type represents a signed integer number with a size of 32 bit, including a reserved bit for the sign. This data type is not able to hold a floating-point value. Therefore, the set bits sum up to the value of 18.

2.1.2 Algorithmic Information Theory

The last section was about the measurement of the absolute information content of objects like strings with the use of Shennon's information theory. An alternative to this model is the algorithmic information theory and its fundamental principle, the Kolmogorov complexity, first introduced by [27]. This complexity enables the possibility of creating a universal distance metric that measures the distance between two objects and is not computably caused by its universality [28]. Moreover, as described in [19] an approximation can be utilized for content-based clustering.

The following provides an overview of the theory of Kolmogorov complexity and the resulting universal metric called Normalized Information Distance (NID). Additionally, the Normalized Compression Distance (NCD) is discussed, which is able to approximate the universal metric.

While this summarizes the most essential characteristics of the algorithmic information distance, [29] is a comprehensive and detailed literature.

2.1.2.1 Kolmogorov Complexity

A central part of the Algorithmic Information Theory is the Kolmogorov complexity, but before going into detail, it is necessary to set up an environment and its formalities. While the cardinality of an object X is notated as $|X|$, the following considers only objects which are binary strings $x \in \{0,1\}^*$. According to [28], [30], it is essential that a set S of binary strings is a prefix set, which has the property that it satisfies the Kraft inequality (see Theorem 2.1). The length of an binary string is defined as $\ell(x)$, and ε describes the empty string.

Theorem 2.1. *As provided by [28], [30], a set S of binary strings $S = \{x \in \{0,1\}^*\}$ is a prefix set and satisfies the Kraft inequality*

$$\sum_{x \in S} 2^{-\ell(x,y)} \leq 1,$$

where ℓ denotes the length of a binary string and $y \in \{0,1\}^*$.

Additionally, [31] describes that a function $\phi : \{0,1\}^* \to \{0,1\}^*$ is partially recursive and is able to print a string x with its description p, formally notated as $\phi(p) = x$. Furthermore, the function uses descriptive language like C, C++, or Java. Therefore, the complexity C of x is defined as the shortest version of

p, with ϕ. Additionally, a more generalized version can be described with the utilization of another input string y, so that the function is $\phi(y, p) = x$.

Theorem 2.2. *As provided by [28], [31], the conditional complexity C, with respect to a partial recursive function $\phi : \{0,1\}^* \to \{0,1\}^*$, of the binary string x with an input string y is defined as*

$$C_\phi(x \mid y) = \min\{\ell(p) : \phi(y, p) = x\}.$$

The unconditional case is similar, with the change that the input string y is replaced by an empty string ε: formally notated as $C_\phi(x) = C_\phi(x \mid \varepsilon)$.

When analyzing the Definition 2.2, one can see that the complexity only refers to ϕ. Due to the fact that the binary representation of the function differs between the languages, compilers, and it is even possible to cross-compile languages, it is complicated to determine which one leads to the shortest binary representation [19], [32]. Therefore the ϕ_0 is the universal function that represents the shortest of all binary programs ϕ that fulfills the given task. Consequently, the function ϕ_0 satisfies $C_{\phi_0}(x \mid y) \leq C_\phi(x \mid y)$, where c is a constant. While in literature one usually refers to C_{ϕ_0} when writing C, the complexity of the concatenated strings $C(xy)$ is not bounded by $C(x)$ and $C(y)$. Additionally, not all partial recursive functions are taken into account. In contrast, this ψ_0, which is a partial recursive prefix function, considers all of them (see Definition 2.5). [28]

Definition 2.5. *As provided by [28], $\forall x, y \in \{0,1\}^*$ let $\psi_0 : \{0,1\}^* \to \{0,1\}^*$ the smallest partial recursive prefix function of all ψ, this satisfies*

$$C_{\psi_0}(x \mid y) \leq C_\psi(x \mid y) + C,$$

where c is a constant. The unconditional case is similar, with the change that the input string y is replaced by an empty string ε: formally notated as $C_{\psi_0}(x) = C_{\psi_0}(x \mid \varepsilon)$.

A problem of this definition is its uncomputability, consequently this is also the case for the Kolmogorov complexity itself. The authors of [33] stated that the problem is that the complexity depends on the function itself rather than on the input. Consequently, the complexity of the function is $C(C(x) \mid x) = c$, which is not computable. Moreover, according to [28], [34] one can write K instead of C_{ψ_0}, therefore Definition 2.5 can be rewritten to $K(x \mid y)$ and

$K(x \mid \varepsilon)$. Furthermore, due to the reason that $K = C_{\psi_0}$, the conditional Kolmogorov complexity of x is defined as the length-shorted program that prints a binary string x with a given input string y, where it is possible that $y = \varepsilon$. The authors of [30] add to this that the amount of information from x which is contained in y is defined as $\mathcal{I}(y : x) = K(x) - K(x \mid y^*)$, where y^* represents the shortest binary program. Furthermore, the publication states that due to the presents of a constant $c > 0$, the Kolmogorov complexity can be rephrased to Eq. 2.4, causes again to the problematic $\mathcal{I}(x : y) = \mathcal{I}(y : x)$ that K is not computable.

$$K(x, y) = K(x) + K(y \mid x^*) = K(y) + K(x \mid y^*) \tag{2.4}$$

The Equation 2.4 leads to the characteristics of Kolmogorov complexity that the conditional complexity is $K(x|y) = K(y \mid x) - K(x)$ and $K(x, y) = K(xy)$, where $K(xy)$ is the length of the shortest program of the concatenation of x and y, without knowing differing between the string objects. This has the consequence that the conditional complexity is $K(x \mid y) \approx K(xy) - K(y)$. [19], [34]

2.1.2.2 Normalized Information Distance

Before being able to define a universal distance metric with the use of the Kolmogorov complexity, it is necessary to give a proper definition of a similarity metric. The authors of [19] defined a set of axioms, represented in Definition 2.6 that needs to be satisfied.

Definition 2.6. *As provided by [30], [19], let D a distance function that calculates the similarity on a non-empty set Ω, such that $D : \Omega \times \Omega \to \mathcal{R}^+$. The resulting value is called the distance between $x, y \in \Omega$. This is a metric on X if $\forall \{x, y, z\} \in X$ it satisfies the following equalities.*

1. *Positivity: $D(x, y) = 0$ if $x = y$*

2. *Symmetry: $D(x, y) = D(y, x)$*

3. *Triangle inequality: $D(x, y) \leq D(x, z) + D(y, z)$*

***Positivity**: The distance to an object itself is zero, therefore, $D(x, y) = 0$ of $D(x, y) = 0$ if x and y are equal, and otherwise, $D(x, y) > 0$.*
***Symmetry**: The order of the objects do not have an impact on the resulting distance.*

Triangle inequality*: The theorem states that the sum of distances from two individual objects x, y to a third one z is always greater than the distance between the objects x and y.*

The Kolmogorov alone is not suitable for an information distance, among others, because of its asymmetric properties. For example, in the case of the conditional complexity of the string x and ε. Therefore, an universal information distance $E(x, y) : x \to y$ is defined: as it is the case for $K(x \mid y)$ it represented the length of the shortest binary program that outputs a string x with a given input y [30], [19]. Moreover, the information distance E for an universal partial recursive function ψ has the characteristic $E_{\psi_0}(x, y) \leq E_{\psi}(x, y) + C_{\psi}$, where C_{ψ} is a constant depending only on ψ. [28]

Definition 2.7. *As provided by [28], let E_0 be a machine-independent information distance concerning a universal partial recursive function ψ_0 between the objects x, y*

$$E_0(x, y) = \min\{l(p) : \psi_0(p, x) = y \textit{ and } \psi_0(p, y) = x\}.$$

Definition 2.8. *As provided by [19], let $E(x, y)$ be the maximum information distance between two objects x, y*

$$E(x, y) = \max\{K(x \mid y), K(y \mid x)\}.$$

As stated in [28], the machine-independent information distance E_0 from Definition 2.7 is equal to the maximum information distance E from Definition 2.8. Furthermore, [30] argues that an essential characteristic of a distance function is that the function is admissible.

Theorem 2.3. *As provided by [30], let D be a normalized information distance such that $D : \Omega \times \Omega \to [0, 1]$ which fulfills the characteristics of a metric and is also symmetric for the constant c*

$$|\{y : D(x, y) \leq e \leq 1\}| < 2^{eK(x)+1}$$

Where $D(x, y)$ satisfies the Kraft theorem

$$\sum_y 2^{-D(x,y)K(x)} \leq 1.$$

Finally, according to [19], [28], [30], the Normalized Information Distance is built based on those definitions, which is based on the Kolmogorov complexity. This metric is also not computable and is defined as

$$NID(x, y) = \frac{\max\{K(x \mid y), K(x \mid y)\}}{\max\{K(x), K(y)\}}, \tag{2.5}$$

, which fulfills the metric equalities.

2.1.2.3 Normalized Compression Distance

Before going into detail about how to approximate the Normalized Information Distance using compression, it is necessary to give a proper definition of a normal compressor. The authors of [19] define a set of axioms, represented in Definition 2.9 that are true for most real-world compressors and are essential to ensure the key characteristics for the later introduced distance metric.

Definition 2.9. *As provided by [19], a compressor C is considered as normal if the following axioms are fulfilled.*

1. *Idempotency:* $C(xx) = C(x)$ *and* $C(\{\}) = 0$
2. *Monotonicity:* $C(x) \leq C(xy)$
3. *Summetry:* $C(xy) = C(yx)$
4. *Distributivity:* $C(xy) + C(z) \leq C(xz) + C(yz)$

Idempotency: *A normal compressor is able to provide idempotency to a required precision. In other words, the union of an object A with itself is the object $A \cap A = A$.*
Monotonicity: *A normal compressor needs to have monotonicity when concatenating objects.*
Symmetry: *The condition of symmetry of means that the order of the compressed objects does not have an impact on the result.*
Distributivity: *While the Kolmogorov complexity satisfies the stronger distributivity $C(xyz) + C(z) \leq C(xz) + C(yz)$, with $C = K$, the real-world compressors do fulfill only the weaker distributivity $C(xy) + C(z) \leq C(xz) + C(yz)$.*

The Normalized Information Distance defines a universal metric, but the disadvantage is its uncomputability caused by the Kolmogorov complexity.

The Normalized Compression Distance approximates the K with the application of normal compressor C and let $C(x)$ be the length of the compressed object x. Furthermore, as is the case at the conditional Kolmogorov complexity $K(x \mid y)$, the conditional compression of two objects $C(x \mid y)$ defines the information of x contained in y. Additionally, the triangle inequality $C(x \mid y) \leq C(x \mid y) + C(z \mid y)$ is fulfilled for the conditional compressed information. The conditional compression is

$$C(y \mid x) = C(xy) - C(x), \tag{2.6}$$

where $C(xy)$ is the length of the compressed concatinated objects x, y. Furthermore, the NCD metric is built upon the NID as defined as

$$NCD(x,y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}, \tag{2.7}$$

where $C(x)$ is the length of the binary compressed object x employing a normal compressor such as GZIP or BZ2. The metrics range is $0 \leq NCD(x,y) \leq 1+\epsilon$, where ϵ is an error rate that is caused by the used compressor. [19], [28], [30], [34]

While the denominator of Equation 2.7 is equal to the one of Equation 2.5, the nominator is more non-trivial to understand. As discussed beforehand, the conditional Kolmogorov complexity is roughly $K(x \mid y) \approx K(xy) - K(y)$, assuming $K(x \mid y) \approx K(xy)$. Therefore, the authors of [19] argue that the nominator of the NID can be reformulated to $\max\{K(xy) - K(y), K(xy) - K(x)\}$, in addition to the authors states that this can be approximated with normal compressors, formally denoted as

$$min\{C(xy), C(yx)\} - min\{C(x), C(y)\}, \tag{2.8}$$

where $min\{C(xy), C(yx)\}$ can be replaced by $C(xy)$ caused by its symmetric characteristics $C(xy) \approx C(yx)$ leading to the nominator of the NCD (see Eq. 2.7).

The Normalized Compression Distance is an active research field focusing on parameter-free feature-fee compression-based clustering in domains like biological genomes [19] and remote sensing [36]. For example, the authors of [35] have tiled two remote sensing images into equally sized patches. A distance matrix has been calculated where each position is the result of the NCD between two patches from the same position in each image. Each position within

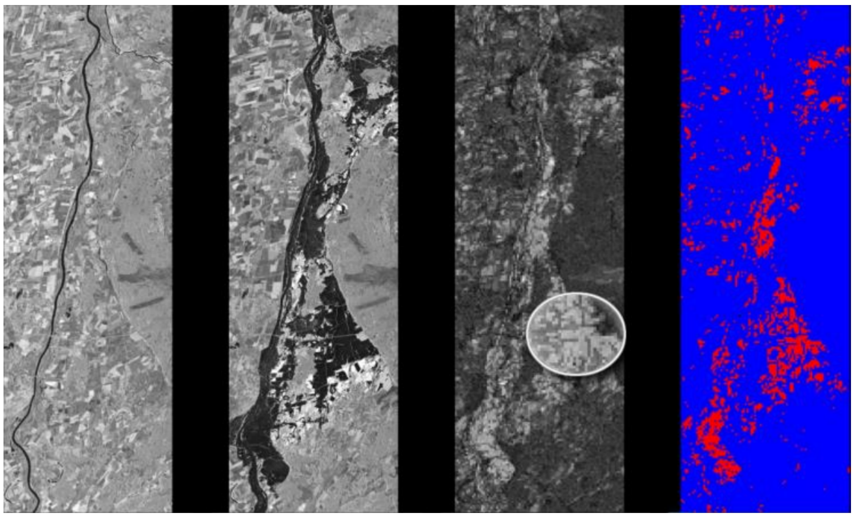


Figure 2.5: Visualization of the use of the Normalized Compression distance. The first two do show remote sensing images before and after a flood. While the third is the calculated distance matrix, the fourth represents the threshold distance matrix indicating where a change is or not (blue: no change, red: change). The figure has first been published in [35].

this matrix shows the change between the selected images. A threshold can then decide whether a change is present in a certain location or not. Figure 2.5 represents the results visually.

2.2 Image Classification and Segmentation

Nowadays, artificial intelligence increased in its importance and is applied in more and more domains. While algorithms are applied to improve, for example, supply-chain services, the aerospace industry have also a large interest in methods that optimize costs, lifetime, and usage of data-producing devices. Anyways, machine learning is a wide field and can be split into several subdomains, such as deep learning which is the scope of this particular section. More precisely, the domain of interest are network architectures that use convolutional layers to make a decision about the content of e.g. an image. Therefore the following gives an insight into the most important concepts, which is the theory of their functionality, common architectures, and the compression of them.

Before discussing convolutional neural networks in detail, it is important to define the two problem families. Patch classification is the first, which is about predicting the content of an image without determining the spatial location of a certain object. An example would be that it is possible to say that an image contains a house with a certain probability without the ability to predict a (specific) location within the image. The task of patch classification is a central task in modern architectures in deep learning. Additionally, the ability to compute the summation of an object has a major role within the domain of transfer learning, such as done on the ImageNet dataset , which is commonly used to pre-train convolutional neural networks, even across the boundaries of the computer vision field. The second category is pixel classification, which is about computing the class affiliation of an individual pixel. An example would be to predict whether an airplane can be as precise as possible located within a remote sensing image. In this particular scenario, there are two types of pixels, *ship* and *no ship*. Without considering the technology that is used to classify the pixels, there are several complexity types, object detection, semantic segmentation, and instance segmentation. While the first one draws bounding boxes around areas where a certain object has been found, the semantic version is only able to detect the class of e.g. a pixel. Compared to object detection, the semantic strategy is not able to detect how many objects from one single type are contained within an single frame. This is the main difference between semantic and instance segmentation, where the latter is able to detect also the count of how often an object is included.

A problem that comes up when considering the class label for both categories, patch classification as well as pixel classification, is their form and structure. While humans label their classes in terms such as *ship* and *no ship* or *forest* and *burnt area*, the algorithms need to transfer those into numerical values such that *ship* is represented by zero and *no ship* by one. The process of one-hot-encoding solves this exact this issue, where a binary vector is created that has a length of the number of different labels, where each position represents a label. Formally described a class set $C = \{activefire, burntarea, smoke\}$ and an image is assigned to the class *active fire*, the transformation function one-hot-encoding T converts the representation, $T : C_i \to (1, 0, 0)$, where i in the class index which is in this particular case one. This converted label is the basis for the training of, for instance, convolutional neural networks.

2.2.1 Convolutional Neural Networks

The basics of feed-forward neural networks build the backbone of modern architectures in deep learning, but when it comes to more complex data, there is a need for specialized model types. A convolutional neural network (CNN) is such a structure that focuses on data that have a grid structure, including spatial dependencies, for local reasons. An example of data like that is an image. It has two grid-structured dimensions and spatial dependencies between the pixels. With the different color channels, the image is a three-dimensional input to the CNN. Furthermore, this network type is not limited to images as input data. Rather they can also be used in natural language processing, sensors' time series predictions, and much more. Moreover, images do have a certain amount of translation invariance – an image put upside down still shows the same image – and the CNN does create features that have a similar pattern due to the reason that the features are extracted from local regions. While earlier layers in the network create low-level features, like edged and corners, later levels (closer to the top layers) do provide high-level features, such as objects [25]. Furthermore, a characteristic of CNNs is that the networks include at least one convolutional layer. This layer uses a convolutional operation which is a dot product between the grid-structured input data and a matrix that holds some trainable weights. [37]

All concepts of the previous section, which describes the feed-forward neural networks are still the same. Consequently, also the fundamental mechanisms and activations are still the same. Even when the structure is slightly different, weights are, in this context, matrix-shaped. As described in [37], next to the convolutional layer, other common ones are pooling and the (ReLU) activation.

As has been the case for fast-forward neural networks, the convolutional layer is a combination of several suboperations and steps. The input data of a CNN is an image of the type $h \times w \times d$, where h is the height, w is the width of the image, and d is the depth. Most commonly, colored images do have three channels $\{red, green, blue\}$; therefore, the depth is $d = 3$, each corresponds to a single color. Each convolutional layer has exactly the input shape $h_q \times w_q \times d_q$, where q indicates the layer id. While the input of the first layer is the source image itself, the matrices are called *feature maps* or *activation maps* for layers $q > 1$. As it is the case for fast-forward neural network, some parameters have been introduced, namely the weights. In CNN, those are organized in a matrix shape and are called *filters* or also *kernels*. The size of the kernel is defined

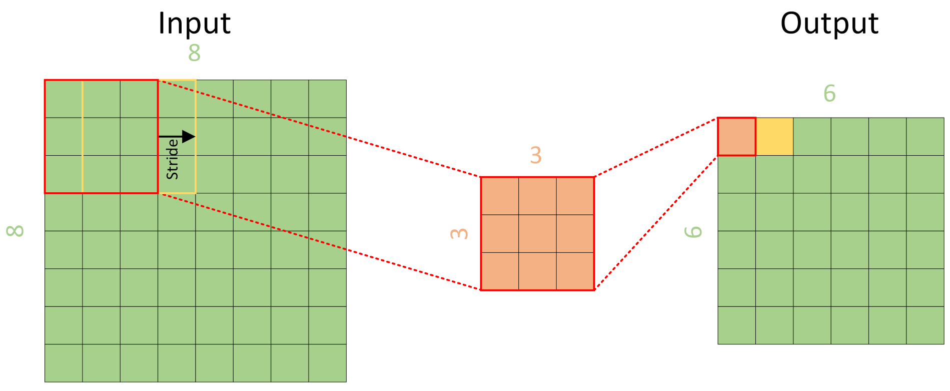


Figure 2.6: Representation of a convolutional layer and its functionality. The input feature map has a size of $8 \times 8 \times 1$. The kernel of the size $3 \times 3 \times 1$ is moved over the feature map, and the cross-correlation is calculated on each position. This leads to the output map with a sit of $6 \times 6 \times 1$.

as $r \times c \times d$, most commonly $r = c$ and $\{3, 5\}$. One can see that the depth of the filter always matches the layer's feature map. Nevertheless, this kernel is placed on each possible position of the image so the kernel fully overlaps. Between each union of the kernel and feature map, the convolutional operation is performed – defined as the cross-correlation – and written to a new feature map. In this way, the single kernel extracts certain features of the input map. Due to the fact that the kernel needs to fully overlap, there are fewer possible positions than pixels in the input. So, when performing the convolution on the q-layer, the feature map for the layer $q+1$ has a height of $h_{q+1} = h_q - r_q + 1$ and a width of $w_{q+1} = w_q - c_q + 1$. Therefore one can clearly see that the size shrinks with the layers. Figure 2.6 visualizes the structure of the convolutional layer, where the down-sampling is visible on the second layer. Furthermore, each layer can have several kernels, each one producing a feature map. So, each of the kernels do create a spatially arranged feature. [38]

The authors of [39] provide an example of the convolutional operation with a 3×3 pixel image and a 2×2 kernel and is as follows:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 5 \\ 7 & 6 & 0 \end{pmatrix} * \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 11 \\ 21 & 10 \end{pmatrix},$$

where $*$ represents the convolutional operation. Additionally, a step-by-step representation is as follows:

$$1 \cdot 2 - 1 \cdot 1 + 2 \cdot 4 + 1 \cdot 3 = 12$$

$$1 \cdot 1 - 1 \cdot 1 + 2 \cdot 3 + 1 \cdot 5 = 11$$

$$1 \cdot 4 - 1 \cdot 3 + 2 \cdot 7 + 1 \cdot 6 = 21$$

$$1 \cdot 3 - 1 \cdot 5 + 2 \cdot 6 + 1 \cdot 0 = 10.$$

In this example, one can see that the kernel is placed in all possible positions where a complete union is possible in relation to its spatial pixel location. Practically, the kernel can be seen as a window that is placed on the top left corner of the feature map, and after computing the cross-correlation, it is moved horizontally and vertically. Moreover, in the above example covering the convolutional operation, the kernel uses a step size of one, so the window gets shifted by one pixel. In order to either decrease the computational costs or to down-sample the image, there is the need to skip positions [25]. To archive that, the parameter *stride* controls the step size of the kernel. [39]

As stated in [39], an activation function Φ is applied after creating the feature map and generating the activation map. More formally described for a kernel ω with a fixed size, the matrix $X^{(q)}$, where q indicated the layer, indices three dimensions, $X^{(q)}_{i,j,k}$ with $1 \leq i \leq hj$, $1 \leq j \leq hj$ and $1 \leq k \leq d$ and represents an output of a layer. A feature map for layer q is then calculated by

$$X^{(q)}_{i,j,k} = \Phi\big(\sum_s \sum_p X^{q+1}_{i+p,j+s,k} \omega^{(q)}_{psk} + b^{(q)}\big), \tag{2.9}$$

where s,p depict the size of the image. Furthermore, b represents the bias for layer q, therefore the output of one layer is the next layer's input. Furthermore, [25] proposes that convolutional neural networks do introduce a certain level of equivariance, which means that if the input feature map changes, the resulting output changes in the same way. More formally, functions f and g are equivariant if $f(g(x)) = g(f(x))$.

Using the current situation, the feature map shrinks with the layer, to control this factor, we apply method padding.

Given this situation, the feature map would shrink from layer to layer. Padding can be utilized to avoid the shrinking affect and keep the map size constant. There are three common methods of padding, where the first one is called *valid* and adds no padding to the image (the situation we are phasing

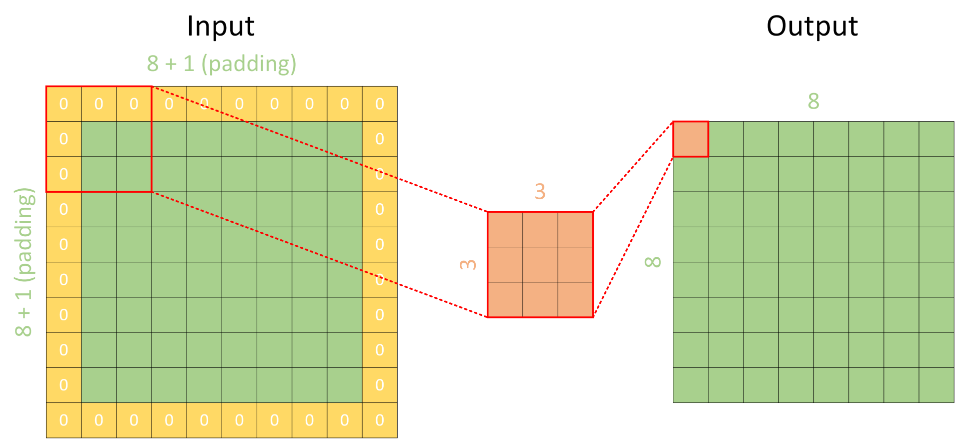


Figure 2.7: Padding avoids the shrinking of the feature map caused by the convolutional operation. One option is to add zeros to each side of the map such that the filter can cover every single pixel; this method is called *zero padding* or also called *same*.

right now, Figure 2.6), which leads to the fact that the image is down-sampled. The second method is called *zero-padding* (sometimes also *same*), which adds enough zeros to the image boundaries such that the size stays the same. Therefore, the output has the same $h \times w$ then the input. The third option results in up-sampling the output, which is reached by adding more zeroes to the image boundaries. Usually, the first two padding methods are more widely accepted. [25]

Another layer type in the convolutional neural network is the pooling layer [40]–[42], where the structure of the layer is similar to the convolutional one, but applies a pooling operation. This function does select certainly local features and promotes them employing their minima, average, or maxima. According to [25], the pooling adds a certain level of translation invariants to the layer, which has the benefit of the focus being set on whether a feature is in the activation map rather than its exact spatial location. Therefore, if a feature is shifted by a small amount, the output on the activation map does not change.

The functionality of the pooling layer can be imagined somehow, like the convolutional layer, with some significant differences. In [38] is described that this layer always produces the same depth d_q on the output than on the input. Furthermore, a window with the size of $P_q \times P_q$ is slid over the input feature map. Then the maximum value within the window is transferred to the new output map; this process is called max-pooling. Equivalent to this, the average-pooling takes the mean of the window, and the min-pooling takes the minimum.

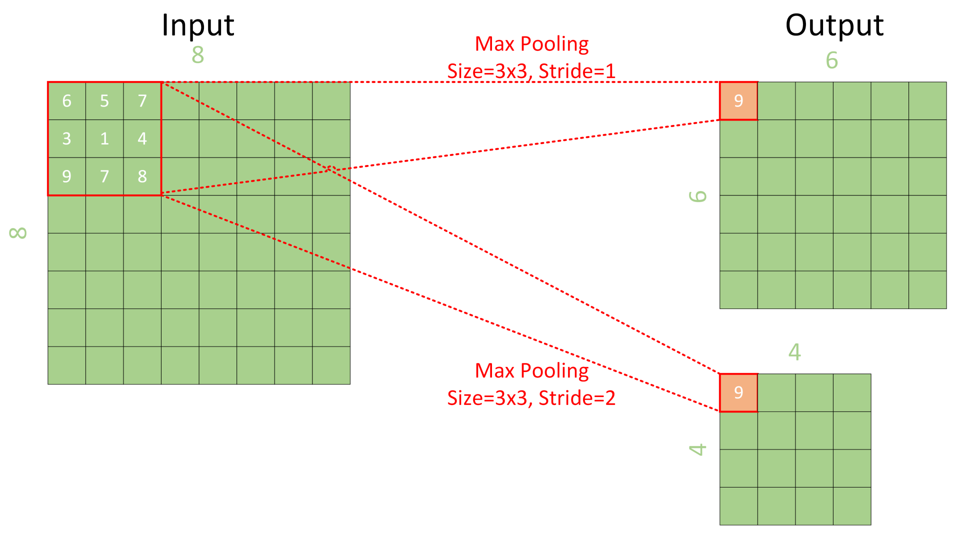


Figure 2.8: Visualization of the pooling operation. A kernel window with a fixed size is moved over the feature map, and either the minimum, average, or maximum value is extracted. In the figure, the window is moved over the input map, and a max pooling is used with a stride equal to one and a second time where the stride is equal to two.

Same as before, the stride parameter controls the step size of the pooling kernel and is able to downsample the feature map, usually, a default stride of two with a kernel size of 2×2 . Therefore, the pooling layer does reduce the spatial size of the feature map and is able to reduce it to a small constant size. Figure 2.8 shows this process visually. Moreover, there are also some other pooling techniques. For example, global average pooling does not use a window, instead, this method reduces an entire channel of the input feature map to a single value by taking its mean [43].

Furthermore, another layer is batch normalization (BN). As described in [44], most commonly used after convolutional layers to speed up the training phase by normalizing the input feature map. The BN is formally defined as

$$x' = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon + \beta}}, \tag{2.10}$$

where x is the input value and x' is the normalized output. Furthermore, γ, β are trainable parameters, μ represents the mean, σ is the variance, and ε is a small number to prevent zero values. These values, including γ and β, do not change during the inference phase. Therefore, it is possible to combine the kernel with the BN layers, which is called BN folding.

In the case of a multi-layer CNN, the last layers consist of high-level features such as objects. On top of the convolutional network are one or more fully connected (FC) layers (also called dense layers), comparable to the layers in a fast-neural network. The last FC layer does apply a particular activation, like softmax, which is selected to have a certain output type. [39]

According to [45], fine-tuning of a CNN is used on previously trained models to recover the accuracy after compression. Moreover, another case is if the network has been pre-trained on a different dataset. In this case, the network does already have some trained weights, but they do not match the ideal values for the focused dataset. Those values are optimized with fine-tuning to maximize the accuracy rather than training the entire network from scratch. While the training is relatively computationally expensive, the inference phase can be performed on edge devices such as an FPGA – some models are to large to even run the inference phase on edge devices.

All in all, the amount of parameters is controlled by a lot of aspects, next to the bias, the kernel values are trained. Therefore, they have a large impact on the total number. Additionally, the more filters are used, the more parameters are involved in the training process [38].

2.2.2 Classification Architectures

Convolutional neural networks were introduced in the late 80s, and since then, they have led to several breakthroughs [44], [46], [47], in the domain of computer vision. Among other reasons, the popularity of those networks is caused by the types of features generated by the layers. While the first layers of the network cover low-level features, such as boundaries and edges, layers located toward the top of the network extract high-level features, like objects [38]. Nevertheless, CNNs are able to solve computer vision tasks such as scene classification [48] and segmentation [49].

The dataset ImageNet is one of the standards when it comes to developing and training convolutional neural architectures. This dataset is part of the ILCVRC challenge [50], which was in 2012 with the architecture AlexNet. Since then, the popularity of those types of networks has raised and deeper architectures have been published. One example is the neural network structure VGG, created by [51], which outperformed AlexNet on ImageNet. While the general structure is similar, VGG is deeper and also includes a larger quantity of parameters. Furthermore, the trend has been to stack layers and create deeper neural architectures. However, the enhance amount of layers does not necessarily lead to accurate and better-quality networks, instead, under some conditions, it is even possible that the error rate can stagnate or even increase over epochs without being overfitted. Optimizing the error rate in this situation is a non-trivial problem. Additionally, the raising count of layers towards a large depth comes with the drawback that the number of hyperparameters does expend too. This cause computationally expensive training processes. Consequently, the smaller modes tend to be more efficient in terms of time consumption. Furthermore, they are also more complex and deeper models are non-trivial to deploy to end devices for instance an FPGAs. [52]

This leads to the current trend of implementing networks that do not simply stack layers. However, nowadays, the model architectures are implemented, which are large but have a technique to speed up the training process. Some other networks do also focus on the ability to deploy to mobile devices. [53], [48]

Nevertheless, the training of a CNN is a computationally expensive process. Even when using non-consumer grade Hardware such as GPU-Servers or other specialized accelerators, some training does require a large time investment. Therefore, it is important to optimize the network architectures. [52]

Next to network compressive methods, such as pruning or quantization, a method to optimize the training is by changing the design, for example, with the employ of skip connections within the network, e.g. in the residual blocks of ResNet or InceptionNet. While VGG is easy to implement and modify, network architectures such as InceptionNet do come with the costs of being more complex to implement. Some other network architectures are designed to be deployed to mobile hardware, like a mobile phone or an FPGA [53].

The following is a selection of the most important network architectures used in this work and gives an overview of their design. Additionally, adding information to the networks will be given on the basis of training the dataset ImageNet from the ILCVRC challenge [50].

VGG: According to [51], the architecture includes five convolutional blocks, which have a kernel size of three, zero padding, and a stride of one. Each block is separated by a max-pooling layer which has a stride of two. Each hidden layer das also apply a ReLU activation. The Architecture is summarized in Table 2.3.

While VGG with 16 layers (VGG16) reaches a top-1 accuracy of 71.3% on ImageNet (ILCVRC challenge 2012), VGG with 19 layers (VGG19) does reach a top-1 accuracy of 71.3%.

Table 2.3: Summarization of the architecture on ImageNet of selected VGG networks. The convolutional layer is noted as conv-<number of filters>-<kernal size> and the dense layer is noted as dense-<number of filters>. The table has been adapted from [51].

Block	16 layers	19 layers
	Input $(224 \times 224 \times 3)$	
Block 1	conv-64-3 conv-64-3	conv-64-3 conv-64-3
	max pool	
Block 2	conv-128-3 conv-128-3	conv-128-3 conv-128-3
	max pool	
Block 3	conv-256-3 conv-256-3 conv-256-3	conv-256-3 conv-256-3 conv-256-3 conv-256-3
	max pool	
Block 4	conv-512-3 conv-512-3 conv-512-3	conv-512-3 conv-512-3 conv-512-3 conv-512-3
	max pool	
Block 5	conv-512-3 conv-512-3 conv-512-3	conv-512-3 conv-512-3 conv-512-3 conv-512-3
	max pool	
	dense-4096	
	dense-4096	
	dense-1000	
	softmax	
#param.	138×10^6	144×10^6

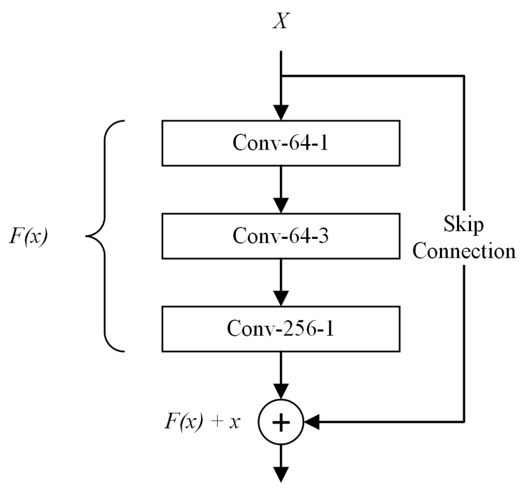


Figure 2.9: Representation of the *bottleneck building block* used to build ResNet with 50 and more layers. The convolutional layer is noted as conv-<number of filters>-<kernal size>. The figure has been adapted from [48].

ResNet: The convolutional neural network architecture ResNet, which is short for residual network and has been introduced by [48]. As stated in this publication, ResNet does use skip connections to help prevent overfitting and optimize the information flow within the network. Nevertheless, the network architecture is constructed using building blocks. For configurations with more than 34 layers, the block-type *bottleneck building block* is used. This block consists of three convolutional layers, where the last one has a four times larger filter size than the previous layers. A skip connection does add a shortcut over the full block. Furthermore, each convolutional layer does have a stride of one by default and applies zero padding. They are additionally followed by batch normalization and a ReLU activation. Figure 2.9 visualizes the configuration of this block.

The architecture consists of four building blocks followed by some top layers. Every block consists of a count of building blocks with the same configuration, except the first one. While the following layers are defined as stated before, the first building block does apply a strait of two. Additionally, another convolutional layer, with the same configuration as the last layer in the building block, is added to the skip connection. While there are several options for shortcut connection, this one is supported by the large deep learning frameworks. A selection of ResNet networks is summarized in Table 2.4. [48]

Table 2.4: Summarization of the architecture on ImageNet of selected ResNet networks. The convolutional layer is noted as conv-<number of filters>-<kernal size> and the dense layer is noted as dense-<number of filters>. Furthermore, the bottleneck building block is stated as bottleneck-<number of filters>. The table has been adapted from [48].

Block	50 layers	101 layers	152 layers
	Input (224 × 224 × 3)		
	conv-64-7, stride 2		
	max pool, stride 2		
Block 1	[bottleneck-64] × 3	[bottleneck-64] × 3	[bottleneck-64] × 3
Block 2	[bottleneck-128] × 4	[bottleneck-128] × 4	[bottleneck-128] × 8
Block 3	[bottleneck-256] × 6	[bottleneck-256] × 23	[bottleneck-256] × 36
Block 4	[bottleneck-512] × 3	[bottleneck-512] × 3	[bottleneck-512] × 3
	average pool		
	dense-1000		
	softmax		
#param.	26×10^6	45×10^6	60×10^6

While ResNet with 50 layers (ResNet50) does reach a top-1 accuracy of 74.9% on ImageNet (ILCVRC challenge 2012), ResNet with 101 layers (ResNet101) does reach a top-1 accuracy of 76.4%, and lastly, ResNet with152 layers (ResNet152) does reach a top-1 accuracy of 76.6%.

Note that there does exist several versions [54], [46] of ResNet family, this one is the first introduced architecture.

SqueezeNet: The neural network architecture SquezeNet focuses on model compression. The more lightweight model is reached by changing the design of the model architecture. The authors of [55] are given the example that with SuqezeNet can create an AlexNet-like model with 50 times fewer parameters than the original architecture. Additionally, the scope is to create models that are compressed in terms of parameters. Among others, the advantages are that the training is optimized, and the models are able to be deployed to mobile hardware accelerators such as an FPGA.

While the work on [55] does also research on applying model quantization to some SqueezeNet configurations, this section concentrates on the vanilla version of the model architecture. Nevertheless, the authors do explain that one problem is that the CNNs are often fed with three-channel images, and subsequent layers do keep this dimension. Another problem of CNNs is the dimension of the kernels. For example, the architecture VGG does apply filters with a size of 3×3 at most layers, which is a computationally expensive operation. All this maximizes the amount of parameters in neural networks. Consequently, SqueezeNet has three strategies to reduce this quantity, the first one is to replace all 3×3 kernel filters with a dimension of 1×1, the second is the decrease the input channels to the filters, which is done by adding a squeeze layer consisting of 1×1 convolutional filters. The third strategy is to maximize the activation map with a late down-sampling in the network. All strategies are applied with a so-called fire module. This block consists of a squeezing part at the beginning, realized by 1×1 convolutional layers, followed by an expansion part consisting of a mix of 1×1 and 3×3 convolutional layers. The final architecture of the SqueezeNet can have several forms, the easiest is to create an AlexNet or VGG similar architecture, and another option is to apply skip connections between several fire modules. A SqueezeNet network is summarized in Table 2.5.

Table 2.5: Sample of the architecture SqueezeNet in its basic form based on AlexNet. The convolutional layer is noted as conv-<number of filters>-<kernal size> and the fire block is noted as fire-<number of filters>. The table has been adapted from [55].

Block	Basic SquezeNet	# bits
	Input ($224 \times 224 \times 3$)	
Block 1	conv-96-7, stride 2	6
Block 2	max pool, stride 2	
Block 3	fire-128	6
Block 4	fire-128	6
Block 5	fire-256	6
Block 6	max pool, stride 2	
Block 7	fire-156	6
Block 8	fire-348	6
Block 9	fire-348	6
Block 10	fire-512	6
Block 11	max pool, stride 2	
Block 12	fire-512	6
Block 13	conv-1000-1, stride 1	6
	average pool	
#param. (unpruned)	1.248×10^6	
#param. (pruned)	421×10^3	

Compared to AlexNet, which archives a top-1 accuracy of 56% on the ImageNet dataset from the 2012 ILSVRC challenge, this version of SquashNet, which does not apply pruning or quantization, reaches a top-1 accuracy of 57.2% with five times fewer parameters. When applying other compression methods to the networks, even dramatically fewer parameters do archive the same accuracy. [55]

2.2.3 Segmentation Architectures

As mentioned previously, image segmentation is the one of two main category when it comes to classifying the content of an image. In summary, in this category, each pixel is classified, rather than before, where a class is assigned to an entire image. While there are multiple types of pixel classification, see above, the scope is set on object detection and semantic segmentation. An

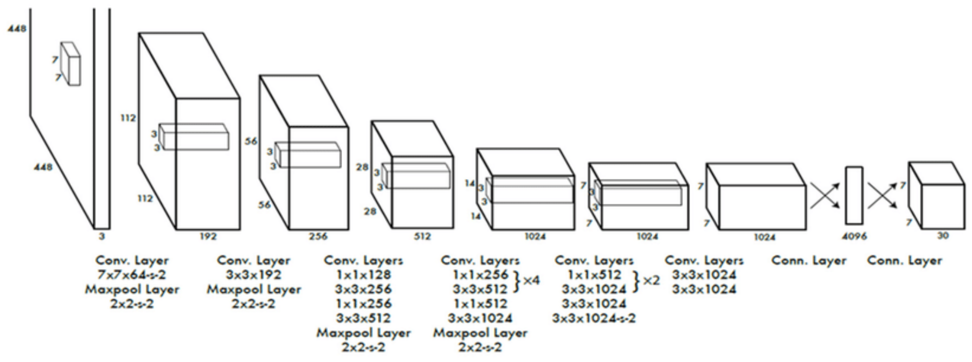

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 2.10: Representation of the YOLO architecture. The figure has been first published in [56].

example could be a remote sensing scene, visualizing a forest that needs to be segmented to detect actively burning wildfires and their affected area. The main principle follows the same strategy and techniques as described before.

Classical, patch-classifying CNNs have a set of dense layers on top to be able to compute a vector that indicates the class of the image. The neural networks architecture YOLO introduced by [56] removes those layers to be able to compute the bounding boxes of certain trained objects. The top dense layers flatten the feature map that is generated by its last convolutional layer to a vector, which is then minimized to a target length which is the number of classes. When removing those layers, one can compute the location and size of the bounding boxes on the basis of the last feature map. Figure 2.10 visualizes the YOLO network architecture.

A commonly state-of-the-art architecture for semantic image segmentation called U-Net has been introduced by [47]. Figure 2.11 visually represents an example of such as model, where one can see that the networks consist of two main components. The first one is the left half is a CNN that scales that image down to a dense representation. Instead of computing the class using a prediction layer, the second component, the right side, scales the image up again. Skip the connection between the different block levels and transmits information to the up-scaling side to help the classification process and prevent errors. The two components do not necessarily need to have the same structure, such as it is done in [49].

Furthermore, the structure of the architecture itself can be freely designed as it is done with patch classifying CNN. Before, the task of transfer learning used pre-trained weights for the full network and only fine-tuned the top layers.

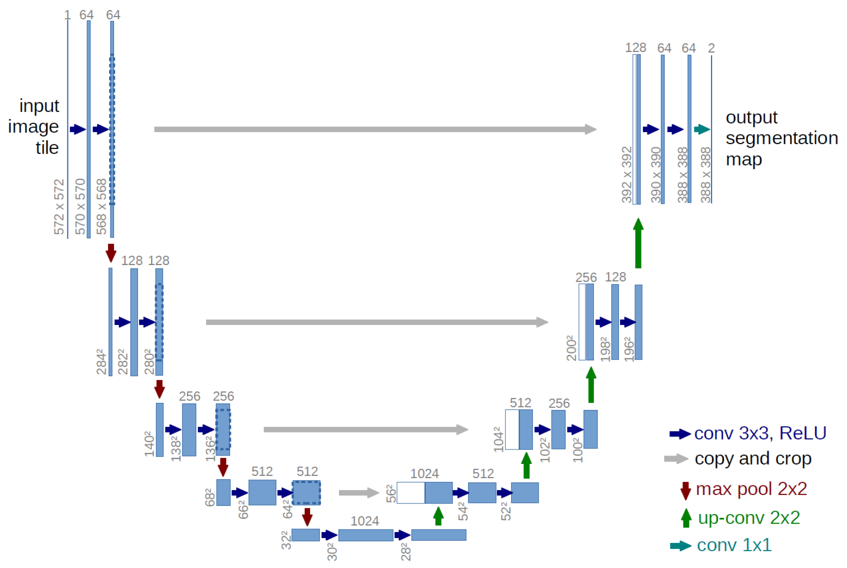


Figure 2.11: Representation of the U-Net architecture that consists of an encoding side (left half) and a side that decodes the image again (right half). The figure has been first published in [47].

It is non-trivial to use pre-trained weights. Anyways, the encoding left part of the architecture can be chosen to have, for example, a MobileNet structure, which opens the possibility to use the weights from ImageNet for this side of the network. In this case, the fine-tuning needs to compute the optimal parameter set for the decoding side of the U-Net architecture.

2.2.4 Modern Model Compression

Model quantization aims to reduce the size of the model and the computational complexity of neural networks. While prunin removes information that does not significantly contribute to the classification result, quantization reduces the precision of mostly weights and activation.

The state-of-the-art in the field of quantization includes a variety of methods and schemes. One is channel-wise quantization. The algorithm proposed in [57] is such an algorithm is called Distribution-Aware-Quantization, which aims to quantize the quantization parameters of the networks. Additionally, the algorithm considers the distinct distribution within each channel of an image. Compared to that, [11] addresses both weights and activation in their algorithm, called Distance-Aware-Quantization, which handles the problem of the undifferentiability of rounding functions leading to the problem of having gradient mismatch. To overcome the mismatch and be able to train the model, the introduced algorithm consists of a distance-aware soft rounding function and a control parameter. Furthermore, [12] states that the quantization error correlates with the accuracy after fine-tuning the model. Because of this reason, this publication proposes a method that uses the functional characteristics of a neural network by permuting the weights to find combinations that are more optimal to quantize. Additionally, a final k-means algorithm can minimize the error caused by the previous steps.

Other publications do focus more on the optimization of training strategies to be aware of the quantization. For example, the scheme called Quantization-aware-training introduced in [58] focuses on exactly this problem. The authors of this publication state that integer arithmetic can be considered as being more efficient than floating-point operations. Those operations do come with the drawback of losing precision, therefore, an loss of accuracy can be expected. Furthermore, the publication proposes a training method that keeps track of the quantization and provides a trade-off between accuracy and performance during inference. Another method is introduced by [59] which is called Once Quantization-aware training (OQAT) and focuses on the stability of the network's accuracy, especially on lower bit rates. While those training methods only consider one single precision type within the neural network, the authors [60] proposed a scheme that is able to train multiple precisions within the same network, such as 16FP precision at some layers including 32FP for others. The experiments within the publication showed that the classification accuracy had been slightly increased compared to a baseline model (ResNet50 with 32FP). Nevertheless, one needs to mention that the problem with mixed precautions is to find a balance between the number of bits available to represent weights, activations and the efficiency of the quantization process itself [61]. Nevertheless, the authors of [45] addressed this training method and created an approach that uses row-wise mixed-Scheme quantization as well as multi-precision. The authors state that this algorithm is able to preserve accuracy while having mixed quantization schemes and multiple precision within a layer. Additionally, the authors also tested their algorithm on an FPGA besides received a speedup of 3.65$\times$ in inferencing ResNet18 on ImageNet compared to a 4-bit fixed-point baseline.

When it comes to very low-precision quantization, one member of this family is the method introduced in [62], which compresses the weights of 32FP neural networks with a range of 3 bits $\{-1, 0, 1\}$. Experiments showed a minimal loss of accuracy compared to baseline classifications with a 16 times smaller model besides half of the computational effort. Due to certain constraints, among others, caused by hardware like FPGAs, there is much research focusing on model quantization using low-precision such as [63] [64], but also on algorithms that do focus on precision ranges that are logarithmic, like [65].

As it is the same for quantization, the state-of-the-art in the field of pruning also includes a wide range of methods and schemes. One example is filter pruning in convolutional neural networks. The authors of [66] propose such a method that has a comparatively small contribution to the classification result. Additionally, this publication states that the importance of filters to the allover network can be calculated using the L1-norm of weights, where small values are considered as having a small impact on the network. Other research does also add complex search algorithms, for example. The authors of [67] created a method that combines channel pruning with neural architecture search. This leads to a technique that is able to search for a compressed version of the given model which meets the given requirements.

Many quantization algorithms have been introduced with a focus on training, related problems and errors. Also, in pruning exists a wide range of research projects which address the training phase. The authors of [68] propose a method called dynamic sparse representation, which is about the limitation of having a high computational cost to reach the point of having fewer parameters without a significant accuracy loss. The project dynamically adjusts the sparsity of the neural network during the training phase, which is done by adding a pruning relevant regularization term to the loss function. In consequence, experiments show that significantly fewer parameters can be reached compared to previously used pruning methods. Moreover, the proposed methods can be used to compress. Pre-trained CNNs without a significant accuracy loss. Other examples that focus directly or indirectly on the training before, during, or after pruning are [69] or [70].

Furthermore, it is also possible to combine pruning and quantization into one method. The authors of [13] propose a method that first uses layered channel pruning to select areas within the filters. The next step is to quantize the selected layers. Next, all unquantized areas get re-trained. This process is repeated until every value has been quantized. So, this approach uses the pruning method as a selection method. Another one is the method proposed by [71], which compresses the networks without a large decrease in accuracy. Among other architecture types, these works compress convolutional neural networks by combining pruning, trained quantization, and a Huffman encoding.

2.3 Set Similarities for Spatial Data

Spatial data do come in various shapes as well as forms, and there are many of them. For example, do the authors of [72] state that this type of data is the basis for lots of different sciences, such as in the social domain or even in the engineering world, for example, when working within the aerospace field. But also in medicine does spatial aspects play an important role, as we saw in 2020 including the following years when the world needed to handle the pandemic situation caused by COVID-19. One special data type is named geospatial, this type is able to capture the geometry of an object with respect to its location in an (Euclidean) space. Anyways, there are roughly two categories that this type of data can be grouped into that is raster and vector shaped. The former one does represent data that is matrix shaped, for instance, remote sensing image that visualizes the surface of the Earth. Considering this example, one characteristic is that each pixel covers a specific area on the ground (resolution, like 10m $\times$10m per pixel and the related coordinates in the metadata), additionally, each pixel is mapped to a specific location on the Earth's surface. Furthermore, the shape of the matrix, as well as the complexity of the content, can differ between the raster-creating device. For example, provides Sentinel-2 [9] 13 multispectral bands or Sentinel-1 [73] two different polarized radar matrices. An example of raster data is shown in Figure 2.12a, in this case, the remote sensing image is visualized where parts are affected by a wildfire. On the other hand, vector data do represent the information in the form of shapes, for example, it can be used to represent the governmental boundary of a country or a district. One common format to express vector data is called *Well Known Text* (WKT), which covers, among others, three main types that are *point*, *line*, and *polygon.* One representative is a trajectory that holds the data of a path that has been tracked, such as the way from home to the working place. An example of vector data is shown in Figure 2.12b, in this particular case, the polygon defines the areas that are affected by the burnt area. Especially trajectories are generated by a large number of people every day by using their phones for the navigation function for their map applications using GPS.

All in all, there are three most important characteristics the geospatial data have. The first one is volume, this defines the amount that is available,

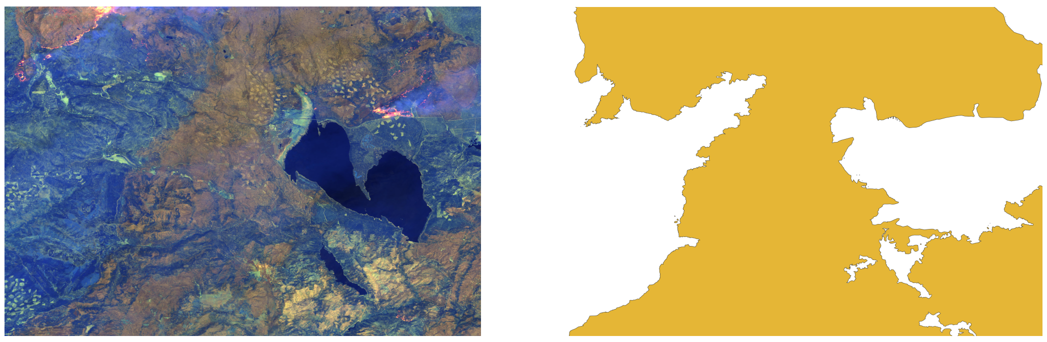

(a) Raster data (b) Vector data

Figure 2.12: In the geospatial domain, there are roughly two types of data. The first is raster which is shaped in the form of a matrix, such as an image. (a) shows an example and depicts and remote sentinel-2 RGB scene, where one can see at some locations and burnt areas. Second is vector data, this type rather defines the shape is data, like the border of a country. An example is visualized in (b), where the polygon states the precise location of the burnt area that is visible in the satellite image.

which is usually massive. For example, does the data center from the German Aerospace Center (DLR) holds approximately more than 60 PB on satellite data from nine missions [1]. The second characteristic is the velocity, which is the amount of data that is generated within a certain time period. Considering this number can be calculated, what would we need to be able to compute the data in real time. The last in the variety that indicates the diversity of the data, for example, spatial data can cover geo-referenced social media data, but also trajectories from ships.

While this section focuses on vector data, this thesis covers both types.

2.3.1 Distance Metrics for Spatial Data

As explained above, geospatial data comes in different shapes, and its volume is often very large. Therefore, it can be a challenge to calculate the similarity between two objects. Because of this reason, some distance metrics need to be mentioned in this work. At the same time, there are information-theoretic approaches, such as the Normalized Compression Distance mentioned in Section 2.1.2.3, that are especially important and commonly used for movement data like trajectories. One is called Intersection over Union (IoU), alternatively named Jaccard Index, which states the similarity of two objects by calculat-

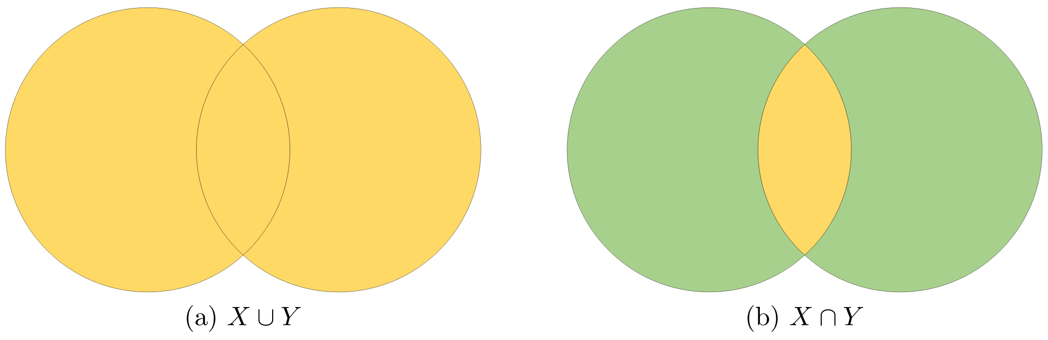


Figure 2.13: The intersection of two objects can be done by using logical operators. One example example is to combine two circles X and Y by $X \cup Y$, this combines the two areas whether they are overlapping or not. On the other hand, if the union is calculated with $X \cap Y$, only the intersection is taken.

ing the overlap, if the overlap is at its maximum, the objects are considered equal. Anyways the IoU is also commonly used in object detection as well as segmentation tasks.

Figure 2.13 visualizes the overlap between two objects. It can be seen that there are two objects, A, B, while there are many forms the objects can overlap, there is an inner as well as an outer intersection. The former one is the overlapping area, and the outer one is the area of the two united objects. Based on principle, the IoU is defined in Definition 2.10.

Definition 2.10. *According to [43] let IoU be the Intersection over Union (also known as the Jaccard index) that calculates the similarity between two sets x and y is formally defined as*

$$IoU(x, y) = \frac{\mid x \mid \cap \mid y \mid}{\mid x \cup y \mid},$$

where a the rage is $0 \leq IoU(x, y) \leq 1$. While zero states the maximal unsimilarity of the sets, one states that they are equal.

According to [74], the Jaccard Similarity $D_{jaccard}$ can be defined by subtracting the IoU from one, consequently, the metric can be defined as

$$D_{jaccard}(x, y) = 1 - IoU(x, y), \tag{2.11}$$

where x and y are two sets.

Another metric that is commonly used for geospatial data, especially for trajectories, is the Frechét distance. In order to calculate the distance between two trajectories, more precisely, the largest distance between two segments. More formally, the Frechét distance is defined as

$$D_{frechet}(x, y) = \inf_{\alpha,\beta} \sup_{t\in[0,1]} D(x(\alpha(t)), y(\beta(t))), \tag{2.12}$$

where $\alpha(t), \beta(t) : [0, 1] \to [0, 1]$ are two points on the trajectories x and y. [74]

2.3.2 Bloom Filter and Distances

This section is an excerpt of the text published in [P5].

Bloom filters are probabilistic data structures first provided by [75] and extended by [76]. The data itself is held within a pre-initialized binary array where its fixed size can be freely chosen, additionally, the data is embedded with hash functions. In more detail, the size m if a filter is commonly chosen in the format of $m = 2^l$, where $l \in \mathbb{R}^+$. Furthermore, the data is embedded with k-pairwise hash functions that compute k positions within the filter that are set to one. Practically said, the insertion of an element e into a filter BF employing k pairwise hash functions h leads to a set of positions within the array $h_i(e), i = 1...k$, where these positions are set to one such that $BF[h_i(e)] = 1, i = 1...k$. While this describes the process of embedding an element, it is necessary to check whether a data item is within a filter. This is simply done by calculating the positions in the same way using the k hash functions and testing if values on the positions are one, if not, the element has not been embedded. An essential characteristic of Bloom filters is that there is not possibility of receiving false negative alerts. Furthermore, the expected error rate can be controlled by the length of the filter including the number of hash functions that are used. Figure 2.14 represents a Bloom filter and the embedding data process.

One important feature of the Bloom filters is called the fraction of zeros, which indicates the percentage of zeros within the filter, defined in Definition 2.11.

Definition 2.11. *According to [6], let FOZ be the fraction of zeros of a Bloom filter allocated with a length m and k hash functions. This is formally defined as*

$$foz(k, m, n) = \left(1 - \frac{k}{m}\right)^n \approx e^{-kn/m},$$

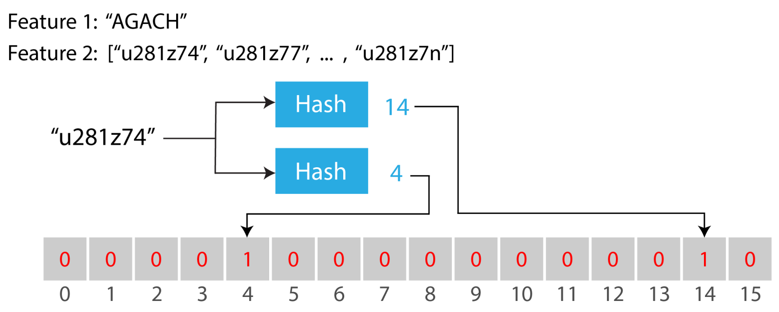


Figure 2.14: The Bloom filter is a binary probabilistic data structure that inserts elements using hash functions. Considering two types of features are generated from trajectories, where one is the segment orientation encoded into letters and the geohash. For each element, the n positions are calculated by n hash functions, which are then set to one. Nevertheless, the reading of data is the reverse operation. If one wants to check if the element "ABC" is in the filter, the positions, if all positions within the Bloom filter are set to one, the element is part of the filter.

where n is the number of inserted elements.

In previous work, the group merged in [76] this idea of handling Bloom filters with the Jaccard distance metric to be able to calculate the similarity between two filters. Furthermore, they describe that it is needed to adapt the IoU or Jaccard index to be able to take a meaningful union. Therefore the amount of elements n_a that have been embedded into the bloom filter is calculated using the FOZ based on a filter BF_A as follows:

$$n_a \approx -\frac{\log(FOZ(BF_A))m}{k}. \tag{2.13}$$

Based on this, the authors said that the number of elements that are part of two filters A and B, can be calculated by logically join them, formally defined as

$$E_{union} \approx -\frac{\log(foz(BF_{A \vee B}))m}{k}. \tag{2.14}$$

Additionally, the authors state that by reformulating the denominator of the IoU equation (see Definition 2.10) to $\mid x \cup y \mid = \mid A \mid + \mid B \mid - \mid x \cap y \mid$, this leads to $E_{intersection} = n_A + n_B - E_{union}$, the adapted Jaccard distance can be described as follows:

$$E_{jaccard} = 1 - \frac{E_{intersection}}{E_{union}}. \tag{2.15}$$

2.3.3 Space-Filling Curves

The motivation for the simplification or compression of high dimensional data is often to reduce the computational effort, minimize the memory footprint, or simply reduce the complexity. The space-filling curves do focus on exactly this task, where this algorithm is used to visit every location within a space with a given precision and a single non-overlapping line. This technique finds its application in geographic information systems but also in image compression, computer graphics, and much more. In geoscience, they are, for example, used to create more efficient indexing systems for spatial data.

One well-known representative is the Hilbert curve. According to [77] the algorithm divides the n-dimensional cube $[0, 2^p)^n$ into 2^p per per side. This results in four locations if $p = 1$. A line is visiting then each location without overlapping. In the next iteration, the space is divided into 2^p subspaces, p is increased, and the routine is applied again. Figure 2.15 shows an example of this proven with three iterations $p = 1, 2, 3$. The first subfigure from the left side shows the first iteration with $p = 1$ and the pattern for the visiting order. While the middle subfigure has $p = 2$, the most right is the last iteration using $p = 3$.

Another representative is the Z-curve, which works in principle the same as Hilbert's curve. An image is divided into a raster with 2^p locations per side, consequently, in the first iteration, an image gets split into four rectangles. In contrast to the visiting order before, considering the first iteration the non-overlapping line starts at the top left area and follows a Z-order. Furthermore, each subarea is assigned to a binary increasing number, following the same Z-order. The next iterations follow the same principle as before, the space is further divided, and the algorithm is applied again. This process is shown in Figure 2.16, where the most left subfigure visualizes the first iteration with $p = 1$, furthermore, the right image depicts the second iteration $p = 2$. One can see that the most right bottom field in case of $p = 2$ can be identified using the binary code 1111. [78]

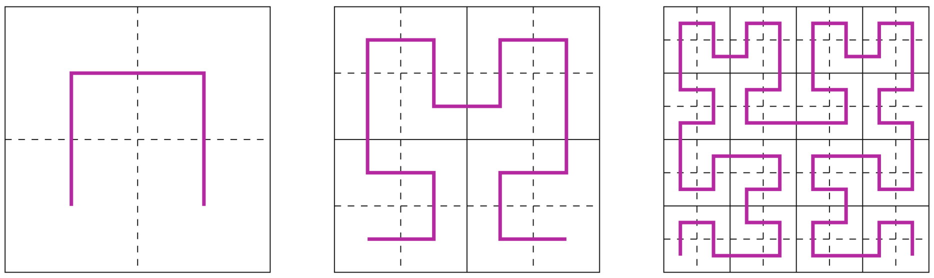


Figure 2.15: This visualizes the process of calculating three iterations of Hilbert's curve. The left image shows the first round with the lowest precision, therefore, the area is divided into four subareas. Each point within the pace is mapped to one of the positions. The middle image depicts the second iteration, which tiles the area into 4 × 4 areas, which provides for each inserted point a smaller error than before. The right image shows the third iteration of this example and therefore provides the smallest error for points within the space. This is because the mapping of the source points is more precise by tiling the patch from iteration one into 16 smaller areas. The figure has been first published in [78].

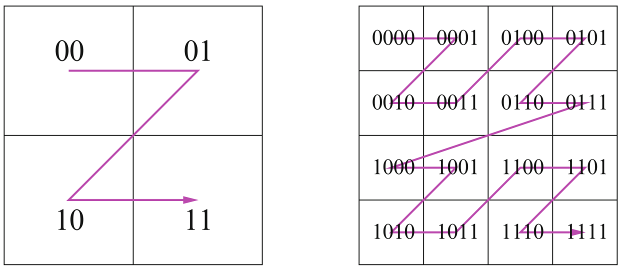


Figure 2.16: This visualizes the process of calculating three iterations of the Z-curve. The left image shows the first round with the lowest precision, therefore, the area is divided into four subareas. Each point within the pace is mapped to one of the positions. The middle image depicts the second iteration, which tiles the area into 4 × 4 areas, which provides for each inserted point a smaller error than before. The right image shows the third iteration of this example and therefore provides the smallest error for points within the space. This is because the mapping of the source points is more precise by tiling the patch from iteration one into 16 smaller areas. While the tiling is similar to Hilbert's curve, each field is assigned to a binary number, and the path of the numbering follows a "Z". The figure has been first published in [78].

2.3.4 Geohash

In addition to the local geometry features of orientation, we employ a global geometry feature based on the Geohash string encoding of spatial location. The Geohash is computed by uniformly splitting space into cells enumerated with a discrete Z-curve. The index of each cell is encoded using a BASE32 encoding to form a string. Thereby, longer strings contain more bits per coordinate and thus refer to a finer grid resolution, as well as prefixes of a string constitute supersets of the string [79]. It is worth noting that the Z-curve nature allows to find the string of a neighbor with simple table lookups, a welcome property for spatial exploration. Figure 2.17 represents a trajectory where the Geohash area of the points is marked.

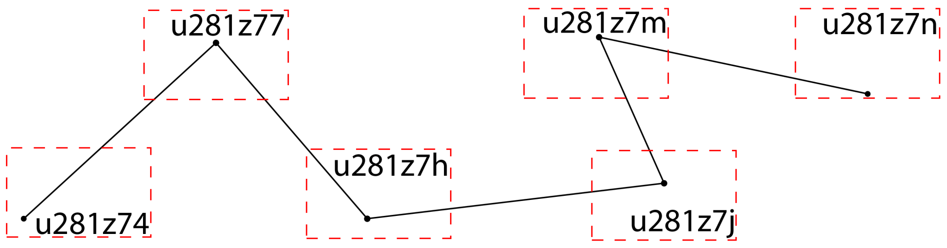


Figure 2.17: The Geohash represents a global grid box as a string. We assign geohashes as the name of the cell a trajectory sample falls in.

2.4 Common Hardware Accelerators

The training of a deep learning model is a computationally expensive task and time-consuming task, where the choice of used hardware accelerator has a large influence on the training speed. One aspect which needs to be considered nowadays is the power consumption of accelerators which has been increasing in the last decades [52]. Especially when deploying those tasks to specialized mobile hardware, such as an FPGA mounted into a satellite, it is important to keep an eye on the power needs because it influences the lifetime, total costs, efficiency, and much more.

To avoid going into detail about the design, usage, and benchmark of different hardware accelerators, this section focuses on the availability of systems and their peculiarity when deploying deep learning models. More detailed information can be taken from [80], [81], [82], which focuses on edge devices, such as FPGAs.

A deep neural network can be trained on different types of hardware accelerators. The first one is consumer CPUs and GPUs, where training on the latter option is preferred due to the better performance. Additionally, next to the consumer versions, there are also data center solutions for instance a GPU server with, e.g., two Nvidia A100 or even special deep learning server solutions like an Nvidia DGX-1. While the throughput is higher on data center solutions, so are the hardware costs. Those accelerators are natively supported by the major deep learning framework such as PyTorch or TensorFlow. Next to consumer as well as non-consumer versions of CPU/GPU systems, it is also possible to deploy the training and inference to embedded devices.

Especially when deploying convolutional neural networks to FPGAs, it is important to consider that only some layers are supported by the FPGA frameworks. For example, when using a Xilinx ZCU102 board, some network architectures cannot be deployed out of the box. This makes it necessary to reimplement some models to make the needed changes based on the layers which are supported by the framework. Furthermore, a quantization followed by compiling the model to the hardware format is needed. [83]

3 Analysis of the Data-driven System's Requirements

The previous chapter introduced the fundamental technologies, methods, and algorithms that are used or required to understand the following sections. Compared to that, this chapter discusses the theoretical perspective of optimizing a data-driven system such as a machine learning pipeline with the use of compression. Note that parts of this chapter has been published in [P8]. In the throughout of this work, the term compression does not only refer to an algorithm like GZIP or BZ2, instead, it is a mechanism that reduces the size of the system with respect to a certain scale. For example, to minimize the number of floating-point operations in a deep learning network to archive a higher throughput.

Machine learning methods and end-to-end processing pipelines that include learning algorithms became a common part of spatial computing. Additionally, data simplification techniques and image manipulation became are commonly used. Moreover, the current research on the topic includes the deployment of computer vision mechanisms in energy, resource, and space-constrained environments, such as on board of satellites in space. Because of the unique constraints there are several problems that need to be solved, that cover especially the efficiency of the data-driven systems.

An example of a classical deployment is provided by Figure 3.1 and visualizes the situation in remote sensing. A satellite is orbiting the Earth and constantly creates data by sensing the surface. Each time a satellite is visible from the ground station, the produced data can be transmitted. There are two challenges in collecting data in space: the large amount of data compared to the small capacity of the downlink and the limited bandwidth of the communication channel. Therefore there is a trade-off between being able to download all created data and the number of ground stations. Furthermore, this leads to a significant investment of resources. While this is a specific challenge for a particular domain, the same type of problem in handling the amount of

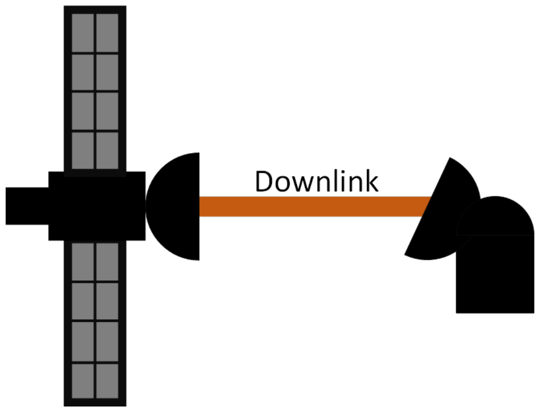


Figure 3.1: The transfer of data from a satellite to a ground station is one representative of classical deployment in geoscience. A satellite is constantly orbiting the Earth and producing data. The station can create a downlink to transfer the data to the ground when the satellite is in the viability window. Unfortunately, there are two constraints, the limited speed and the time window.

data exists in the geoscience domain in manifold situations. Other examples would be autonomous devices, such as cars, where the cameras and sensors produces frequently a large amount of data. Nevertheless, to eliminate the computational issue is to scale the system horizontally or vertically, or to solve it by (software) design. Considering the remote sensing example again, one possibility is to optimize the efficiency of the transmission channel by deciding whether a scene is relevant (send) or not (drop). This means, a satellite with the task of detecting wildfire would only transmit suspicious scenes that needs to be checked. The consideration of only using relevant data leads then to more efficient usage of the downlink. A more technical perspective to this situation is depicted in Figure 3.2 showing two main components of Earth observation systems, the satellite on the left and the data receiving ground station on the right. One can see that the data is sent to an edge AI chip over a channel with a large bandwidth, like an onboard bus, and the downlink is the system's bottleneck. Consequently, sending only relevant data over this challenge while still analyzing all observations using onboard computing reduces the amount of data that is transmitted over the communication channel.

To further illustrate the challenge, let us consider real-world satellite missions in orbit. The German aerospace centers stated in 2018 that they were hosting 90 PB of Earth observation data and made the assumption that in 2030 the volume of 185 PB is exceeded [1]. Similar results are reached by large-scale autonomous driving projects. From the technical point of view,

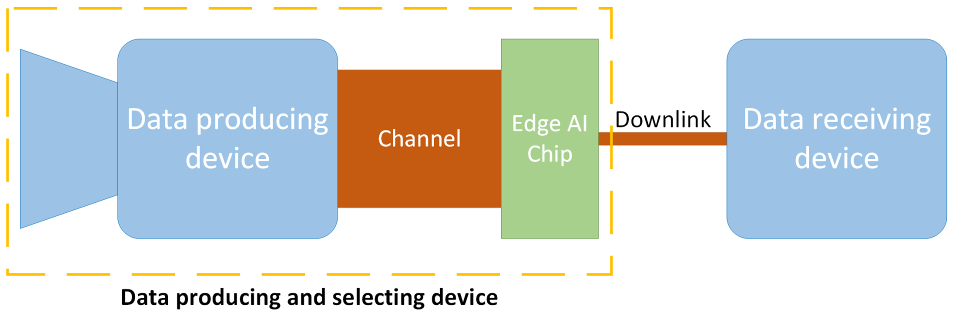


Figure 3.2: In remote sensing, there are two main components. One is the data-receiving ground station, and the second is the satellites. One can see that the communication within the satellite is larger than the wireless connection to the ground. In order to increase the efficiency of the system is to embed an ai chip into the satellite that is connected to the data-producing component with a large bandwidth. The onboard chip could decide whether a scene is relevant for the defined use case. If the decision is negative, the image is pruned and does not block important time on the downlink. As a consequence, the communication is compressed and increases the efficiency of the entire system.

one option of being even able to process such an amount of data is to scale the systems hardware, for example, an Nvidia DGX SuperPOD or even to LRZ's SuperMUC-NG. Both options lead to high operational costs. While this is an extreme case, it should visualize the importance of optimizing algorithmic processes toward maximum efficiency without lowering the accuracy under a certain level.

Those examples show that an onboard machine learning algorithm, for example, in space missions can lead to more data-efficient communication. While this is only one building block of a data-driven that has been considered, compression can be applied to all components. This optimizes each individual item and can lead to a maximization of the system's efficiency in terms of throughput, power consumption, and time needed for the computation. Scaling the system's hardware is still an option, but before it is needed to scale the system's performance by the algorithm's design. Moreover, it is also necessary to investigate the requirements and objectives for the future of spatial data science.

Based on the aforementioned examples, we derive the requirements given in Table 3.1 for a data driven system and derive hypotheses and from it.

Table 3.1: Overview of the requirements for the future of spatial computing.

Identifier	Requirement
Req-1	The data-driven system should process any incoming data stream in real-time.
Req-2	The data-driven system's power consumption should be minimized keeping the models accuracy degradation in a small acceptable range.
Req-3	The amount of data transmitted over the downlink should be minimized without degrading the performance on the observational task.

Important to reach them to be able to be prepared for the future in spatial computing. Therefore is in needed to investigate the potential of compression.

Objectives: As mentioned above, how we treat data and the algorithms need to be designed more efficiently, because the presented issues are only a selection of scenarios where the computation of a large amount of data is non-trivial. This work focuses on optimization by compressing the different building blocks. Therefore the objective of this work can be summarized as follows:

To research the role of compression in spatial computing.

Furthermore, some questions arise when considering the above stated objectives. First is which information quality is needed when a data-driven system is designed, for example, when considering a satellite. The next question is about the capacity of a communication channel, more explicit would be the speed of the downlink between the satellite and the ground station. In alignment with that, the question is whether the processing pipelines of data-driven systems can be designed to be more energy efficient. Furthermore, from a more theoretical perspective, how much information is minimally needed to archive the same result can be asked. For instance, how much information can be pruned, such as decreasing the resolution or shrinking the bit depth.

The following section of the chapter states the hypotheses for the thesis and discusses the role of compression from its theoretical side. While Section 3.2 introduces the data-driven system, the following Sections 3.3-3.6 discuss the compression of every single aspect of the system. Finally, the consequences for the data-driven system is discussed in Section 3.7.

3.1 Hypotheses

The research questions that are stated in Section 1.1 do cover the compression of each element within a data-driven system. Due to the research that has already been done in the field of compression (e.g.in computer vision), it makes sense to concretize the questions to narrow the topic that is addressed. Therefore the following hypotheses are based on the research questions and underlying this dissertation.

H1: A complexity reduction of the input data and an adaptation of the information representation for the systems input can significantly reduce the computational complexity with limited degradation in functionality.

H2: In Earth observation applications, an image compression with a comparably high information loss has a relatively small negative impact on the performance of the trained models.

H3: Deep learning on embedded hardware accelerators, is able to reduce the amount of data that is transmitted over the communication channel and therefore have an impact on the entire system's needs, like energy consumption or memory footprint.

H4: Compressing Deep Learning algorithms in the sense of reducing the number of weights and the connectivity of neurons does significantly reduce energy consumption and computational complexity with controllable amount of quality degradation.

H5: Data-driven systems can be prevented from having a large generalization error by determining whether the computed outcome is trustworthy.

3.2 Compression Scheme

In this thesis, the topic of compression is formalized as depicted in Figure 3.3. Data takes a path along a processing pipeline from input data over multiple processing steps each connected with a communication channel. Note that these processing steps can be deployed in different locations and thereby the communication channel can be within a computing device, where it is comparably fast, or over a slow network link. On a different axis, each processing step is decomposed into an algorithm that acts on the data and a set of parameters that influence the behavior of the algorithm. Considering this figure, we

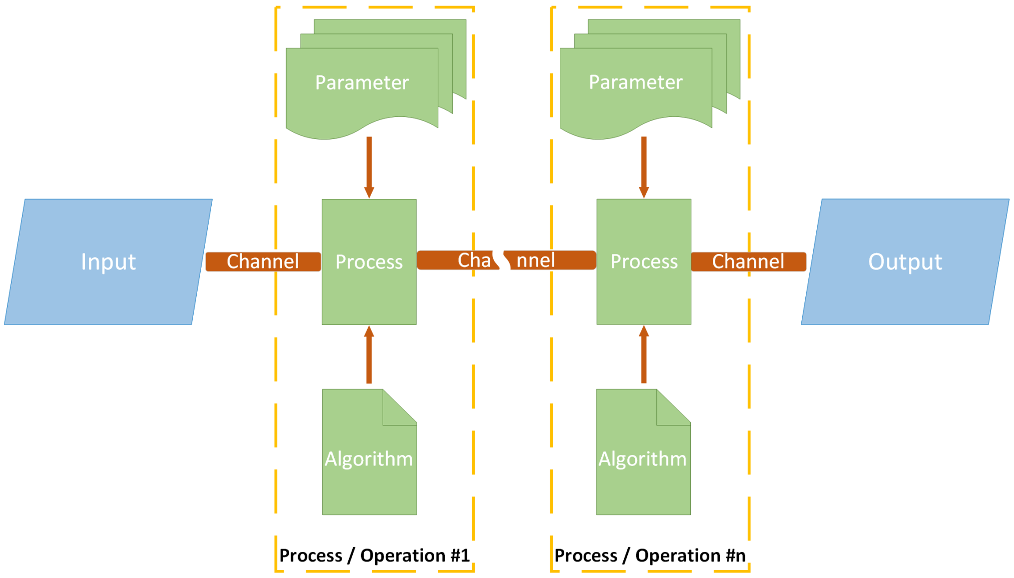


Figure 3.3: Schematic overview of the components of a machine learning processing pipeline consisting of multiple parts. The first is the input data that is transported through the channel to the operations where a specific task is performed, e.g., predicting the content of a data item. It is necessary to investigate the options to compress each individual aspect of the full processing chain. Only then is a statement about the impact and consequences of compression on the entire system feasible.

consider algorithm compression in the sense of compressing either the input, the algorithm in terms of the number of operations, the parameters in terms of their number and size, or the output of a processing step or across a pipeline.

As already mentioned, the input element represents the data that is on the input without restricting it to a specific type. Therefore, it is irrelevant if the data, for example, is matrix or vector shaped. The data on the input of the data-driven system has an repercussion to the entire system – in average the larger a single data item becomes, the more computational effort is needed for the process. Furthermore, more data needs also larger resources within the channel. Furthermore, it is questionable if the entire provided information contains only highly relevant data. Rather the input includes features of high higher and lower importance. Consequently, the compression or even pruning of unimportant aspects leads to higher efficiency of the entire system due to having only relevant data to process. A practical example would be that images can be quantized to a lower dimensional space to highlight specific details. The compression of data leads to a smaller number of bits that requires to be loaded from the disk into the main memory. Additionally, some formats are more efficient in their ability to be processed than others. Furthermore, this influences the entire system, considering a convolution neural network that trains data within batches where the data is too large to be held entirely in main memory and therefore needs to be read periodically. It can be seen that input compression has an effect on the entire system by holding out irrelevant information and therefore decreasing the computational effort.

The operation which processes the data has two important components which are coming hand in hand and need to be considered together at the same time, namely the algorithm and its parameters. There is no restriction on the algorithm type, even when the selected scenarios focus on machine learning tasks. Nevertheless, the goal is to reduce, on the one hand, the number of instruction calls that are needed, as well as the minimization of the algorithm's memory footprint. Considering a convolutional neural network, the weights of the model refer to the parameters. The number of floating-point operations have a direct impact on the performance similar to the number of iterations in a loop. One way to compress the network and decrease its complexity is to reduce the depth while still being able to keep the prediction accuracy in the required performance range. Another option to decrease memory and computational needs by changing the weights, for example, in the form of weight sharing or model pruning. Furthermore, the compression of the operation effects the total energy consumption of the system because there is a saving in time if memory and instructions call.

The output can have various shapes and differs between the use cases. Generally, compressing the output can means to provide only data which is relevant to its use case. For example, a satellite with the task of detecting wildfires does not necessarily need to transmit data where no affected position is included. Another example would be the transmission of satellite scenes from multispectral satellites, which are entirely covered by clouds. The filtering of such results would cause to a reduction only in outputs that are relevant for the following process or entity. Consequently, output compression has an impact on the follow-up resources, whether from computational nature or the hardware side.

Beyond compressing the different building blocks of the described data-driven system, efficiency can be reached on different ways. One is to share the gathered knowledge between the elements, an example would be a model that takes constant measures and only communicates/generates an output if there is a change. More concrete would be to have a satellite with a camera and an AI receiver onboard only transmitting images of detected images if they are not sending an identification of other unusual behavior occurs. This would lead in this specific scenario to only transmit data through the channel that is relevant for the receiver.

3.3 Compression of the Input

Currently, most commonly, we tend to store data in full resolution. For example, an image from Sentinel-2 has a storage need of approximately 800 MB, additionally, the product itself contains a lot of overhead files – from the perspective of a computer vision task. With compression, the memory footprint of a Sentinel-2 product can be reduced to a minimum. Furthermore, this is not only true for images but also for a wide range of different data, such as trajectories or texts. Another question is if the information is lost during compression, this depends highly on the algorithm and parameters. When applying compression to any data D_x, the goal is to change the representation in a way such that memory is saved. Therefore, a compressor C can be defined as a map $C : D_x \rightarrow Dy$, where Dy is the compressed version. There are two large groups of compression algorithms. When $x = y$ after decoding, the details are fully recovered, this is called lossless compression. On the other hand, if $x \approx y$ includes $x \neq y$, some aspects cannot be restored, this is called lossy compression. Typically, some error measure (e.g., bit error rate) needs to be below a certain limit. In the following common input compression methods are presented.

3.3.1 Lossless Compression

The goal of lossless compression is to encode the data in a way that the initial state can be completely restored. A compression algorithm C if it has an inverse algorithm D such that $D(C(x)) = x$ which holds for all valid inputs. This includes for example the case that many algorithms are defined only for certain inputs (e.g., images) where they are lossless, but others are defined for all data (ZIP) where they are lossless. Figure 3.4 depicts a concrete example of lossless compression, the left image is the source showing black (255) pixels on the top half and gray (128) pixels on the bottom half. This matrix-shaped data is then coded employing e.g. PNG with a compression level of nine (middle matrix). When reading the data, an algorithm needs to decode the image from

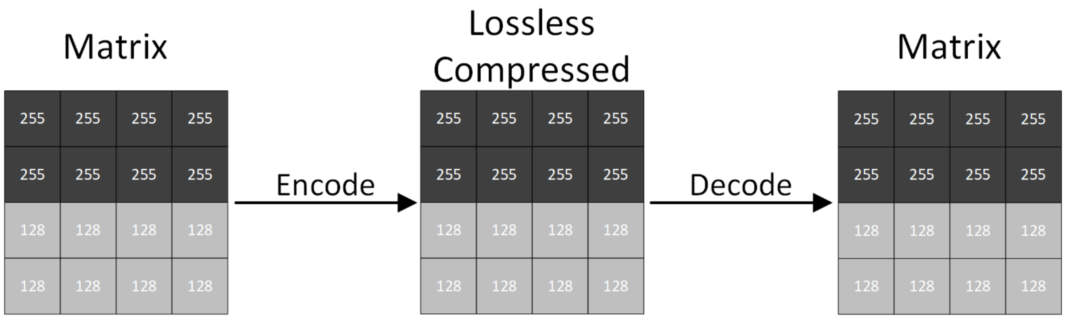


Figure 3.4: When using a lossless compressor, the input image is compressed with, for example, PNG with a compression level of 9 – the maximum for this format – and can be uncompressed without losing information. Consequently, nothing is lost during encoding and decoding. The visualization displays this process using an 8-bit grayscale image with a size of 4 × 4 pixel. It can be seen that the left matrix is the same as the right one.

the file to a matrix-shaped data that is understandable for the program. The right image shows the extracted image, and it can be seen that the result is the same as the original input. This describes the benefit of having a lossless compression: not having a change between the original (left) and the recovered version (right). While this example is about matrix-shaped data, the concept is the same for vector data such as trajectories or any other types of data.

The impact on the entire system, which is described in Figure 3.3, cannot be generally said, even not when only considering images. Therefore, let's consider the aspects into the types of repercussions: memory footprint, and time consumption needed for processing. Depending on the compression algorithm and the corresponding parameters, it is expected that the memory footprint shrinks on average, but one needs to consider that some data is easier to compress than others. For example, the data visualized in Figure 3.4 is more trivial to compress because it only includes two colors within a defined pattern. Other data which might be harder to compress without losing information tend to have a larger memory footprint compared to its initial state. The next aspect is the time consumption, this depends on the number of bits that need to be read from the disk as well as the computational effort which is needed to encode the data. Consequently, the two aspects are partly proportionally dependent on each other. Moreover, it is expected that those aspects also have an impact on the total energy consumption during reading and writing, which is mainly caused by the execution time.

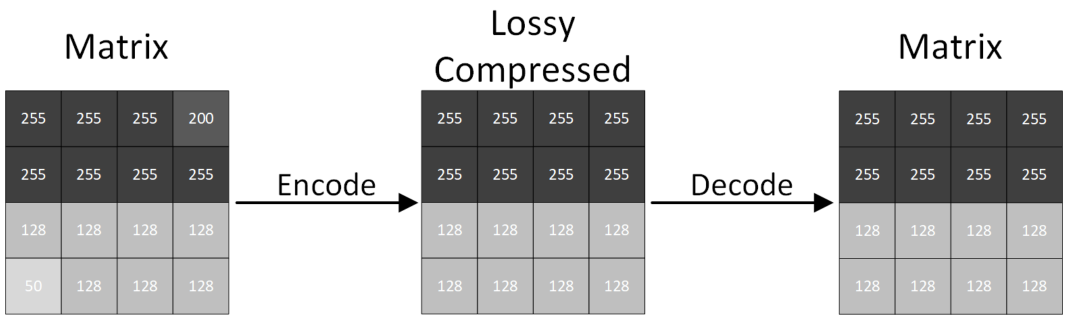


Figure 3.5: With lossy compression, the size of the image shrinks with the costs of losing information without the possibility of recovering. How much is lost depends on the algorithm and on the used parameters. This is visualized in the figure, where it can be seen that the matrix has two pixels that have a deviation, particularly pixels $(4, 1)$ and $(1, 4)$ have slightly different values. In this example, the encoding employing a lossy compressor leads to the consequence that those values are changed and are not visible in the decoded output matrix.

3.3.2 Lossy Compression

Lossy compression is the opposite of the before-discussed method. The goal is to compress the method to reduce the memory footprint, but it is excepted that there is an loss that leads to an error between the reconstructed data and its source. A pair of two algorithms C and D, where $C(x)$ is smaller than x on average $|| \ D(C(x))\breve{} x \ ||$ is small, where $|| \cdot ||$ is some measure of quality. An explicit example is represented in Figure 3.5, which is based on matrix-shaped data. The left image shows a grayscale that has black (255) pixels on the top half and gray (128) pixels on the bottom half. Furthermore, there are two positions with slightly different values, specifically $(4, 1) = 50$ and $(1, 4) = 200$. This image gets encoded with an algorithm based on lossy compression that results in the matrix in the middle. One can see that the pixels with the unique values have changed, which is caused, in this case, by their uniqueness. If the image is then decoded, it shows only the compressed version without the possibility of recovering the unique values, leading to the fact that aspects has been lost. While this example was about images, the same concept works for other data types. For instance, one algorithm used for trajectories is called Douglas Pucker, which simplifies them based on a threshold. Nevertheless, another difference to the above method is that more aggressive compression rates can be reached.

The impact is similar to the lossless compression with some differences. The compression rate and the memory footprint depends on the used algorithm and the corresponding parameter. Additionally due to the partly proportionally dependency the time shrinks with the needed space with some limitations. The compression rates can be more extreme, which results in a loss of information if this makes a difference within the algorithm, depending on the task. Considering a convolutional neural network, the decoding time might have a comparatively small influences than the number of bits that need to be read but might come with the cost of an increased error rate due to missing details. Same as before, a faster reading can be explained by the reduced number of bits which can lead to lower energy consumption during the process.

3.3.3 Quantization

One particular simple and important lossy compression algorithm is quantization. The individual data items are changed in representation to a smaller integer by assigning a range of values to each integer, for example instead of taking a 32-bit floating-point number, an 8-bit integer is used. Often, the smallest integer (0) and the largest for a given bit width (e.g., 2^{k-1}) are extending to capture all values larger than the largest value and smaller than the smallest value for the quantization. Therefore quantization is if the information $D_x \in \mathbb{R}^{n \times n}$, where n is a positive natural number, is quantized using $C_{quantization} : D_x \rightarrow D_y$ where $D_y \in \mathbb{N}^{n \times n}$. One important type of quantization is uniform.All in all, quantization does not necessarily mean $\mathbb{R} \rightarrow \mathbb{N}$, quantization is the reduction of representing bits, independent from the type of numerical space. The Figure 3.6 visualized the quantization process on the basis of a matrix-shaped input. The image on the left is the same 8-bit grayscale image as before and is the input source. The destination space has a range of one single bit to represent a single pixel, therefore, the input gets mapped to a black (1) and white (0) image. This does not only work for images but also for different data types, such as signals where a discrete signal gets transformed into a digital one.

Before, the space dimension to represent a single value did not change, therefore, it was not necessary to differentiate between space needed on disk and within the main memory even when there can be a difference but distinguishable small. First, let's take a look at the memory footprint on disk, which depends on the data format if the data is stored with a lower complexity if so,

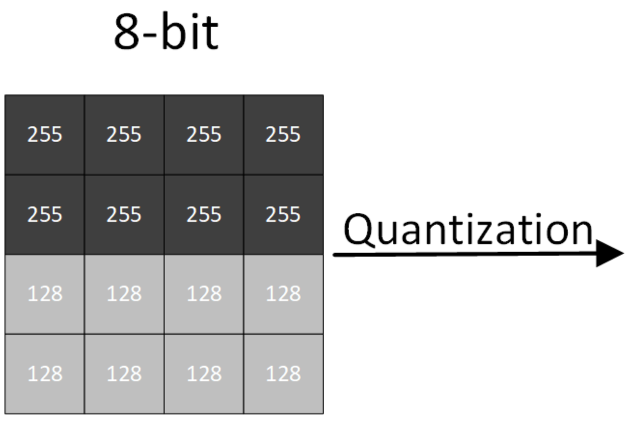


Figure 3.6: Most commonly, a traditional image has a depth of 8 bits. When using quantization to compress the data, in this case, a grayscale image, the depth is reduced. This leads to a loss of information because mapping an 8-bit color space to a lower dimension. The visualized example shrinks the input space to 1-bit, which can only represent either black or white. Consequently, the value of 128 from the input cannot be probably mapped and causes an irretrievable loss.

the space shrinks in relation to the target complexity. The size in the main memory is reduced in theory, which depends on the datatype. Nevertheless, the behavior changes with respect to the variable type used, therefore, there is a difference between a 32-bit floating point number and a single bit. When taking a convolutional neural network – which is a sensitive structure – to predict the content of an image with a 24-bit RGB image needs a specific time and computational effort. The reduction space dimension using quantization can highlight, in certain cases, some specific features, especially edges, and corners. The more prominent features can help, in theory, to keep increasing the prediction accuracy by a constant or lower computational time. On the other hand, when considering machine learning algorithms, very low spaces can tend to be more overfitting.

3.3.4 The Impact of Input Data Compression

Data comes in different shapes, structures, and facets, like trajectories or Twitter tweets. Similar is the scenario for the compression methods that are designed to compress particular data types. Due to this reason, answering this question in a generalized form is non-trivial. While it makes sense to focus on a particular type, the overall message is commonly shared between the domains.

In spatial computing, the images are often provided in their uncompressed full complexity. Such as from the Sentinel mission, which delivers multispectral data employing JPEG2000 and SAR by using the TIFF format. Similar is the situation for Earth observation datasets in which are large portion is provided in a TIFF container without applying an compression algorithm.

The first task of a data-driven system is to read the data from the disk. Especially for datasets that do not fit into the main memory, this task is done periodically during runtime and even multiple times per image. Depending on the programming language, a data item is loaded and embedded into a pre-defined data structure that does not change during the computation, assuming no quantization or pruning is applied. As a consequence, the shape and complexity of uncompressed data are constant during the full algorithmic procedure. This situation is similar to images that have a compressed representation on disk. While the consequences are that compression does not have an effect on the large parts of the system's pipeline, the reading process does highly depend on the memory footprint on the disk. Inefficient compression (like uncompressed images) leads to a high wallclock time as the data cannot be read fast enough from disk and a compression that is computationally complex leads to a larger wallclock time as the reconstruction of the initial state of the data is consuming CPU time. Even when a compressed image needs to be encoded, the choice of compression does influence the runtime of the program. This repercussion directly depends on the size of the selected dataset. On the contrary, the pruning of information can be a problem for some algorithms, such as training a deep learning model.

When integrating input data compression into a system, the compression typically implies among others five effects to the performance of the system. (1) the wallclock time can be reduced when the system has been bound to the speed of the I/O subsystem (IObound) by reducing the footprint on disk. (2) the wallclock time can be increased when the compression takes up CPU time needed for other processing tasks (CPUbound). (3) the energy consumption can rise or shrink depending on the actual energy consumption of the subsystems and their activity. (4) when adding compression to a fully connected distributed system, the number of nodes in a distributed system can be reduced whereby the communication complexity shrinks quadratically. (5) in case of a ring-based distributed system (e.g., Apache Cassandra), the number of nodes is reduced whereby the performance of the replication and synchronization along the ring increases. Therefore, a rigorous benchmark with performance metrics at least including application performance (e.g., accuracy, F1 score, precision, recall for deep learning), wallclock behavior (e.g., frames per second, etc.), and energy behavior (e.g., Joule/frame) needs to be conducted.

Compression methods have in common that the memory footprint is lower compared to the uncompressed source data. The minimization of storage needs, on disk and within the main memory, leads to the consequence that fewer bits required to be processed, regardless of whether the task is to read the data or to perform any other operation. The faster processing conduces to higher throughput and probably lower total energy consumption. Furthermore, in theory, this means that not all data is provided by the full complexity essential for the processing to fulfill the given task. If this hypothesis holds true in practical scenarios needs to be investigated.

3.4 Compression of the Algorithm

The last section was about the compression of the input data, which showed theoretically that the algorithms could work more efficiently with certain preprocessing, like removing information whiles less relevant and having a smaller effect on the result. While those decisions had an impact on the entire system, the algorithm has not been actively compressed, which is the scope of this section. The motivation behind that is to get algorithms that ideally only perform steps and work on data aspects that are absolutely essential to the end result without affecting its accuracy. The decrease in instruction as such is performed can lead to two aspects, a lower memory footprint and reaching a higher efficiency. Thus, to lower needs in time and energy. Additionally, the compression of an algorithm can enable the possibility to on-board processing.

One technique to compress an algorithm is called loop unrolling. This method optimizes the loops by replacing the original loop with multiple instructions from the body. Let's consider a simple loop that does $n = 1\,024$ iterations and multiplies each entry of an n-long array with a random number. Instead of iterating over every single step, one can do the given task for i, $i+1$, $i+2$, and $i+3$ and then increment the i by four. This leads to a reduction of cycles within the loop, additionally, the body of the latter loop can be parallelized more easily. Nevertheless, a smaller algorithm in terms of the number of instructions needed to get the same outcome is reached in multiple ways. When considering machine learning algorithms, one way to compress them is to reduce the number of parameters. In neural networks, one technique is to

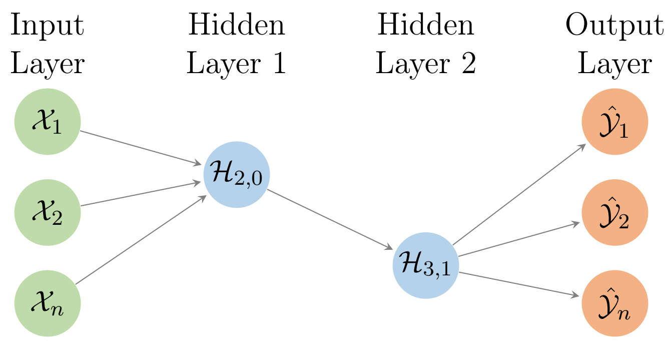


Figure 3.7: The visualization shows a pruned feed-forward neural network. The unpruned version originally had four nodes in each hidden layer. During pruning removed, some nodes in the hidden layers were because the contribution to the result was comparatively small. Therefore, not all information provided by the input is needed to calculate a prediction for the given task.

cut out neutrons which do have a comparatively small contribution to the final prediction result. Dropout is a technique that influences locally by considering only a subset of the network. Therefore, the active impact is restricted to only a small number of layers. The passive effect has an repercussion on all subsequent layers caused by the reduced number of information which is forward. On the one hand, the neural network does tend less to overfit, on the other hand, the compression leads to the consequence of having fewer instruction calls and a smaller memory footprint.

Compared to this locally applied technique, pruning methods act globally. The pruning algorithm removes nodes within a neural network which does not largely contribute to the final decision. Figure 3.7 visualizes this process on an example of a feed-forward neural network but is valid for a large range of other deep learning architectures. Furthermore, considering the previous section, which made the assumption that not all information within an image is relevant, can be seen here again. Theoretically said, if a convolutional neural network can be pruned without a loss in accuracy, there is information that where irrelevant to the task. Moreover, the number of instructions or in neural networks, the number of floating-point operations is reduced, the memory footprint is minimized.

The discussed regularization and pruning methods do shrink the number of floating-point operations needed to calculate the values for the output layer. Another aspect is to compress the feature maps itself to shrink the instruction calls. Figure 3.8 shows an example where an input map with a size of $8 \times 8 \times 1$

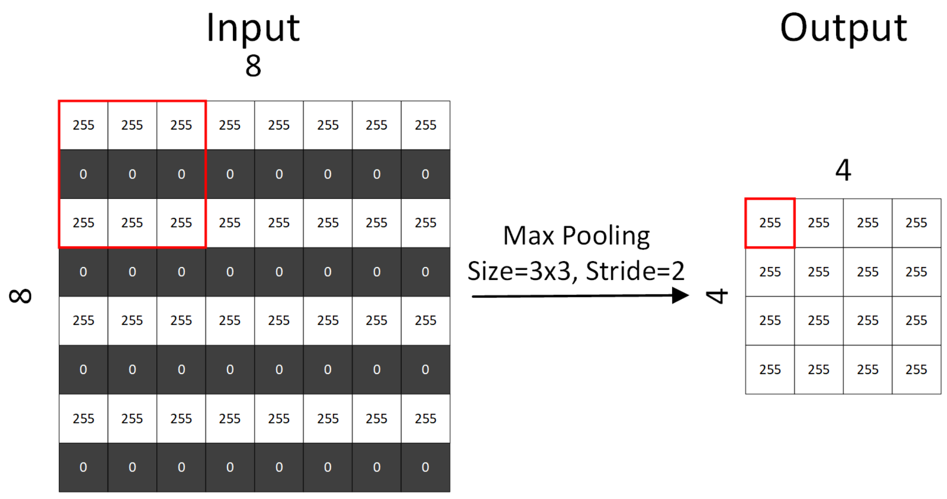


Figure 3.8: The goal of pooling is to shrink the size of the input feature map by a given value. The visualization shows at the input a feature map with a size of $8 \times 8 \times 1$ pixels. Considering a convolution layer that has a single kernel with a size of 3×3, a stride of 1, and zero padding, there are 729 floating point operations (FLOPS) needed to create the output map. The pooling operation described in the figure shrinks the size of the feature map to 4×4 pixels. The compression leads by the same operation as before to a number of flops of 225 to create the output map.

is compressed using pooling to $4 \times 4 \times 1$. Considering a convolutional layer that utilizes a single 3×3 kernel, 729 floating point operations (FLOPS) are esential to compute the output map. Compared to that, after pooling the map needs for the same procedure 225 FLOPS, which is less than the third then before. Same as in regularization, it is the active process only locally within a single layer but influences the entire system in the form of forwarding less information. Whether it is a neural network or another algorithm, these methods have in common that the number of instruction calls is reduced. The decrease leads to higher efficiency in terms of time required and the total energy to finish the task.

There are two methods that also need to mention, early stopping and ensembles of algorithms. While the first one stops the training process under a certain condition, the latter one is where multiple models with a lower precision are combined into a single one. For example, the application sample in Chapter 4.3.1.3 ensembled abstaining models which are specialized to a specific task, such as detecting if a tweet is related to a building. While the first method compressed the footprint of a model by preventing it from performing unneeded training steps, the ensemble creates one single algorithm from multiple ones and supports a generalization. Moreover, there is a wide range of methods to compress the algorithms to create more efficient models.

3.4.1 The Impact of Algorithm Compression

All in all, the compression of the algorithm led to two consequences, (1) the smaller number of instruction that is needed to finish the task and (2) the lower memory footprint on disk and in memory. Those consequences have in common that the algorithm has a higher efficiency and saves time. Furthermore, the optimization and savings in time can result in lower energy consumption of the entire system. From the theoretical perspective, there are two things that should be investigated using a practical example. First, it is similar to the hypothesis from Section 3.1, that is, if all data from the full resolution is relevant. The second perspective is the memory footprint that can lead to an optimized algorithm in relation to its memory consumption. When considering a pruning technique, the goal is to remove neurons that do not have a large contribution to the result, therefore, information provided by the data can be irrelevant. Also, when considering pooling, which highlights certain features by taking, for example, the maximum value within a window, it results to the consequence that the data is compressed to destroy spatial dependencies and remove unimportant intonation. Tho points lead to the hypothesis that full-resolution data does provide a smaller amount of information than assumed. The question is if current baselines can be reached by using efficient models in terms of their number of instructions.

Both discussed compression perspectives lead to smaller algorithms that can enable the possibility of more efficient computing and prevent unneeded scaling of the hardware. Additionally, some models can be deployed in its compressed format to on-board processors, for instance a mobile device or an FPGA.

Nevertheless, such as the application samples in Section 4.2.2 provided an end-to-end system to classify trajectories from handwritten characters and mobility data. Unfortunately, the most important similarity measures for trajectories, like the Fréchet distance, have quadratic complexity. Using the characteristic that the distance measure approximates the Kolmogorov complexity with normal compressors, the fixed-sized Bloom filters can serve as lossy compressors where the informal content is empirical evaluated. In addition, simple geometry measures, such as the direction of a single segment can be embedded. Furthermore, the joint compression is calculated by logical OR operation of two filters. This reduces the complexity of the algorithm itself by changing the representation of a trajectory. Furthermore, the metric is evaluated using a k-nearest neighbor algorithm that reaches the correct baseline of sample data sets.

While this focuses on trajectory data, another domain is Earth observation which uses deep learning models to estimate the content of remote sensing data. The compression of, for example, convolutional neural networks, do commonly have an impact on the result where the quantity depends on the method and on the compression rate. While pooling is commonly applied in a large number of architectures, quantization reduces the complexity that is available to represent a single weight. On the other hand, pruning removes nodes that do not form importance for the final prediction results. While the domain of spatial computing employs this rarely, especially quantization is often required to train or inference models on edge computing hardware such as an FPGA. The model's footprint is significantly reduced. This comes with the codes of losing accuracy, depending on the method and the compression rate. Moreover, the pruning has the advantage that the number of floating point operations is reduced that need to be done during training which leads to a performance increase. In conclusion, model compression is a trade-off but enables running models on smaller hardware more efficiently. In addition, the use of embedded devices increases in importance in, for example, the space industry. The satellites are constantly producing data that is required to be transmitted over a downstream to Earth. One constraint is that the channel has a limited speed and visibility window where it is needed to up and download all information that is required. This is especially a problem for time-critical tasks. The utilization of on-board deep learning is able to detect whether a created information is useful, for example, a sensed scene of a satellite that detects wildfire. In this way, the channel is optimized by only transmitting data that is relevant.

3.5 Compression of the Output

The last two sections discussed the importance of input data to create models that are more efficient in the form of a reduced memory footprint or a reduction of the computation time. Another aspect that needs to be considered

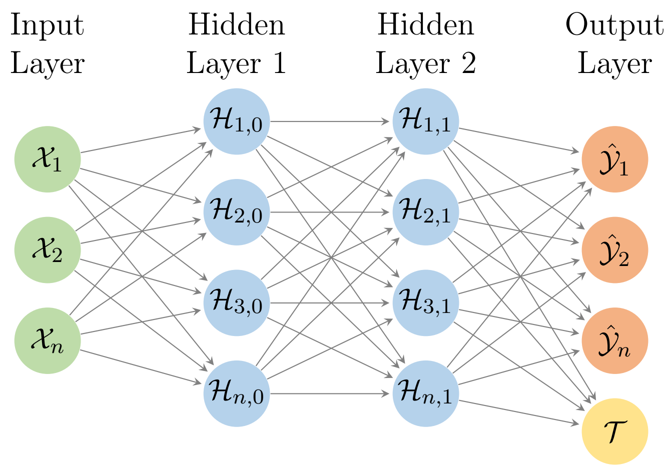


Figure 3.9: When performing a classification using a machine learning algorithm, one question is whether a result lacks confidence. The figure shows a feed-forward neural network that uses abstaining in the form of an additional node $\mathcal{T}$ that indicates if the result can be trusted or not.

is whether the output of an algorithm is important or irrelevant. Therefore, in comparison, this method does not have a direct effect on the efficiency of the result-generating part of the processing chain, rather, the focus is to only further process data which are relevant to the given task. This can be done in multiple ways, one is called abstaining, which focuses on the decision whether a result from an algorithm is trustful. How the decision is dreaded depends on the use case, for example, one option is to take no action, which results in a reducing the amount of output data. Figure 3.9 shows abstaining in a feed-forward neural network. One can see that the structure is quite similar, with an additional output node $\mathcal{T}$ indicating the confidence of the output.

Another strategy is to create an algorithm that determines if the output has a specific pattern. Based on the outcome of the algorithm can be decided whether the output is worth it to process further or needs to be traded differently. Figure 3.10 visualized the result of such an algorithm. A neural network can be trained to decide if there is a specific pattern within the images, such as clouds. Due to the reason that clouds do hide certain target objects, those could be ignored for further processing. More concretely, the neural network can be deployed onboard a satellite and decides if a scene is transferred down to Earth. Otherwise, irrelevant images can be ignored and therefore withdrawn. This leads to compression in the form of only handling important data. On the other hand, filtering constantly removes data.

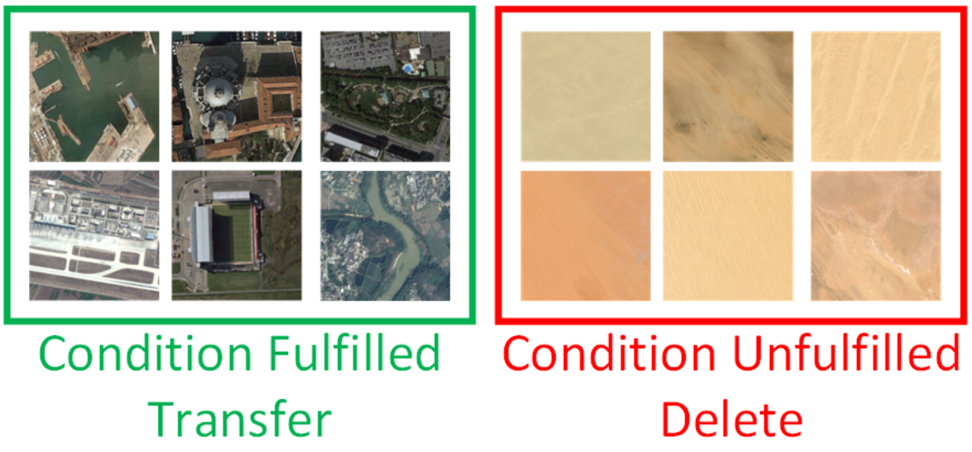


Figure 3.10: Considering a convolution neural network that detects if a specific condition is fulfilled and filters the output based on its result, for example, based on cloud coverage or, as stated in the figure, desert and urban area. Filtering can be, for instance, applied onboard a satellite to decide whether a scene is relevant and needs to be transferred to the base station.

In the domain of remote sensing using satellites, the consequence would be to reduce the amount of data that is transmitted over the downlink, therefore, output data compression can be useful to only give relevant data, especially with respect to direct receiving. But filtering does not only find its application in space missions, rather, it is worth it to be considered in all areas procuring a large amount of output. Another example a generative AI model, without considering a specific scenario, the software always provides an output to an input. This can be misleading, especially when the input does not contain any useful information or even a wrong assumption. Consequently, suppressing the output can optimize the accuracy and performance in some cases.

From the theoretical perspective, one question is the impact on the processing pipeline and the hardware resources of such methods. Taking the memory footprint into account, compressing or even pruning the output can increase the performance of follow-up processes, such as the communication channel between the satellite and the ground station. Furthermore, there is also a save in time when irrelevant data is not further processed.

3.5.1 The Impact of Output Compression

The output of machine learning models can have various forms and shapes, such as images and text, in the field of generative artificial intelligence. Another possible outcome is the class for either an entire image or an single pixel. In all cases, the results are sometimes either suspicious or wrong. Due to the fact that most instances in the dataset did not have a clear label, the machine learning system had to learn the concept from a very small and heavily imbalanced dataset. However, the (partial) ability of selected machine learning models to assess the certainty of their own output can be used to improve the overall situation. A solution to this problem is to determine whether the model's outcome is trustworthy. If this is not the case, the result can be rejected. This strategy led to a higher robustness model of the entire system that has been built and opened the possibility of assigning function types to unknown buildings. Therefore, it is necessary to verify the outcome and exclude it from further processing if needed.

3.6 Computing with Compressed Representations

The first section discussed the compression of the input, still needing a comparable large amount of bits to cover the data. Unfortunately, some hardware has constraints that make the computation of data in its source representation hard to impossible. For example, while the current state-of-the-art quantum computer covers 433 qubits [84], an average CPU server can simulate only 16 qubits. An extreme case of input compression is representation learning, where data is represented employing only a limited number of bits. Representation learning cannot only be useful for quantum computing but also efficiently process data. Considering the example given in Figure 3.11, x represents the input data, more concretely an image with a size of 64×64 pixels, with the task to predict the input with a quantum machine learning (QML) algorithm. Even the largest quantum computer currently available cannot handle this amount of data, especially not when simulating on an average CPU/GPU server. Within the example, an autoencoder can be taken to train a lower representation of the input image: the encoder vector. From the theoretical perspective, the most important data is mapped to a small number of bits, which can be the input for another algorithm. As it is stated in [85], this mapping to a very small

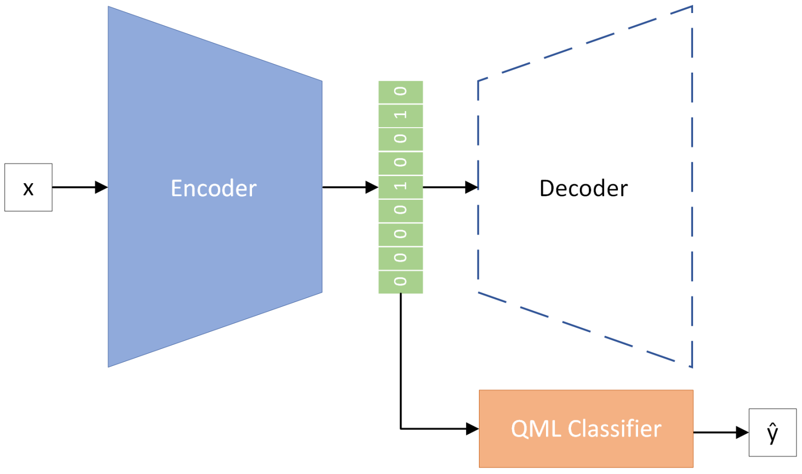


Figure 3.11: Nowadays, quantum machine learning is a highly researched field that needs either special hardware or a large number of computational resources to simulate q-bits. Unfortunately, there is currently a small number of hardware that is able to compute – whether simulated or not – with more than 60 bits. This problem is that lots of data need more than some bits for a single data item. One solution is representation learning. One way to learn a smaller version of the data is to take the encoder vector of an autoencoder as input for a quantum-based machine learning algorithm.

space comes with the costs of losing information that might be relevant to keep the prediction accuracy as high as the reference baseline. Furthermore, besides that, another factor is that there is another algorithm needed to compute the lower denominational representation. On the other hand, the memory footprint can be significantly reduced, enabling the possibility of processing data in real-time. If this sums up, an advantage or disadvantage needs to be considered for a concrete case. From the theoretical perspective in relation to quantum computing and image processing, it is questionable if it makes sense to apply two algorithms with quadratic complexity to be able to use today's quantum hardware, even when a feasibility study is essential for research.

In alternative is to lower the dimentional representation by embedding the data into Bloom filters in linear time. The example in Figure 3.12 shows that a defined type of data can be embedded into a binary Bloom filter with a fixed size, in this case, 32 bit. The change of the representation to a fixed, constant and well-defined way enables to use of the generalized algorithm in different types of data. Furthermore, it is possible to extract the data by asking whether a data item is in the filter. While there are no false negatives when extracting

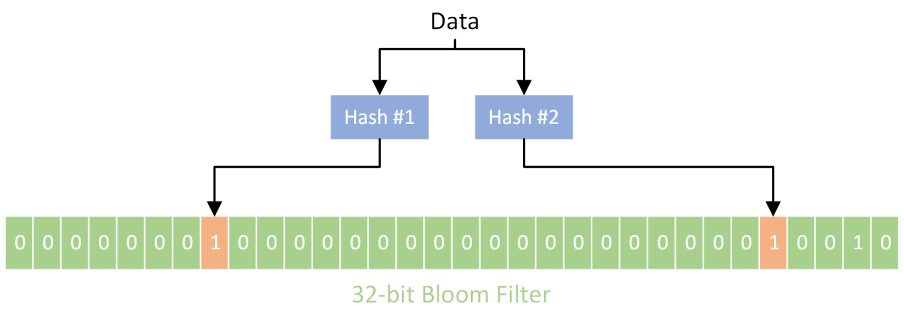


Figure 3.12: A single data item is embedded with of two hash functions setting certain bits within the Bloom filter. The data structure is able to hold the data using a lower dimension, additionally, the data can be represented with constant size.

data from the Bloom filter, there is a risk of having false positives.

Another form of computing with compressed representations is feature-free data mining. One algorithm which is doing that is the normalized compression distance (detailed explanation see Section 2.1.2.3), which utilizes normal compressors to represent that data and calculate the similarity based on the length of the compressed version. An advantage is that it is theoretically possible to compare two different types of data without changing the algorithm. Similar to the situation with the autoencoder, compression using normal compressors is the bottleneck in the process. On the contrary, the described Bloom filter provides nothing else than a compressed form of data. This would reduce the complexity of a linear problem, leading to higher efficiency. When replacing the compressor with a Bloom filter, one problem which arises is that the length of the Bloom filter does not accommodated any information reading to the comprehensibility of data, and a comparison is useless due to its constant size. As is has been done in the application sample in Chapter 4.2.2 an solution would be to use binary operations, such as the union between two Bloom filters, to measure its similarity. Another option is to compare the entropies of the two filters. A practical test will be successful if this is possible without a significant loss in accuracy when comparing types of data.

3.6.1 The Impact of Computing with Compressed Representations

A small representation can lead to a significantly smaller memory footprint at the cost of loosing accuracy. If an algorithm still is able to compute the expected result without a large increase in the error rate, it can be deduced that the provided information either can be compressed to a minimum without a loss or there is something that is not relevant. Nevertheless, the computation with compressed representations is able to generalize algorithms, such as when using abstaining.

3.7 The Consequences for the Data-driven System

This section answers the three research questions stated in Chapter 1.1 concerning the hypotheses and the investigated current compressive methods and their impact on the entire data-driven system. Furthermore, the current state-of-the-art literature servers as an additional foundation for the provided answers to the questions.

RQ-1: *What is the minimum of the complexity and informational content of heterogeneous data required for a data-driven system?*

When it comes to the compression of data, there are several techniques and methods that have been furnished for a large range of different data types. For example, JPEG compression for images or Douglas Peuker for trajectories. Additionally, it provides Shannon's and Kolmogorov's information theory the ability (whether it is computable or not) to make an assumption about the complexity of information contained in the data. Considering the algorithm that is used in a data-driven system as a machine learning model, there are many ways to estimate a corresponding label to the data. The authors of [86] describes that the two theories focused on the transmission and complexity of data rather than which amount of information is relevant and meaningful. Additionally, the authors state that, in theory, lossy source encoding techniques based on those theories can be adapted to make an assumption about the relevant content contained in data. One way to compress data is quantization, while this reduces the complexity that is available to represent a particular value or area within a given item, the information that is of less importance and does not contribute to algorithms prediction might be still present is some cases [86]. Nevertheless, another method to reduce the amount of information, particularly in images, is by using compression-based file formats. While lossless compression does not have an repercussion on the prediction accuracy of the algorithm's outcome, the compression rates are lower than for lossy methods. The latter comes with costs and might be unable to provide the best optimum calculation result. Another method to compress data, especially images, is to learn the representation with deep learning. While [85] uses, for example, an autoencoder trained to map a given image to a 16-bit vector, the authors of [87], [88] employ a deep learning model that learns to remove details that is less important for the visual representation.

The contributions of this thesis to this research question are outlined in Chapter 4.1 and can is in relation to two hypotheses: first is if a data-driven system's performance can be optimized by reducing the compressing of the input (H1), and second if this can be done without losing precision on the outcome (H2). While the algorithm can be clearly optimized by changing the representation using lossless compression, this does not influence the outcome. Furthermore, more aggressive methods provide a higher efficiency but with the costs of losing precision on the algorithm's result. The application experiment (Chapter 4.1.2) showed that a moderate compression of data does have either a positive or no repercussions on the algorithm's precision on its outcome. Smaller memory footprints lead to the consequence that the program needs less time to load the data from the disk to the main memory, which has an effect on the total system's runtime. Therefore it is important to prune the non-meaningful part to optimize the program in terms of its runtime. Besides, this is a trade-off between the amount of information contained in the data that are pruned and the ability to run the program without or less impact on the result.

Therefore, we conclude that the answer to this research question is that the use of compression does not always significantly impact the precision of the algorithm's outcome. Rather the results show that not all information that is furnished by data, especially remote sensing images, is relevant. Therefore, the utilization of compression on the input data is relevant to increase the efficiency of a data-driven system.

RQ-2: *What is the potential and limitation of (aggressive) algorithm compression and randomized data structures in an edge computing scenario?*

The theoretical situation is similar to the complexity of heterogeneous data and consequences for a data-driven system of reducing them. The quantization of parameters does lead to more efficient architecture in relation to their memory footprint and the computational effort to compute an optimal parameter set but does not prune parameters with low importance [89]. Furthermore, deep learning architecture areas that do not significantly contribute to the prediction result can be removed to create an algorithm with a lower footprint and fewer instructions (less floating point operations) [37]. Therefore, similar then the input algorithms/operations such as neural networks contain details that is not relevant and some that are essential to the output. Furthermore, there is a range of research that [90], [91], [69] proves either theoretically or practically that the parts of a deep learning model can be removed with less to no negative impact on the result. The research field *deep learning model compression* addresses this particular scenario – removing elements of an algorithm that are of less importance. Several methods have been introduced that focus on, for example, channel pruning, layer pruning, and filter pruning, while all of them have in common that the number of floating point operations has been reduced. While there is often a trade-off between accuracy and compression [92], the authors of [93] showed that removing unnecessary information is able to support the algorithm towards a higher degree of accuracy on the predicted output. In addition, the [94] states that the compression of models is especially interesting when deploying models to edge devices that have limited computing power and memory footprint. In contrast to this approach to compress the algorithm is by taking randomized data structures, which are able to suppress the quadratic nature of some algorithms. The probabilistic data structure Bloom filters are widely used among several domains [95] [96], [97]. In previous work, the group proposed in [98] an approach to embed trajectory features into those filters and measured their similarity. Additionally, [6] showed that the filters are able to hold large datasets (like Twitter data) in the main memory.

The contributions of this thesis to this research question are outlined in Chapter 4.2 and can is in relation to two hypotheses: if the use of the channel between a remote device and its base station can be optimized (H3) and if the compression impacts the total runtime of the entire data-driven system (H4). A data-driven system in the context of spatial computing has several constraints that are also caused by the data content's complexity. The runtime can be reduced by applying compression but comes with the cost of loss of precision of the outcome. Nevertheless, the deployment of custom hardware is essential to reduce hardware needs and increase the efficiency of the systems.

RQ-3: *How can abstaining and model ensembles facilitate learning of a data-*

driven system?

The compression of the output can have several motivations, like rejecting results that are not trustworthy. As stated above in Section 3.5 and in [99], [100], abstaining is able to increase the robustness of a deep learning model against noisy data. An application experiment presented in Chapter 4.3.1 uses this technique to train a classifier for building function classification based on Twitter data where only a small portion of the dataset is relevant. Abstaining is able to determine if the output is meaningful. In contrast to this model, ensembles combine several small algorithms to reach a generalized model that maximizes their prediction performance in terms of the quality of the outcome without increasing the model's variance. While there are several commonly employed methods to reach model ensembles, such as boosting, bagging, and stacking, the authors of [101] describe the combination of models does come with the costs of the need to search a large parameter space to find the optimal set. The application experiment in Chapter 4.1.1 various trained models and created an ensemble using the top-k ones. The proposed approach reached relatively high accuracies, but comes with the costs of increased computational effort compared to other methods.

The contributions of this thesis to this research question are outlined in Chapter 4.3 and can is in relation to one hypothesis, which is about increasing the robustness of algorithms against data that is suboptimal to train (H5). The experiment showed that it is possible to use abstaining and ensembles in a spatial context to train to reach a robust model trained on a large-scale dataset that consists of a large amount of label noise.

3.7.1 The Role of Compression in Spatial Computing

The investigation provided an analysis of how a data-driven can be compressed and the resulting impact on the computed. A careful combination can maximize the performance in relation to the precision of the outcome, runtime, and energy consumption. The use of moderate input compression (for example, as it is the case for JPEG quality factor of 75 or PNG with a compression level of 9 in the case of images, see Chapter 4.1.2) causes a marginal form in accuracy when training a convolutional neural network but provides a trade-off between the effort that is needed for reading the prediction performance. Additionally, the energy consumption, especially on edge computing hardware accelerators, is shortened by a large portion, dependent on the size of the dataset and the main memory. Additionally, if a deep learning model is trained without having a large generalization error supports the algorithm's compression with quantization and pruning. While the former is required to minimize the memory footprint and for the deployment to an FPGA, the latter optimizes the efficiency by reducing the number of floating point operations. The resulting model comes with the cost of losing accuracy but has the advantage of being optimized in relation to computational performance and energy consumption. Additionally, for some given tasks, this enables even the possibility of performing the computation without scaling the hardware.

4 Application Experiments and Results

The previous chapter discussed the foundational aspects of the thesis and defined some requirements. This part provides some samples to the theoretic chapter and should highlight their importance as well as their feasibility. As described, a machine learning processing pipeline can be split into several components, which are input, algorithm, and output. The following provides some application examples for each of them. The first section investigates the compression of input data. In line with the theoretic foundation, image data is compressed employing different forms, such as the manipulation of color spaces as well as the use of compression algorithms. The following section is about the complexity reduction of algorithms, which uses two different methods. On the one hand, the algorithm is compressed by utilizing probabilistic data structures. On the other hand, an example is given on how the communication of satellites with their ground station can be reduced. The last section provides an application example of how to compress the output of the machine learning operation by determining whether the outcome is trustworthy.

4.1 Examples of Input Compression

The first element from each machine learning pipeline is the complexity and the volume of the input data. Anyways, the data has a significant impact on the processing speed of the entire system. When considering the complexity that is available to represent a single color of a pixel, the memory that is needed increases with this value. A higher memory demand can also be problematic when an item is required to be copied or moved (disk $\longleftrightarrow$ memory) during the computation of the algorithm. Additionally, another factor is the representation of the data on disk, for example, some file formats are more efficient to load than others. One application example shows that if considering JPEG over TIFF to store a satellite image, there is a notable performance increase during the training phase of a deep learning model. This comes with the cost of losing accuracy, which is vanishingly small. While this section focuses on images only, the results are also applicable to different data types, but the quantity of the impact changes – also, the processing of how the input of compressed differs.

The following provides two examples of how the input can be compressed to support the performance. The first uses a genetic algorithm to change the color space of the training images such that the model that is trained reaches higher accuracies. The second application example investigates the impact of images quantization and compression on the time that is needed for training a deep learning model as well as the accuracy.

4.1.1 Genetic Algorithm for Transfer Learning

This section has been published in our work [P3] and is structured as follows: An introductional motivation is provided first, followed by methodology in Section 4.1.1.2. Furthermore, the Section 4.1.1.3 describes the experiments and results. Finally, a summary is given is Section 4.1.1.4.

4.1.1.1 Motivation

During the last decade, convolutional neural networks have been invented and quickly become a foundational element of computer vision. These deep learning methods are consistently outperforming state-of-the-art methods reaching surprising performance on many computer vision tasks.

One famous dataset to train neural networks, especially CNNs, in the domain of classification is ImageNet furnished by ILSVRC 2012 [50]. This set is constructed by manually labeling 1 200 000 items into 1 000 categories. While there are also newer versions of the dataset, the 2012 version is still the most common one to benchmark models in the computer vision domain and is commonly used for transfer learning.

A comparable dataset in the domain of earth observation might be BigEarthNet given by [4], which is a collection of land cover and land use images from the mission Sentinel-2. The dataset provides nearly 600 000 multispectral images, which is approximately half of the size of ImageNet, additionally, a single one can include multiple land use categories. Moreover, the spatial auto-correlation characteristics add another complexity to the datasets. Additionally, the images from the Earth's surface exhibit a higher density of structures – a mountain compared to a riverbed – than the different classes of ImageNet. Due to these constraints, it can be said that the given task for BigEarthNet is considerably more complex than ImageNet. Other satellite datasets are available but are not as suitable to train domain specific neural networks as BigEarthNet.

There are two main components that are given in computer vision leading to a revolution in image processing that is not given in the field of Earth observation. The first problem is that the number of available training datasets is relatively small in relation to the complexity of tasks such as classification and segmentation. Second is that it is non-trivial to apply the investigated algorithmic improvements because the deep learning community does **not focus** on the computational efficiency of spatial images. Moreover, currently, datasets are generated to fit the requirements of computer vision algorithms instead of employing remote sensing requirements. Furthermore, [102] provides an overview of the deep learning methods that have been applied to the remote sensing community, moreover, authors of [103] discuss information fusion techniques that have been introduced into the field.

The problems of having a low number of datasets, overlapping labels, and noise in the classes increase the complexity of training an efficient high, quantified machine learning algorithm in remote sensing. There are mainly two solutions to those difficulties: (1) reduce the complexity of the algorithm, (2) transfer learning across domains.

The first option is to reduce the model complexity by pruning parameters that have a comparatively minor impact. Another example is to lower the depth of the neural network. Anyways, one needs to know that many transitional clustering methods, e.g. a support vector machine or decision trees, are capable outperform deep learning methods when the region is either very large or even global. One example where this happened is when the water surface has been globally computed. This has been done by only transitional image analyzation algorithms due to the ability to embed knowledge in an efficient way. In other words, points on the convex hull in the color space were used to set the scope of specific features within the data.

The second option to solve the mentioned problems is to utilize transfer learning across domains, for example, to pre-train a model on ImageNet (or any other set) and employ the knowledge to predict labels from BigEarthNet. The scope of this part of this work is to use the transferred knowledge across domains combined with the idea of using points on convex hulls. In detail, this means while changing the input from the deep neural network by changing the points that lead to a highlighting of different features, the model's parameters stay constant. Consequently, the complexity is reduced due to training only a small portion of the architecture instead of taking the pre-trained values into consideration.

4.1.1.2 Methodology

Before going into the definition of the proposed approach, one needs to know about color space manipulation. Especially for remote sensing data, the number of channels per image varies, for instance, it does provide Sentinel-2 13 bands. While there are applications that utilize all the provided information, some use cases create three band combinations to visually highlight specific areas in the image. One example is wildfire, while Sentinel-2's $\{R, G, B\}$ bands show the smoke, the other band combinations give a details view of the actively burning areas. Usually, CNNs are pre-trained on well established computer vision benchmarking datasets and fine-tuned on the data of interest. Unfortunately, the complexity as well as the color space differ from remote sensing, this makes it necessary to transform the space of the images to highlight features and shapes in order to maximize the prediction accuracy. A practical example is an image of a gray airplane on a landing strip. The colors are rather monotone than having a large variability, of course, depending on the season where the snow could change the problem. Therefore the spectrum of the image in this particular case is relatively small. Nevertheless, CNNs are sensitive to edges, corners, and textures, but are also to colors, consequently, a change in the space can increase the performance.

The transformation proposed in this part of the work is based on the archetypical analysis, which is able to identify extrema in the convex hull of the color space. Moreover, another characteristic is that all pixels can be represented by the archetypical values. This representation using extreme points is able to highlight certain color areas and transforms the image to highlight certain areas. This leads to an efficient model.

The proposed method is based on color archetypes and the mapping function archetypal projection, which are defined before presenting the experimental design.

Definition 4.1. *(Color Archetype, see [P3]). Let D^n be the color space with* *n dimensions and A be an archetype that is formally defined as*

$$A = \{a : a \in \mathbb{R}^1, a \preceq a\}^{D^n}, A \neq \emptyset,$$

the number of dimensions can, for example, be the available multispectral channels provided by a satellite.

Definition 4.2. *(Archetype Projection, see [P3]). Let f be an archetypal projection function that maps an input image with n-dimensional dolor space to a set of n-dimensional archetypes. Every single pixel is mapped to a set of distances to the given archetypes. Per default, the Euclidean distance metric is used, and all values are normalized to an 8-bit range, $[0, 255] \cap \mathbb{N}_0$.*

To put things into practice, the task is to not re-train a given pre-trained neural network, instead, only the top layers are fine-tuned. To maximize the prediction accuracy, the color space of the input data is modified to make some features more prominent. Therefore the task is to find a set of archetypes that can optimally map the pixels to support the model's performance. This type of analysis was first introduced by [104] roughly said the authors are describing the search for the archetypes that are prominent convex combined values that are able to represent the entire data item such as an image. All in all, the benefit is that the archetypes are able to estimate a convex hull.

In order to find the best suitable values for the archetypes, to avoid a brute-force search a genetic algorithm is used, leading to an optimized time efficiency with the costs of what might not be the globally best values. The initial values for the algorithm are the archetypes generated by taking some random values and the original data, in this particular case using the dataset Gaofen [5]. Each RGB image is represented by three archetypes that are mapped to the channel with the archetypal projection function. The resulting data is then employed to fine-tune the top layers of CNN pre-trained on ImageNet, that is, in this case, a VGG16 model with two top layers. The first one is a fully-connected layer with 32 units and ReLU activation followed by a softmax prediction layer. Additionally, a 0.5 dropout has been applied between them. The model has been fine-tuned for 15 epochs on 50% of the data without any overlap between the train, validation, and test set. Furthermore, the genetic algorithm has trained models for approximately eight hours.

The validation dataset is used to calculate the performance of each individually generated item of one round of the genetic algorithm. The top-k archetypes are then selected towards their fitness on the basis of those new

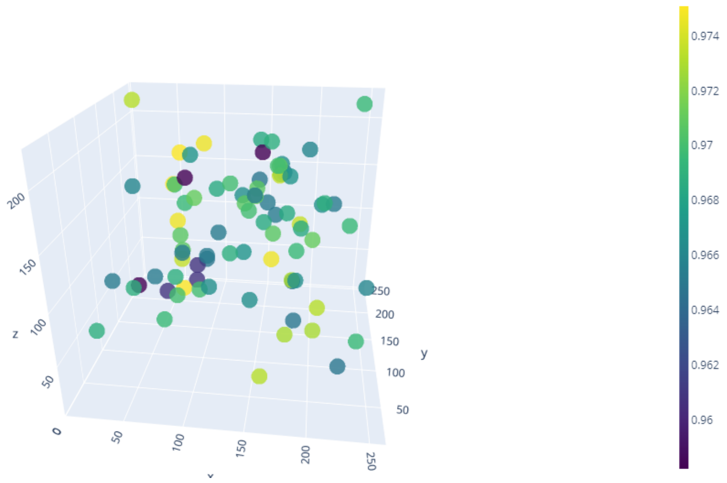


Figure 4.1: Visualization of the different identified local extrema, called archetypes, within the feature space that has been generated during the genetic algorithm, where it is not visible if there are correlations between the single values. A model that is trained on RGB images does use three archetypes. The space is large enough to provide the diversity to might support the model begging. First published in [P3].

convex combinations that are generated. The new values are created based on two parents, therefore, $k = 2$ variables X_k, X_l, and a random variable $\alpha = [0, 1]$. In addition an offspring is added to the given archetypes to distribute the newly generated ones in the space instead of having them in its origin. This is done with the equation

$$o_j = \alpha a_k + (1 - \alpha) a_l + \mathcal{N}(0, \sigma), \tag{4.1}$$

where o is the offspring, and a normal distribution $\mathcal{N}$ is used to randomize the values, with $\mu = 0$ and σ as deviation.

4.1.1.3 Experiments and Results

In contrast to gradient-based training methods, genetic algorithms have the advantage of being able to track more than a single local maxima. Additionally, also used to avoid brute-force searching algorithms, leading to a possible decrease in computational complexity. For the proposed basgging method, it is essential that there is no relation between the models, even when the archetypes are based on the same parents, only then can the algorithm improve the prediction performance. On the other hand, the recombination of

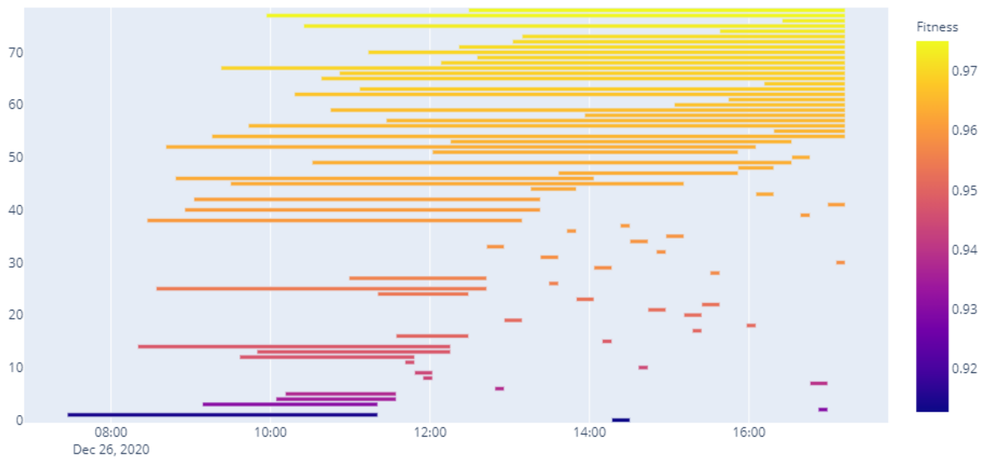


Figure 4.2: Visualization of the lifetime of the models in relation to their prediction performance. It can be seen that models generated later in the time span tend to have higher accuracies. First published in [P3].

related convex archetypes does lead to a good exploration within the problem space. Therefore two questions need to be answered: the first one is if the proposed method is able to provide the flexibility and efficiency to find more optimal areas within the feature space. Second is if a new generation in the algorithm identifies better types based on the encoded information of the previous generation.

Baseline performance: To be able to compare later generated models with the proposed approach, it is necessary to train a reference that uses the above described CNN trained on data without manipulation within the color space. The following experiments utilize the Gaofen dataset that has been provided by the 2020 Challenge on Automated High-Resolution Earth Observation Image Interpretation [5]. The model does reach an accuracy of 94.0%.

Stability of Diversity: When dealing with bagging or boosting algorithms, a challenge is that the modes have less diversity in finding the local minimum. Especially then, the different models do find the same optimal point within the problem space, a combination does not lead to any performance increase. The experiments showed that the models do provide the multiplicity to avoid a performance over time. The genetic algorithm is able to find enough local extrema to support a bagging of models without stagnating in

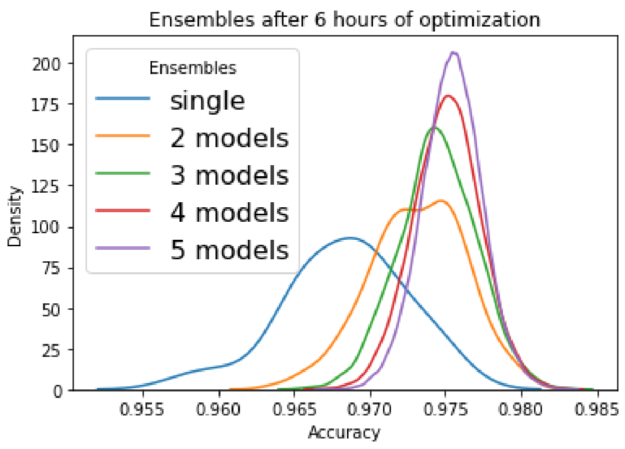


Figure 4.3: Model ensembling is done aggregating the models by their softmax's means. The visualization shows the ensembling of one to five models. It can be seen that the number of models is inversely proportional to the accuracy's variance. First published in [P3].

performance. Figure 4.1 shows the positions of located extreme within the archetypal space that has been selected for further processing. While the figure itself does not show the correlation between the individual archetypes, the models do use different areas within this space. Due to the difference within the color space, certain features are highlighted, and the pre-trained models vary from each other. This assumption can result to the possibility that the models are uncorrelated to each other and increase in performance when using bagging.

Genetic knowledge exploitation: To be able to interpret the ability of models to explore the archetype space with linear interpolation, it is necessary to take investigate the lifetime of the different models. Figure 4.2 does provide an overview of when a model has been created and when it has been rejected due to low fitness. Therefore, models with higher performance have longer life than bad ones. When analyzing the model creation timestamps with respect to their accuracy, it can be seen that there is a trend of providing better models over time. The experiments showed that the optimization on a single GPU (Nvidia RTX 2080 Ti) is able to boost the performance using bagging and archetypal transformation to 97.5% of a single model. Moreover, the computational effort to perform the linear equation is comparatively small.

Model bagging performance: One item essential point of bagging is the aggregation of models, this is done by taking the mean of the softmax layers outcome. Figure 4.3 provides an overview of the beg performance of one to five members. The main archived aspect is that the variance of the accuracy is indirectly proportional to the number of aggregate models, additionally, the accuracy has reached over 97.5% on average for considering five models when considering the unmodified source RGB data.

All in all, it can be stated that the performance and stability increase when ensembling uncorrelated modes using bagging. This is caused by the diversity of focusing on different features within the given data item.

4.1.1.4 Summary

This part of the work showed that color space manipulating computer vision techniques could be used to increase the performance of predictions employing neural networks that are focused on Earth imagery. The bagging of models using color space transformation based on extreme points, called archetypes, was able to push the performance. A following ensemble of multiple models increased the stability by reducing the variance of the accuracy when testing the models on the reference data. Additionally, this enables to apply methods that are designed for three channel images on multi or even hyperspectral data.

There are some directions for future work that are open to be answered or need to be investigated. First is a benchmark of different, more radical transformation function and to test the concept to actively prune channels of multispectral images. The second is to combine the proposed method with model pruning to prove slimmer models for edge devices.

4.1.2 Image Compression for Communication Reduction

This section has been published in our work [P11] and is structured as follows: An introductional motivation is provided first, followed by the related work in Section 4.1.2.2. Furthermore, the Section 4.1.2.3, furnishes a discussion about the proposed approach and the corresponding fundamental principles. While Section 4.1.2.4 introduces the relevant datasets, the Experiments are then evaluated in Section 4.1.2.5 and finally summarized in Section 4.1.2.6.

4.1.2.1 Motivation

Currently, many Earth orbiting satellites are used for different tasks: navigation, satellite television, to name a few. One of the tasks is the observation of the surface to detect the impact of disaster events such as wildfires or to monitor urban development. For those proposes, the satellites are equipped with a wide range of different sensors, multispectral- and hyperspectral, that generate a large amount of data that serves as the backbone of today's environmental research. While the quality of those data, like the resolution of images, is central for research, the amount of memory that is needed to store this data makes the accessibility a non-trivial task. Even experiments, as in urban development, are often done on a selected local spatial area rather than running global investigations. While self-hosted solutions often face computational and storage problems in this domain, large companies, e.g. Google and Amazon, provide hosting solutions that are able to host this amount of data. This is, done for the Sentinel data that is available on Amazon Web Services. This comes with the costs that the companies have a vendor lock-in and the power over the data itself.

Another aspect that is worth it to be considered when it comes to data, especially data that has been generated from optical devices, is that the majority does not help to solve a given particular problem. If a scene is from interest for a specific task, all 13 band provided by Sentinel-2 needs to be downloaded. On the other hand, the data that is included in this product will be entirely employ only in rare cases, as it is the case for the classification of land use and land cover. Additionally, the full complexity of a remote-sensing image is often ignored by many projects and products.

A lowering of the data's complexity could reduce the amount of data that required to be transmitted over the communication channel. Additionally, the pruning of information minimizes the memory footprint and the energy that is needed to solve a particular problem. Therefore, this part of the thesis focuses if compression can be applied to optimize the training and inference phase of a convolutional neural network in relation to its runtime while having a marginal loss in accuracy. To investigate the images are firstly quantized to an extensive range of target bit depths, in addition, embedded into compression-applying file formats, for instance JPEG. The evaluation of the time that is needed for the training on GPU and the archived accuracy is based on the deep learning architectures VGG-16 [51], ResNet [48], and MobileNet [53], which are pre-trained on ImageNet.

4.1.2.2 Related Work

In this section, we give an overview of the related work as aspects that are relevant to this work. Therefore the main parts which are discussed are the state-of-the-art of compression in deep learning and how data is currently stored.

Images and the art to store them: When researching in the context of optimizing the machine learning pipeline for remote sensing images using compression, there are many dataset types that are essential. One prominent dataset is ImageNet ILSVRC 2012 which has been introduced in [50] and is common in transfer learning. Moreover, the most convolutional neural network are benchmarked on this dataset. Compared to computer vision datasets, the domain of satellite scenes has many forms, such as radar data from Sentinel-1. There are even considerably differences between the sensing devices of the different satellites, for example, comparing Sentinel-1 and TerraSAR-X were both creating Synthetic Aperture Radar (SAR) images. In this work, we are focusing on multispectral data, where one contributor is Sentinel-2. According to [9], those satellite provides 13 spectral channels and delivers them using JPEG2000. The number of bands and the data format differs between the missions. Another aspect is that satellite scenes are furnished with different stages of pre-processing. When considering datasets based on Sentinel-2's multispectral images, one version option is the raw data, another one is with already applied pre-processed, which can cover steps as tiling and labeling, as it has been done in [P2]. An extreme case builds hyperspectral images (HSI), which give a vast count of (multispectral) images. The amount can be challenging for deep learning models in both cases,especially with respect to its computational complexity.One dataset is, e.g., BigEarthNet provided by [4], which includes 590 326 tiles with a depth of 16-bit extracted from Sentinel-2 scenes. Each one of the 13 bands has been stored as an individual TIFF which leads to 7 674 238 files. Other datasets stored them using an 8-bit representation [10] or delivered only the $\{R, G, B\}$ bands [105]. Other datasets provide more than three bands, and storing them into one image format [106].

Instead of taking standard formats like TIFF, PNG, or JPEG to store each image individually in the file system of the operating system, one can use a hierarchically structured file format. This data structure is able to hold entire datasets within one single file. Those formats are especially relevant when the computation is performed on high-performance computing (HPC) systems. One candidate in this group is the format HDF5 invented by [107], which is able to hold data in various forms. Furthermore, one advantage of this data structure is the possibility to chunk the data, which enables it to define pre-define either batches or even entire subsets. Another representative is TensorFlow's TFrecords provided by [108] and is similar to HDF5 with the constrained of not being able to chunk the data, nevertheless, the format is serialized to feed the deep learning model created by the framework and is also suitable to be applied on systems with a file limit in a directory as it is the case for some HPC systems.

One needs to mention that there are lots of other possibilities to store the data, for instance using databases. An example is Google's Bigtable [109] which is a high-performance database specialized in handling vast amounts of data. Furthermore, without going into detail, it is essential to know that file systems have a theoretical and practical limit. When considering EX4, the theoretical limit per directory is four billion in files, while in practice, in some scenarios, a count of $\leq 100\,000$ files can already produce problems.

Compression in deep learning: The compression of images is a field that is heavily investigated from all sides, a reason might be that the use reduces the footprint of the data and saves on this way resources. Additionally, the reduced size can also lead to optimized characteristics in computing. Furthermore, this domain is also present in the field of machine learning, for instance, current research tries to optimize data compression with deep learning algorithms. Nevertheless, compression comes with the drawback of having the possibility of losing information and the creation of artifacts, e.g. blocking, that have a notably impact on the precision of algorithms. In this part of the work, the consentration is set on the impact of compression on the classifier in relation to their performance and accuracy. One option to decrease the footprint of an image is to apply quantization. The authors of [8] do research in this direction, more specifically, the experiments focus on quantization and pruning of details. Before the classification, the authors quantized into several spaces with different sizes, additionally, information that seems to be less important has been pruned. Finally, the evaluation of the compression has been done on CNNs linke VGG16 and ResNet50. While the scope of the described work is about images, instead in the domain of computer vision and not based on remote sensing images that cover, e.g. land cover or land use, the data is compressed with pruning and not on normal compressors such as ZLIB or file formats like JPEG. On the other hand, the authors of [7] do target on compression with file formats and their impact on learning algorithms. While the authors investigated the compression itself, this project direct on machine learning and its performance itself, which is also the scope of this section. The limiting factor of their work is that only JPEG is investigated instead of all common formats.

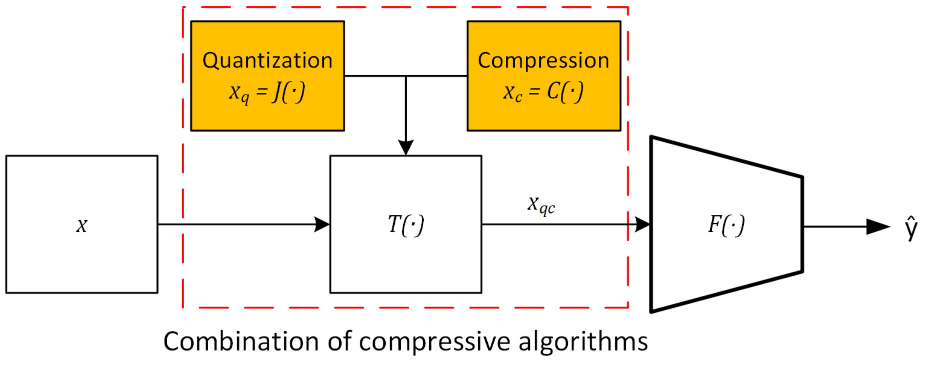


Figure 4.4: An end-to-end pipeline is designed to evaluate the impact of compressing the input to a deep neural network. In the beginning, the data x that consists of images (and the corresponding labels) is compressed by a function $T(\cdot)$. This function has two steps, first is to quantize to a lower dimensional color $x_q = J(\cdot)$, e.g. by representing each pixel color by one bit instead of eight. The second step is to embed the image into a selected file format. While, PNG provides a lossless representation, JPEG compresses the images by removing high frequencies, which can cause artifacts. The last step of the pipeline is to train a convolutional neural network $F(\cdot)$ and evaluate its results. The figure has been adapted from [P11].

The methods for image archiving in the domain of remote sensing [110]–[112] are, for example, based on hashing-based image retrieval. Those methods can significantly increase the throughput of searching and storing satellite scenes with full resolution. In contrast, the scope of this work is to investigate the impact of compression on deep learning models in the training phases as well as the inference in custom hardware.

4.1.2.3 Proposed Approach

In order to evaluate whether the compression of input images does have an impact on the CNN in terms of the time needed to train a model and its final accuracy, an end-to-end pipeline has been defined. The focus is set on two types, first is the reduction of the color space with the use of quantization, and the second is the embedding into file formats, where some of them prune information permanently. The end-to-end approach is beneficial to automatically evaluate each configuration pair against each other. The goal is to check if it is possible to compress the input information to reduce the training time with as less classification performance loss as possible. This ensures the optimization of the deep learning pipeline for spatial data without scaling the hardware.

The end-to-end pipeline to evaluate the impact of compression on deep learning models is visualized in Figure 4.4, and will be built during this section. The first part of the architecture is the input X that is part of the dataset in its source form. The next phase is the compression $X_{qc} = T(\cdot)$ that corresponds of two sub-steps. The first is to quantize the image into a range of levels such that a target bit depth is reached. Afterward, the image with the smaller color space is embedded into a set of file formats, where some of them do apply a lossy compression. The formats TIFF, JPEG, PNG, and BMP are selected because they are common and nativity supported by the TensorFlow framework. Each configuration pair is then trained $F(\cdot)$ using a separate CNN and evaluated against all other possibilities in terms of the impact of compression on the training time and the classification accuracy.

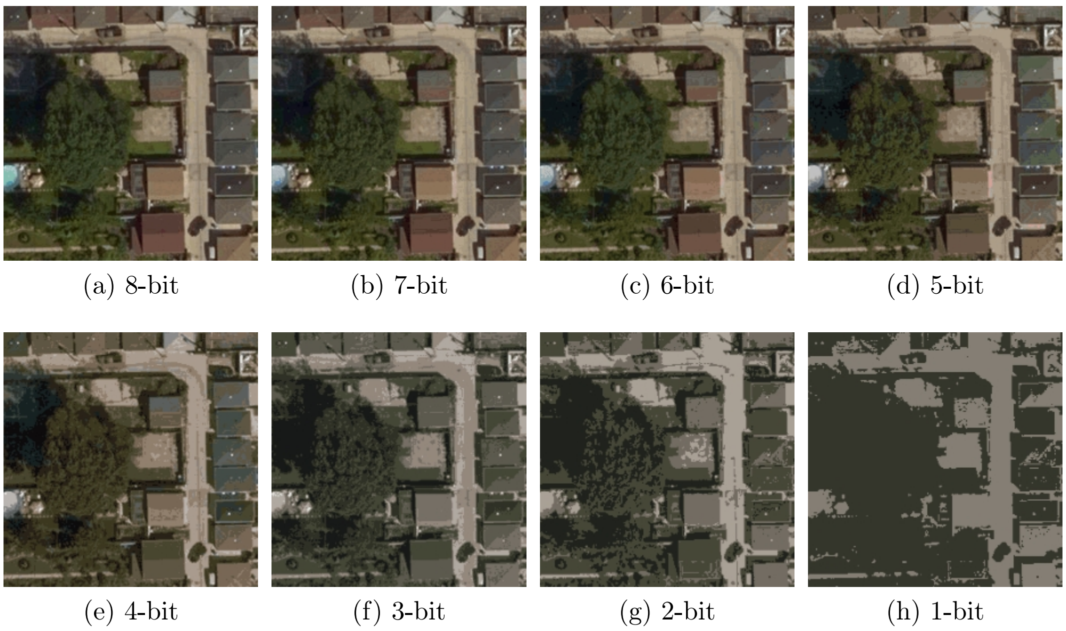
The problem definition. Dataset d consists of images x and the corresponding labels y, where each item pair build a set $\{x, y\}$. Furthermore, a neural network $\mathcal{F}$ with a set of parameters Θ is used as a classifier, formally defined as $\mathcal{F}_\Theta(\cdot)$. This classifier is able to compute an estimated label $\widehat{y}$ based on a given input x, formally said $\widehat{y} = \mathcal{F}_\Theta(x)$. Additionally, the neural network is trained by minimizing the loss function $\mathcal{L}(y, \widehat{y})$, which can be done by the following function:

$$\Theta^* = \arg\min_{\Theta} \sum_{(x,y)\in D} \mathcal{L}(y, \mathcal{F}_\Theta(\mathcal{T}(x, b))), \tag{4.2}$$

where Θ^* depicts the optimal set of parameters and $\mathcal{T}$ defines the transformation function that compresses the images to x_{qc}. The first quantized to a target bit depth 2^b and then embedded into a selected file format. In consequence, it needs to be evaluated how large the impact of $\mathcal{T}$ on Θ^*.

In summary, the end-to-end pipeline consists of four parts, quantization, embedding into file formats, training of the neural networks, and the final evaluation. In order to investigate the impact of compression following paragraphs discuss those components in detail.

Image quantization: The first compressive method that is applied on the architecture from Figure 4.4 is the process of quantization. This takes an input and reduces the space that is available to represent a pixel of a channel, formally $J : x \to x_q$, where J represents the quantization function, furthermore, $x_q \leq x$ with respect to their bit depth. Moreover, it is necessary to differentiate between two compression strategies: *lossless* and *lossy*. The


(a) 8-bit (b) 7-bit (c) 6-bit (d) 5-bit

(e) 4-bit (f) 3-bit (g) 2-bit (h) 1-bit

Figure 4.5: An standard RGB image consists of the common of three channels that have a total bit depth of 24 bits, so eight bits per layer. When reducing the space that is available to represent a single pixel from a channel, also shrinks in size. The images above depict various quantization levels between eight bit per pixel per layer down to a single bit. While the eight bit version shows a scene in its full complexity, the smaller the color space gets, the more are only edges visible. This figure has been first published in [P11].

former defines methods that are able to change the representation of given data such that no details are lost. Consequently, the source data can be reproduced in all its details based on only the compressed version. On the other hand, lossy methods allow an error between the two representations, such that the source data cannot be entirely reproduced without differences. While those strategy archives higher compression rates, it does come with the costs of pruning information. Image quantization is part of the latter compression strategy family, which reduces the bit depth by shrinking the color space, and therefore, information is lost. [113]

All in all, there is an extensive range of quantization algorithms and strategies that are currently used. A primary and classic method is clustering, where each pixel is assigned to a cluster that represents a color. In this part of the work, we utilize the k-means algorithm for quantization [114] to decrease the bit depth of each image.

The k-means clustering algorithm is a rather basic and well-known method.

Nevertheless, each data item that is in this case a $\{R, G, B\}$ pixel, is placed into the space where all possible distances are computed. This is done utilizing a similarity measure $\mathcal{V}$, while the Euclidean norm is common, the choice of the metric depends on the type of the given data as well as on the space itself. Firstly, k-means initializes k random clusters centers μ_k. Afterward, the distances to all centers are calculated for each data item x_n within the space and assigned to the closest μ_k. Formally defined as

$$\mathcal{V}(x_n, \mu_k) = \parallel x_n - \mu_k \parallel^2, \tag{4.3}$$

where $\parallel \cdot \parallel$ donates the Euclidean norm. Furthermore, while each item is now assigned to a cluster center, the membership of the cluster is stated with the use of a vector of binary indicators r_{nk}, where $r_{nk} = 1$ if $k = \arg\min_j \parallel x_n - \mu_j \parallel^2$ and $r_{nk} = 1$ if $j \neq k$. The combination of this with eq 4.3 builds the k-means clustering algorithm, formally defines as

$$\mathcal{J}(x, k) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \mathcal{V}(x_n, \mu_k), \tag{4.4}$$

where N is the number of x in D. In this term item, x represents a $\{R, G, B\}$ pixel that is assigned to one of the clusters that represent the possible clusters. Therefore, the amount of clusters control which color is available, practically said if 64 centers do lead to a bit depth of $log_2(64) = 6$ bit available to represent one pixel of a single channel. In this way, the values in the images change with the size of the color space. All in all, the quantization uses two inputs that are the input x and the number of bits b that are available to represent a single value. Formally defined as

$$x_q = \mathcal{J}_{RGB}(x, 2^b), \tag{4.5}$$

where x_q is the quantized picture that has a bit depth of b. The effects that are caused by this compression type are visualized in Figure 4.5, which shows all the quantization levels from eight bit per layer (commonly known as RGB/sRGB) down to one bit of a three-channel image. [113], [114]

Image compression using file formats: The first compressive method that is applied in the architecture from Figure 4.4 is to embed the images x into selected file formats. Typically, those consist of two major parts: the compressor and the encoding. The former algorithm is utilized to minimize the memory footprint to keep the file as small as possible. For example, the format PNG does apply the ZLIP compressor on the images that come with several modes that define the degree of compression [115]. In contrast, the encoded is responsible for changing the representation to embed into the file format. While some do save the raw matrix, such as a grayscale, other formats like JPEG do emply more complex methods. Nevertheless, as explained above in image quantization, there are two compression strategies that find their application also in file formats. Similar to the previous definition, some formats can use a lossless compression $C_{lossless}$, with the goal of being able to reconstruct the source image without any error from the embedded representation. The following defines the related formats:

$$\mathcal{C}_{lossless} = \begin{cases} TIFF(x) \\ PNG(ZLIB(x)) \\ BMP(x). \end{cases} \tag{4.6}$$

In contrast to this, the second strategy is to use lossy compressors C_{lossy} that accept a certain error between the source image and the reconstruction based on the embedded data. The following defines the related formats:

$$\mathcal{C}_{lossy} = \Big\{ JPEG(DCT(x)). \tag{4.7}$$

In order to generalize this a set can be defined that includes the listed file formats from Eq. 4.6 and Eq. 4.7 that defines this type of image compression $\mathcal{C} = \{\mathcal{C}_{lossless}, \mathcal{C}_{lossy}\}$. In summary, the quantized image x_q is compressed by embedding into selected file formats, formally said

$$x_c = \mathcal{C}(x_q), \tag{4.8}$$

where x_c is the compressed version of the input. The following gives an overview of the relevant image formats.

Portable Network Graphics (PNG). The first format is PNG, $C_{PNG} = PNG(ZLIB(x))$, and part of the lossless compression file formats. Furthermore, it is possible to embed images with up to 8 bit per layer and up to three color channels $\{R, G, B\}$ plus an alpha band. The ZLIB compressor does support a range of compression levels (CL), from zero to nine. While zero indicates an uncompressed image, a level of one is focused on the performance, and the concentrate of level nine is set on the maximum compression. The impact of

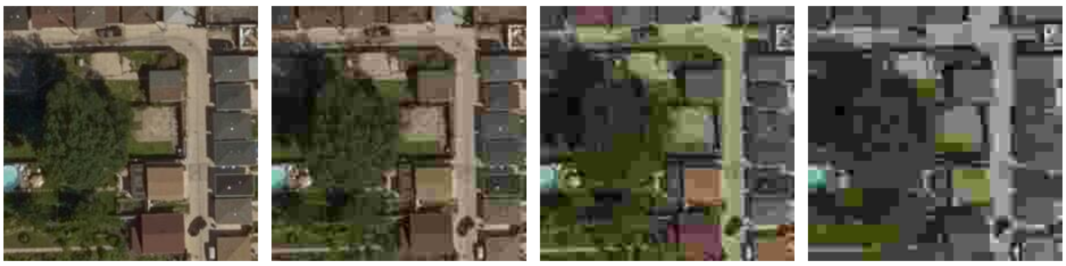
(a) CL 0 (b) CL 1 (c) CL 9

Figure 4.6: The image format PNG is able to hold up to three channels (plus an alpha band) with a maximum bit depth of 24-bit in total. Additionally, ZLIB is used to compress the data to reduce the memory footprint. The compressor has nine levels in total, while the lowers indicates that no compression is applied, and one sets the concentration on the performance. Level nine focuses on maximum compression. Furthermore, PNG is a lossless image format, therefore, the source data can be reproduced by the compressed version without any error. This is also the reason why there is no visible difference between the sub-figures. This figure has been first published in [P11].

the compression levels from ZLIB is depicted in Fig. 4.6, where no significant difference is visible due to the lossless nature of the file format. [115]

Bitmap (BMP). The bitmaps (BMP) $C_{bmp} = BMP(x)$ embed the raw pixels without applying compression, therefore, this file format is at least theoretically part of the lossless formats. The missing compressor leads to the disadvantage that items might be stored inefficiently compared to other formats that even prune parts of the image. Furthermore, bitmaps do support a bit depth up to 24 bit, three-layer pictures $\{R, G, B\}$ plus an alpha channel. In consequence, if the image x or its quantized form x_q exceeds those limits, whether it is the number of bands or the bit depth, it is necessary to scale the affected aspect. [116]

Tagged Image File Format (TIFF). Another format that does not apply a compressor by default, but has the option, is TIFF $C_{tiff} = TIFF(x)$, and therefore also lossless. While the last format was able to hold only images with up to three plus one channel, this format is more flexible and supports a wide range of bit depths and the number of layers. Additionally, this format is often used in the field of remote sensing due to the possibility that it is



(a) QF 75 (b) QF 10 (c) QF 5 (d) QF 1

Figure 4.7: The image file format JPEG is theoretically able to hold more than three/four bands and a bit depth of 24 bit per layer. Commonly the codecs only support embedding three channels (plus an alpha channel) and 24 bits in total. Furthermore, JPEG applies lossy compression to the images, which reduces the memory footprint, this is controlled by a parameter called quality factor (QF). While a value of 100 is a theoretically uncompressed image, the compression rate increases by lower values. With lower values, the occurrence of image artifacts, such as blocking, increases. This figure has been first published in [P11].

possible to store e.g. all 13 bands of a Sentinel-2 scenes into one single file. While images with a higher bit depth than 24 bit can be embedded as well, this work focuses on 24-bit standard $\{R, G, B\}$ to be able to compare the file formats against each other. [117]

JPEG (JPG). Compared to all other formats that have been introduced, JPEG $C_{jpeg} = JPEG(DCT(x))$, named after the Joint Photographic Experts Group, is a lossy compression algorithm. Commonly images with a bit depth of up to 24 bits are embedded by codecs that have between one (grayscale) and four ($\{R, G, B\}$ and an alpha channel) bands. Compared to the common usage, the JPEG standard also supports images that have a bit depth of 24 bit per channel. Furthermore, the compression factor is controlled by a parameter called *quality factor* (QF), where the minimum value of one represents the maximum compression and 100 is theoretically uncompressed. The impact of different values is depicted in fig. 4.7, where it is visible that the image with $QF = 75$ does only have small to no distortions. On the other hand, in the case of $QF = 1$, a large number of artifacts are visible, and even a loss in color. [118]–[120]

Image classification using a CNN: The next step in the end-to-end pipeline that is described in Fig. 4.4 is the training and classification using a

CNN. Before being able to estimate a label $\widehat{y}$ based on the image x_{qc}, it is necessary to finalize the compression function $T(\cdot)$ by combining the quantization with the embedding process. The former applies the described function $\mathcal{J}_{RGB}(x, 2^b)$ that reduces the bit depth to a target value b for every single pixel of each $\{R, G, B\}$ channel. While low quantization rates have a more negligible impact, edge cases lead to a high amount of information such that only the structures of elements are present. Conversely, the second component is the embedding into file formats that are done using the function $C(x)$, where the level of compression and the occurrence of artifacts are based on the format itself. While PNG is a lossless format that ideally has no impact on the loaded image, JPEG prunes the high frequencies to a certain defined level which can cause a high amount of distinctions, for instnace blocking. Anyways, the combination of those components builds the compression transformation function $\mathcal{T}$, which is formally defined as

$$\mathcal{T}(x, 2^b) = \mathcal{C}(\mathcal{J}_{RGB}(x, 2^b)), \tag{4.9}$$

where x is the input image, and b the target bit depth.

In addition to the compression, the final step before the evaluation is the training and classification of the data. Therefore let $\mathcal{F}(\cdot)$ be a CNN classifier and Θ be the parameter set that is used. The task is to compute an estimated label $\widehat{y}$ utilizing $\widehat{y} = \mathcal{F}_\Theta(x)$ based on an input image x. Additionally, the model is trained by minimizing the loss function $\mathcal{L}(y, \widehat{y})$, formally defined as

$$\Theta^* = \arg\min_{\Theta} \sum_{(x,y)\in D} \mathcal{L}(y, \mathcal{F}_\Theta(x))), \tag{4.10}$$

where Θ^* is the optimal parameter set and (x, y) the image label pair that is part of the dataset D. The full architecture is the combination of the $\mathcal{T}$ with the compressor $\mathcal{C}$, therefore, the architecture is formally defines as

$$\widehat{y} = \mathcal{F}_\Theta(\mathcal{T}(x, b)) = \mathcal{F}_\Theta(\mathcal{C}(\mathcal{J}_{RGB}(x, b))). \tag{4.11}$$

Evaluation: *The scope is not to outperform existing state-of-the-art baselines, rather to have a reference to compare.* The compression of images can have large manipulating characteristics that affect the color and the complexity available for each pixel. Furthermore, in edge cases where the compression factor is high, there is also a significant impact on the footprint in memory. The scope of this work is not to prove that higher accuracies that the current

Table 4.1: Overview of the datasets. This table has been table published in [P11].

Name	No. Images	Source Format	Storage Size
AID	10 000	JPEG - QF 100	2 616 MiB
EuroSAT	27 000	TIFF	3 904 MiB
RSI-CB256	24 956	TIFF	4 73 MiB

baselines can be archived, nor to compete with state-of-the-art compression algorithms. The focus is to investigate the impact of compression on modern classification algorithms and to check if compressed representations can improve performance without a significant decrease in accuracy. Therefore, each image is compressed in two dimensions, one is to reduce the complexity available for each pixel using quantization, and the second is to embed the image into various common file formats. The training and inference results are then compared to each other in terms of time consumption and prediction performance. Furthermore, selected models are then deployed to an FPGA, namely an ultrascale+ architecture with a 10 digit number hosted on a Xilinx ZCU102 evaluation board, to measure the time and energy consumption during the inference.

4.1.2.4 Datasets

This section describes the datasets which are used in this part of the work. To be able to interpret the results from the experiments indispensable to provide additional details about their file formats, storage size, and other key feature indicators. The baseline classifications are furnished later in this section. Furthermore, essential information about the datasets is summarized in Table 4.1.

AID: The dataset AID has been provided by [121], and its task is to predict the land use and land cover type. The images from the dataset are extracted from Google Earth and are grouped into 30 classes, where a class is, for example, *Airport*. The full set covers 10 000 individuals with on size of 600 × 600 pixels. Furthermore, the images do only given the optical red, green, and blue bands with a bit depth of 8 bit per channel. Additionally, the dataset is not geo-referenced. The missing spatial metainformation leads to the problem that it is not ensured that there is a spatial correlation between the test, train and validation set. Nevertheless, it is unlikely that this affects the model performance in terms of over-estimation, therefore, it does not affect the planned investigation, which is relevant for this section of the work. Furthermore, the dataset has a total size of 2 616 MiB and is available in JPEG files utilizing a quality factor of 100. Furthermore, there is also an updated version of this dataset published in [122] containing 400 000 images within the 30 classes. Unfortunately, this dataset is not publicly available.

EuroSAT: EuroSAT has been provided by [10], [106] and its task to predict the land use and land cover type. The dataset consists of 27 000 geo-referenced images, which are extracted from Sentinel-2. Furthermore, the dataset comes with all 13 multispectral bands sensed from the satellite, including the red, green, and blue channels. Within this work, mostly the optical channels red, green, and blue are considered for the experiments. Each one of the images has a pixel-wise resolution on the ground of 10 meters and a size of 64×64 pixels per patch. The images are labeled and correspond to one of the 10 classes. The dataset is close to being balanced, where each class contains between 2 000 to 3 000 elements. In addition, the dataset has a total size of 3 904 MiB and is available in TIFF files with a bit depth of 8 bit per layer. There is a second version of the dataset, which employs JPEG images with only providing the optical $\{R, G, B\}$ bands, which has not been used in this work.

RSI-CB256: RSI-CB has been delivered by [105], and its task is to predict the land use and land cover type. The dataset exists in two varieties, RSI-CB128 and RSI-CB256, differing in the size, count of classes, and quantity. This work considers only the letter one, which consists of 24 000 images with a size of 256×256 pixels and corresponding to 35 classes. Furthermore, each image provides the three optical bands, red, green, and blue, with a bit depth of 8 bit per channel. Unfortunately, the dataset is not geo-referenced and therefore leads to the same problems and consequences as with the previous dataset AID. Moreover, the images are provided taking the format TIFF with a total size of 4 734 MiB on disk.

4.1.2.5 Classification of Compressed Images

This section evaluates the impact of compression on deep learning with the architecture introduced above. Therefore, three different convolutional neural networks have been chosen for training and inference: VGG16, ResNet50, and MobileNet. Those particular architectures were selected because of their sensitivity to colors and shapes, their comparable trivial structure, and their ability to reach high prediction results. While all images from the datasets are scaled to a size of 224×224 pixels, all networks are pre-trained on ImageNet from 2012 [50]. Furthermore, to correctly evaluate the experimental results

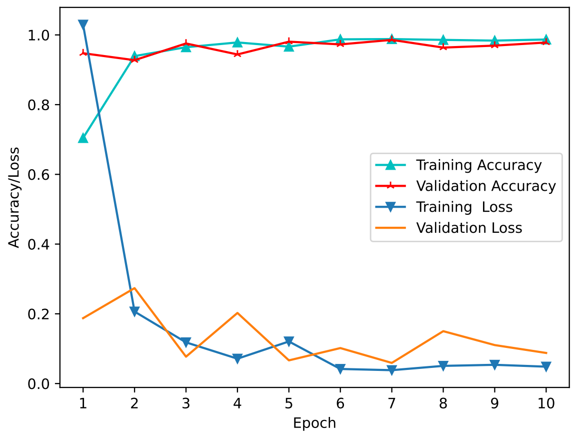


Figure 4.8: In order to evaluate the impact of compression on CNNs, it is necessary to create baseline models. The goal is not to outperform the related current state-of-the-art literature, rather, it is needed to create robust models the do not tend to have a large generalization error. The visualization shows the training process of the uncompressed dataset RSI-CB256 using VGG16. This figure has been first published in [P11].

and ensure the arability of comparing these results even with experiments from other papers, the original top layer configuration defined in the corresponding papers [51], [48], [53] is employed. Moreover, each model is trained for ten epochs with a batch size of 128, except for AID@ResNet50, where a 64 was taken. Additionally, the default learning rate is 10^{-4}, where only AID@VGG16 and AID@ResNet50 used a value of 10^{-5}. Additionally, experiments were done on two *Nvidia A100* GPUs á 80 GB memory.

The following paragraphs focus on defining baseline models to compare the models against an uncompressed version. In the following, all models are trained as described above and evaluated with respect to their classification accuracy. This is then followed by a comparison of the runtime needed to train each model. While this is done on GPU, another common aspect is to perform the inference on custom hardware, such as an FPGA. Since the upcoming of radiation-safe custom hardware, the space industry is interested in deploying deep learning on satellites which comes with several restrictions. Later, the models are also deployed to an FPGA, in Section 4.2.1.4, and its characteristics, like the energy consumption, are investigated.

Classification baseline: The first is to train baseline models to be able to accurately evaluate the impact of the compression in a spatial context.

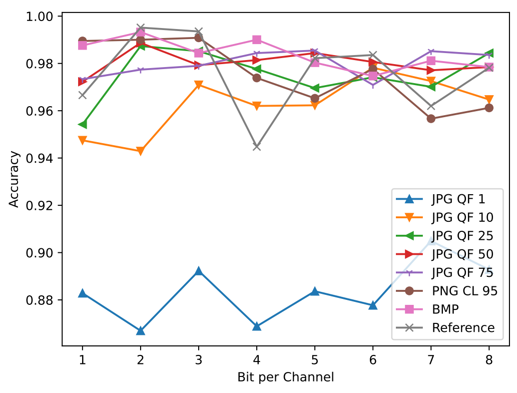
Table 4.2: Overview of the baseline classification results. This table has been first published in [P11].

Network	RSI-CB256	EuroSAT	AID
VGG16	98.00%	94.64%	87.60%
ResNet50	98.82%	96.00%	89.60%
MobileNet	98.27%	93.60%	85.20%

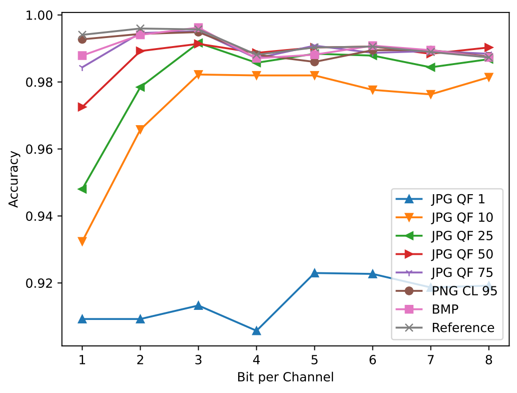
While the intention is not to outperform baselines from current literature, rather, it is more relevant to create models that are stable and robust without a tendency for overfitting. This is due to the motivation to reduce a generalization error to a minimum across the wide span of quantization as well as compression with image file formats. All baselines are trained with the scheme defined earlier in this application sample, so pre-trained on ImageNet and the top layers from the related papers. Additionally, the data is split into three subsets, where one held-out set is used only for evaluation. Figure 4.8 visualizes the training process of the dataset RSI-CB256 as a sample, the performances are summarized in Table 4.2.

Evaluation on the basis of the accuracy: The architecture supposes two options to compress the image, one is quantization, and the other one is to embedding into file formats. The former focuses on quantizing the image to a target depth with the k-means clustering algorithm. The first step is to randomly select a large number of pixels from the dataset to be able to build the clusters using $k = 2^b$, where $b = \{1, 2, 3, 4, 5, 6, 7, 8\}$. While each cluster defines a color within the space, each pixel gets assigned to one the nearest, which results in an (new) image that has the targeted depth. One advantage is that more prominent shapes and colors are highlighted when transforming the data into a smaller color space. This can emphasize dominant and high-contrast spatial structures that may support the network to extract the essential features. The second compression type is to embed the data into image formats that vary in their encoding strategy. For this experiment, we have chosen some of the most widely considered image file formats, including TIFF, BMP, and PNG, using ZLIP and a compression level $CL = 9$, as well as JPEG with the quality factors $QF = \{1, 10, 25, 50, 75\}$.

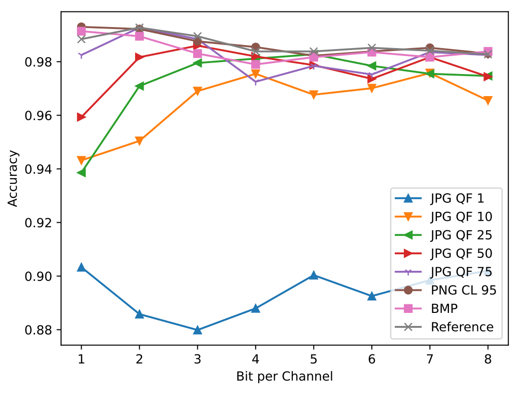
When analyzing the results and compare them to the related baselines, it is expected that higher compression rates cause and decrease in accuracy. On



(a) RSI-CB256@VGG16



(b) RSI-CB256@ResNet50



(c) RSI-CB256@MobileNet

Figure 4.9: The result of the training from the dataset RSI-CB256 with respect to the **accuracy**. This figure has been first published in [P11].

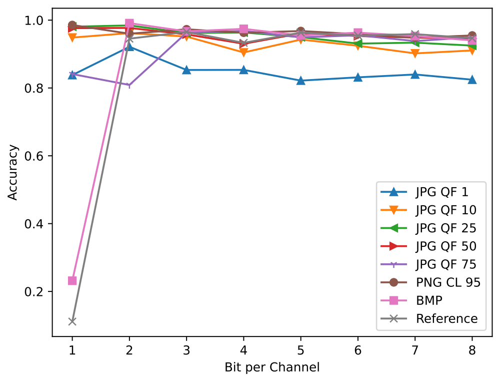
the other hand, moderate rates could support CNN towards reaching the same or even higher prediction precision. Considering the results from the dataset RSI-CB256 trained on VGG16 that is represented in Figure 4.9a, one can see that most of the parameter combinations reach acceptable results. More concretely, the baseline uses the TIFF and requires a size of 4 734 MiB in memory. While all bit depths do reach high prediction precision, the source of 24-bit reaches an accuracy on the test set of 98.00%. All over, other compression parameter configurations do reach comparable results. One example is the dataset embedded into PNG applying ZLIB, $CL = 9$, additionally, some bit depths even reach slightly higher accuracies than the reference model. When the data is quantized with $b = 7$, an accuracy of 98.53% is reached, additionally, the memory footprint is reduced to 958 MiB. In the case of JPEG with $QF = 10$, the situation is slightly different. While the test set was able to reach 97.24%, which is a decrease of less than one percent, with the advantage that the needs memory shrinks to 107 MiB. This means that the datasets require only 2.26% of the storage than taking the original configuration. Furthermore, as expected when considering a stronger compression rate as it is the case for $QF = 1$, too many features have been pruned from the image, leading to a considerable decrease in the performance.

The results for ResNet50 in Figure 4.9b and MobileNet in Figure 4.9c are quite similar to each other. While it is the same case that more aggressive compression rates cause lower accuracies, it is also visible that the robustness of the results decreases. When considering higher bit depths, it can be seen that the range of the accuracies is relatively narrow. Compared to that, the higher the compression rates get, the more comprehensive the range of values.

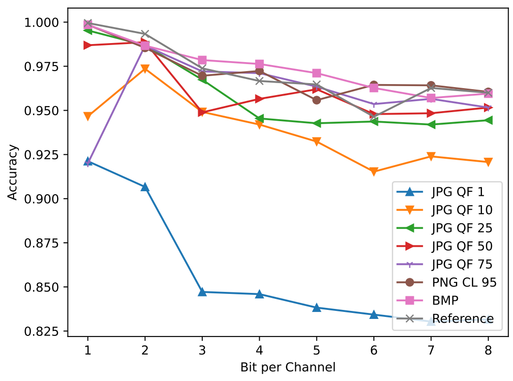
All in all, what can be seen in the case of RSI-CB256 is that higher quantization rates have an impact on the model's accuracy. On the contrary, all three network types, the selection of the file format and its parameter caused a comparable slight loss in accuracy but had the advantage of significantly decreasing the memory footprint. Therefore, based on the first dataset, it seems that some configurations, namely JPEG $QF = 75$, save a large amount of memory with the cost of losing some accuracy. While the loss of $< 3\%$ accuracy is negligible for lots of use cases (especially when considering the variation of results), some tasks cannot accept this performance drop.

Comparing the results of RSI-CB256 to EuroSAT visualized in the figures 4.10a–4.10c one can see a schematically slightly similar pattern. While the different combinations reach high accuracies, JPEG with $CF = 1$ has a low performance. As before, this is caused by the JPEG compression algorithm that removes high frequencies from an image and causes, in its extreme form, image artifacts, therefore, this leads to a loss of spatial details. Even when the performance drop is significantly more prominent in this example. Furthermore, another aspect that is more prominent than before is the fact that lower color spaces tend to have higher accuracy, especially when utilizing ResNet50 and MobileNet. This can have two reasons, (1) shallow color spaces tend to have more notable generalization errors, and (2) quantization is able to highlighter certain features within the images and support the classifier to reach high performance. We assume that both is the case, while (1) is more present in aggressive quantization, (2) is more likely to be present in moderate values. Considering also the results in RSI-CB256, one can see that it is likely that images are quantizable without a considerable impact on the accuracies.

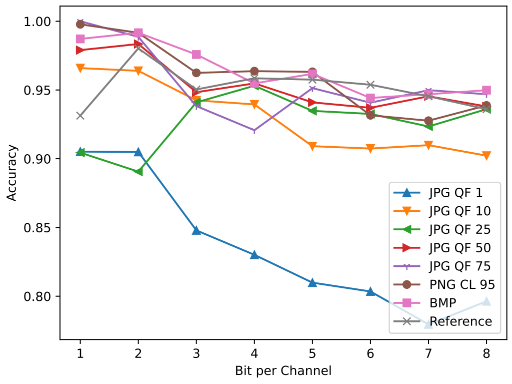
The AID dataset visualized in Figure 4.11a–4.11c supports these assump-



(a) EuroSAT@VGG16



(b) EuroSAT@ResNet50

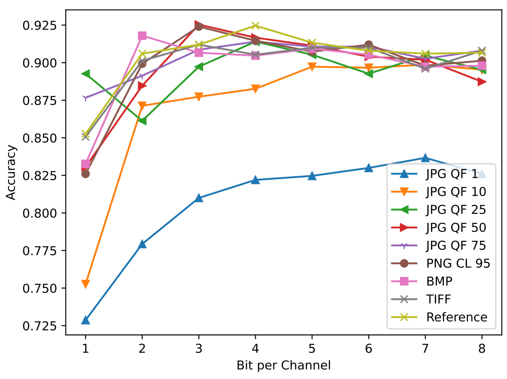


(c) EuroSAT@MobileNet

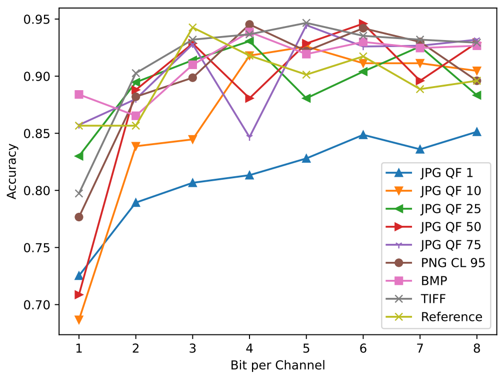
Figure 4.10: The result of the training from the dataset EuroSAT with respect to the **accuracy**. This figure has been first published in [P11].

tions due to the schematically similar pattern while having some different characteristics. One is that the accuracies are distributed over a larger area, especially when using high quantization rates, even when the data set is similar to the other ones except for the number of images and their sizes. The results and models are getting more robust when utilizing larger color spaces for each pixel. While the reference combination, JPEG with $QF = 100$, creates a baseline, it is interesting to see that compressed versions such as PNG with $CL = 9$ or JPEG with $QF = 10$ do reach comparable results. Especially latter supports the assumption that compression can help a neural network to focus on only relevant features from the images.

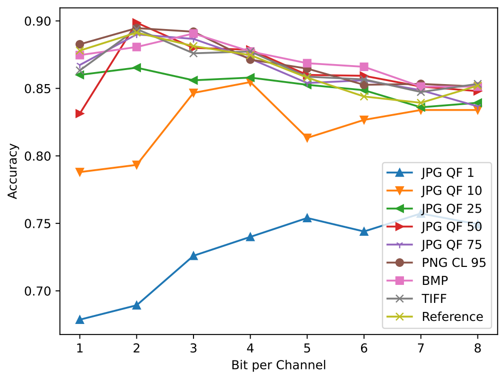
Considering all available results, the general pattern is quite similar. Low compression rates employing the maximum color space lead to good performances, caused by the fact that the entire complexity is provided. On the other hand, the higher the compression rate is, whether it is the



(a) AID@VGG16



(b) AID@ResNet50

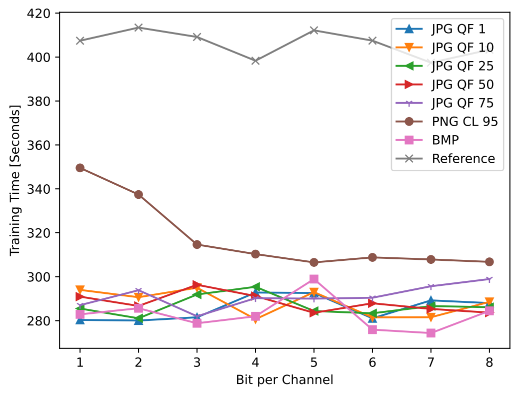


(c) AID@MobileNet

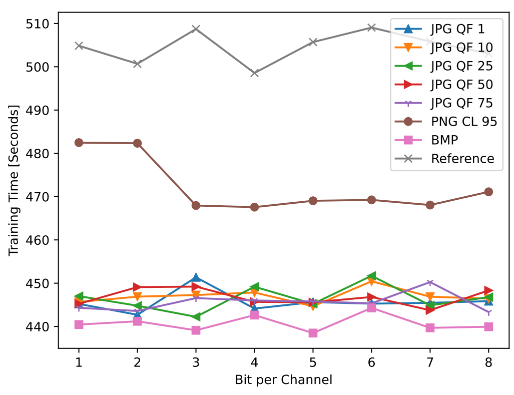
Figure 4.11: The result of the training from the dataset AID with respect to the **accuracy**. This figure has been first published in [P11].

compressor of the file format or the reduction of the bit depth, the accuracy shrinks, namely when considering JPEG with $QF = 1$, $b = 1$. Another aspect that needs to be mentioned is that overfitting is more likely when considering high compression rates. At the same time, it is clearly visible that some combinations do reach high-performance values while providing a smaller amount of information. Consequently, not every detail is relevant that is processed with the full complexity of the sensed image. While moderate information pruning reaches similar performances to the baseline, even edge cases in terms of compression reach performances $> 80\%$. If those techniques do have an impact on the training and inference time is investigated next.

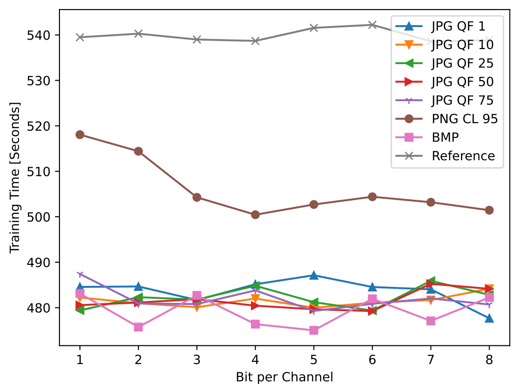
Evaluation on the basis of the training time: Especially the training phase in deep learning is computationally expensive and often time-consuming. This frequently leads to either a horizontal or vertical scaling of the hardware,



(a) RSI-CB256@VGG16



(b) RSI-CB256@ResNet50

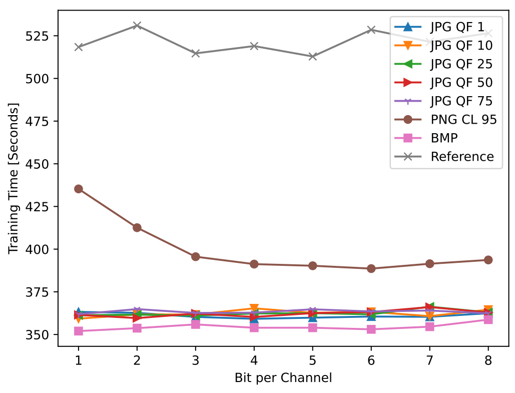


(c) RSI-CB256@MobileNet

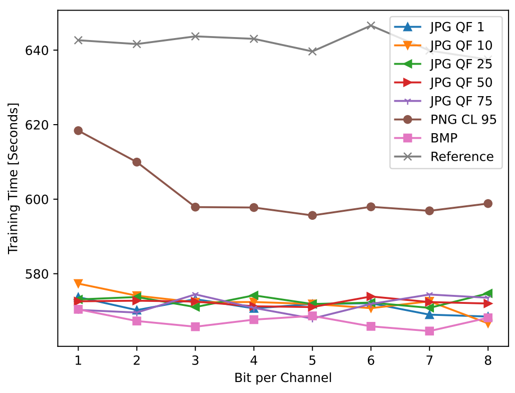
Figure 4.12: The result of the training from the dataset RSI-CB256 with respect to the **runtime**. This figure has been first published in [P11].

which can cause high costs. Smaller models reduce the computation time, therefore, the scope of this part of the work is whether compression of the input also rescues the time needed for the training. While the time has been measured for all combinations used before, it is not relevant which hardware is taken. More important is to compare the length of the relative time frame against each other. Consequently, the scope is how many percent of the training time can be saved by simply compressing the images by either quantization or image file formats.

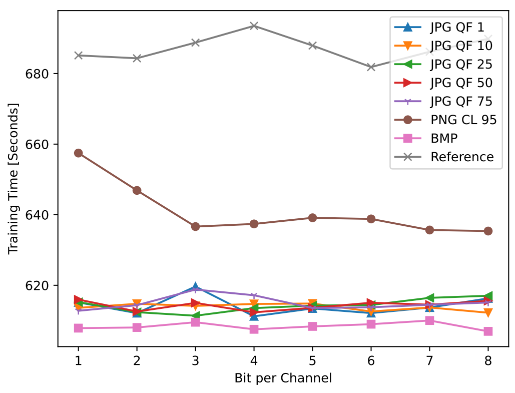
The measured training time for the datasets RSI-CB256 visualized in Figure 4.12a–4.12c is again investigated first. Considering the results from VGG16, one can see that the reference baseline that used the format TIFF was the slowest for all quantization rates with a needed time between 400s and 420s. Compared to this, the file format PNG took 350s when $b = 1$ and



(a) EuroSAT@VGG16



(b) EuroSAT@ResNet50



(c) EuroSAT@MobileNet

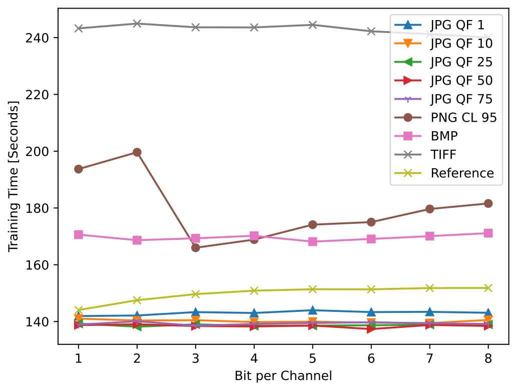
Figure 4.13: The result of the training from the dataset EuroSAT with respect to the **runtime**. This figure has been first published in [P11].

310s if $b = 8$. Due to the lossless compression of this file format, it is expected that the classification results will be similar then to the baseline, but it is interesting to see that the training time is noticeably lower. While this does not make a slight difference for a single run, this has a large impact when training n models with a genetic algorithm or a brute-force method to find suitable training parameters. Moreover, when analyzing the results for JPEG (and BMP), one can see that the time has been minimized to a range around 300s, which is a time reduction of $\approx 25\%$ on average. This situation is similar for ResNet50 and MobileNet, even when the time reduction is smaller than in the case of VGG16 and the training times differ from each other.

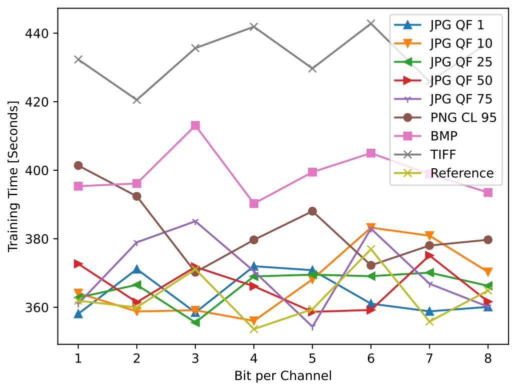
The results for the EuroSAT dataset, visualized in the figures 4.13a–4.13c, show schematically similar patterns than in the case of RSI-CB256. Considering VGG16, the reference model that uses TIFF is the slowest, with approximately 525s on average for the training. It is not surprising that the needed time differs from dataset to dataset due to the different specifications. While PNG $CL = 9$ reaches a better performance than the baseline, BMP and JPEG are much faster in all quantization levels, specifically, the training time has the bounds $[350s, 375s]$. This situation is similar for the ResNet50 and MobileNet, even when the time reduction is smaller than in the case of VGG16 and the training times differ from each other.

The results for the dataset, visualized in the Figures 4.14a–4.14c, underlines the statements from above with their similar patterns and their differences. While the other datasets took the TIFF format as a reference, this dataset has been stored with JPEG with $QF = 100$ as a baseline that needs 2 616 MiB on disk. Furthermore, the training took 152s on average over all quantization levels in the case of VGG16. Additionally, the reference configuration employed $b = 8$ reached an accuracy of 90.7%. At the same time, TIFF with $b = 8$ reached 90.8% in approximately 240s training time, PNG with $CL = 9$, $b = 8$ landed at 90.13% in 182s, and JPEG using $QF = 75$, $b = 8$ ended up with 90.7% in 139s. Those examples, comparing the results of $b = 8$, show that the assumption from above is that not all information still holds. Moreover, not only the performances reached similar values, the embedding into compressive file formats reduced the training time by a large factor. Especially considering the worst-case scenario of TIFF, JPEG with $QF = 75$ reached a similar accuracy but reduced the training time by $\approx 42\%$. While this is similar to MobileNet, The results for ResNet are very unstable; the reason for this case can only be assumed. Additionally, this is in line with the results related above, where the accuracies from AID@ResNet50 also showed unstable patterns.

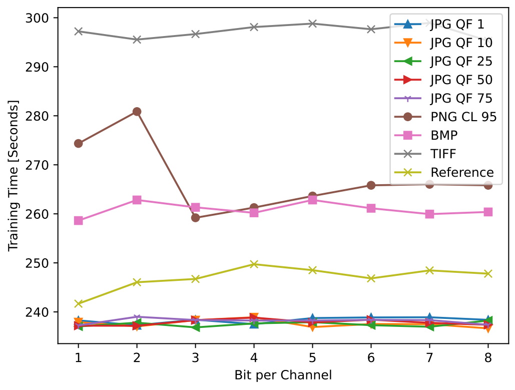
As a consequence of those results, it can be said that compression can significantly reduce the time that is needed to train a model. Especially when embedding the images into JPEG with high-quality factors maximized the performance with low to no costs in accuracy. This goes in line with the comparison based on the accuracy, therefore, this shows that not every piece of information that is provided by the total complexity is helpful to reach a good model, instead, with all details leads to a performant training process. Moreover, the utilize of JPEG with $QF = 75$ is a compromise between pruning information with its costs of losing accuracy.



(a) AID@VGG16



(b) AID@ResNet50



(c) AID@MobileNet

Figure 4.14: The result of the training from the dataset AID with respect to the **runtime**. This figure has been first published in [P11].

4.1.2.6 Summary

In this part of the work, we have investigated different image compression methods and the if they impact the training process of CNNs in terms of time and accuracy. More concretely, the image has been first quantized to a (smaller) target bit depth, followed by an embedding into (compressive) file formats. Then we trained multiple CNNs and measured accuracy and runtime on GPU. Finally, results are compared with respect to the compression (quantization and file format).

Three CNN types (VGG16, ResNet50, and MobileNet) have been selected and trained using three different remote sensing datasets. As mentioned above, the color space has been reduced to a range of target bit depths with quantization followed by embedding into various file formats (TIFF, BMP, PNG, JPEG). The benchmark of each combination proved that the time needed for the training could be reduced by compressing the images while paying the costs with accuracy. For example, while TIFF reached high classification performances, the training process is notably slower than, for example, taking PNG, which reached a similar accuracy but with shorter training. Even faster has been JPEG with all considerable quality factors, while very high compression factors minimize the training time at the expense of accuracy, it seems that JPEG with moderate compression is a good computerize due to its performance and nearly no loss classification precision.

It can be said that not every piece of information within an image is relevant for a classifier such as a CNN. In fact, the compression of an image highlight more prominent features and characteristics such that they support neural networks. Moreover, the time needed for the training can be minimized while losing only a comparatively small percentage in accuracy. In consequence, a scaling of the hardware might be preventable by pruning the unimportant information of the image. Additionally, the focus has been set on the training phase, Section 4.2.1.4 investigates the impact on the power consumption and performance during inference on an FPGA.

4.2 Examples of Algorithm Compression

The second component of each machine-learning pipeline is the operation that is applied to the data. While the discussed theoretical foundation is valid for algorithms in general, the scope in this section is set on machine- and deep learning models. The goal of this part is to provide samples that show the advantages and consequences when compressing this component. An application sample that is provided in the following deals with maximizing the throughput of a communication channel between a satellite and a ground station. In other words, a satellite is constantly orbiting the Earth, and its data is employed to detect wildfires. Due to the fact that a fire can be considered an outlier, the satellite transmits a large quantity of information that is more or less not useful. When applying deep learning on-board, a model can decide whether a sensed scene is relevant or not. In the following, only data is transmitted which are marked as critical.

The following furnishes two samples of how algorithms can be compressed to support their performance. The first application is the change of a trajectories representation during computation which leads to suppression of the quadratic nature of trajectories. The second application sample that is provided is the above-mentioned example which uses compression to minimize the footprint of AI space scenarios.

4.2.1 Minimal Footprint AI for Space Application

This section is based on our publications [P11], [P10], [P9], [P6], [P7], [P13], [P12], [T1], [T3], [T2] and is structured as follows: An introductional motivation is provided first, followed by the related work in Section 4.2.1.2. Furthermore, Section 4.2.1.3 divides the machine learning in space use cases into several scenarios. Section 4.2.1.4 describes then in-depth the scenario *Heavy Image Classification* and has been published within [P11]. Finally, Section 4.2.1.3 concludes this section with a summary.

4.2.1.1 Motivation

Nowadays, satellites are used for a large about of day-to-day operations that are common. One employs the navigation of smartphones to reach a particular place. Other examples are the television or the observation of the Earth's surface. While those tasks are common, a problem that occurs when using satellites is the remote communication that is required to transmit the signal from orbit down to a ground station. Furthermore, this communication channel has a limited bandwidth that can be considered. Additionally, the transmission tasks a specific time and has, therefore, a specific amount of delay that depends on the height of the satellite's orbit. This is especially a problem for tasks that are time critical, whether it is a device intrinsic failure on extrinsic motivation. The utilization of machine learning onboard satellites can reduce the demand for manual action it is required, such as navigation, in the case of the risk of collision with another object. Moreover, in the field of Earth observation, there is also the need to solve time-critical tasks automatically to enable semi-real-time data-driven processing. A practical example is the detection of wildfires, which are near to an outlier when considering the massive amount of entire produced data. While currently, all the data needs to be transmitted to the ground station over the speed-limited downlink, the use of machine learning algorithms on the satellite itself can decide whether a scene is relevant and only select data from transmission if a suspicious area is detected. In this way, the amount of data that is required to be transferred and investigated is actively limited already in orbit, which does lower the resources that are demanded on

Earth.

When it comes to the deployment of machine learning algorithms on a satellite in space, the use of FPGAs gets more and more attention, especially by the industry [2], [3]. Nevertheless, the use of such edge devices has some challenges that need to be solved. One is the limited memory size and the processing capability, especially when it comes to inferring deep learning models. Additionally, the power consumption, as well as the heat production that is caused by the computation, is essential to be considered as well. Moreover, in terms of deep learning, there is only a tiny portion of datasets that fulfill the requirement of being non-commercial, non-classified, and accurately labeled. This is a problem when training models for a specific purpose, as it is the case in the wildfire demonstration that has been mentioned above. Furthermore, the task of computing the weights for models can be expensive. Therefore energy consumption might be a problem for both the training and inference phase. Another challenge is the constrained computing environment of a satellite in space, which is the situation that the energy is collected with solar panels. Additionally, the heat that is generated by the components needs to be transferred away to avoid overheating. The cooling of the components is a non-trivial task within a vacuum.

Due to the limited research that has been done, in research as well as in industrial environments, there is the demand to define requirements that are needed for deployment but also for the performance evaluation on-device. The reference implementation is required that is deployable on the most common hardware devices. The tasks that can be solved using machine learning on board a satellite are very diverse and cannot be generalized. Therefore, this section provides the definition of several scenarios, where selected ones are investigated in detail, namely *Image Classification Heavy*.

4.2.1.2 Related Work

There is less to no particular research which covers the exact case of deploying deep learning models on board a satellite that orbits the Earth. However, when splitting the entire specific scenario into its smaller subareas, each is either investigated in detail or the attention is rising. Therefore, this part focuses on the aspect of deploying models on FPGAs and the detection of specific elements within remote sensing images.

The deployment of deep learning models on FPGAs raises attention and has some non-trivial challenges to solve. The authors one [2], [3] split the deployment into several scenarios, which are related to the computational load that is on the device itself when performing the computation. This scenario, namely *image classification heavy*, split the state-of-the-art convolutional neural networks into two groups. For example, a representative of the firstly mentioned group is ResNet50, and MobileNet for the other one. In contrast to this, the authors of [83] provide a framework that is able to compile deep learning models that are trained by another framework such that it is able to run on an FPGA with the use of DPUs. The limitation is that some layers are not supported, which might require that the architecture is re-implemented.

4.2.1.3 Scenarios

As described above, the deployment of machine learning and deep learning algorithms to devices that are located in constrained environments, in this particular case, the computation utilizing on-board hardware on satellites in space, is needed to consider the problems and costs that come with the environment. Therefore this section defines reference use cases for each individual scenario to split the problem into smaller junks. The reason is that it is non-trivial to define generalized requirements that fulfill the needs of all cases, thus the detection of anomalies in telemetry data or the task of segmenting active fire in remote sensing images. While the ideal case would be that each scenario type includes a baseline implementation as well as a reference dataset, the focus is the classification and segmentation of image data. Additionally, all use cases are deployed to an FPGA, namely a Xilinx Zynq Ultracsale+ MPSoc ZCU102, which has been chosen because of the support of multiple common hardware acceleration frameworks (VitisAI, FINN, and MATLAB's Deep Learning HDL toolbox). If not mentioned differently, each reference implementation of a baseline is trained on GPU, compiled/deployed with VitisAI, and inferenced on FPGA. Moreover, to evaluate the performance, the power consumption of the board, the accuracy of the model, and the network's throughput are considered. The scenarios are described in the following.

Anomaly Detection – Light: When it comes to space unmanned space missions as the operation of satellites, a failure of hard- or software systems can be critical. Additionally, due to the time delay during the transmission and the channel's bandwidth, the industry is currently researching to detect failures as well as anomalies automatically with the consideration of machine learning algorithms. One type of data that is relevant in this task is the telemetry data, which is also the scope of this scenario. A challenge that occurs when using this type of data is its large quantity, which is caused by the permanent satellite's movement in orbit. The authors of [123] describe that multivariant labeled telemetry data can be classified with the employment of an LSTM model, which serves as a basis for this scenario. Additionally, the baseline is created with the NASA Anomaly dataset that contains labeled data from MSL and SMAP devices.

Radio Classification: Various satellites do send and receive various types of signals; therefore, one task in machine learning deals with the type prediction of unknown signals. While this is one example, this use case is about maximizing the capacity and performance of dynamically shared channels. For this purpose, different types of neural networks can be taken to predict the signal's type and its modulation of a certain channel. During the MLAB project, the Open RadioML Synthetic Benchmark dataset is used for the benchmark. The technique to classify the signals is done in line with the method described in [124], which describes that multiple residual blocks are connected to a prediction layer with a softmax activation function. Additionally, the authors of [125] state that when transforming the IQ plane to a vector representation, deeper layers can be simulated more precisely.

Image Classification – Heavy: This scenario focuses on the task of detecting whether an object/characteristic is within the image or not. This class is assigned to an entire image instead of classifying individual pixels, as it is done in segmentation. A CNN can have different shapes and depths. While smaller versions, for instance MobileNet, are able to run on edge devices, large architectures like VGG or DenseNet include a high number of trainable parameters that leads to a need for higher computational resources. The focus of this scenario is set on the latter one, which is defined here as *heavy* networks in relation to their computational effort required for training and inference. Additionally, this use case does only consider the three $\{R, G, B\}$ color channels.

Image Classification – Light: This scenario is equivalent to the previous one, with the significant difference that only architectures are taken that are designed to be applied on edge devices, such as MobileNet. The tiling of the use cases is done because it is supposed that tiny convolutional natural networks that have a limited number of parameters need fewer computational resources and, therefore, a diminutive amount of energy.

Image Classification – Multispectral: While the last two scenarios covered only images with three color channels, this is about to handle multispectral satellite images, like Sentinel-2, which provides 13 bands. To detect the content, it is necessary to tile the large images into smaller patches; based on the last example, an area of 64×64 pixels would represent an area of 0.4096 km^2 on the ground (considering a resolution of 10 m per pixel).

Additionally, the smaller sub-images are then categorized into N classes; in the case of wildfire, the classes could be *fire*, *burnt area*, and *smoke*. This dataset is then classified using a convolutional neural network. It needs to be considered that pre-trained weights from computer vision datasets do often provide only three channels; therefore, the pre-training is essential to be in one tile to the constraints. Finally, the class of a certain region within the source satellite image is defined based on the smaller tiles.

Image Object Detection: The task of object detection is to check if a dedicated object exists within an image and define its location by placing a bounding box around it. When considering CNNs, one option is to take the YOLO architecture, which has been chosen in the experiments described in [P10] and covers the scenario of the MLAB project. Furthermore, the Airbus Aircraft dataset has been chosen for this purpose, additionally, the accuracy is measured in this case with average precision.

Image Segmentation: While object detection places bounding boxes around, segmentation is about classifying each individual pixel that is assigned to a class. Therefore this scenario describes the task of semantic image segmentation. U-Nets are part of the family of convolutional neural networks and have been chosen for this task with ResNet50 as a base model. As before, the Airbus Aircraft dataset is used but with the common intersection over union (IoU).

Those scenarios serve as a collection of use cases that are considered relevant when it comes to deploying machine learning and deep learning algorithms to devices in the space environment. While each of the use cases is important to investigate, the scope of this section is set especially on image data. Therefore, the rest of this section focuses only on a single scenario, namely the Image Classification Heavy. Those selections do cover the two main classification problems, that is, a pixel- and patch-based prediction of the class membership.

4.2.1.4 Scenario: Image Classification Heavy

The classification of images tries to predict whether a certain is contained within an image or not without stating its location. Therefore an image is labeled based on its content. One example could be the prediction of different land use or land cover types within a tiled satellite image. The scope of the heavy classification type is to consider convolutional neural networks with a large number of parameters that need to be computed during the training process. Deployment of those models onboard, for example, satellites, can decrease the information that needs to be transmitted over the channel to the base station. A use case for this is, for example, the detection of wildfires where only images are transmitted to the ground, which has a risk of containing affected regions. Therefore the scope of this scenario is to investigate large-scale models and their deployment to FPGAs with respect to their prediction performance as well as their time and energy consumption.

The first step is to train models on different datasets considering a wide range of networks. To keep things as simple as possible, a dataset is selected to be investigated in detail. The next step is to quantize the models of the selected dataset, which is required for the compilation of the target platform. The last step is to measure the performance and other characteristics on the FPGA itself

Datasets: The following provides an overview of the datasets that are considered in this part of the work. While some of the data is already described in detail in the application sample furnished in Section 4.1.2, this will detailedly cover only the additional sets. Additionally, Table 4.3 provides all essential key features from all datasets.

Resisc45: The Resisc45 dataset has been provided by [126] and covers the problem of patch classification in the field of land cover and land use detection. The dataset delivers 31 500 satellite images that have a size of 256×256 pixels and are categorized into 45 classes. Moreover, each image has a bit depth of 24 bits and includes the three $\{R, G, B\}$ channels. As is the case for the EuroSAT dataset, the dataset is not geo-referenced, therefore, a spatial split into subsets cannot be done, which leads to the problem that

Table 4.3: Overview of the datasets.

Name	No. Images	classes
AID	10 000	30
EuroSAT	27 000	10
RSI-CB256	24 956	35
Resisc45	31 500	45

non-overlapping in relation to the spatial position cannot be ensured.

Training on GPU: This part of the work describes the training process, its result, and the quantization of the neural networks. Anyways the above-described publicly available datasets, except the wildfire dataset, are considered within this scenario. Those are trained on a wide range of different datasets, including the architecture families VGG, ResNet DenseNet, and MobileNet. Each network is trained without pre-computed weights as well as using the ones from ImageNet. Furthermore, there is the requirement that each model needs to have an accuracy $> 80\%$. Otherwise, it's not considered and, therefore, not listed in the tables. Additionally, some models are usually not pre-trained on ImageNet, such as ResNet34 and DenseNet161.

Table 4.4 states the detailed results for all datasets and architecture types. One can see that in the case of AID, the table only includes a few classification results that are above 80% and those very close to the lower boundary. In contrast to this, the EuroSAT dataset reaches results higher than 90% for half of the networks in both cases, not pre-trained and using ImageNet weights. Especially the pre-trained version of the VGG and ResNet family has a high performance. This is different for the RSI-CB256 dataset. While the ImageNet case has very high accuracies above 97%, the other case does perform partly poorly in terms of not reaching the required lower bound. The Resisc45 dataset performs even worse, while the not-pre-computed use case has only two architectures that reach the lower bound. The other case has a performance that is on average lower than, especially, the one from EuroSAT. All in all, when considering both cases, with pre-trained weights and not, the EuroSAT dataset has the most stable performance. On the other hand, the RSI-CB256 dataset performs best when only considering the ImageNet case.

Table 4.4: Overview of the accuracy of the datasets and models that have been trained. Due to the requirements of the scenario *heavy image classification*, only prediction results are included that reach an accuracy above 80%. The table has been adapted form [T2].

Model	AID		EuroSAT		RSI-CB256		Resisc45	
	None*	ImageNet*	None*	ImageNet*	None*	ImageNet*	None*	ImageNet*
VGG16	–	–	85.85%	96.06%	87.80%	98.70%	–	88.69%
VGG19	–	88.06%	86.64%	95.18%	86.44%	97.62%	–	–
ResNet50	–	–	88.64%	98.13%	96.60%	99.21%	80.14%	91.94%
ResNet101	–	–	94.83%	98.12%	93.80%	99.62%	–	90.58%
ResNet152	–	–	92.61%	91.53%	96.04%	99.40%	–	92.20%
DenseNet121	75.73%	86.36%	95.48%	86.36%	–	98.06%	–	84.65%
DenseNet169	78.13%	87.21%	95.70%	87.21%	–	96.79%	–	82.79%
DenseNet201	75.60%	85.22%	94.96%	85.22%	–	98.49%	–	86.87%
MobileNet	–	86.29%	93.01%	94.58%	–	97.84%	93.01%	94.58%

(*) Indicates on which dataset the model has been pre-trained.

While the goal is to measure the performance of the networks on FPGA, this requires the deployment of the models employing two steps. First is the quantization of the networks, where the models are compressed to ensure a light weights version, which is required due to the limited resources on the FPGA and the characteristics of the platform. The second step is the compilation of the target platform's source code type. Only then can the FPGA predict the labels for the data items with the considering of DPUs. The scope is to select a dataset that performs well across both use cases and all networks. While the RSI-CB256 dataset is best suitable in the case of using ImageNet weights, unfortunately, the other case does perform poorly. Therefore, the EuroSAT dataset is selected to further investigate the impact of the quantization process.

All models from the selected architecture are quantized from their complete precision to a target complexity of an integer using 8 bits. Table 4.5 states the prediction performance before and after this process. Similar then before, only accuracies below 80% are not considered but are listed in the table. One can see that the performance decreases after quantization, which is because of the limited complexity which is available to represent a single weight. In comparison, some networks are quite stable, which as VGG16, which has an accuracy loss of 1.55%, other networks do lose significantly on performance, such as ResNet101 without weights, which has a loss of 81.97%, or MobileNet with ImgeNet weights, which has a loss of 41.36%. Additionally, DenseNet169 with the weights from ImageNet, delivered an error during the deployment process and, therefore, could not be quantized.

To keep things easier and to avoid unnecessary information, the evaluation of the FPGA itself is done on a selection of those networks. Additionally, to be able to compare the results to the ones from the application sample in Section 4.1.2, only the networks use the pre-trained weights from ImageNet. Furthermore, the selections should be diverse in their architectural depth but also should differentiate significantly by the number of weights.

Image classification on custom hardware: The performance of a CNN consists not only of the accuracy of the prediction result; instead, it is necessary to consider other aspects, such as the data throughput within a specified timeframe and the size of the model. In line with the application sample in Section 4.1.2, the choice of the compression of an image has a considerable impact on those aspects. Moreover, next to the speed of the neural network, the type of hardware accelerator and its features also play a major role in

Table 4.5: Results from all networks that have been trained on the dataset EuroSat. Each network has been trained without any weights and with pre-trained ones from ImageNet. Additionally, the label shows the results before and after the quantization process that is required from the deployment to the FPGA hosted on a Xilinx ZCU102 evaluation board using VitisAI. Each weight has been quantized to an integer with eight bits. The table has been adapted form [T2].

Model	After Training		After Quantization	
	Not pre-trained*	ImageNet*	Not pre-trained*	ImageNet*
VGG16	85.85%	96.30%	85.95%	94.75%
VGG19	86.64%	97.30%	85.19%	96.96%
ResNet34	94.69%	–	64.62%	–
ResNet50	88.64%	97.20%	81.68%	94.79%
ResNet101	94.83%	94.46%	12.86%	92.32%
ResNet152	92.61%	93.52%	86.02%	92.84%
DenseNet121	95.48%	95.77%	93.53%	91.14%
DenseNet161	96.69%	–	95.80%	–
DenseNet169	95.70%	94.35%	94.84%	–
DenseNet201	94.96%	93.83%	94.64%	59.09%
MobileNet	93.01%	95.26%	90.20%	53.90%

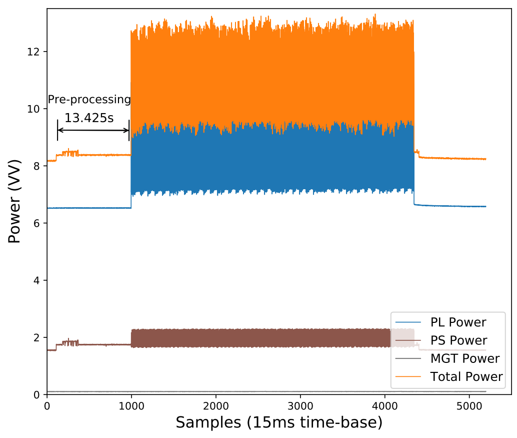
(*) Indicates on which dataset the neural network has been pre-trained.

the entire system. One form is, for example, the energy and time that is needed to preprocess the dataset, followed by predicting the labels. While those measures have an economic impact on computation on consumer and non-consumer grade GPU devices, the consequences are more relevant when performing on-board training or inference on satellites or other devices that are highly constraint. Therefore, the next part focuses on inferencing datasets on an FPGA, namely an ultrascale+ architecture with a 10 digit number hosted on a Xilinx ZCU102 evaluation board, to investigate the impact and opportunities of data compression in line with results from other application experiment examples. In summary, the objective is to take the results from the given experiment (on GPU) to evaluate the performance of the FPGA during the runtime of the end-to-end inferencing program.

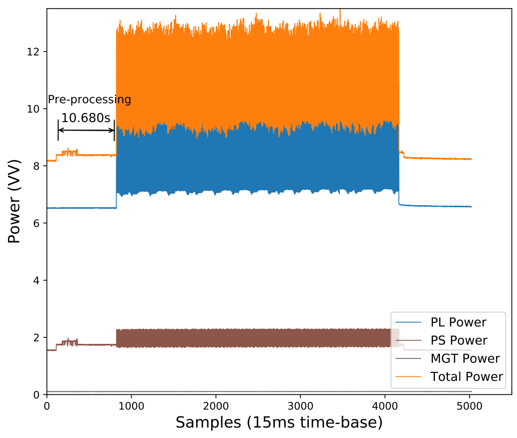
To avoid unnecessary complexity, it makes sense to only run the performance tests on a single dataset and a small number of configurations (MobileNet, depth: 8-bit, format: JPEG–$QF = 100$). The selection is made on the basis of the listed performances as well as in line with the given results from the previous experiment. Therefore, while the accuracy of the model still is relevant, it is of interest to make a selection with respect to the robustness of a model over all available bit depths. Additionally, it is required to maximize the difference in the time needed for the training between the baseline and the other tested image formats. This should ensure that the performance impact on FPGA is clearly visible, constant, and stable across multiple inference sessions. Moreover, a diminutive number of parameters and a smaller depth should move the spotlight to the data transfer rather than on the computation of the feature maps. Because of this reason and to be able to compare the performance to the already available results, it makes sense to focus on the smallest models from the VGG (VGG16, 153.7M parameters, 16 layer) and the ResNet family (ResNet50, 25.6M parameters, 107 layer). Furthermore, considering the listed criteria, the EuroSAT dataset seems to be best suitable for the evaluation.

Compared to the inference on GPU, the next step is to deploy the selected models to the FPGA, where the quantization of the models is the first task, followed by the compiling to the target platform. Anyways the quantization process using the Xilinx native VitisAI framework requires only the operation of layers within the models that are supported, which leads to the requirement to re-implement the networks of interest. In this particular case, the complexity that is available to represent a single weight after quantization is an integer with eight bits. This may cause a drop in accuracy before and after this process. Table 4.5 states the performances before and after the quantization; for completeness, the accuracies for all implemented models are listed. During the compilation of the target platform, three DPUs have been configured for the inference phase. Furthermore, a random set of 1 000 images are selected from the entire dataset to be referenced on FPGA.

Embedded devices as FPGAs as well as other embedded systems with the ability to train and inference neural networks, are designed to consume a low amount of energy. The exact level that is needed to predict a label is also influenced by the preprocessing steps, but also by the model itself. Moreover, there are three relevant rails that have the main impact consumption of the system and therefore need to be taken into account to evaluate the total power consumption: The first is the consumption of the programmable logic (PL) components of the FPGA power management, the I/O Rail MGT represents the consumption from video signals and codecs. The last rail that is relevant for power measurement is the processor system (PS). The global power of consumption of the considered components within the FPGA is the sum of



(a) Reference (TIFF)



(b) JPEG QF50

Figure 4.15: Visualization of the energy consumption during the inference of 1000 samples from the dataset EuroSAT using VGG16 and a **single thread** on the FPGA. This figure has been first published in [P11].

the rail's energy.

The experiments in Section 4.1.2 concluded that the use of JPEG is able to reduce the training time of GPU. Herefore, the reference file format needs to be compared with JPEG, and a $QF = 50$, has a balance between compression rate and performance, mainly because there is only a small difference between applying a quality factor of 75 and 25. Additionally, the first case is to investigate the computation with a VGG16 and a single DPU, and one thread. The total power consumption, as well as the energy level of the single thread, is visualized in Figure 4.15 samples with 15ms, where the left diagram represents the baseline with TIFF as a file format. In general, the FPGA has an idle power which indicates the consumption level without a load and can be seen in both cases in the first few samples of the line for the total power. After a few milliseconds, the program is started, which is indicated by the energy jump to approximately 8.1–8.3W. The first task within the program is to read all images into its main memory; this is done on the CPU, therefore, the load of the PS power rises, and the total consumption is above its idle level. The next step starts with index 995, where the DPU starts to estimate a label for each image. During this phase, the energy consumption increases significantly for the PL Power and consequently for the total value. The prediction phase reaches a total power consumption level of 13.31W on maximum. While each

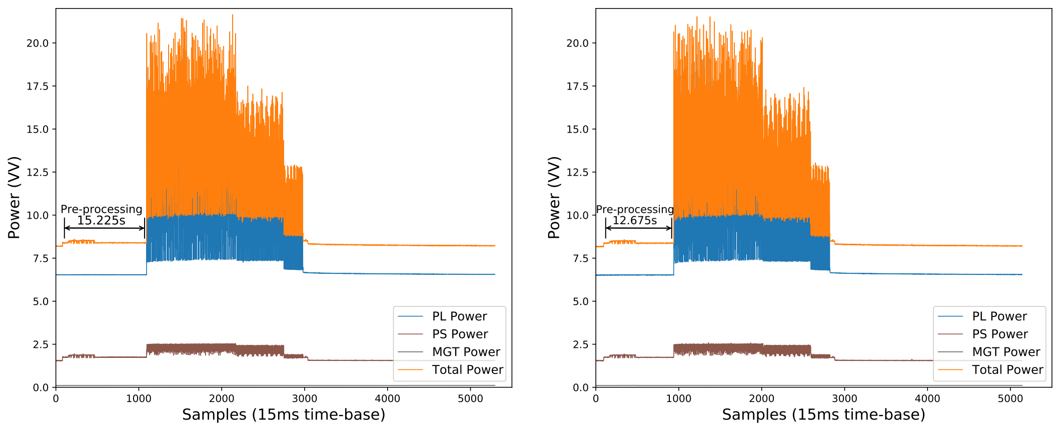
Table 4.6: Runtime on FPGA of the inference program. This table has been first published in [P11].

Model	Threads	Pre-processing Time	Total Time
TIFF@VGG16	1	13.425s	63.720s
JPEG@VGG16	1	10.680s	60.945s
TIFF@VGG16	4	15.225s	43.575s
JPEG@VGG16	4	12.675s	41.085s
TIFF@ResNet50	1	10.755s	25.830s

rail has different levels, one can see that the MGT power line does only idle, which is caused by the fact that the program processes images rather than any video signals. Anyways, considering the results from both file formats, the power consumption does not notable differ from each other. The reason is that when the image is read, whether it is from JPEG or TIFF, the complexity and dimension of the resulting matrices are equal. On the other hand, the energy might still be on the sample level, but when considering the time essential for the preprocessing, one can see that in the case of TIFF, 13.4s are needed to read the images, and 10.7s for JPEG. This leads to a time savings of 25.23%, which can have an enormous impact when processing large datasets, additionally, this is in line with the results from the experiments on GPU. The runtime and timespan for the preprocessing for all models and use cases are summarized in Table 4.6.

The use of a single thread leads to inefficiency during the computation, especially during the prediction phase. When increasing the number of threads to four, the total inference time is significantly reduced. On the other hand, the preprocessing time is comparable to the previous case (TIFF: 15.2 s, JPEG: 12.7 s), see Figure 4.16, which results to the fact that the factor for the impact of JPEG over TIFF increases. As a consequence, the right choice of threads and image file formats do have a considerable economic and computational impact on the program in terms of time and energy that needs to be invested.

In both cases, the single and multi-threading inferencing the use of different file formats did have a comparable considerable impact on the computation time. As a consequence, it needs to be investigated if the network architecture has an impact too. While VGG16 is an architecture type that includes 153.7M parameters, the skip connections of ResNet50 prevent the depth of the model and lead to only 25.6M. The results in Figure 4.17 depict that the effort needed for predicting the labels with ResNet50 with a single thread and TIFF is



(a) Reference (TIFF)   (b) JPEG QF50

Figure 4.16: Visualization of the energy consumption during the inference of 1 000 samples from the dataset EuroSAT with VGG16 and a **four thread** on the FPGA. This figure has been first published in [P11].

lower than using the same configuration with VGG16. This shows that the architecture choice in terms of their size required to be taken into consideration when computing on-board.

4.2.1.5 Summary

This part of the work investigated the feasibility of deploying machine learning and deep learning technologies to devices that operate in constrained environments, such as satellites in space. Due to the reason the different tasks of a satellite could not be generalized, several scenarios have been defined that should serve as a baseline for future operations within this field of interest. While the use cases are diverse in their domain, the scope of further investigating the scenarios has been set on the classification and segmentation of images.

The scenario, namely *heavy image classification*, focused on convolutional neural network architectures that have a high number of weights that needs to be computed during the training phase. First, four datasets were trained on multiple architectures and then quantized to be able to investigate the impact of the deployment process on the classification performance of the models. Afterward, the energy consumption and needed time for the complete inference phase were analyzed. Additionally, the data has been compressed using the image file format JPEG. As expected, the switch of the compressive file format

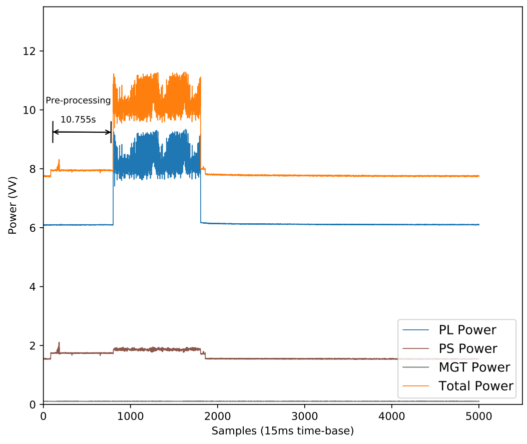


Figure 4.17: The energy consumption of an FPGA Xilinx ZCU 102 inference the file format TIFF using ResNet50 and the use of a single thread. This figure has been first published in [P11].

had no impact on the energy consumption during the inference phase, but the time to process the images beforehand was reduced by more than 25% at its peak without losing a large quantity of accuracy.

It is feasible to perform deep learning methods on edge devices as an FPGA, but with the limitation that the energy consumption of the entire inference system depends on the model that is selected. Additionally, another aspect for the energy consumption height is the shape and complexity of the fed data. While the original source data does provide high accuracies, compression using, for example, JPEG has a significant impact on the time that is needed to process the images.

In the future, it is recommended to perform the different scenarios on various target platforms. Additionally, it is required to provide a more extensive investigation of state-of-the-art network architectures.

4.2.2 Trajectory Similarity using Compression

This section has been published in our work [P5] and is structured as follows: An introductional motivation is provided first, including the related work, followed by a discussion about trajectory encoding into a sequence of discrete orientations in Section 4.2.2.2. Furthermore, a metric that is based on compression is introduced in Section 4.2.2.3 and evaluated in Section 4.2.2.4. Finally, the section closes with a summary in Section 4.2.2.5.

4.2.2.1 Motivation

The employ of trajectories is indispensable for modern computations of map generation [127] and urban analysis [128], but also in fields as biology [129]. An important task is to compare those trajectories toward their similarity, where the main computational problem is that the most central distance function, dynamic time warping [130] and Fréchet distance [131] to name some, have a non-linear time complexity. While there are some metrics that are optimized to suppress this problem by ignoring particular from the spatial data because of its precision, the most common ones are those with a high complexity degree. Furthermore, the ACM SIGSPATIAL GIS Cup 2017 tried to find a solution to this problem by focusing on the Fréchet distance only. The best submissions of this challenge used techniques e.g. simplification [132] or bounding box representations [133].

This part of the work tries to solve the quadratic complexity issue by taking compression as a distance metric. Firstly, simple geometry features, namely the orientation of a single trajectory segment, are generated and then embedded into a fixed-sized Bloom filter. While the normalized compression distance utilize normal compressors, like GZIP, to estimate the Kolmogorov complexity, the created filters serve as a lossy compression where the informational content is empirically evaluated. Furthermore, the joint Kolmogorov complexity of two objects is required to be able to compute the similarity of two objects in the case of the normalized compression distance. It is approximated by compressing both objects together. In contrast to this method, the proposed approach with Bloom filters is by uses the logical OR operator. This approach is evaluated on different trajectory datasets, including real-world GPS data, and a k-NN algorithm.

The following it is examples of how the geometry features from the trajectories are computed. The next part discussed in-depth the embedding of the features into the Bloom filters as well as the proposed similarity metric that is adapted from the normalized compression distance. Furthermore, the following evaluation provides an analysis and benchmark using four different datasets that cover, among other fields, GPS data and handwritten characters. Finally, a summary of the main findings is given.

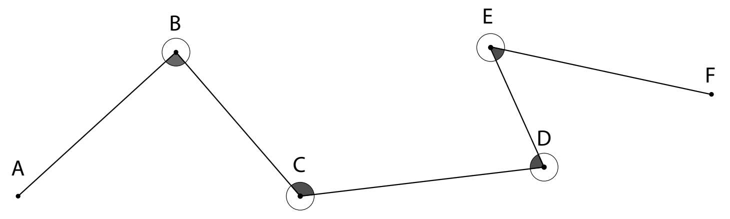


Figure 4.18: The local orientation is a feature that represents the trajectory by a sequence of direction changes. More concretely, each angle between two segments is calculated and encoded to a letter, for example, by dividing the unit circle into eight directions. This adds a rotation-, translation-, and scaling invariance to the data. The darker regions around sub-positions depict the angle of interest. Figure has been published in [P5].

4.2.2.2 Trajectory Representations as Strings of Discrete Orientations

A trajectory is a sequence of spatial location information within an n-dimensional space, a representative is, for instance, a GPS signal. Additionally, some other related characteristics, such as the timestamp for each single point, can be included as well. Therefore, a trajectory is built of linear interpolated vectors that can include related attributes. While one option is to sequentially list all coordinates as a list, another representation form is to remember the angles between the segments and encode them into letters. This would transfer the trajectory into a string of characters. Depending on the calculation of the angles and the encoding, this adds a translation-, rotation-, and scaling invariance to the data. If the size of the trajectory is still relevant, one can repeat a sequence of letter with respect to the segment length.

There are two options to calculate the angles between two segments that end up with different characteristics. The first one is to calculate the angles in a local context. Consequently, this represents the trajectory by a sum of direction changes, *turn left* or *turn right*. As it has been done in [98] reach trajectory has been discretized using the angle and the distance. The corresponding values have been assigned to a letter by dividing the unit circle into n elements, where each section is represented by a specific letter. Furthermore, each letter has been repeated with respect to the length of the segment. In this way, the representation is no anymore scaling invariant. A sample of this procedure is represented by Figure 4.18, which visualizes a trajectory and the letters that correspond to the angles that are relative to the previous segment.

In contrast to the local representation, The global orientation calculates the

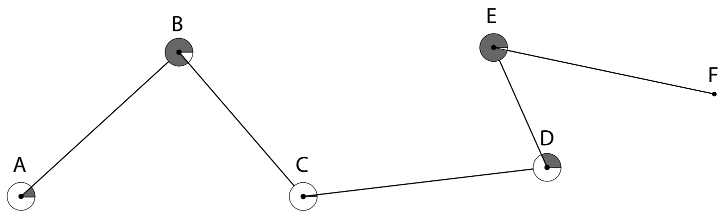


Figure 4.19: The global orientation is a feature that represents a trajectory by a sequence of directions with respect to the Euclidean space. More concretely, the angle of each segment is calculated individually and encoded into a letter. For example, by dividing the unit circle into eight directions. This adds a translation-, and a scaling invariance to the data. Compared to the local directions, this is not rotational invariant due to the encoded global aspect. The darker regions around sub-positions depict the angle of interest. Figure has been published in [P5].

angles based on the cartesian coordinate system rather than on the previous segment. Therefore, each letter represents a direction in space, *North* or *South*. Similar then before, all angles between 0 and 2π are quantized into N equal-sized parts, each section representing a letter. Figure 4.19 shows the same trajectory as before but with globally calculated orientations.

4.2.2.3 Compression Distance of Trajectories

This section proposes a method that applies the approach of the algorithmic information theory, more specifically NCD, to location data. It is a non-trivial task to find a suitable measure for the complexity of trajectories, especially not when it comes to joint complexity. While the NCD does apply normal compressors to compare two objects, this part of the work adopted the Bloom filter as a lossy compressor. More specifically, a trajectory was converted to a sequence of features, as their global orientation, and embedded into a Bloom filter. Instead of embedding a single character after each other, sets of words (n-gram) are injected into the filter to highlight the direction changes within the single sequence of location points. The use of this specific data structure enables the possibility to apply logical operations efficiently, which is useful when calculating the joint complexity. In detail, the proposed method can be split into three parts. First is the embedding into the Bloom filters. Second, to use the filter as a lossy compressor, and finally the adaption of NCD.

Bloom Filter and trajectory features: The first step of the proposed distance measure is to extract features and embed those into a Bloom filter, acting as a basis for the distance measure. While there are lots of different feature types that can be generated from trajectories, this work employs two types, namely the local/global orientation for each segment and the geohashes from the coordinates. Before actually embedding the features, the former characteristic is encoded into a set of symbols. When using a probabilistic data structure, a problem that might occur is information overlay, which is caused by a trajectory that visits the same grid cell multiple times or folds multiple segments that are pointing in the same direction. This can lead to the disadvantage of not providing enough spatial details to accurately measure the similarity between two trajectories, consequently having a large amount of miscalculation and, therefore, a bad performance. A solution to this problem is to take the number of occurrences for each letter into account. By appending the actual, this incrementing suffix to each letter adds enough information to prevent a large amount of overlay. For example, consider the trajectory that has been encoded to the letters $AGAH$, after adding the suffix, the letters are transferred into the set $\{A_1, G_1, A_2, H_2\}$. Note that a second A would have the suffix two, which allows the embedding of the letters without a loss. This process of creating the Bloom filters is represented in Algorithm 1.

Algorithm 1: Initialization of the Bloom filters and the embedding of generated features.

```
Input: A finite set $T = \{t_1, t_2, \ldots, t_n\}$ of trajectories
Input: bfLen: the size of the Bloom filter
Input: k: the number of hash functions
Output: A set of $n$ Bloom filters
1  for $i \leftarrow 0$ to $n$ do
2      $filter \leftarrow newBloomFilter(bfLen, k)$
3      $f_1 \leftarrow geohashes\ from\ t_i\ as\ set\ \{h_1, h_2, \ldots,\ h_j\}$
4      $f_2 \leftarrow orientations\ from\ t_i\ as\ set\ \{s_1, s_2, \ldots,\ s_{j-1}\}$
5      for $x \leftarrow 0$ to $j$ do
6          if $x < (j-1)$ then
7              $letter \leftarrow addIncrementalSuffixToChar(f_{2,x})$
8              $filter.add(letter)$
9      end for
10     $BloomFilters[i] \leftarrow filter$
11 end for
12 return $BloomFilters$
```

The use of Bloom filters to hold data brings some advantages, that is, the constant length. The filters are allocated with a length that is independent of the embedded data, leading to a suppression of the quadratic complexity of trajectories. Furthermore, due to this characteristic and the binarity logic, operations can be applied without the overhead. On the other hand, one needs to consider that the information distribution within filters does matter and has an impact on the ability to compare the filters.

Compression using Bloom filter: The Normalized Compression Distance (NCD) has already been explained in detail in Chapter 2.1.2.3, but in summary, this metric approximates a Kolmogorov complexity $K(x)$ utilizing the length of the compressed version of an object $C(x)$ that is generated by normal compressors e.g. GZIP or BZ2. Moreover, the relative complexity of two objects $K(x, y)$ is defined by jointly compressed objects $C(x, y) = C(x \cup y)$, where $\cdot \cup \cdot$ represents the concatenation of two objects. The calculation of this metric is computationally expensive because of normal compressors to approximate K. Previous projects namely [P1] did successfully apply this technique to remote sensing data, but the computational needs do limit the applicability in real-world scenarios.

A solution to the expensive computing of NCD is to replace the normal compressors with Bloom filters that are adapted to be considered as lossy compressors by considering the information distribution within each filter. A characteristic of the filters is that the number of zeros is inverse-dependent to the amount of uniquely inserted elements. This leads to a binary array of embedded features, where its information-theoretic complexity can be calculated with Shannon's entropy $H(x) = -x \log_2(X) - (1-x) \log_2(1-x)$. To be able to apply the entropy H as a similarity measure B like its done when using normal compressors, the complexity of x is calculated by $B(x) := H(FOZ(BF(x)))$, where FOZ (see Definition 2.11) is the fraction of zeros of a Bloom filter BF.

In relation to this definition, it is also necessary to introduce a measure for the joint complexity to approximate $K(x, y)$. This is done by taking the union of two Bloom filters, which is directly computable by computing using the logic OR operation. More formally defined, $BF(x \cup y) = BF(x) \vee BF(x)$, consequently the joint information-theoretic complexity can be calculated by $B(x, y) = H(FOZ(BF(x) \vee BF(y)))$. This measure fulfills all necessary definitions as the idempotency, such that $B(x, x) = B(x)$ that is increasing with the dissimilarity of the two given trajectories, and the symmetry $B(x, y) = B(y, x)$.

An advantage is that only Bloom filters for each individual trajectory are needed to compute the proposed measure. This result to a reduction of the memory footprint. Additionally, the employ of logic operations to estimate the joint complexity of $K(x, y)$ and the avoidance of using normal compressors such as GZIP or BZ2 cause a reduction of the computational complexity.

NCD using Bloom Filter: The definition of the information-theoretic complexity measures $B(x)$ allows the construction of the NCD with Bloom filters as lossy compressors. There is the limiting factor that the defined measure $B(x) = H(FOZ(BF(x)))$ is only increasing if the fraction of zeros of the affected Bloom filters is below 0.5. Higher percentages can lead to distortions in terms of getting negative distance values.

There are three solutions to get rid of this limitation: (1) is to increase the size of the filters. This has the consequence that it is required to recompute the filters for all trajectories. This increases the computational effort as well as the memory footprint, where the motivation is to reach the opposite. (2) is to directly use the fraction of zeros and skip the calculation of the entropy, such that $B(x) = FOZ(B(x))$. While this is not the scope of this work due to having intentionally faulty units, it cannot be said if this results in good results. One can see that the metric produce negative distance values if the union of two filters from individual trajectories leads to a higher number of individual features than it is for the separated filters. Therefore, (3) is to take the nominator's absolute values from the equation, and the Bloom Compression Distance can be formulated by Definition 4.3.

Definition 4.3. *Let NBD be the Normalized Bloom Distance that is built on top of the NCD. The calcuated by using an information-theoretic approach and estimates the dissimilarity between two objects x,y where their characteristics have been embedded into Bloom filters. The distance measure is defined as*

$$NBD(x, y) = \frac{\mid B(xy) - min\{B(x), B(y)\} \mid}{max\{B(x), B(y)\}},$$

where the range is $0 \leq NBD(x, y) \leq 1+\varepsilon$*. While a value of zero indicates two similar objects, the dissimilarity increases with the number to its maximum of one. Furthermore,* ε *is the error that is caused by sampling problems with small filters and distribution problems with its origin of using real-world hash functions with non-uniform collisions. If* $H(x) = H(Bf(x))$ *is fulfilled, NBD approximates the Kolmogorov complexity.*

Experiments using the defined NBD showed that (3) improves the results compared to the solution (1) and (2), additionally, the performances are improved.

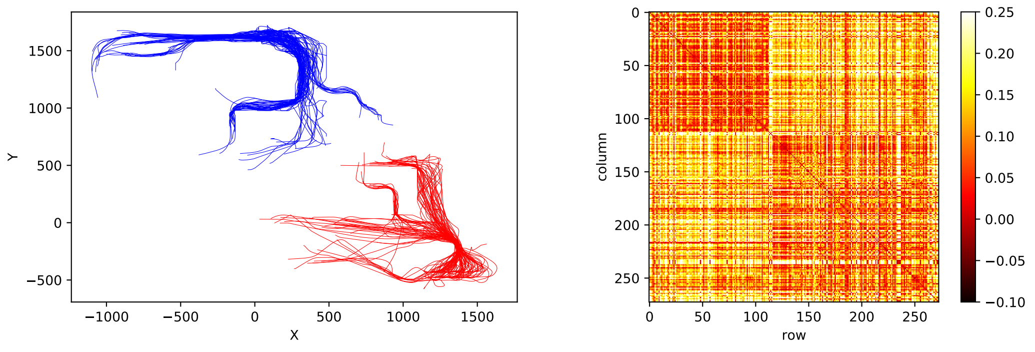
4.2.2.4 Experiments

The classification and clustering of trajectories can be a non-trivial task, where small changes and characteristics of single segments can have a large impact on the accuracy of a selected distance metric. The method that is introduced in Section 4.2.2.3 embeds the features of an individual trajectory into a Bloom filter that functions as a lossy compressor to approximate the Kolmogorov complexity. This section applies the proposed metric to classify the trajectories into categories. The experiment is divided into several tasks, preprocessing, filter generation, and classification.

Before being able to perform a classification, it is necessary to preprocess the given data. The first is to assemble the individual coordinates into segments. Additionally, a Douglas Pucker algorithm is applied to simplify the trajectories and avoid large jumps. Furthermore, traces that are below a threshold are removed from the dataset. The next step is to generate a Bloom filter for each individual trajectory. Therefore the selected features, namely *global directions*, that differed from dataset to dataset have been extracted and embedded into filters. As a result, each trajectory is represented by characteristics that are embedded into an individual filter. The last step within the processing chain is the classification itself, where the k-nearest neighbor algorithm is used to predict the trajectory's class based on the Bloom filters and the proposed distance metric. There are several parameters that need to be considered. Next to the number of neighbors for the k-NN it also the feature generation can be adjusted, for example, by defining the amount of directions. Additionally, the size of the Bloom filters does also have a significant impact on the final result. Therefore, those parameters are found by a permutation of a set of possibilities.

Several experiments have been performed to evaluate the normalized Bloom distance and its performance, to be able to rate the descriptive power. Datasets have been selected that represent different types of trajectories, such as characters of mobility, but also different problem tasks, like binary- or multi-class classification.

Results for Prague-Teams: The first dataset that is taken to evaluate the proposed Normalized Bloom distance is Prague-Teams provided by [98] and



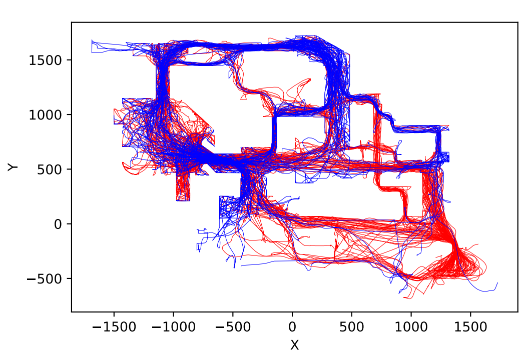
(a) Prague-Teams (b) Similarity Map

Figure 4.20: Visualization of the dataset Prague-Teams (left), where the colors indicate the team membership, and the corresponding similarity map (right). This figure has been published in [P5].

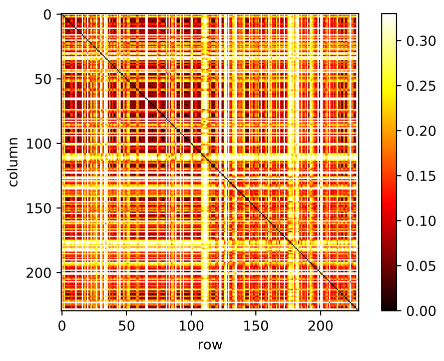
consists of 273 individual trajectories. The dataset has been generated with the computer game Urban Terror where five players played the map Prague. After the spawn of each player, the first 128 location points, which is equivalent to the first 6.4 seconds of gameplay, are sampled to a single trajectory. The dataset consists of two classes, corresponding to the team members of each player, where the distribution is 40/60. In conclusion, the task is to predict the team members of the trajectories. The dataset is visualized in Figure 4.20a, where the colors represent the teams.

The Bloom filters are constructed with a length of 256 bit and two hash functions. Due to the spatial separation of the data, it is necessary to exclude any location information from the generated features, therefore, the global directions have been selected using 16 different possible orientations. Those characteristics lead to a fraction of zeros of 47.51% on average and a memory footprint of 8 kib for the entire dataset. Consequently, a single coordinate pair is represented by a 2 bit.

To evaluate the metric, a k-NN classifier has been trained using the Normalized Bloom Distance and taking 19 neighbors into consideration. This led to an overall prediction accuracy of 91.59%. While the matrix factorization approach reached comparable results, the n-gram free version ended up with 81.03% [98]. In addition, the generation of a similarity matrix sorted by team membership, visualized in Figure 4.20b, shows clearly two separated regions that correspond with the classes. While a noise pattern is visible, it does not affect the classification.



(a) Prague Ego-Shooter



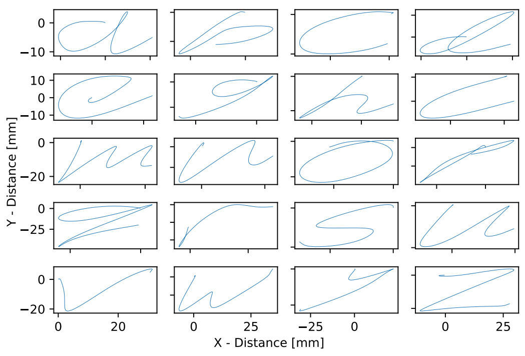
(b) Similarity Map

Figure 4.21: Visualization of the dataset Prague Ego-Shooter (left), where the colors indicate the team membership, and the corresponding similarity map (right). This figure has been published in [P5].

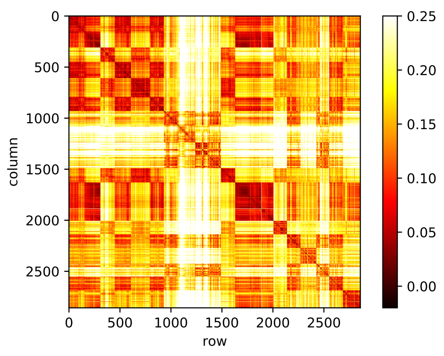
Results for Prague Ego-Shooter: The second dataset is called *Prague Ego-Shooter* and is created using the same game as before with the difference in its playing mode. This time several players tried to capture and hold an item on its position within the game. Additionally, due to the structure of the map, the generated trajectories are self-redundant. The dataset itself contains 275 trajectories, including 244 675 individual coordinates. Compared to the last dataset, the trajectory has been sampled using the players traced through each full lifetime cycle. The dataset is visualized in Figure 4.21a, where the colors represent the teams.

The configuration of the Bloom filters and the features are equivalent to the previous experiments, with the difference that each filter has been initialized with a length of 1 024 bit and three hash functions. Those characteristics lead to a fraction of zeros of 73.89% on average.

The evaluation of the metric using k-NN and the same scheme as before, with the difference that 13 neighbors have been taken into account, reached a prediction accuracy of 90.52%. Moreover, the similarity metric, visualized in Figure 4.21b, is interesting because it shows, on the one hand, a similar structure and noise to the last experiment but with significantly smaller differences between the classes. This might be because of the fact that each player respawns near the origin, and the target that needs to be captured leads to trajectories that contain a similarity. This makes the classification problem harder and, therefore, the similarity metric less intuitive. Furthermore, this classification task shows that the proposed metric is able to perform well on



(a) Characters



(b) Similarity Map

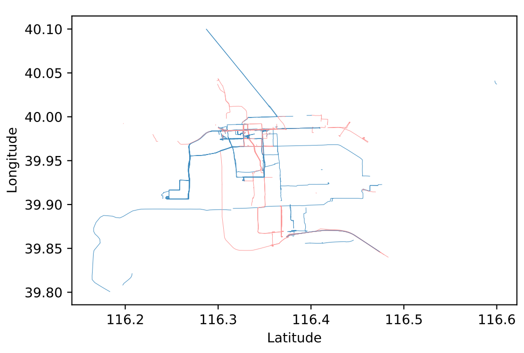
Figure 4.22: Visualization of the dataset Characters (left), where the colors indicate the team membership, and the corresponding similarity map (right). This figure has been published in [P5].

more complex two-class problems.

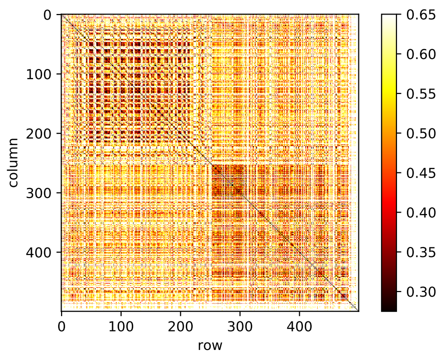
Results for Characters: The third dataset to evaluate the proposed metric is provided by the *UCI Machine Learning Repository* and is called Characters. As the authors from [134] state, it consists of the dataset of 2 858 trajectories that are traced from handwritten characters by only considering letters that can be written with a single stroke. Consequently, each trajectory is assigned to one of the 20 provided classes, where each holds between 125 and 175 individual items. The raw data has been smoothed using a Gaussian filter, therefore, the preprocessing of this dataset includes an additional step to reversing this step. A sample for each class is visualized in Figure 4.22a.

The Bloom filters are constructed with a length of 256 bit and two hash functions. The global orientations of a segment with eight different directions have been selected as features to be embedded. Those characteristics cause a fraction of zeros of 74.40% on average.

The evaluation of this dataset is done with k-NN and 15 neighbors that are considered, causing an accuracy of 81.52%. Surprisingly, the matrix factorization approach in [98] using the same parameters reaches a prediction accuracy of 74.2%. Considering the individual letter that is included in the dataset, one can see that some of them are structurally very similar to each other, such as V and U. This leads to distortion when predicting the classes and increases the complexity of this task. These artifacts are reflected in the corresponding similarity matrix, which is visualized in Figure 4.22b. All in all, this example shows that the metric is able to handle multi-class problems



(a) Geolife



(b) Similarity Map

Figure 4.23: Visualization of the dataset Geolife (left), where the colors indicate the team membership, and the corresponding similarity map (right). This figure has been published in [P5].

with slightly structurally overlapping patterns.

Results for Geolife: The fourth dataset is called Geolife, provided by [128]. This real-world dataset included mobility data tracked by 182 users equipped with GPS trackers and was collected between April 2007 and August 2012. Furthermore, the dataset consists of 24.8 million location points that correspond to 17 621 trajectories that are assigned to 10 classes. Due to its complexity, this work took a balanced subset of Geolife, specifically, the first 500 trajectories from the classes *Bus* and *Taxi*. The Figure 4.23a visualizes the used subset, where the color represents the class of each trajectory.

While the Bloom filters are constructed with a length of 1 024 bit and two hash functions, next to the global orientations using 16 possible directions, the geohash with a length of seven characters has been calculated or each location of a segment as a feature as well. Those characteristics lead to a fraction of zeros of 93.11% on average, where the high count related to the number of features and average trajectory length indicates a possible information overlay within the filters.

The evaluation with this dataset is similar to the other ones, with the difference that only four neighbors are considered for the classifier. The configuration leads to an overall accuracy of 82.00% for the subset. A consideration of the entire dataset could not be successfully performed, where a reason could be the distribution of the ones within the filter. The computed similarity matrix and sorted by classes, visualized within Figure 4.23a, shows clear patterns representing the different mobility types.

4.2.2.5 Summary

In this part of the work, a similar metric was introduced that is based on the normalized compression distance. The proposed metric adapted Bloom filters as lossy compressors to approximate the Kolmogorov complexity. To evaluate the approach, trajectory features, like encoded segment orientations, were generated and embedded.

An advantage compared to the normalized compression distance is that it is sufficient to have only the Bloom filters and, therefore, the data in a compressed format, which reduces the memory footprint as well as the computational effort. Especially the fact that the computation of the joint complexity is done with logic operators on the compressed data directly has a significant impact on its performance. Due to the nature of the simple features, the structure of trajectories, and the use of Bloom filters, a single location point were represented by a low number of bits. Therefore the information-theoretic metric is effective in classifying real-world trajectory data.

Furthermore, even with simple features namely the orientation of a segment encoded into a letter were able to reach baseline results from other publications. Moreover, compared to common trajectory distance measures for instance DTW and Frechet distance that have a quadratic complexity, the Normalized Bloom Distance suppressed this by replacing the normal compressors with Bloom filters.

The approach of replacing the normal compressors with a probabilistic data structure opens the possibility of efficiently computing the similarity between two objects. While the focus has been set on trajectories, it can be assumed that this is adaptable to a wide range of spatial objects.

The distance measure based on compression reached in the provided examples comparable results to using the complete available information, consequently, one can assume that the data does contain some details that is not needed in some scenarios. By removing the boilerplate, the algorithm's performance can be increased.

4.3 Example of Output Compression

The third component of a machine learning pipeline is the outcome of the operational algorithm, for example, the prediction result. Currently used generative AI models does generate text on the basis of user input, but it is not able to decline an input because of its incorrectness. Therefore the model generates an output even when the generated outcome is not trustworthy. This section focuses on the compression of the output in terms of deciding whether the result can be trusted or not.

4.3.1 Information-Optimal Abstaining

This section has been first published in our work [P4] and is structured as follows: An introductional motivation is provided first, including a related work. Next, Section 4.3.1.2 furnishes a brief summary about the text classification. Furthermore, abstaining is described in Section 4.3.1.3 and the ensemble of models in Section 4.3.1.4. In addition, while Section 4.3.1.5 describes how a dataset consisting of building polygons and Twitter texts are built, Section 4.3.1.6 evaluates the dataset using sparse text mining models. Moreover, the ensembled models have been investigated in Section 4.3.1.7. Finally, the model is concluded with a summary in Section 4.3.1.8

4.3.1.1 Motivation

Nowadays, the is a tendency that in the future, people will live in cities rather than in the countryside [135]. Nevertheless, urbanization does bring global problems and consequences [136]. There are two types of data sources that are able to monitor the behavior of urban development, namely social media and remote sensing. The fusion of these data types is a growing research field [103], where the challenge is to combine highly accurate satellite data and social media data. Some of the latter mentioned data has a spatial location attached within its metadata and is processed in the form of tagged photos [137], social media text [138], and mobility data [139]. When it comes to monitoring urban areas with the use of those combined data sources is the determination of a building function. While the remote sensing data provide parameters such as the footprint of the building, the social media texts can give insights about the usage. There is a wide range of building types that can be assigned. Two classes that are essential for society are *residential* and *commercial*, which build a large portion of an average city. A problem that occurs when only considering those two building types is the fact that not everything fulfills the criteria of being a member of this function type.

This part of the thesis focuses on the challenge of the feasibility of using a Twitter text collection to assign the function types of *residential* and *commercial* buildings. A problem that occurs is that it is expected that only a tiny portion of the tweets that are near a building has actually related to it. Therefore it is necessary to propose a classification system that is able to filter out data that is irrelevant. As a consequence, when it is expected that such a system filters most of the data, the model needs to be trained on a comparatively small portion of the dataset. Another challenge that might be faced is an unequal class imbalance. While the most significant part of a city consists of a *residential* area, it is expected that most tweets are about *commercial* buildings. Furthermore, there is research in the field of detecting anomalies and outliers [140]. Unfortunately, those scenarios do not fit the requirements of this research, instead, it is needed to understand the minority class.

To solve the issues of overlapping and ambiguity classes and the filtering of irrelevant data, a technique called abstaining [141] is applied, where the classifier has the option to decide whether a result is trustworthy. An aspect that needs to be considered is if the costs of abstaining from a classifier are tenable and comparable to the cost of misclassification. While this is a trade-off in cost-sensitive classification [142], the use of probabilistic classifiers can help to address those challenges without subjective cost settings.

Furthermore, the following demonstrates that the ensemble of ensemble sparse text-mining models is able to extract labels for a portion of the unlabeled building with the use of geo-referenced social media data and how those texts can contribute to building classification.

4.3.1.2 Text Classification

The content that is available on the internet is enormous in all its facets, one datatype is the textual data that is accessible on webpages, but also in the form of collections namely Wikipedia, news pages, as well as social media text blocks. The latter one is the scope within this part of the work, more specifically, the focus is set on Twitter tweets. Nevertheless, while generative models focus on the generation of text. The classification aims to categorize its content. Therefore, features such as language patterns, words, or their number of occurrences. Due to the short and informal structure of the data, low-level structural features are used, including words and characters. Additionally, n-grams of characters are used to be able to cover syllables.

In text classification systems, the first step is to split each document within a given corpus D into small components, in this case, into words. This processing step is called tokenization. All words that are included within the documents are held in a vocabulary, additionally, a vector of natural numbers is created for each document that shows the amount of occurrences for each word that is in the vocabulary. Due to the characteristics of the term frequency (TF) vector, the corpus D can be represented by a sparse matrix S.

The employ of raw frequencies of terms can have the problem of including information that is not useful for the classification task, those are language-specific terms filling words for instance we, have, will, are, and for. It is recommended to remove those types of words from the vocabulary to prevent the model from learning unimportant information. Conversely, rare words can lead to sparsity in the dataset, and it is hard to map the meaning by its context. Therefore, it is essential to build a set of vocabulary that does not contain stop words and words that are below a threshold of term frequency. Especially the latter is a trade-off and does not exclude information that might be important. The combination builds the Term-Frequency Inverse-Document-Frequency (TF-IDF) that indicates how important a word is, the higher this value is, the more important a word. This value can be normalized by the expected frequency the words within a document might have.

A problem that tweets face is their length, which makes a document-word metric like the TF-IDF partially successful for learning methods. Therefore, there is a particular risk of overfitting and the use of basic learning methods e.g. logistic regression and multinational Naïve Bayes. Additionally, due to the nature of the problem of building function detection and the length of each text, it is clear that not every post contributes to the decision of the prediction algorithm; only some of the words that are included within the vocabulary do contribute to the algorithm's decision about the function type of a building.

All in all, there are approximately two methods that address the dedicated text classification problem of having a slight overlap of the words between two vocabularies. (1) topic mining, where selected elements of a vocabulary are assigned to a topic group. (2) text embedding, where words are assigned to a location within a low-dimensional space. The meaning and topic group can then be computed with a distance metric as the L1 or L2 norm. Anyways, what the two techniques have in common is the large amount of data that is needed for training and the precise definition of topics over the entire corpus.

4.3.1.3 Abstaining

The data used in this part of the work comes with some problems, for instance class imbalance. Conversely, there is also a specific risk of having blurred data, unfortunately, methods for class imbalance that are are not necessarily designed to handle this type of problem. Nevertheless, the drawback of blurred data is that sometimes a class cannot be safely assigned to a single data item. Considering the defined problem, not every building is part of the categories *commercial* or *residential*. At the same time, some buildings can also have two or more classes at the same time (e.g., *apartments* and *shops*), while other buildings are from different class types, like *industrial* buildings. Considering the content of tweets with respect to their location, the situation gets even more complex. While some texts contain some helpful information that is related to the function of a building, other tweets within the same area are totally irrelevant. As a consequence, it cannot be expected that a learning algorithm is able to perform well overall classes or even being able to predict certain classes.

A solution to these limitations is abstaining, which has been well studied from the theoretical point of view by [141] and its applications by [143]. Let ϕ be a bilistic classifier that assigns as class probability vector to ϕ_i

Abstaining refers to the strategy of making a prediction when a model is unsure about the corrective of the output. This method can be beneficial for enhancing the overall accuracy of the model by preventing it from making erroneous predictions. More formally said, let ϕ be bilistic classifier that assigns as class probability vector to ϕ_i based on an data item x_i and an vector of decision thresholds τ is introduced,

$$y_l = \arg\max \left(\frac{\phi_i(x_l)}{\tau_i} \right), \tag{4.12}$$

where $0 < \tau_i \leq 1$. Furthermore, this threshold parameter can be used to vary the weight of each class. While one possibility is to guess the values for τ, another way is to apply the mutual information, defined in Theorem 2.3, to optimize the decision thresholds. When using abstaining in classification tasks, the equation or rule defined in Eq. 4.12 is extended to cover additional classes $m + 1$. This extension is defined as

X	**Y**				
	1	2	...	m	$m+1$
1	c_{11}	c_{12}	$\cdots$	c_{1m}	$c_{1(m+1)}$
2	c_{21}	c_{22}	$\cdots$	c_{2m}	$c_{2(m+1)}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
m	c_{m1}	c_{m2}	$\cdots$	c_{mm}	$c_{m(m+1)}$

Table 4.7: Confusion Matrix. This table has been first published in [P5].

$$y_l = \begin{cases} \arg\max\left(\frac{\phi_i(x_l)}{\tau_i}\right) & \text{if } \max\left(\frac{\phi_i(x_l)}{\tau_i}\right) \geq 1 \\ m+1 & \text{otherwise} \end{cases}, \tag{4.13}$$

where m indicated the class. A non-trivial challenge is still to find suitable values to the vector τ, where one way is to include external expert knowledge. While this is a subjective choice that differs from case to case, a more objective strategy that is, for example, based on information theoretics that is comparable to techniques SMOTE introduced by [144] or Chow's rule from [141]. Another way is to reject based on a geometric mean over a large number of datasets with different tasks and characteristics, single- and multi-class. Due to the parameter-free aspects of his approach, it is selected to be used as a basis and is expended in this part of the work. One traditional way to measure the relation and dependency between two variables is the normalized mutual information that is defined within Definition 4.4.

Theorem 4.1. *Let H be Shennon's entropy, formally defined as*

$$H(X) = -\sum_{m}^{i=1} P(X=i)\log_2(P(X=i)),$$

where X is a random variable.

Definition 4.4. *Let NMI be the Normalized Mutual Information, formally defined as*

$$NMI(T,Y) = \frac{I(T,Y)}{H(T)},$$

where X and Y are two random variables.

As mentioned above, it is a non-trivial task to optimize the values for τ. One method that has been introduced by [145] uses mutual information to build a confusion matrix and to solve the optimization problem. Table 4.7 describes a confusion matrix of X and Y that has an additional column for the abstaining case $m+1$. Furthermore, as [146] shows, the mutual information for the entries of the matrix can be approximated by

$$I(X,Y) \approx I(C) = -\frac{\sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}\log_2\left(\frac{c_{ij}}{C_i\sum_{i=1}^{m}\frac{c_{ij}}{N}}\right)}{\sum_{i=1}^{m} C_i \log_2 \frac{C_i}{N}}, \tag{4.14}$$

where N is the total number of samples and C_i is the sum of the i-th row. By iterating until a m handles the drawback of the mutual information of not considering rejections, even if this is formally not quite correct compared to [145].

Optimizing abstaining classifier: As is the case for most learning algorithms, one factor that has a large impact on cost efficiency in terms of computing power is finding an optimal set of parameters. In this case, Powell's grid search algorithm proposed by [147] is used to find good values for the decision threshold τ and parameters to optimize the model's performance and maximize the mutual information, formally said

$$\tau^* = \arg\max NMI(t, y = \phi^\tau(x)), \tag{4.15}$$

where τ^* is the optimal set of values and ϕ^τ represents based on the probability from a classifier ϕ likewise as it is defined in Eq. 4.16.

4.3.1.4 Ensembling Models

Already in 1984, [148] stated that it is commonly done to take the average of operations like forecasts; therefore, the task of ensembling classifiers to archive a higher all-over prediction performance of machine learning models has been widely accepted. A generic formulation of ensembling is to build many unique classifiers for a certain problem and build an averaged model on the bases of the previously created ones. Anyways, the scope in this part of this is set on basic approaches to increase the performance of the final model. A trivial approach of combining n probabilistic classifier ϕ is by averaging

$$\phi_*(x) = \frac{1}{n}\sum_i \phi_i(x) \tag{4.16}$$

where x is the data item that corresponds to a label y, and ϕ is the combined model, which is more robust than each ϕ_i, where $i = 1, \ldots, n$, by itself.

The performance of the final classifier rises, especially if a single ϕ_i reaches a high prediction accuracy. While this method is called *model blending*, another approach is model stacking, which creates ϕ_* by predicting the label y on the basis of a vector of all $\phi_i(x)$. The latter version has the advantage of providing more stable and complex model combinations.

All in all, there is a large number of options for how to ensemble models. The proposed methods are chosen to be taken in this application sample because of their common usage and high archived performance.

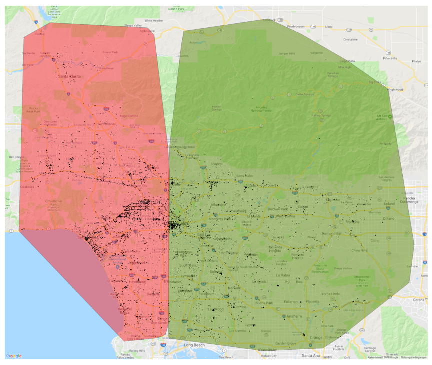
4.3.1.5 Datasets

To be able to perform the experiments, a dataset has been created where a large quantity of Twitter tweets located in Los Angeles has been assigned to two classes. Therefore tweets are located within a defined area and are mapped near buildings that are labeled within OpenStreetMap (OSM). The following gives a detailed insight into the creative process and the dataset itself.

Twitter data preparation: Until recently, it was possible to mine Twitter tweets using their API in the amount of one percent. During a certain time period, 4 TB of tweets have been collected for the dataset. This data has been filtered to only cover tweets that have a precise geolocation. Furthermore, it is expected that most data do have a relation to their location that has been addressed by the application. One needs to know that there is also the possibility to distort the source location of a tweet, such as is done by some bots, but it is assumed that the number is comparable small such that they do not impact the classification heavily. Furthermore, the results of the experiments do confirm this assumption.

OSM buildings: The area of interest is the city of Los Angeles, where the OSM data contains more than 24 800 polygons that either function as residential or commercial. Those two classes are the focus of the experiments that are done in the next section. Additionally, it needs to be mentioned that it is assumed that label accuracy is high.

Spatial join of tweets and OSM buildings: To join the OSM and the Twitter data, each tweet is assigned to its nearest building of the selected OSM polygons. One problem that occurs is that some tweet locations are too far away from any labeled polygon such that it does have a relation to each other. To eliminate such data points, a Euclidean distance is used within a WGS84


(a) Test-train split and the OSM buildings marked in black.

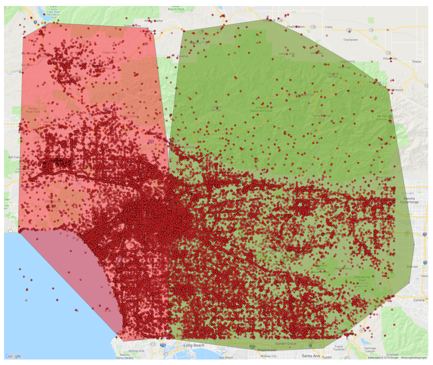

(b) Test-train split and the Twitter tweets marked in red.

Figure 4.24: Visualization of the dataset in the area of Los Angeles. The map has been taken from Google Maps in 2018. This figure has been first published in [P5].

coordinate space that measures the distance. In consequence, all Twitter data is removed from the dataset that is further away than approximately 100m from the nearest polygon.

Dataset split: In order to be able to evaluate the proposed approach correctly, it is important to have to spatial overlap between the training and the test set. On the other hand, it is necessary to have the same class distribution within both splits to be able to train the classifier optimally. Therefore, the data is within its amount into two sets but also geographically into half. Figure 4.24 visualizes the two subsets, where the color indicates the membership.

Datasets: Los Angeles Tweets: The final dataset contains more than 1 200 000 tweets in the area of Los Angeles that are assigned to OSM building polygons. All tweets are geo-referenced and collected between November 2017 and May 2018. Additionally, the data has been split geographically into near-balanced halves. Each split does have two types of classes, nevertheless, whether they correspond to the western or eastern half of the city. The second type of class is the building type, these classes are *commercial* and *residential*, where each class contains 16 133 samples. This builds a dataset that does include tweets where most do have a relation to a building type. Figure 4.25 visualizes the distribution of tweets per building. It can be seen that only most tweets are related to a small portion of buildings. This means in detail

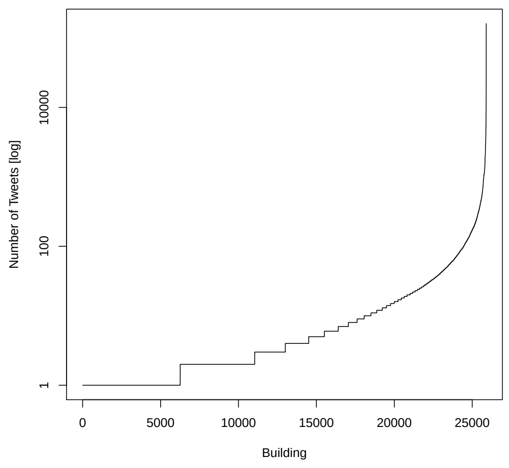


Figure 4.25: Distribution of the number of tweets per building. This figure has been first published in [P5].

that while the average number of tweets per building is 47.33, only 30% of the buildings do have more than ten assigned tweets. Moreover, only 5.6% of the buildings do have more than 100 related tweets.

4.3.1.6 Sparse Text Mining Models

The first task is to create baseline models using a wide range of sparse text mining models. Only then is it formally correctly possible to evaluate the performance of either information-optimal abstaining and the impact of ensembling to models. Therefore it is necessary to create classifiers for the area of interest that analyze the given data on a tweet basis rather than the tweets per building. This is due to the expectation that some tweets are abstained due to missing information about any building, while other tweets are clearly related. Therefore, as described above in Section 4.3.1.2, a vocabulary is created, and then the TF-IDF vector using the normalized term frequency. Furthermore, based on this vector, a sparse matrix is calculated that includes 71 994 columns.

To provide a baseline that is representative, a wide range of models have been trained, that includes the algorithms: Ridge regression, Passive-aggressive classifier [149], k-NN, random forest (100 trees), support vector machines (SVM), neural network (including L1 and L2 regularization), as well as Naïve Bayes. Moreover, an SVM with L1 regression has been taken to extract features from the tweets that are then trained using an SVM with L2

Classifier	**Training**		**Test**	
	Commercial	Residential	Commercial	Residential
Ridge	0.97	0.97	0.50	0.50
Perceptron	1.00	1.00	0.50	0.48
kNN	0.72	0.79	0.40	0.57
RF	1.00	1.00	0.49	0.53
X-Tree	1.00	1.00	0.50	0.52
SVC-L2	0.99	0.99	0.50	0.50
SVC-L1	0.91	0.91	0.50	0.51
ElasticNet	0.77	0.70	0.54	0.42
MN-NB	0.99	0.99	0.52	0.52
SVC-L1/2	0.94	0.94	0.50	0.50

Table 4.8: Overview of the classification results in case of sparse text representation, where the metric is precision. This table has been first published in [P5].

regression.

The results of the selected classifiers do have poor performance due to overfitting and lack of generalization. Even so, it is expected that the introduction of abstaining and ensembling in Setion 4.3.1.7 is able to increase the performance to a large quantity.

Table 4.8 provides an overview of the performance of the models during training and evaluation. When analyzing the individual prediction accuracies with respect to each other, one can see that the discrepancy between the two values is relatively high. Therefore, the models do have the problem of overfitting, such that the evaluation is near to a random guess. When considering only the values from the training, it seems that some knowledge is valid and is extracted from the tweets. As a consequence, the goal is to use abstaining to use this information. While most data instances do not have any indicators that allow a conclusion to a function of a close building, the information collected by the classifiers needs to be selected and used for the classification process.

The information-optimal abstaining mentioned above is applied to the data in the next step to finding decision threshold values that support the defined problem. Furthermore, this is only concerned with classifiers that do provide from their own probability, this is caused by the fact that only then abstaining can be applied without any concerns. Anyways, this still requires an additional subset, which leads to the problem of reducing the amount of data that is

Classifier	**Abstain-Rate**	**Commercial**		**Residential**	
		Precision	Recall	Precision	Recall
MN-NB1	63%	0.54	0.19	0.57	0.21
MN-NB2	72%	0.53	0.14	0.58	0.17
MN-NB3	89%	0.55	0.04	0.62	0.09
SGD-L2	99%	0.17	0.00	0.83	0.01
SGD-L1	96%	0.56	0.01	0.76	0.04

Table 4.9: verview of the classification results with using abstaining, where the metric is the precision and recall. This table has been first published in [P5].

available. In general, while it is unclear if the calibration does work on a disjoint spatial set, this does lead to the point of having small urban areas for the tasks. The size of the region doe matter in this case because some districts of the city most likely only contain one single type of building. Such as, in the inner city, it is tending to find shops and residential areas rather than industry buildings.

One characteristic of abstaining is that the precision is increased with the costs of having a low recall, therefore, the F1-score is replaced by its class-based components. Anyways the classifier Naïve Bayes are trained with a selection of smoothing parameters that are MN-B1 = 0.001, MN-B2 = 0.01, and MN-B3 = 0.1. While those values adjust the probabilities of words of the test set, this does also affect the abstaining.

Table 4.9 provides an overview of the results of using a more significant smoothing parameter for Naïve Bayes. It can be seen that this also increases the abstaining rate. On the other hand, while the recall is decreasing, the precision increases, leading to a more significant gap between those values.

Figure 4.26 visualizes the impact of the Laplacian smoothing parameter on the models. One can see that an immense amount of regularization does increase the performance of the prediction of minority classes. At the same time, the amount of classified elements shrinks. In conclusion, this leads to a trade-off between predicting the minority classes or a more conservative classification.

All in all, it can be said that it is non-trivial to assign functions to a building based on tweets. In general, the trained classifier dent to overfitting and poor generalization. Conversely, the use of abstaining showed that there is valuable information within the large amount of data that contribute to the defined

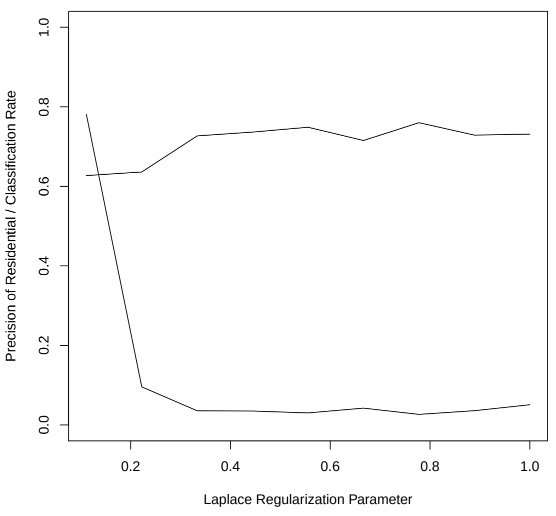


Figure 4.26: Visualization of the impact of Laplacian smoothing. This figure has been first published in [P5].

problem and enables the possibility to classify correctly more than 80% for one percent of the unlabeled OSM buildings. This means that 168 buildings are labeled correctly from the test set. Furthermore, combined with (human) expert knowledge could increase the performance of mapping buildings in OSM data.

4.3.1.7 Ensembling Abstaining Models

While the previous section depicted that it is possible to use a comparatively small amount of information to train a classifier to assign the function to buildings, this section focuses on ensembles of models to improve performance. In more detail, the idea is to train multiple models, each learning a different fraction of the valuable information. The union of those models should lead to higher robustness and a maximization of the prediction performance. First, it is necessary to remove words from the building vocabulary that are most tending not contributing to the task, this is, for example, among others, stop words and smileys. Therefore, words are reduced that have a probability $< 0.1\%$ to occur in the text but also those who have a higher likelihood, specifically $> 80\%$. In this way, rare words and frequent words are removed from the vocabulary and lead to a set of 1 032 words that are considered. Based on this data, multiple different classifiers, such as SVMs, are trained and evaluated. An overview of the models and their performance is given in Figure 4.27, where

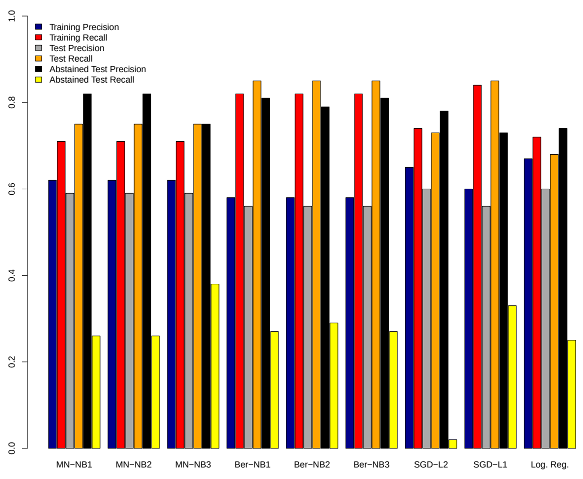


Figure 4.27: Visualization of the performance of the ensembles of the abstained models. This figure has been first published in [P5].

the naming convention is the same as before. Furthermore, the performance is depicted for the *residential* class and over the metrics for all subsets.

Analyzing the image, one can see that the bias is reduced by simply removing rare and frequent words from the vocabulary. Additionally, the precision and recall are in the range of 60% to 80% for both the training and the test set. The increase in precision was reached by abstaining from each classifier separately. Moreover, an ensemble of all models is built by a weighted average. The weights for building this new probabilistic classifier are computed by the precision that is expected from the residential buildings. This is due to the need to have high correctness when predicting this building function. This leads to a classifier that correctly labeled 1 937 buildings, which refers to an accuracy of 85%. For all other cases, the decision was not trustful enough to make a decision about the functionality based on the given tweets, and therefore no prediction has been made.

While the focus has been set firmly on the low-represented class *residential*, negative effects on the other class *commercial* have been ignored until now, nevertheless, the results for this particular class are still relatively high with a precision of 61% and a recall of 72%, especially, with the taken actions. All in all, the two classes combined enabled the possibility to label more than 31 000 buildings within the area of interest.

Table 4.10 provides an overview of the prediction performance from the ensembled models compared to selected baseline models from the lase section.

Classifier	**Abstain-Rate**	**Training**		**Test**	
		Commercial	Residential	Commercial	Residential
BIRP	54 %	0.60	0.78	0.82	0.26
HRF1	58 %	0.70	0.23	0.75	0.38
AVE	-	0.59	0.85	0.73	0.41
AVE-A	16 %	0.61	0.72	0.75	0.37
AVE-F1	-	0.59	0.86	0.74	0.52
AVE-F1-A	16 %	0.61	0.72	0.75	0.37

Table 4.10: Overview of the result from the ensemble classifier. This table has been first published in [P5].

This includes models with the highest individual *residential* precision (BIRP) reached by Naïve Bayes with high smoothing as well as the model with the highest residential F1-score (HRF1) reached by Naïve Bayes with low smoothing. Compared with the ensemble models with averaging (AVG), averaging using F1-score (AVG-F1) as well as both models with information-optimal abstaining.

Each classifier suffers from different trade-offs. For example, do BIRP and HRF1 have the same problem of high abstaining rates. Conversely, the enabled versions of these models could improve their performance. Considering all results, the performances are within a range of $[73\%, 75\%]$ for the precision of the residential class and a recall of $[37\%, 52\%]$. Furthermore, one can also see that the F1-score enable versions do provide a generally high value for their metrics.

4.3.1.8 Summary

In this part of the work, social media data are used to predict the building functions. The massive amount of tweets brings, on the one hand, a large amount of data but has the drawback that only a tiny portion of the information is engaging for the task, therefore, it is a non-trivial task to locate the details that largely contributes to the performance of the classifier. In this work, methods such as abstaining and ensembling models are taken to only use interesting data. While the models did reach high performances, even with the mentioned techniques, a generalization has not been reached. A reason for that is not on the model, instead, it is caused by learning data that is not relevant to the task.

To evaluate the models for the given problem task a wide range of different models have been trained that included abstaining and ensembling techniques. The results show that some of them reach high predictions but with a small amount of classified buildings, caused by a high number of results that lacked truthfulness and are therefore abstained. All in all, it can be concluded that social media data like tweets are not enough to detect the function type of a building. Conversely, the introduction of ensembling models to this problem was able to increase the performance of the trained models. In order to investigate a more extensive set of parameters, it would be necessary to have a larger set of data.

5 Conclusion

In spatial computing, deep learning is employed to discover the content of data to make predictions about, for example, urban development or land use. Often, the models that are common, provide the maximum possible complexity, like large convolutional neural networks using 32-bit floating point numbers. The same applies for data, such as images, where information is included that is not relevant or meaningful. The use of the entire complexity leads to challenges for computational resources, storage systems, and real-time access to data. This is especially the case for non-governmental institutions. Reducing the amount of information, that is utilized to solve a particular problem, can reduce the amount of technological as well as economic resources.

The impact from the technical perspective. The minimum of the complexity and informational content of heterogeneous data is the point where only required information is left. Crossing this point and pruning more information leads to a decrease of the systems outcome's accuracy. The theoretical analysis showed that data as well as algorithms do include information that does not significantly contribute to the final computed prediction of a data-driven system. More specifically, the information theory from Shannon focuses on entropy but ignores which portion of an object, such as an image, is from high relevance. This is the same for the application examples, where a moderate compression of the input data led to a lower runtime with similar to slightly higher accuracy values. This also applies to algorithms where, for instance, deep learning architectures furnish complexity or even parameters that are not important in terms of reaching a high quality on the predicted outcome. Especially the use of a lossless compression methods reduces the memory footprint and time needed to move data from the disk into the main memory without impacting the algorithm's accuracy. Moreover, removing non-meaningful information can emphasize other features and lead to slightly better predictions of the system's results. In addition, the compression including pruning from the generated results are able to increase the robustness of the models.

The impact to the society. While this excerpt focuses on the technical perspectives, the compression of data-driven systems can also impact the industry and society. The operational costs of vastly scaled (generative) AI, such as it has been done with the large language model, are enormous. In addition, does the increase of machine learning to automate business use cases have an impact on global energy consumption, therefore those technologies has the potential to raise the carbon footprint[1]. Furthermore, the larger complexity of the models and datasets utilized for training requires scaling the available computing hardware, which indirectly affects our environment[2]. Therefore the employ of compressing the data-driven system, in terms of pruning information that is from less relevance and meaning, is able to optimize the energy consumption, required computational hardware resources, and time that needs to be invested to either train or inference a machine learning program. Thus, the utilization of compression does also have an impact on operational costs as well as the carbon footprint.

5.1 Open Problems and Future Work

This thesis provides an investigation of the role of compression in spatial computing, where the scope is rather a feasibility study than the demand in an industrial context. While the theoretical aspects are followed by application samples that give a foundation to create more efficient data-driven systems, the intense focus on satellite images leads to a lack of covering the different types of spatial data. Furthermore, there are less datasets in Earth observation that cover either a particular problem or are publicly available, which is a problem, especially for tasks that are relevant to be performed on board a satellite, for example, the detection of wildfire. Another limitation of this thesis is that the investigation on edge computing hardware is only done on one specific FPGA.

Furthermore, there are several research questions that are not answered within this thesis: (1) Is there an ability to compress the hardware component itself to optimize the performance characteristics of a data-driven system, such

[1]Wired: AI Can Do Great Things–if It Does not Burn the Planet (https://www.wired.com/story/ai-great-things-burn-planet/) [Accessed on 08.06.2023]
[2]Wired: The Generative AI Race Has a Dirty Secret (https://www.wired.co.uk/article/the-generative-ai-search-race-has-a-dirty-secret) [Accessed on 08.06.2023]

as the runtime and the energy consumption, without losing precision on the outcome, (2) can the program be compressed, (3) can the compression of inter-program communication optimize the performance of the entire system. Each of this question is from relevance but crosses is beyond scope of this work. Moreover, the creation of datasets and the use of a more diverse set of hardware accelerators would lead to an overcome of the named limitations. In addition to this, future research projects should also focus on compressing the result of machine learning algorithms.

Bibliography

[1] R. Unterstein, *60 petabytes for the german satellite data archive d-sda*, German Aerospace Center (DLR), Ed., 2018. [Online]. Available: `https://www.dlr.de/eoc/en/desktopdefault.aspx/tabid-12632/22039_read-51751`.

[2] V. J. Reddi, C. Cheng, D. Kanter, *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2020, pp. 446–459, ISBN: 978-1-7281-4661-4. DOI: `10.1109/ISCA45697.2020.00045`.

[3] Y. Umuroglu, N. J. Fraser, G. Gambardella, *et al.*, "Finn," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, J. Greene and J. H. Anderson, Eds., New York, NY, USA: ACM, 2017, pp. 65–74, ISBN: 9781450343541. DOI: `10.1145/3020078.3021744`.

[4] G. Sumbul, M. Charfuelan, B. Demir, and V. Markl, "Bigearthnet: A large-scale benchmark archive for remote sensing image understanding," in *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, 2019, pp. 5901–5904.

[5] X.-Y. Tong, G.-S. Xia, Q. Lu, *et al.*, "Land-cover classification with high-resolution remote sensing images using transferable deep models," *Remote Sensing of Environment*, vol. 237, p. 111 322, 2020, ISSN: 0034-4257. DOI: `https://doi.org/10.1016/j.rse.2019.111322`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0034425719303414`.

[6] M. Werner, "Globimaps - a probabilistic data structure for in-memory processing of global raster datasets," in *27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '19)*, 2019.

[7] J. Tichonov, O. Kurasova, and E. Filatovas, "Image classification for jpeg compression," *Advances in Science and Technology Research Journal*, vol. 12, pp. 29–34, Jun. 2018.

[8] Q. Qin, J. Ren, J. Yu, *et al.*, "To compress, or not to compress: Characterizing deep learning model compression for embedded inference," in *2018 IEEE International Conference on Big Data and Cloud Computing (BdCloud)*, 2018, pp. 729–736.

[9] European Space Agency, *Sentinel-2 user handbook*, 2015.

[10] P. Helber, B. Bischke, A. Dengel, and D. Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 7, pp. 2217–2226, 2019, ISSN: 1939-1404. DOI: 10.1109/JSTARS.2019.2918242.

[11] D. Kim, J. Lee, and B. Ham, "Distance-aware quantization," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, 2021, pp. 5251–5260, ISBN: 978-1-6654-2812-5. DOI: 10.1109/ICCV48922.2021.00522.

[12] J. Martinez, J. Shewakramani, T. Wei Liu, I. Andrei Barsan, W. Zeng, and R. Urtasun, "Permute, quantize, and fine-tune: Efficient compression of neural networks," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2021, pp. 15 694–15 703, ISBN: 978-1-6654-4509-2. DOI: 10.1109/CVPR46437.2021.01544.

[13] Y. Fan, W. Pang, and S. Lu, "Hfpq: Deep neural network compression by hardware-friendly pruning-quantization," *Applied Intelligence*, vol. 51, no. 10, pp. 7016–7028, 2021, ISSN: 0924-669X. DOI: 10.1007/s10489-020-01968-x.

[14] S. Thulasidasan, T. Bhattacharya, J. Bilmes, G. Chennupati, and J. Mohd-Yusof, "Combating label noise in deep learning using abstention," *arXiv preprint arXiv:1905.10964*, 2019.

[15] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948, ISSN: 00058580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[16] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, 10. printing. Urbana: Univ. of Ill. Press, 1964.

[17] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Second edition. Hoboken, NJ: Wiley-Interscience, 2006, ISBN: 0471241954. [Online]. Available: `http://www.loc.gov/catdir/enhancements/fy0624/2005047799-d.html`.

[18] W. Hamilton, *Discussions on Philosophy and Literature, Education and University Reform Chiefly From the Edinburgh Review : Corrected, Vindicated, Enlarged : In Notes and Appendices*. Longman, Brown, Green and Longmans, 1852.

[19] R. Cilibrasi and P. Vitanyi, "Clustering by compression," *IEEE Transactions on information theory*, vol. 51, no. 4, pp. 1523–1545, 2005, ISSN: 0018-9448. DOI: `10.1109/TIT.2005.844059`.

[20] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge: Cambridge University Press, 2003, ISBN: 978-0-521-64298-9. [Online]. Available: `http://www.loc.gov/catdir/description/cam032/2003055133.html`.

[21] R. M. Gray, *Entropy and Information Theory*. Boston, MA: Springer US, 2011, ISBN: 978-1-4419-7969-8. DOI: `10.1007/978-1-4419-7970-4`.

[22] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951, ISSN: 0003-4851. DOI: `10.1214/aoms/1177729694`.

[23] R. M. Fano, *The Transmission of Information*. Massachusetts Institute of Technology, Research Laboratory of Electronics . . ., 1949, vol. 65.

[24] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952, ISSN: 0096-8390. DOI: `10.1109/JRPROC.1952.273898`.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (Adaptive computation and machine learning). Cambridge, Massachusetts and London, England: The MIT Press, 2016, ISBN: 9780262035613.

[26] *Ieee standard for floating-point arithmetic*, Piscataway, NJ, USA. DOI: `10.1109/IEEESTD.2019.8766229`.

[27] A. N. Kolmogorov, "On tables of random numbers," *Theoretical Computer Science*, vol. 207, no. 2, pp. 387–395, 1998, ISSN: 03043975. DOI: `10.1016/S0304-3975(98)00075-9`.

[28] P. M. B. Vitányi, F. J. Balbach, R. L. Cilibrasi, and M. Li, "Normalized information distance," in *Information Theory and Statistical Learning*, F. Emmert-Streib and M. Dehmer, Eds., Boston, MA: Springer US, 2009, pp. 45–82, ISBN: 978-0-387-84815-0. DOI: `10.1007/978-0-387-84816-7{\textunderscore}3`.

[29] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*. New York, NY: Springer New York, 2008, ISBN: 978-0-387-33998-6. DOI: `10.1007/978-0-387-49820-1`.

[30] M. Li, X. Chen, X. Li, B. Ma, and P. Vitanyi, "The similarity metric," *IEEE Transactions on information theory*, vol. 50, no. 12, pp. 3250–3264, 2004, ISSN: 0018-9448. DOI: `10.1109/TIT.2004.838101`.

[31] M. Ferbus-Zanda and S. Grigorieff, "Kolmogorov complexity in perspective part i: Information theory and randomness," in *Constructivity and Computability in Historical and Philosophical Perspective*, ser. Logic, Epistemology, and the Unity of Science, J. Dubucs and M. Bourdeau, Eds., vol. 34, Dordrecht: Springer Netherlands, 2014, pp. 57–94, ISBN: 978-94-017-9216-5. DOI: `10.1007/978-94-017-9217-2{\textunderscore}3`.

[32] P. D. Grünwald and P. M. B. Vitanyi, *Algorithmic information theory*, 2008. DOI: `10.48550/arXiv.0809.2754`.

[33] P. M. B. Vitányi, "How incomputable is kolmogorov complexity?" *Entropy (Basel, Switzerland)*, vol. 22, no. 4, 2020. DOI: `10.3390/e22040408`.

[34] N. Tran, "The normalized compression distance and image distinguishability," in *Human Vision and Electronic Imaging XII*, B. E. Rogowitz, T. N. Pappas, and S. J. Daly, Eds., ser. SPIE Proceedings, SPIE, 2007, p. 64921D. DOI: `10.1117/12.704334`.

[35] M. Coca, A. Anghel, and M. Datcu, "Normalized compression distance for sar image change detection," in *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, 2018, pp. 5784–5787, ISBN: 978-1-5386-7150-4. DOI: `10.1109/IGARSS.2018.8518126`.

[36] M. Coca, A. Anghel, and M. Datcu, “Unbiased seamless sar image change detection based on normalized compression distance,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 7, pp. 2088–2096, 2019. DOI: 10.1109/JSTARS.2019.2909143.

[37] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and quantization for deep neural network acceleration: A survey,” *Neurocomputing*, vol. 461, pp. 370–403, 2021, ISSN: 09252312. DOI: 10.1016/j.neucom.2021.07.045.

[38] C. C. Aggarwal, *Neural Networks and Deep Learning*. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0.

[39] O. L. Calin, *Deep learning architectures: A mathematical approach* (Springer eBook Collection). Cham, Switzerland: Springer, 2020, ISBN: 978-3030367237. DOI: 10.1007/978-3-030-36721-3.

[40] J. Nagi, F. Ducatelle, G. A. Di Caro, *et al.*, “Max-pooling convolutional neural networks for vision-based hand gesture recognition,” in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, IEEE, 2011, pp. 342–347, ISBN: 978-1-4577-0242-6. DOI: 10.1109/ICSIPA.2011.6144164.

[41] R. Nirthika, S. Manivannan, A. Ramanan, and R. Wang, “Pooling in convolutional neural networks for medical image analysis: A survey and an empirical study,” *Neural computing & applications*, vol. 34, no. 7, pp. 5321–5347, 2022, ISSN: 0941-0643. DOI: 10.1007/s00521-022-06953-8.

[42] A. Zafar, M. Aamir, N. Mohd Nawi, *et al.*, “A comparison of pooling methods for convolutional neural networks,” *Applied Sciences*, vol. 12, no. 17, p. 8643, 2022. DOI: 10.3390/app12178643.

[43] M. Elgendy, *Deep learning for vision systems*. Shelter Island, NY: Manning, 2020, ISBN: 1617296198.

[44] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML’15, Lille, France: JMLR.org, 2015, pp. 448–456.

[45] S.-E. Chang, Y. Li, M. Sun, *et al.*, "Rmsmp: A novel deep neural network quantization framework with row-wise mixed schemes and multiple precisions," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, 2021, pp. 5231–5240, ISBN: 978-1-6654-2812-5. DOI: 10.1109/ICCV48922.2021.00520.

[46] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17, AAAI Press, 2017, pp. 4278–4284.

[47] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, ser. Lecture Notes in Computer Science, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., vol. 9351, Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24573-7. DOI: 10.1007/978-3-319-24574-4{\textunderscore}28.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016, pp. 770–778, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90.

[49] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, pp. 5967–5976, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.632.

[50] Olga Russakovsky, Jia Deng, Hao Su, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[51] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[52] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*,

vol. 105, no. 12, pp. 2295–2329, 2017, ISSN: 0018-9219. DOI: 10.1109/JPROC.2017.2761740.

[53] A. G. Howard, M. Zhu, B. Chen, *et al.*, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. DOI: 10.48550/arXiv.1704.04861.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9908, Cham: Springer International Publishing, 2016, pp. 630–645, ISBN: 978-3-319-46492-3. DOI: 10.1007/978-3-319-46493-0{\textunderscore}38.

[55] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size*, 2016. DOI: 10.48550/arXiv.1602.07360.

[56] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[57] Jonathan Frankle and Michael Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7.

[58] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, 2018, pp. 2704–2713, ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00286.

[59] M. Shen, F. Liang, R. Gong, *et al.*, "Once quantization-aware training: High performance extremely low-bit architecture search," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, 2021, pp. 5320–5329, ISBN: 978-1-6654-2812-5. DOI: 10.1109/ICCV48922.2021.00529.

[60] Paulius Micikevicius, Sharan Narang, Jonah Alben, *et al.*, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[61] W. Chen, P. Wang, and J. Cheng, "Towards mixed-precision quantization of neural networks via constrained optimization," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, 2021, pp. 5330–5339, ISBN: 978-1-6654-2812-5. DOI: `10.1109/ICCV48922.2021.00530`.

[62] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan, *Ternary weight networks*, 2016. DOI: `10.48550/arXiv.1605.04711`.

[63] T. Han, D. Li, J. Liu, L. Tian, and Y. Shan, "Improving low-precision network quantization via bin regularization," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, 2021, pp. 5241–5250, ISBN: 978-1-6654-2812-5. DOI: `10.1109/ICCV48922.2021.00521`.

[64] P. Chen, J. Liu, B. Zhuang, M. Tan, and C. Shen, "Aqd: Towards accurate quantized object detection," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2021, pp. 104–113, ISBN: 978-1-6654-4509-2. DOI: `10.1109/CVPR46437.2021.00017`.

[65] S. Oh, H. Sim, S. Lee, and J. Lee, "Automated log-scale quantization for low-cost deep neural networks," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2021, pp. 742–751, ISBN: 978-1-6654-4509-2. DOI: `10.1109/CVPR46437.2021.00080`.

[66] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2017. [Online]. Available: `https://openreview.net/forum?id=rJqFGTslg`.

[67] Y.-J. Zheng, S.-B. Chen, C. H. Q. Ding, and B. Luo, "Model compression based on differentiable network channel pruning," *IEEE transactions on neural networks and learning systems*, vol. PP, 2022. DOI: `10.1109/TNNLS.2022.3165123`.

[68] H. Mostafa and X. Wang, *Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization*, 2019. DOI: `10.48550/arXiv.1902.05967`.

[69] M. Shen, P. Molchanov, H. Yin, and J. M. Alvarez, "When to prune? a policy towards early structural pruning," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2022, pp. 12 237–12 246, ISBN: 978-1-6654-6946-3. DOI: `10.1109/CVPR52688.2022.01193`.

[70] P. Wimmer, J. Mehnert, and A. Condurache, "Interspace pruning: Using adaptive filter representations to improve training of sparse cnns," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2022, pp. 12 517–12 527, ISBN: 978-1-6654-6946-3. DOI: `10.1109/CVPR52688.2022.01220`.

[71] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, 2015. DOI: `10.48550/arXiv.1510.00149`.

[72] M. Werner and Y.-Y. Chiang, *Handbook of Big Geospatial Data.* Cham: Springer International Publishing, 2021, ISBN: 978-3-030-55461-3. DOI: `10.1007/978-3-030-55462-0`.

[73] European Space Agency, *Sentinel-1: Esa's radar observatory mission for gmes operational services*, K. Fletcher, Ed., Noordwijk, 2012.

[74] M. Werner, *Indoor Location-Based Services.* Cham: Springer International Publishing, 2014, ISBN: 978-3-319-10698-4. DOI: `10.1007/978-3-319-10699-1`.

[75] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970, ISSN: 0001-0782. DOI: `10.1145/362686.362692`. [Online]. Available: `https://doi.org/10.1145/362686.362692`.

[76] M. Werner, "BACR: Set Similarities with Lower Bounds and Application to Spatial Trajectories," in *23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL 2015)*, 2015.

[77] J. Skilling, "Programming the hilbert curve," in *AIP Conference Proceedings*, AIP, 2004, pp. 381–387. DOI: `10.1063/1.1751381`.

[78] M. Bader, *Space-Filling Curves.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 9, ISBN: 978-3-642-31045-4. DOI: `10.1007/978-3-642-31046-1`.

[79] G. Niemeyer, *Geohash*, 2008.

[80] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, IEEE, 2014, pp. 10–14, ISBN: 978-1-4799-0920-9. DOI: `10.1109/ISSCC.2014.6757323`.

[81] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating cnn inference on fpgas: A survey," *arXiv preprint arXiv:1806.01683*, 2018.

[82] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, Mar. 2019, ISSN: 1936-7406. DOI: `10.1145/3289185`. [Online]. Available: `https://doi.org/10.1145/3289185`.

[83] AMD Xilinx, *Vitisai user guide*, 2023.

[84] H. Collins and C. Nay, *Ibm unveils 400 qubit-plus quantum processor and next-generation ibm quantum system two*, IBM Newsroom, Ed., 2022. [Online]. Available: `https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two`.

[85] J. M. Zollner, "Quantum classifiers for remote sensing," in *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '22, Seattle, Washington: Association for Computing Machinery, 2022, ISBN: 9781450395298. DOI: `10.1145/3557915.3565537`. [Online]. Available: `https://doi.org/10.1145/3557915.3565537`.

[86] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[87] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *International Conference on Learning Representations*, 2017.

[88] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," in *International Conference on Learning Representations*, 2017.

[89] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A survey of quantization methods for efficient neural network inference,” in *Low-Power Computer Vision*, G. K. Thiruvathukal, Y.-H. Lu, J. Kim, Y. Chen, and B. Chen, Eds., Boca Raton: Chapman and Hall/CRC, 2022, pp. 291–326, ISBN: 9781003162810. DOI: `10.1201/9781003162810-13`.

[90] S. V. Naik, S. K. Majjigudda, S. Naik, *et al.*, “Survey on comparative study of pruning mechanism on mobilenetv3 model,” in *2021 International Conference on Intelligent Technologies (CONIT)*, IEEE, 2021, pp. 1–8, ISBN: 978-1-7281-8583-5. DOI: `10.1109/CONIT51480.2021.9498400`.

[91] X. Ma, S. Lin, S. Ye, *et al.*, “Non-structured dnn weight pruning—is it beneficial in any platform?” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 4930–4944, 2022. DOI: `10.1109/TNNLS.2021.3063265`.

[92] M. Zhu and S. Gupta, “To prune, or not to prune: Exploring the efficacy of pruning for model compression,” *arXiv preprint arXiv:1710.01878*, 2017.

[93] S. Babakniya, S. Kundu, S. Prakash, Y. Niu, and S. Avestimehr, “Federated sparse training: Lottery aware model compression for resource constrained edge,” in *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*, 2022.

[94] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “Model compression and acceleration for deep neural networks: The principles, progress, and challenges,” *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018. DOI: `10.1109/MSP.2017.2765695`.

[95] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, “Theory and practice of bloom filters for distributed systems,” *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2012. DOI: `10.1109/SURV.2011.031611.00024`.

[96] J. Bruck, J. Gao, and A. Jiang, “Weighted bloom filter,” in *2006 IEEE International Symposium on Information Theory*, 2006, pp. 2304–2308. DOI: `10.1109/ISIT.2006.261978`.

[97] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019. DOI: `10.1109/COMST.2018.2889329`.

[98] M. Werner and M. Kiermeier, "A Low-Dimensional Feature Vector Representation for Alignment-Free Spatial Trajectory Analysis," in *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems (MobiGIS'16)*, 2016.

[99] M. Ganaie, M. Hu, A. Malik, M. Tanveer, and P. Suganthan, "Ensemble deep learning: A review," *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105 151, 2022, ISSN: 0952-1976. DOI: `https://doi.org/10.1016/j.engappai.2022.105151`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S095219762200269X`.

[100] H. Salman, J. Li, I. Razenshteyn, *et al.*, "Provably robust deep learning via adversarially trained smoothed classifiers," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2019/file/3a24b25a7b092a252166a1641ae953e7-Paper.pdf`.

[101] A. Mohammed and R. Kora, "A comprehensive review on ensemble deep learning: Opportunities and challenges," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 2, pp. 757–774, 2023, ISSN: 1319-1578. DOI: `https://doi.org/10.1016/j.jksuci.2023.01.014`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1319157823000228`.

[102] T. Hoeser and C. Kuenzer, "Object detection and image segmentation with deep learning on earth observation data: A review-part i: Evolution and recent trends," *Remote Sensing*, vol. 12, no. 10, p. 1667, 2020.

[103] S. Salcedo-Sanz, P. Ghamisi, M. Piles, *et al.*, "Machine learning information fusion in earth observation: A comprehensive review of methods, applications and data sources," *Information Fusion*, vol. 63, pp. 256–272, 2020.

[104] A. Cutler and L. Breiman, “Archetypal analysis,” *Technometrics*, vol. 36, no. 4, pp. 338–347, 1994.

[105] H. Li, X. Dou, C. Tao, *et al.*, “Rsi-cb: A large-scale remote sensing image classification benchmark using crowdsourced data,” *Sensors (Basel, Switzerland)*, vol. 20, no. 6, 2020. DOI: 10.3390/s20061594.

[106] P. Helber, B. Bischke, A. Dengel, and D. Borth, “Introducing eurosat: A novel dataset and deep learning benchmark for land use and land cover classification,” in *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, 2018, pp. 204–207, ISBN: 978-1-5386-7150-4. DOI: 10.1109/IGARSS.2018.8519248.

[107] The HDF Group. “Hierarchical Data Format, version 5.” https://www.hdfgroup.org/HDF5/. (1997-2022).

[108] T. Developers, *Tensorflow*, version v2.8.2, Specific TensorFlow versions can be found in the "Versions" list on the right side of this page.<br>See the full list of authors <a href="https://github.com/tensorflow/tensorflow/graphs/contr ibutors">on GitHub</a>., May 2022. DOI: 10.5281/zenodo.6574269. [Online]. Available: https://doi.org/10.5281/zenodo.6574269.

[109] F. Chang, J. Dean, S. Ghemawat, *et al.*, “Bigtable: A distributed storage system for structured data,” *ACM Trans. Comput. Syst.*, vol. 26, no. 2, Jun. 2008, ISSN: 0734-2071. DOI: 10.1145/1365815.1365816. [Online]. Available: https://doi.org/10.1145/1365815.1365816.

[110] B. Demir and L. Bruzzone, “Hashing-based scalable remote sensing image search and retrieval in large archives,” *IEEE transactions on geoscience and remote sensing*, vol. 54, no. 2, pp. 892–904, 2015.

[111] Y. Sun, S. Feng, Y. Ye, *et al.*, “Multisensor fusion and explicit semantic preserving-based deep hashing for cross-modal remote sensing image retrieval,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022. DOI: 10.1109/TGRS.2021.3136641.

[112] Y. Sun, Y. Ye, X. Li, *et al.*, “Unsupervised deep hashing through learning soft pseudo label for remote sensing image retrieval,” *Knowledge-Based Systems*, vol. 239, p. 107 807, 2022, ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.107807. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095070512101008X.

[113] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics), Softcover reprint of the original 1st edition 2006 (corrected at 8th printing 2009). New York, NY: Springer New York, 2006, ISBN: 9781493938438.

[114] G. Cheng and J. Wei, "Color quantization application based on k-means in remote sensing image processing," *Journal of Physics: Conference Series*, vol. 1213, no. 4, p. 042 012, 2019, ISSN: 1742-6588. DOI: 10.1088/1742-6596/1213/4/042012.

[115] G. Roelofs, Ed., *PNG: The definitive guide ; [creating & programming portable network graphics*, 1. ed. Beijing and Köln: O'Reilly, 1999, ISBN: 1-56592-542-4. [Online]. Available: http://www.loc.gov/catdir/enhancements/fy0915/00702376-b.html.

[116] S. Leonard, *Windows image media types*, 2016. DOI: 10.17487/RFC7903.

[117] A. D. Association *et al.*, "Tiff revision 6.0," *Adobe Systems Incorporated, Mountain View*, 1992.

[118] E. A. Belyaev, C. Mantel, and S. O. Forchhammer, "High bit depth infrared image compression via low bit depth codecs," in *Infrared Remote Sensing and Instrumentation XXV*, M. Strojnik and M. S. Kirk, Eds., SPIE, 6/08/2017 - 10/08/2017, p. 9, ISBN: 9781510612631. DOI: 10.1117/12.2275542. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10403/2275542/High-bit-depth-infrared-image-compression-via-low-bit-depth/10.1117/12.2275542.full.

[119] G. Hudson, A. Léger, B. Niss, I. Sebestyén, and J. Vaaben, "Jpeg-1 standard 25 years: Past, present, and future reasons for a success," *Journal of Electronic Imaging*, vol. 27, no. 04, p. 1, 2018, ISSN: 1017-9909. DOI: 10.1117/1.JEI.27.4.040901.

[120] G. K. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992, ISSN: 00983063. DOI: 10.1109/30.125072.

[121] G.-S. Xia, J. Hu, F. Hu, *et al.*, "Aid: A benchmark data set for performance evaluation of aerial scene classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 7, pp. 3965–3981, 2017, ISSN: 0196-2892. DOI: 10.1109/TGRS.2017.2685945.

[122] P. Jin, G.-S. Xia, F. Hu, Q. Lu, and L. Zhang, "Aid++: An updated version of aid on scene classification," in *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, 2018, pp. 4721–4724, ISBN: 978-1-5386-7150-4. DOI: 10.1109/IGARSS.2018.8518882.

[123] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18, London, United Kingdom: Association for Computing Machinery, 2018, pp. 387–395, ISBN: 9781450355520. DOI: 10.1145/3219819.3219845. [Online]. Available: https://doi.org/10.1145/3219819.3219845.

[124] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Dec. 2015, arXiv:1512.03385 [cs]. DOI: 10.48550/arXiv.1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[125] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018. DOI: 10.1109/JSTSP.2018.2797022.

[126] G. Cheng, J. Han, and X. Lu, "Remote sensing image scene classification: Benchmark and state of the art," *Proceedings of the IEEE*, vol. 105, no. 10, pp. 1865–1883, Oct. 2017, ISSN: 1558-2256. DOI: 10.1109/jproc.2017.2675998. [Online]. Available: http://dx.doi.org/10.1109/JPROC.2017.2675998.

[127] T. Alsahfi, M. Almotairi, R. Elmasri, and B. Alshemaimri, "Road map generation and feature extraction from gps trajectories data," in *Proceedings of the 12th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, ser. IWCTS'19, Chicago, IL, USA: Association for Computing Machinery, 2019, ISBN: 9781450369671. DOI: 10.1145/3357000.3366140. [Online]. Available: https://doi.org/10.1145/3357000.3366140.

[128] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data(base) Engineering Bulletin*, Jun. 2010. [Online]. Available: https://www.

microsoft . com / en - us / research / publication / geolife - a - collaborative - social - networking - service - among - user - location-and-trajectory/.

[129] A. Karbalayghareh, U. Braga-Neto, and E. R. Dougherty, "Classification of single-cell gene expression trajectories from incomplete and noisy data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 1, pp. 193–207, 2019.

[130] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561–580, Oct. 2007, ISSN: 1088-467X.

[131] T. Devogele, L. Etienne, M. Esnault, and F. Lardy, "Optimized discrete fréchet distance between trajectories," in *Proceedings of the 6th ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data*, ser. BigSpatial'17, Redondo Beach, CA, USA: Association for Computing Machinery, 2017, pp. 11–19, ISBN: 9781450354943. DOI: 10.1145/3150919.3150924. [Online]. Available: https://doi.org/10.1145/3150919.3150924.

[132] K. Buchin, Y. Diez, T. van Diggelen, and W. Meulemans, "Efficient trajectory queries under the fréchet distance (gis cup)," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2017, pp. 1–4.

[133] F. Dütsch and J. Vahrenhold, "A filter-and-refinement-algorithm for range queries based on the fréchet distance (gis cup)," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2017, pp. 1–4.

[134] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[135] H. Taubenböck and M. Wurm, "Globale urbanisierung–markenzeichen des 21. jahrhunderts," in *Globale Urbanisierung*, Springer, 2015, pp. 5–10.

[136] B. Cohen, "Urbanization in developing countries: Current trends, future projections, and key challenges for sustainability," *Technology in society*, vol. 28, no. 1-2, pp. 63–80, 2006.

[137] S. Paldino, I. Bojic, S. Sobolevsky, C. Ratti, and M. C. González, "Urban magnetism through the lens of geo-tagged photography," *EPJ Data Science*, vol. 4, no. 1, p. 5, 2015.

[138] A. Crooks, A. Croitoru, A. Stefanidis, and J. Radzikowski, "# Earthquake: Twitter as a distributed sensor system," *Transactions in GIS*, vol. 17, no. 1, pp. 124–147, 2013.

[139] M. Veloso, S. Phithakkitnukoon, and C. Bento, "Urban mobility study using taxi traces," in *Proceedings of the 2011 international workshop on Trajectory data mining and analysis*, ACM, 2011, pp. 23–30.

[140] M. Kiermeier, M. Werner, C. Linnhoff-Popien, H. Sauer, and J. Wieghardt, "Anomaly detection in self-organizing industrial systems using pathlets," in *Proceedings of the 18th Annual International Conference on Industrial Technology (ICIT)*, IEEE, 2017, pp. 1226–1231.

[141] C. Chow, "On optimum recognition error and reject tradeoff," *IEEE Transactions on information theory*, vol. 16, no. 1, pp. 41–46, 1970.

[142] C. Elkan, "The foundations of cost-sensitive learning," in *International joint conference on artificial intelligence*, Lawrence Erlbaum Associates Ltd, vol. 17, 2001, pp. 973–978.

[143] T. Pietraszek, "Classification of intrusion detection alerts using abstaining classifiers," *Intelligent Data Analysis*, vol. 11, no. 3, pp. 293–316, 2007.

[144] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[145] B.-G. HU and Y. WANG, "Evaluation criteria based on mutual information for classifications including rejected class," *Acta Automatica Sinica*, vol. 34, no. 11, pp. 1396–1403, 2008.

[146] H. Bao-Gang, H. Ran, and Y. Xiao-Tong, "Information-theoretic measures for objective evaluation of classifications," *Acta Automatica Sinica*, vol. 38, no. 7, pp. 1169–1182, 2012.

[147] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, vol. 7, no. 2, pp. 155–162, 1964.

[148] C. W. Granger and R. Ramanathan, “Improved methods of combining forecasts,” *Journal of Forecasting*, vol. 3, no. 2, pp. 197–204, 1984.

[149] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, “Online passive-aggressive algorithms,” *Journal of Machine Learning Research*, vol. 7, no. Mar, pp. 551–585, 2006.

List of Own Publications

[P1] C. O. Dumitru, G. Dax, G. Schwarz, C. Cazacu, M. C. Adamescu, and M. Datcu, "Accurate monitoring of the danube delta dynamics using copernicus data," in *SPIE Remote Sensing*, 2019, pp. 1–13. [Online]. Available: `https://elib.dlr.de/129121/`.

[P2] C. O. Dumitru, G. Schwarz, G. Dax, V. Andrei, D. Ao, and M. Datcu, "Active and machine learning for earth observation image analysis with traditional and innovative approaches," in *Principles of Data Science*, ser. Transactions on Computational Science and Computational Intelligence, H. R. Arabnia, K. Daimi, R. Stahlbock, C. Soviany, L. Heilig, and K. Brüssau, Eds., Cham: Springer International Publishing, 2020, pp. 207–231, ISBN: 978-3-030-43980-4. DOI: `10.1007/978-3-030-43981-1_10`.

[P3] G. Dax, M. Laass, and M. Werner, "Genetic algorithm for improved transfer learning through bagging color-adjusted models," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, IEEE, 2021, pp. 2612–2615, ISBN: 978-1-6654-0369-6. DOI: `10.1109/IGARSS47720.2021.9554380`.

[P4] G. Dax and M. Werner, "Information-optimal abstaining for reliable classification of building functions," *AGILE: GIScience Series*, vol. 2, pp. 1–10, 2021. DOI: `10.5194/agile-giss-2-1-2021`.

[P5] G. Dax and M. Werner, "Trajectory similarity using compression," in *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*, IEEE, 2021, pp. 169–174, ISBN: 978-1-6654-2845-3. DOI: `10.1109/MDM52706.2021.00035`.

[P6] M. Ghiglione, A. Raoofy, G. Dax, G. Furano, R. Wiest, C. Trinitis, M. Werner, M. Schulz, and M. Langer, "Machine learning application benchmark for satellite on-board data processing," in *European Workshop on On-Board Data Processing*, 2021. DOI: `10.5281/zenodo.5520877`.

[P7] A. Raoofy, G. Dax, M. Ghiglione, M. Langer, C. Trinitis, M. Werner, and M. Schulz, "Benchmarking machine learning inference in fpga-based accelerated space applications," in *Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware*, 2021.

[P8] G. Dax and M. Werner, "The role of compression in spatial computing," in *PhD Colloquium of the DGK Section on Geoinformatics 2022*, Braunschweig, 2022. [Online]. Available: `https://www.geoinfo.uni-bonn.de/DGKGeoinfo2022/pdf-dokumente/09_DGKGeoinfo2022_DAX-WERNER_paper_2684.pdf`.

[P9] M. Ghiglione, V. Serra, A. Raoofy, G. Dax, C. Trinitis, M. Werner, M. Schulz, and G. Furano, "Survey of frameworks for inference of neural networks in space data system," in *DASIA 2022*, 2022.

[P10] A. Raoofy, G. Dax, V. Serra, M. Ghiglione, M. Werner, and C. Trinitis, "Benchmarking and feasibility aspects of machine learning in space systems," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, L. Sterpone, A. Bartolini, and A. Butko, Eds., New York, NY, USA: ACM, 2022, pp. 225–226, ISBN: 9781450393386. DOI: `10.1145/3528416.3530986`.

[P11] G. Dax, S. Nagarajan, H. Li, and M. Werner, "Compression supports spatial deep learning," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 16, pp. 702–713, 2023, ISSN: 1939-1404. DOI: `10.1109/JSTARS.2022.3226563`.

[P12] A. Koch, G. Dax, M. Petry, H. Gomez, U. Raoofy Amir Saroliya, M. Ghiglione, G. Furano, M. Werner, C. Trinitis, and M. Langer, "Reference implementations for machine learning application benchmark," in *European Data Handling and Data Processing Conference (EDHPC 2023)*, 2023.

[P13] A. Koch, M. Petry, M. Ghiglione, A. Raoofy, G. Dax, G. Furano, M. Werner, C. Trinitis, and M. Langer, "Machine learning application benchmark," in *20th ACM International Conference on Computing Frontiers: Special Session on Computer Architectures in Space — CompSpace '23*, ACM, 2023, ISBN: 979-8-4007-0140-5/23/05. DOI: `10.1145/3587135.3592769`.

Project Reports

[T1] A. Raoofy, G. Dax, V. Serra, M. Ghiglione, M. Werner, C. Trinitis, R. Wiest, M. Schulz, and M. Langer, “Project mlab: Dataset and model selection document,” Airbus, Project Report 1.2, Jan. 2022.

[T2] A. Raoofy, G. Dax, V. Serra, M. Ghiglione, M. Werner, C. Trinitis, R. Wiest, M. Schulz, and M. Langer, “Project mlab: Ml algorithm submission description document,” Airbus, Project Report 1, Jun. 2022.

[T3] A. Raoofy, G. Dax, V. Serra, M. Ghiglione, M. Werner, C. Trinitis, R. Wiest, M. Schulz, and M. Langer, “Project mlab: Ml benchmarking description document,” Airbus, Project Report 1, Jun. 2022.