TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Faculty of Computer Science

Automatic image recognition in upload filters - computing of transparent decisions (XAI), with the help of Deep Learning methods

Bachelor thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Business Information Systems and Management

Author:

Timo Bohnstedt

Student ID: 301 8484

Assessor:	Prof. Dr. Alfred Holl
Supervisor:	Prof. Dr. Florian Gallwitz

Hinweis: Diese Erklärung ist in alle Exemplare der Abschlussarbeit fest einzubinden. (Keine Spiralbindung)

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Bohnstedt Vorname: Timo Matrikel-Nr.: 3018484

Fakultät: Informatik Studiengang: Wirtschaftsinformatik

Semester: Sommersemester 2020

Titel der Abschlussarbeit:

Automatic image recognition in upload filters - computing of transparent decisions (XAI), with the help of Deep Learning methods

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Erlangen, 17.07.2020 

Ort, Datum, Unterschrift Studierende/Studierender

Erklärung zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit ☒ genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

☐ genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von 5 Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Erlangen, 17.07.2020 

Ort, Datum, Unterschrift Studierende/Studierender

Formular drucken

Datenschutz: Die Antragstellung ist regelmäßig mit der Speicherung und Verarbeitung der von Ihnen mitgeteilten Daten durch die Technische Hochschule Nürnberg Georg Simon Ohm verbunden. Weitere Informationen zum Umgang der Technischen Hochschule Nürnberg mit Ihren personenbezogenen Daten sind unter nachfolgendem Link abrufbar: **https://www.th-nuernberg.de/datenschutz/**

Abstract

Deep learning models boast remarkable predictive capabilities, and they will work on image recognition tasks within upload filters. But what else can these models tell about their decisions? If these models are used in practice, they shall not only be work excellent but be transparent. And yet the task of making models transparent appears underspecified if it comes to the usage of these models in upload filters. Papers provide diverse solutions for transparent models and offer myriad notions of what attributes render models transparent. In this paper, I seek to refine the discourse on transparent models for image recognition. First, the motivation and underlying interest in transparent models get examined, finding them to be diverse. Then, model properties and techniques thought to confer transparency gets addressed, identifying SHAP values as a suitable method of doing so. Based on a prototype, the feasibility of an explainable model within an upload filter gets discussed. Furthermore, the assertion that explainable artificial intelligence can improve the user experience of upload filters within social media gets questioned.

Preface

Machine Learning has grown large in both research and industrial applications, especially with the success of deep learning and so-called neural networks, so extensive that its impact and possible after-effects are unpredictable. Interpretability, explainability and transparency of machine learning algorithms have thus become pressing issues. For this reason, I started to gain intrinsic motivation about machine learning, deep learning and explainable artificial intelligence.

I first came into contact with Data Science through an IT project. I modified a remote-controlled car so that it was possible to control it with a Raspberry Pi. The final step was to implement a neural network for autonomously driving. The whole project was very challenging. But I had a lot of fun, and my passion was awakened. My knowledge and motivation were immediately tested in Hong Kong, while I focused on machine learning during my semester abroad. The mathematical theorems that were expected to be solved were genuine hurdle because of my gaps in mathematics. Tanks to the enormous effort, however, I achieved huge breakthroughs, was able to close the gaps, master the tasks and finally shine at the practical programming part.

This experience strengthened my wish to specialize further in this direction. Throughout this thesis, I will question the oft-made assertions that linear models are transparent and that deep neural networks are not. I think this is important since there is a lot of ambiguity when it comes to the use of deep learning methods within upload filters.

Contents

Chapter 1.

Introduction

> "Most times, the way isn't clear, but you want to start anyway. It is in starting with the first step that other steps become clearer."
>
> — Israelmore Ayivor, Leaders' Frontpage: Leadership Insights from 21 Martin Luther King Jr. Thoughts

1.1. Upload Filter in the public discussion

The "Directive on Copyright in the EU Digital Single Market", which came into force on the 7th of June in 2019, introduced a requirement for an automated upload filter. The discussion refers to Article 17 (previously Article 13) because it forces content-sharing providers such as Facebook, Google and YouTube to be responsible for copyright violations. From this point on, it is no longer possible to check contents manually [Woollacott, 2019].

Figure 1.1 shows protesters in Berlin. For them, the fear of an upload filter is real. They are concerned that these filters can harm their business as small content creators or their freedom of speech.

Different parties, organisations and scientists are arguing for and against the copyright directive. For example, the Christian Democratic Union (CDU) and its representative at the European Parliament Axel Foss support the directive. He said that it protects the freedom of expression on the internet and a diverse media landscape [Europarl, 2017]. Critics as Julia Reda from the Pirate Party (also a member of the European Parliament) denies this statement. She described it as a change into a dark future for internet freedom, when upload filters would check every content automatically [Woollacott, 2019].

Figure (1.1): In Berlin, opponents of the copyright directive are protesting. They fear that Article 13 will lead to upload filters, which they see as a danger considering different aspects [Tagesspiegel, 2019].

More precisely, she assumes that technology does not know the nuances of particular copyright law yet. And even more, she fears that small content creators are not able to interpret the decisions of the upload filters [Reda, 2019]. Dr Gallwitz is a professor at the Technical University of Nuremberg where he is researching in the field of pattern recognition. He criticised that there is no smart software that intelligently recognises patterns [Gallwitz, 2019].

Instead of using smart software, a real-world upload filter is realised through fingerprinting. This technique allows comparing blocked content with the latest user-generated content. The hardware requirements which are needed to examine the contents are proportionally small. Fewer hardware requirements and a lack of transparency and reliability are reasons why most companies still use older techniques (e.g. hash values), even they can develop smart software for upload filters [Spoerri, 2019] [Wagner, 1983].

The Microsoft software Photo-DNA identifies child pornography with the fingerprinting method, e.g. if an image in Microsoft's database is marked as pornographic. That means YouTube can detect when someone wants to upload an image to the platform which contains offensive content [Microsoft, 2013]. The Software Content ID uses the same technique to identify copyright offences. It is used and developed by YouTube. That shows how technology companies are responsible for the methods of choice when it comes to uploading filters. Even though scientific research has shown that different approaches are performing

well, the companies have to see this decision from a business point of view [YouTube, 2010].

Using deep learning models as upload filters in the future could be possible, but among this usage are many problems. One of them is to get labelled data. Therefore, labelled data is necessary to train a machine learning model, e.g. which can detect patterns from images. Another problem is to define the patterns or the labels itself [Waltermann and Hess, 2019]. Labels are categories that are assigned to an image [Goodfellow et al., 2016] e. g. attach the label cat to an image with a cat. In the case of copyright violations or offensive content, any item that has labels with copyright content will be blocked. A machine learning algorithm which can detect every offensive content must be very complex because the complexity from machine learning algorithms is increasing with the number of growing input features and an increasing number of possible labels [Yao et al., 2017]. In machine learning, a feature can be any input which can feed into a machine learning algorithm. For example, an image with 32 x 32 pixels has 1024 input features because every pixel is a feature [Goodfellow et al., 2016].

1.2. Upload filters and Explainable Artificial Intelligence

As mentioned in the introduction to the Directive on EU Copyright and Upload Filters (Section 1.1), a machine learning algorithm, which is used to detect copyright infringements would be very complex. So, it is not often used in practice right now. Joel Dudley, director of the Institute for Next-Generation Healthcare at the Ichan School of Medicine, said: "We can build these models, but we don't know how they work [Knight, 2019]. His statement was part of an interview with the journalist Will Knight from MIT's journal Technology Review about explainable artificial intelligence. A precise definition of explainable artificial intelligence does not exist. The idea is that we use a machine learning algorithm to make decisions which must be in natural language. For example, if we use an image as an input for a machine learning algorithm, the task would be to label the images with a label for cancer or no cancer. A doctor who wants to use this information could be interested in the question of why the algorithm predicts a cretin label for this image [Samek et al., 2017].

Apart from the Technology Review, newspapers like The New York Times, Financial Times and The Register have also reported that an explainable artificial intelligence is necessary when a machine learning algorithm is used for automatic decision making [Kuang, 2017] [Robinson, 2017] [Waters, 2017]. David Gunning explains in his research project about explainable artificial intelligence that machine learning can bring a huge benefit to the transportation sector, as well as to the finance, security, legal, medicine and military sector. Gunning claims that algorithms are limited because they can not explain their actions and

decisions to users in an understandable way. He assumes explainable artificial intelligence will be essential if users want to understand, trust, and effectively manage machine learning algorithms in their applications [Gunning, 2019]. Figure 1.2 shows an example of how transparent upload filters can be applied to uploaded images. When we build an upload filter with deep learning methods, it has to be an explainable artificial intelligence [Waltermann and Hess, 2019]. Otherwise, there will be no explanation of how the decision from an upload filter was made.

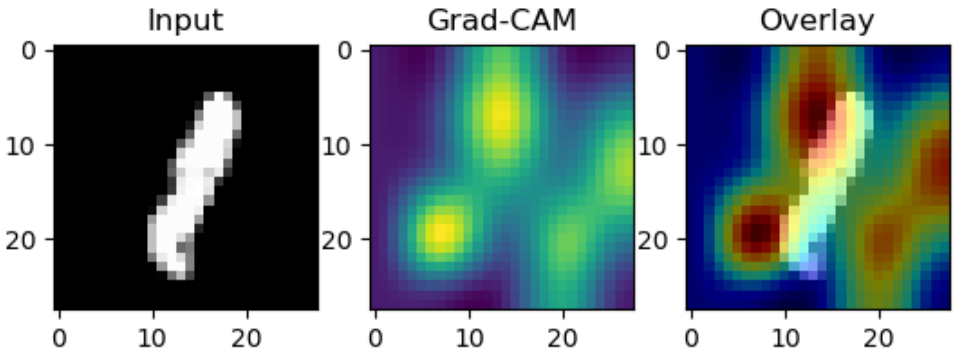


Figure (1.2): Example of how explainable artificial intelligence can be applied to uploaded images [Versloot, 2019]

1.3. The Objective of this Thesis

Even though automatic image recognition with an explainable artificial intelligence is just a research topic, it is important to evaluate how to generate transparent decisions (XAI). So, the main objective of this thesis is to find an answer to the following question:

> *"How can automatic image recognition be implemented in such a way that the resulting decisions are transparent?"*

For this purpose, a prototype will be developed, which is able to recognise patterns on images (automatic image recognition) automatically. If a pattern is recognised by the algorithm, it assigns a label to the image. In the pre-field, it gets defined which patterns shall recognised by the deep learning algorithm.

Afterwards, the results of the algorithm are presented in such a way that the decisions become transparent (explainable artificial intelligence). What transparent actually means is hard to tell, because it depends on the algorithms and techniques, which are used to implement the filter. Anyway, for this thesis, transparent shall be defined as follows:

> *Imagine a social media platform like Instagram, where everybody shares its images. If someone shares an image and it is getting blocked by a filter (e.g. because of copyright violation), the user should be able to understand why the decision was made. The explanation can be an automatically generated text, which says that the algorithm had detected a copyright issue and it highlights specific parts in the blocked image. The highlighted details tell the user why the algorithm came to a decision.*

In contrast to my example, a real-world application is much more complicated. First, in a real-world application, has more labels. Second, it is not only possible to upload images. Instead, also text and sound have to be checked. This is why the application is only a prototype. Anyway, the research question will be answered anyways at least from a theoretical perspective. In the end, the prototype is used to transfer the insights gained through the prototype to a more general scenario.

1.4. Thesis Structure

The Introductions objective is to present context information around the topic and to resent the research question, the structure and related work (Chapter 1). While the terms were already briefly introduced in the introduction the Background and Theories chapter objective to present the terms and the functions in concern to the research question (Chapter 2). The Requirements chapter is a description of the possible scope of the prototype and a collection of the functional requirements for the prototype (Chapter 3). The Functional Specifications chapter aims to present set of functional specifications which can be used to decide how the technical side of the implementation will be done (Chapter 4). The IT-Specifications shall present a collection of instruments for the prototypes implementation which will be used together with the functional specification set to implement the prototype (Chapter 5). The evaluation shall present is an overview of the number of fulfilled and unfulfilled requirements of the prototype (Chapter 6). The Generalization chapter is an analysis of the obtained evaluation results in a more general context (Chapter 7). The Conclusion is a personal statement regarding the results of the evaluation (Chapter 8).

The Methodological Table (Figure 1.3) also presents the thesis structure. It was prepared to get an overview of the topic and was created with the help of Prof. Dr. Alfred Holl. It was beneficial to bring the questions of interest in a structured and logical order.

Chapters	Questions of interest	Methods	Objectives
	1. How can automatic image recognition be implemented so that the following decisions become transparent (XAI)? It shall be decided whether or not there is a specific pattern on an image (upload filter).	Case study, prototyping	Erkenntnisse über die Implementierung von Uploadfiltern mittels XAI.
Introduction	1. Why is there a discussion about the use of upload filters within social media platforms? 2. How can explainable artificial intelligence improve the user experience of upload filters within social media? 3. What is the objective of this thesis? 4. How to proceed within this thesis to reach its objective? 5. What results have already been published covering the thesis objective?	Literature research	Context information around the topic and an introduction of the basic concepts. An introduction to the research question, the structure and related work.
Background and Theory	1. What steps in the past leads to the usage of upload filters today? 2. How can supervised machine learning used for and upload filter implementation? 3. How can deep learning methods help to improve supervised machine learning methods for upload filters? 4. How convolutional neural networks help to improve deep learning methods for upload filters? 5. How can explainable artificial intelligence help to make convolutional neural networks more transparent? 6. How does a specific explainable artificial intelligence method work in detail? 7. What the results of a specific explainable artificial intelligence method mean?	Literature research	The terms were already briefly introduced in the introduction. In the current chapter, the terms and the functions behind them will be described in greater detail.
Requirements	1. What is the objective of the prototype? 2. What kind of data will feed into the prototype implementation? 3. What happens with the data within the prototype implementation? 4. What is the output data of the prototype? 5. How can the prototype be evaluated?	Use of the results worked out so far. A personal selection of the collected requirements during the literature research.	Description of the possible scope of the prototype and collection of the functional requirements for the prototype.
Functional-Specifications	How the corresponding theory sections and the requirements can be linked together to get the expected results for the following points: 1. To get the input data? 2. To divide the data into different parts to evaluate the prototype implementation? 3. To implement a classifier which can recognize cats and dogs on images? 4. To use the classifier to get predictions on unseen images? 5. To use the classifier and XAI to make the decisions transparent?	Create functional specfications Literature research	Get a functional specification which can be used to decide who the technical side of the implementation will be done.
IT-Specifications	1. What is the prototypes implementation structure to transform the functional-specifications into practice? 2. What is the primary Programming language to transform the functional-specifications into practice? 3. What libraries are used to transform the functional-specifications into practice? 4. What software libraries used to get transparent decisions? 5. What is essential than it comes to implement the prototype?	Literature research	Collection of instruments for the implementation of the prototype with the help of the functional-specifications.
Evaluation	1. How does the prototype perform when it comes to recognizing dogs and cats on images? 2. How does the prototype perform when it comes to computing metries which are essential to make the decisions made by the prototype transparent? 3. How does the prototype perform when it comes to visualizing a model transparently?	Target-performance comparison of the techniecal concept with the finished prototype.	Overview of the number of fulfilled and unfulfilled requirements of the prototype (quality).
Generalisation	1. How does the performance of the prototype affect the statement that image recognition will be a common task in the future? 2. How does the performance of the prototype affect the statement that methods for explainable artificial intelligence can increase the user experience of an upload filter? 3. How does the performance of the prototype affect the statement that methods for explainable artificial intelligence can support user which are affected by decisions made by deep learning models?	Generalisation from the results obtained so far.	Analysis of the obtained evaluation results in a more general context.
Conclusion	1. What is my interpretation of the results? 2.How do I think they will affect the feature of deep learning or upload filters?	Presentation of my own opinion	My personal conclusion regarding the results of the evaluation.

Figure (1.3): Methodological table which was prepared in the run-up to the bachelor thesis in cooperation with the assesor Dr Prof Alfred Holl.

1.5. Related Work

Katharina Blandina Weitz master thesis "Applying Explainable Artificial Intelligence for Deep Learning Networks to Decode Facial Expressions of Pain and Emotions" is about programmes used in hospitals, which detect pain in facial expressions. According to her, this software should use deep learning to support human practitioners. In her thesis, Ms Weitz finds out that these deep learning methods work well in recognising pain in human facial expressions, but they are hard to interpret. From her abstract, it can be seen that explainable artificial intelligence does not depend on specific data sets. Rather, according to her statement, the methods can be applied to any dataset that was previously was processed by deep learning before[Weitz, 2018]. As described in the corresponding section the contexted will be changed in this thesis (Section 1.3).

The Defence Advanced Research Projects Agency is an agency of the United States Department of Defence. They focus on the development of emerging technologies that the military can use in the future. Because both the number of applications and the complexity of machine learning models have increased, a project has been started to specialise in explainable artificial intelligence. It aims for more transparent models while maintaining a high level of prediction accuracy. Furthermore, the program wants to enable human users to understand, appropriately trust, effectively manage the output of a machine learning algorithm [Robinson, 2017]. The Defense Advanced Research Projects Agency presents its projects as a concept (Figure 1.4). The concept aims to move away from models which human users cannot understand towards more transparent models. A step towards to transparent models is also the main objective of the prototype implementation (Section 1.3).

Of course more scientific papers are published on explainable artificial intelligence, but to present a complete bibliography is not the objective (Section 1.3).

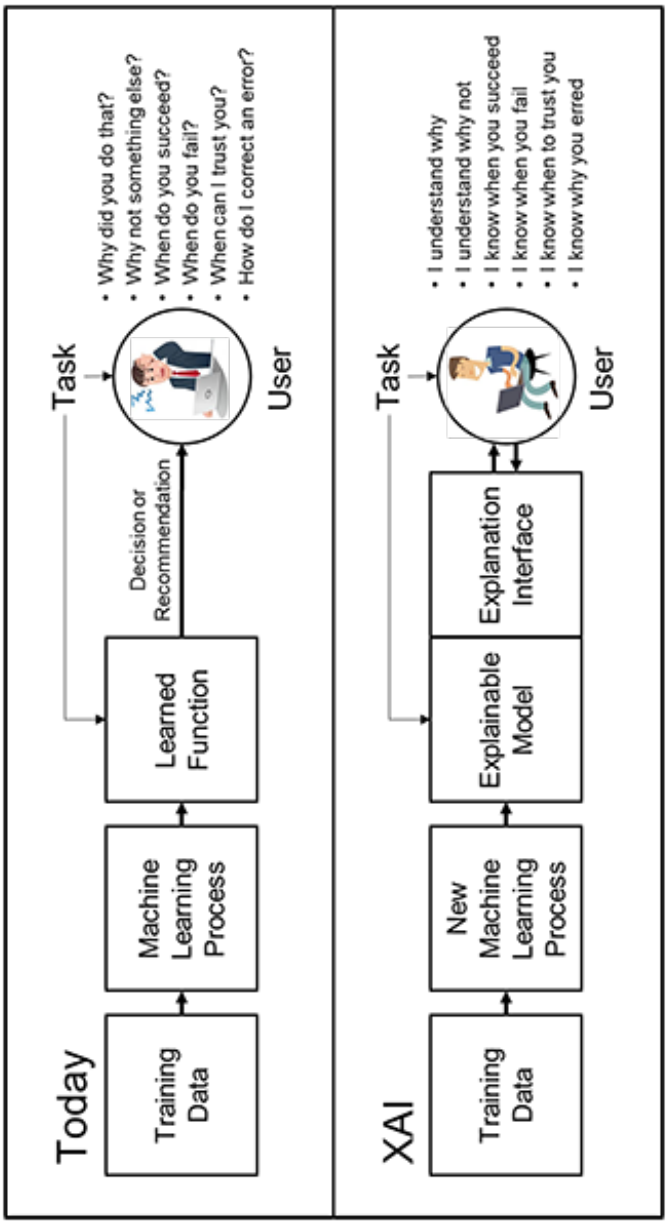


Figure (1.4): Explainable artificial intelligence concept of the Defense Advanced Research Projects Agency [Robinson, 2017]

Chapter 2.

Background and Theory

> "Math is like water. It has a lot of difficult theories, of course, but its basic logic is very simple."
>
> - Haruki Murakami, IQ84

This chapter is a summary of all the fundamental aspects of machine learning. More specifically, the crucial elements for an understanding of the prototype-functionality. First of all, the historical development of machine learning will be covered briefly. As soon as the reader has an understanding of the historical development of machine learning aspects and algorithms of machine learning, which are relevant to talk about the topic of this thesis are discussed. In the final part of this chapter, two different approaches of explainable artificial intelligence are introduced. Furthermore, a specific kind explainable artificial intelligence is mentioned, such that the reader can understand how predictions can become transparent.

2.1. History of Machine Learning

An example of machine learning is to take a lot of images and try to recognize cats (patterns) on them. The machine learning algorithm would be able to predict whether it is a cat or not. In this example, machine learning can be seen as

> *"[...] a set of methods which can automatically detect patterns in data, and then use the uncovered patterns to predict unseen data or to perform other kinds of decision making under uncertainty" [Murphy, 2012, p. 1].*

But there is more than one definition which describes machine learning. Arthur Samuel was the first researcher who used this term and defined it as "Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed"

[Samuel, 1959]. The computer scientist Mitchel provided a more formal definition of machine learning, which is quoted in hundreds of papers[1]. In his book from 1997, he says "A computer program is said to learn from experience E concerning some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [Mitchell, 1997].

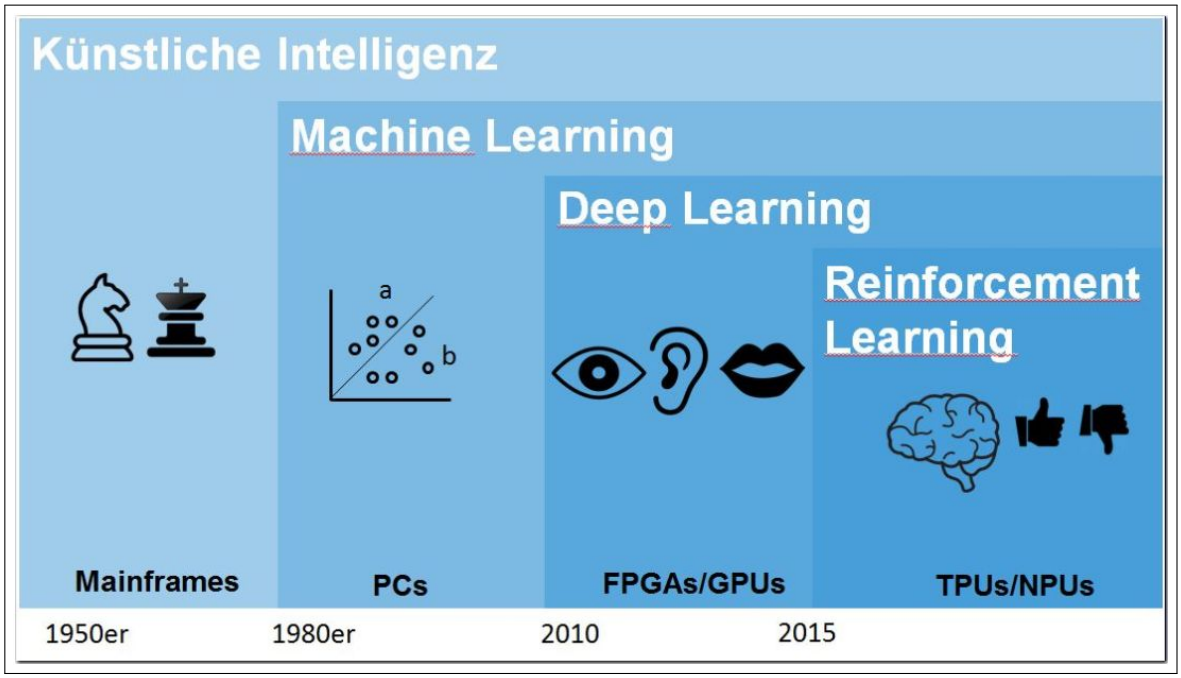


Figure (2.1): Determining critical technologies within the domains of artificial intelligence(extended representation by Dmitri Gross according to a presentation from Michael Copeland) [Gross, 2017] [COPELAND, 2017]

According to the science magazine, the most influential computer scientist, in the field of pattern recognition, Michael Jordan wants to give another modern definition. His objective is to give a neutral statement, which defines the word artificial intelligence. He says:

> *"It is one of today's rapidly growing technical fields, lying at the intersection of computer science and statistics, and at the core of artificial intelligence and data science" [Bohannon, 2016] [Jordan and Mitchell, 2015]*

Furthermore, with this definition, Jordan wants to connect the terms of machine learning and statistics. In his opinion, these two terms are the essential components of artificial intelligence [Michael Jordan, 2018]. Machine learning has changed over the last decades. Machine learning became popular in the eighties, as a result of the increasing use of personal computers. Furthermore, deep learning gained on popularity until 2010, caused by the development of more powerful graphical processor units. This statement is criticized as

[1] 998 results on google scholar while searching for the original meaning from Mitchel (date: 24 Jan 2020)

well. Mainly because of the unclearly defined terminology. Anyways it is useful to get an overview of the field and the historical evolvement over the last years (Figure 2.1).

Figure (2.2): The Netflix documentary Alpha Go is part of the hype around deep learning methods. Critics claim until the problem of transparency is not solved this results are not crucial because these models can not be used in a real-world application [Rocke, 2019]

Alpha Go, which is developed by the Google Research Laboratory, is a good example of the popularity of deep learning methods. It shows how an enormous amount of time and hardware resources can lead to success as the artificial player has been the world ranked number one in the board game Go. But this success is criticized as well, because it is not a real-world application and gots pushed by a Netflix documentary, scientist claim that it is just an overhyped topic as long as the model could not explain its decisions (Figure 2.2).

2.2. Suprvised Machine Learning for Upload Filters

By doing the following steps, a so called cost function (Definition 2.2.5) will be optimized to get a prediction on unseen data, as defined by Mitchel (Section 2.1). An example application is an upload filter which determines if a dog or a cat is on an image:

1. Eliminate empty entries and statistically irrelevant data e.g. duplicates
2. Rescale the input data in the required form
3. Split the Data into a test and a training set
4. Initialize the parameters of the model
5. Learn the parameters for the model by minimizing the cost function (Definition 2.2.5)
6. Use the learned parameters to make predictions (Definition 2.2.1)
7. Test the ability to predict patterns on unseen data correctly

In the following, these steps are described in greater detail. The example of an Instagram upload filter is a classification task which can be solved by a supervised machine learning algorithm. It is designed to learn by example, [Murphy, 2012, p. 3 - 4]. Examples of cats and suitable counterexamples are assumed to be given before a supervised machine learning algorithms get developed. (Figure 2.3).

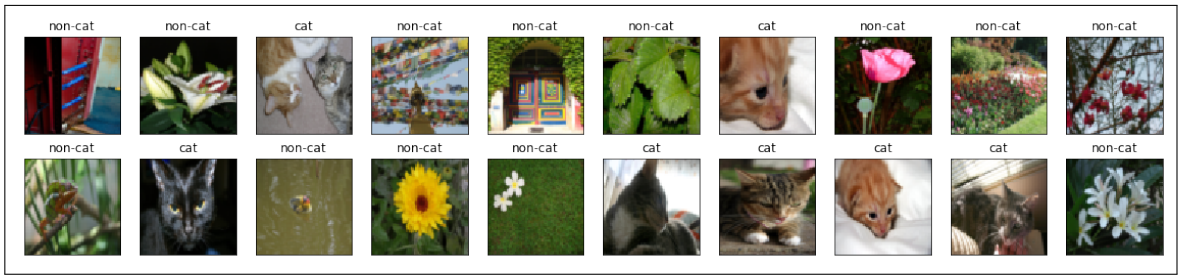


Figure (2.3): Examples sampled from the the ImaegNet data set. The pattern on these images are similar to the patterns which the prototype shall recognise [Chollet, 2016].

The name supervised learning comes from the idea that training this type of algorithm is like having a supervisor which observes the whole learning process, e.g. recognize cats on images after having seen enough samples of the same distribution. At the same time, a "trainer" leads the algorithm to the correct result [Goodfellow et al., 2016, p. 103] [Murphy, 2012, p. 3]. Technically spoken, these input data is called training data. First, training data consists of input data, for example, values that represent an image. Second, training data consists of output data, for example, the class which an image belongs to. During training, the algorithm will search for patterns in the input data that correlate with the desired outputs. After training, the supervised learning algorithm will feed with new unseen examples such that its ability to predict unseen data can be tested. So, the algorithm determines which labels correspondent to the unseen examples, based on optimized parameters. Such an algorithm can be expressed as follows [Murphy, 2012, p. 3]:

Definition 2.2.1

$$f(X) = \hat{y}$$

where

$f()$ = Mapping function which assigns the labels to a given input.
X = A matrix of input value where each column in X stands for a single example.
$\hat{y}$ = The determined labels for each column in X represented by a vector.

Where $\hat{y}$ is the predicted output, which is determined by a mapping function f that assigns a label to a single value or a set of multiple-input values donated by X, the function connects an input features to a probability to belong to a certain class. The logic behind this function is also called machine learning model [Murphy, 2012, p. 3].

Before a machine learning algorithm gets trained, the training data has to be prepared. This step is called preprocessing. An example of this is a machine learning algorithm which detects cats on images. Therefore training data is required to be preprocessed. This means the data has to be prepared such it fits into the mapping function. In the example of an machine learning algorithm which detects cat and dogs, this means the images transformed such that they are represented by a matrix. Within this matrix, every column stands for an example. Furthermore, the corresponding labels shall be in the form of a vector. Therefore, every value within this vector corresponds to a column in the input matrix. The vector represents the labels by zeros and ones. For example, a zero at the first position in the output vector means that the first column in the input matrix shows a cat [Brownlee, 2019].

When talking about the training of a machine learning algorithm, the optimization of a function is always meant by that. Further, the function to be optimized is the mapping function which assigns an output value to a given input. Therefore the parameters of the function will be optimized during the training process, such that the output values match the actual output values as closely as possible. In other words, such that the real values as close as possible to the predicted values (Definition 2.2.5). Usually, that is an optimization problem and is solved with different techniques. The technique also determines the required form of the input and output data, as suggested in step two. If we used a logistic regression to classify cats, a structure as can be seen in Definition 2.2.2 is necessary. The logistic regression is a model which is applied to determine the probability of a certain class or event exists such as pass/fail, win/lose, alive/dead or healthy/sick. It can be extended to predict several classes of events such as determining whether an image contains a cat, dog, lion, etc.

Definition 2.2.2

$$X \in \mathbb{R}^{n \times m}$$
$$y \in \mathbb{R}^{m}$$

where

X = A matrix of input values. Each column in stands for a single example.
y = A vector with labels. Each value belongs to a column of the input matrix
n = Total amount of input features
m = Total amount of examples
$\mathbb{R}$ = All values are in real number space

A pixel image of a cat, as you can see on the left-hand side in Figure 2.4, is represented as three matrices (Figure 2.4, centre). With the help of mathematical functions from the field of linear algebra, the images are transformed from the matrix form into a vector form (Figure 2.4, right) [Brownlee, 2019, p. 276]. Even if it is not necessary, it is recommended to normalize the values. Otherwise, the calculations could become very slow and very memory intensive. [Goodfellow et al., 2016, p. 57].

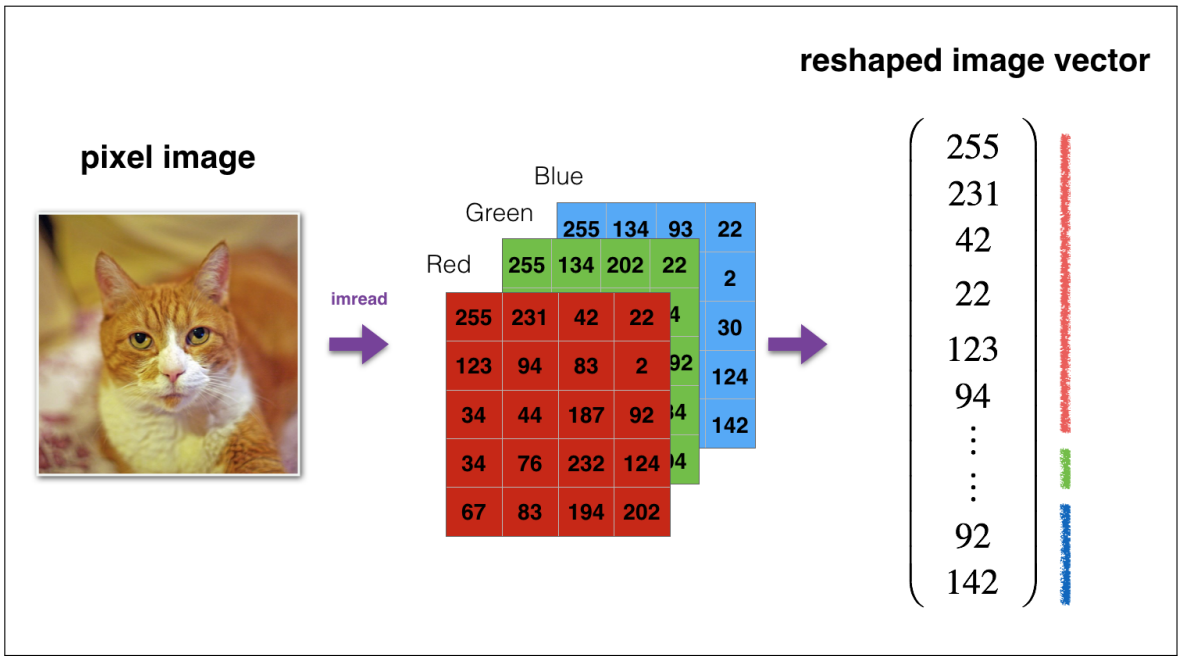


Figure (2.4): An image is represented by three matrices. Each for one colour channel (red, green and blue). After transforming it with linear algebra it becomes a vector.

The objective is to optimize the cost function (Definition 2.2.5) for many examples. This is why the input gets feed into a matrix (Definition 2.2.2). After transforming an image, we get a vector, so it is mandatory to store every single vector in the column of a matrix. Matrix representations are one of the reasons why machine learning algorithms are much more efficient. Nowadays, memory is cheaper than computing power. Without matrix multiplication, more loops are required. Therefore, loops require a relatively large amount of computing power, especially for extensive data. Matrix multiplication requires a lot of memory but requires fewer loops and therefore, less computing power. Due to the extreme amounts of data that are processed during the calculation of machine learning algorithms, these effects are multiplied by each other, which makes an efficient computation even more important [Ng, 2008].

The next step in designing a machine learning algorithm is to create a specific function which can be optimized. As a first step a linear function which looks like the following can be used:

Definition 2.2.3

$$z = WX + b$$

where

z = The results of the linear function
W = A matrix with parameters assigned to every row of the input matrix X
X = A matrix of input value where each column in X stands for a single example.
b = Intercept added in a linear equation called bias parameter.

Next step in order to get the results is to pass the results z to a sigmoid function. Every function which is used after calculating the linear function itself is a so-called activation function. Activation functions are mathematical equations to determine the output of a neural network. Furthermore, an activation function can be used for different tasks. In the pretend example the sigmoid function (Definition 2.2.4) maps a value between 0 and 1 to each output. In such a way, that this value can be interpreted as a probability (Definition 2.2.4). Finally, a label can be assigned to the input by using a rule-based approach. For example, it could be the case that if the probability is greater than 50% that a certain case will occur, a positive label will also be assigned (the probability that a cat is on the image is greater than 50%, then the image has the label cat).

Definition 2.2.4

$$\hat{y} = sigmoid(z)$$

where

$\hat{y}$ = The determined labels for each column in X represented by a vector.
z = The results of the linear function

The next step for the example of an Instagram upload filter is to get a criterion, which can be used to optimize the parameters. In other words, to quantify the success of the training process. In order to get such a criterion, a so-called error value is formed. This is calculated from the actual label (y)and the predicted label ($\hat{y}$). How exactly this calculation is done is determined by the loss function. For example the loss function could be defined by the cross-entropy function (Definition 2.2.4), which compares the given values with the

calculated values. Every function in machine learning that could be used to compute such an error can be used as a loss function as well. Finally, all error values of the individual examples are summed up and divided by the total number of examples. The function which will be used to do so is therefore called error function and denoted by $\mathcal{J}$. $\mathcal{J}$ maps a single value to all errors. Finally, the error function $\mathcal{J}$ gets optimized during the training process such that the parameters (W and b) adjusted such that the error function $\mathcal{J}$ reaches a global minimum. While in the first step, the so-called forward propagation, the error values and calculations are calculated, in the second step, the weights of the function will be optimized. This step uses the chain rule to form the gradient of the weights and adjust them according to this gradient. In the literature, the second step is commonly called backward propagation. This step is explained in greater detail in Section (2.3).

Definition 2.2.5

$$\mathcal{J} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^i, y^i)$$

where

m = Number of examples e.g. number of cat and no cat images
$\mathcal{L}$ = Cross-Entropy function
$\hat{y}^{\text{i}}$ = Calculated label for the ith example
y^{i} = Correct label for the ith example
$\mathcal{J}$ = Cost Function which can be optimized s. t. it is at its global minimum

The cost function (Definition 2.2.5) should not be optimized as close as possible to the training data. A function which fits perfectly is called over-fitted.

Overfitting means models perform well on the training data but do not generalize well for new data. It happens when the model is too complex relative to the amount and noisiness of the training data. If the accuracy on predicting patterns within the training set is high, but the accuracy on predicting patterns on images from the test set is low, the machine learning algorithm is likely overfitted. So, it is time to take corrective measure [Goodfellow et al., 2016, p. 110]. Anyway, the goal of supervised learning as in the Instagram upload filter example is to achieve high performance on unseen data. To do that the data has to be divided into a test and a training set. The training set is used like described before (preprocessing the data and approximate the cost function to its global minimum). In contrast, the test data set would only be preprocessed and then used to make predictions about unseen cat images. So it is possible to test the neural network of its ability to predict unseen data Section (2.1).

In other words, every step is necessary to train a machine-learning algorithm to predict if it is a cat or not. In Figure 2.5 is the process of predicting an $\times 64$ image visualized. Therefore the notations are explained in Definition 2.2.6.

Definition 2.2.6

x^{th} = exists for each pixel of the ith example
$W_{\text{x}} = x$th parameter assigned to each pixel of x^{th}
$Wa + b$ = the linear function which computes a first output (Definition 2.2.3)
σ = the sigmoid function to get the labels (Definition 2.2.4)

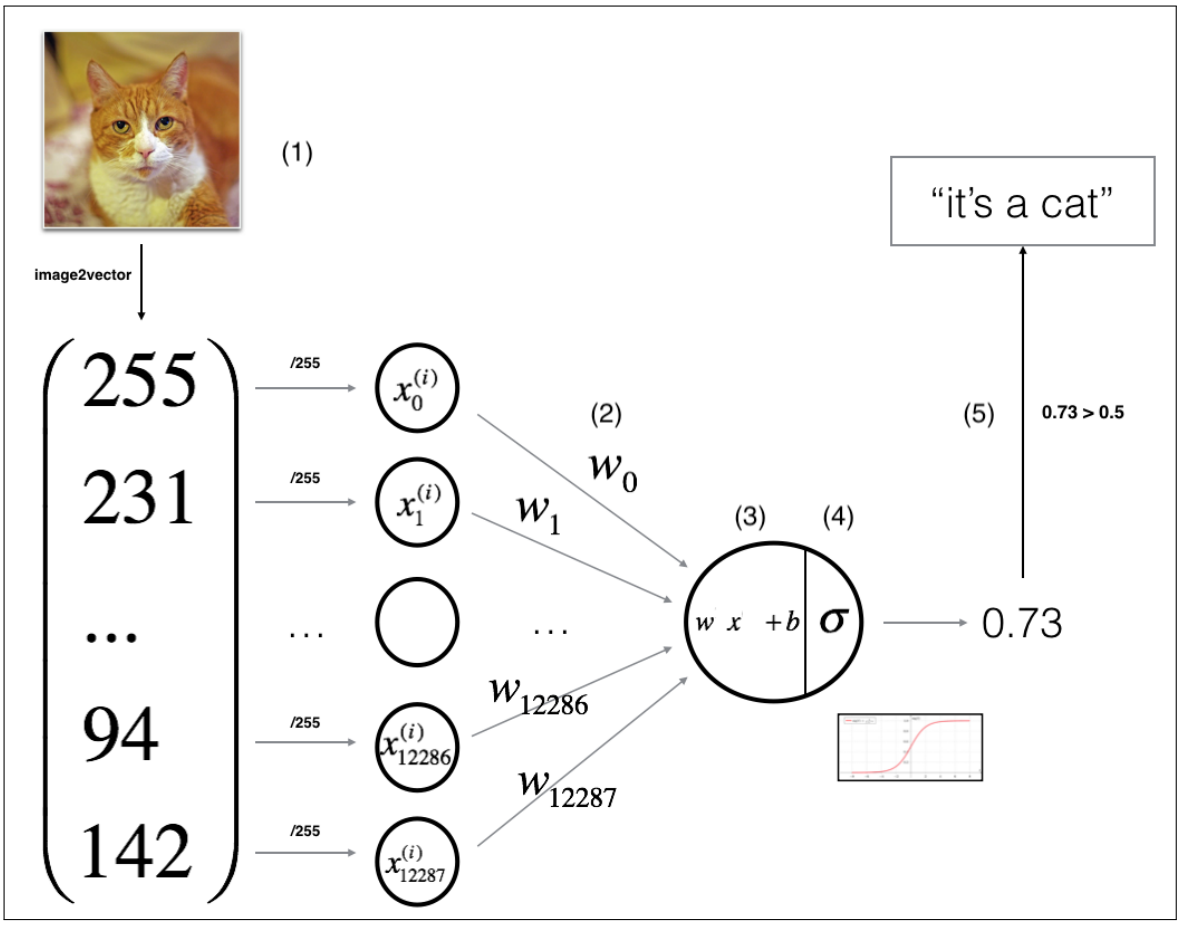


Figure (2.5): The image visualizes the whole process of using an image (1) and putting this into a single neuron (4). First, the neuron calculates a linear function and uses the result as input to a sigmoid function (becomes an interpretable value between zero and one). The network can make a prediction (5) while using a decision boundary (e.g. if the probability is higher as 0.5 it is a cat) if the forecast is not correct, the parameters of the function (2, 3) getting adjusted as long as the function is optimized. Finally, we get an approximation that fits all training examples as close as possible.

An algorithm that works like this can be considered as a simple neural network [Britz, 2015]. The term neural came from the fact that at first scientists tried to recreate the functionality of neurons in the human brain. Besides the early beginning, neural networks have not much in common with the brain because actually, it seems the brain is much more complicated as it seems before [Kriesel, 2007]. However, it is called a network because usually different neurons working together. In the simple scenario from Figure 2.5 we had just one neuron (Figure 2.6)

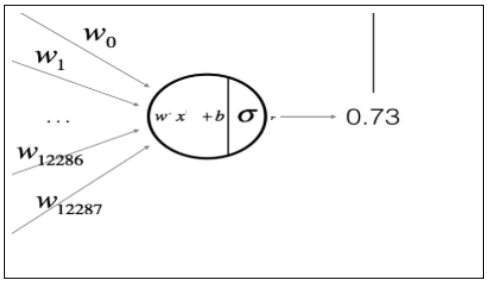


Figure (2.6): An example for a single neuron. First, a linear output will be calculated by a simple linear function with the parameters W and b. Afterwards the output will be normalized to get probabilities.

For the Instagram upload filter, such a neural network[2] achieves a test accuracy of 70% (Listing 2.1).

```
1 Cost after iteration 0: 0.693147
2 Cost after iteration 1000: 0.214820
3 ...
4 Cost after iteration 1800: 0.146542
5 Cost after iteration 1900: 0.140872
6 train accuracy: 99.04306220095694 %
7 test accuracy: 70.0 %
```

Code Listing (2.1): Test accuracy is 70% after iteration 2000 times and using 209 examples with 12287 features (64×64 pixels). This is not state of the art but very good if considering that this is a linear classifier on a high dimensional feature space.

It is crucial to achieving a higher accuracy before creating a transparent neural network which is easy to understand. Within the next section, machine learning algorithms will be introduced to get higher accuracy.

[2]The achieved accuracy is not always precisely the same, because the optimization just reaches a local minimum rather than a global minimum. In practice, this change is hardly relevant. Besides, the results can differ depending on the implementation. The results presented here were obtained with the programming language Python using frameworks from the field of data science.

2.2.1. Deep Learning Neural Networks

Figure 2.7 shows how the same network as in Figure 2.5 would look like if we would use two layers. Therefore the notations are explained in Definition 2.2.7.

Definition 2.2.7

x_{th} = exists for each pixel of the ith example
a_{th} = a so called neuron which gets the results from the first layer
$WX + b$ = the linear function which computes a first output (Definition 2.2.3)
σ = the sigmoid function to get labels (Definition 2.2.4)

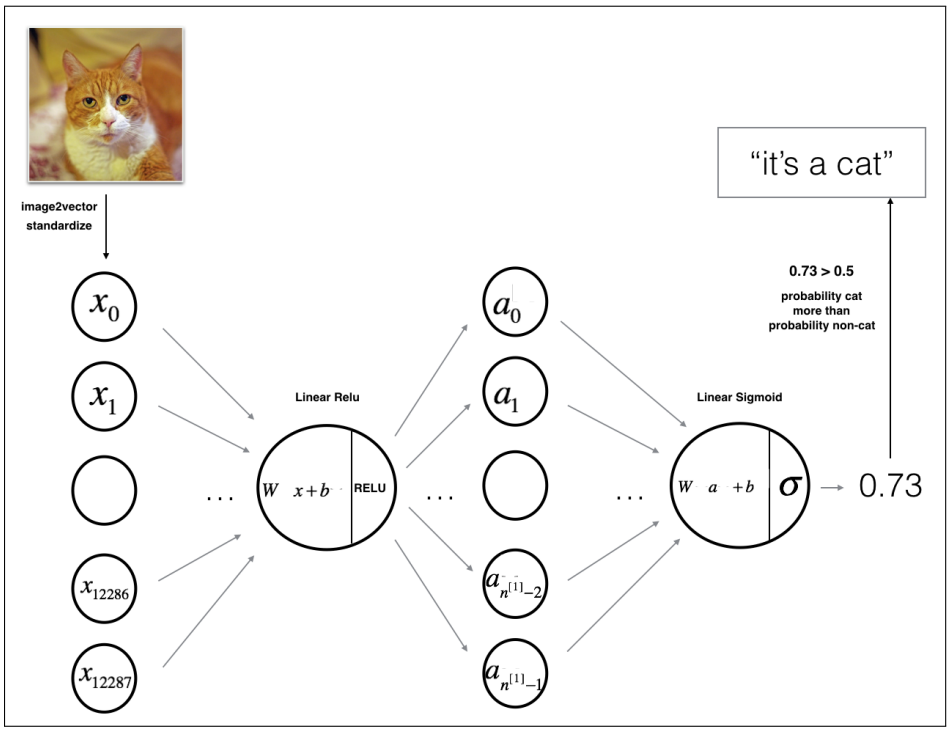


Figure (2.7): As before in the Figure 2.5 The image visualizes the whole process of using a neural network to recognise patterns of cats and dogs. Instead of just on layer now the machine learning algorithm uses two layers.

On the one hand, more layers resulting in an increasing amount of parameters which the algorithm has to optimize [Goodfellow et al., 2016, p. 21]. For example, to find the global minimum of a cost function in our Instagram upload filter example, it would take significantly more time and requires more examples. And even then it is not sure if such a global minimum exists. On the other hand, a neural network with more layers would outstand a neural network with just one layer by far. Listing 2.2 shows that the algorithm detects up to 80% of the images. So, the deep neuronal network, compared with a single layer neural network, has increased by 10%.

The results could be better if more examples were used as input data. The required time and computing capacity would be comparatively low with such an implementation. That is why the neural network is just used with the same amount of examples [Goodfellow et al., 2016, p.167] [Murphy, 2012, p.995 - 997].

In general, the term deep within deep neural networks is related to the number of layers. Andrew Ng suggests that neural networks with more than one layer should be called deep [Ng, 2008]. Furthermore, every layer which is not the output and not the output layer itself would be called the hidden layer. Whereas the output layer counts to the total amount of layers, the input layer is usually not considered as a layer. There is not an exact definition of what a layer exactly is. Most sources are excluding the input and including the output layer. Steps in which an activation function activates the neurons do not count as independent layers. They belong to the previous layer [Ng, 2008] [Kriesel, 2007] [Goodfellow et al., 2016].

A problem with the neural network, which is not deep is its capability of training with larger images. So, the network was built to recognize patterns on images with 4096 input features ((64 × 64 pixels). Suppose the input is a (300 × 300 RGB image, the first layer of the network has 90000 neurons, and each one is fully connected to the input. That means that each neurone in the previous layer is connected to each node in the following layer. The number of parameters which the training process has to optimize would be calculated with the following formula [Vasudev, 2019][Trask et al., 2015].

Definition 2.2.8

$$W_{\text{ff}} = F_{\text{-1}} \times F$$
$$B_{\text{ff}} = F$$
$$P_{\text{ff}} = W_{\text{ff}} \times + B_{\text{ff}}$$

where

W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer
B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer
P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer
F = Number of neurons in the FC Layer
$F_{\text{-1}}$ = Number of neurons in the previous FC Layer

In the equation above, $F_{-1} \times F$ is the total number of weights (connections between layers) from neurons of the previous fully connected layer, to the neurons of the current fully connected layer. The total number of biases is the same as the number of neurons (F). So, for

```
...
Cost after iteration 2300: 0.100897
Cost after iteration 2400: 0.092878
train accuracy: 98.5645933014 %
test accuracy: 80.0 %
```

Code Listing (2.2): Test accuracy is 80% after iteration 2400 times and using 209 examples with 12288 features (64×64 pixels). This is not state of the art but very good if considering that this is an algorihm wich is not spezialised to recoginize images.

the example given above (300×300 pixels, 100 Layers in the first hidden layer, RGB and fully connected) the calculation would look like this:

$$W_{\text{ff}} = 270000 \times 100 = 27000000$$
$$B_{\text{ff}} = 100$$
$$P_{\text{ff}} = W_{\text{ff}} + B_{\text{ff}} = 27000000 + 100 = 27000100$$
$$F = 100$$
$$F_{\text{-1}} = 270000 \ (300 \times 300 \text{ pixels} \times 3 \text{ colour channels})$$

If the image gets even bigger and the amount of layers increases, the weights can not be optimized with standard hardware, because the number of parameters is too big. As a guideline, there are about 14000000000 parameters where it is still possible to train without special hardware (e.g. on your own computer). Everything above this is difficult to realize on a personal computer without any hardware adjustments. Besides, the models could not predict in any case in a pleasant amount of time [Trask et al., 2015]. Instead of the introduced neural network, there are particular implementations of deep neural networks that can calculate a forecast more efficiently.

The objective of the next section is to determine how to get excellent results in a reasonable time, even if the input images are greater or equal to 300×300 pixels.

2.2.2. Convolutional Neural Networks

The most popular deep learning models leveraged for computer vision problems are convolutional neural networks. To present the convolutional neural networks, the example given at the beginning of this chapter will be changed. Instead of cats, the network shall now recognize ten different digits which are simulated with the hands as you can see in Figure 2.9, where:

- **y** is a vector with the length of the possible outputs, e.g. five-digit imitating means the length of a single output vector would be five

- and each element of **y** is a binary value which determines if the input belongs to a class on the i^{th} position within the vector

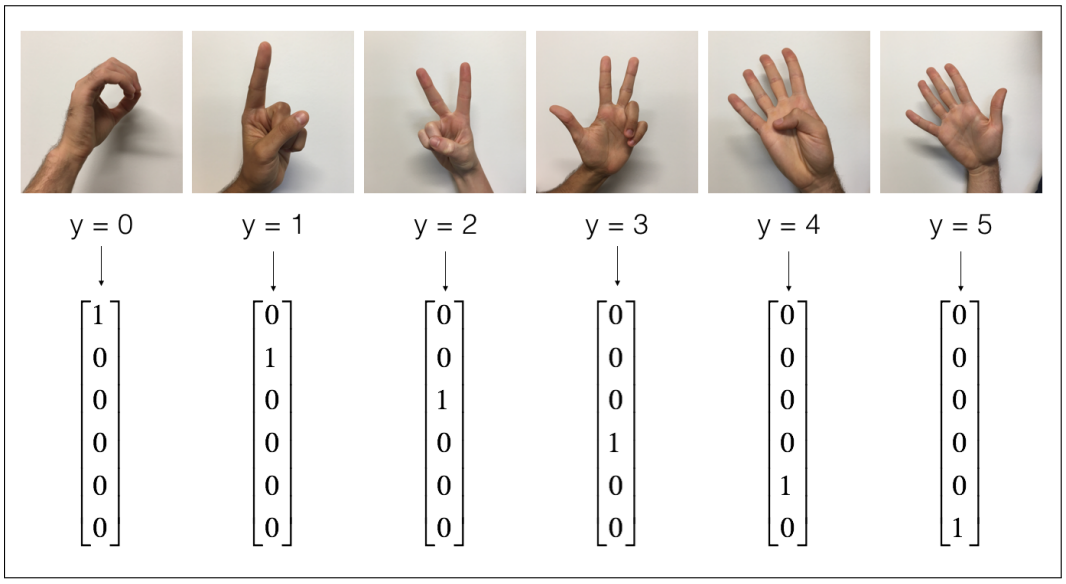


Figure (2.8): Examples of gestures which are imitating digits

This is a more realistic case of an upload filter because similar to this, the "Hitlergruß" gesture or another forbidden gesticulation can be identified. Even if the strength of a convolutional neural network is to recognize images with high dimensions, the dimensions will be low as in the example before. The calculation with inputs of higher dimensions takes to long such that it can be trained along with this thesis. [Trask et al., 2015]

Before I used a so-called densely fully connected neural network (Figure 2.7). The network consists of a certain amount of neurons which are arranged in different layers (Figure 2.5). Each neuron is fully connected with the neurons in the previous layer. To see how complex such a network can be, in Figure 2.9 such a network can be seen. The Figure shows that such a network can become complex. Consider that each connection is represented by a weight. This weight has to be initialized and adjusted. If it comes to understand when and why which weight was adjusted to ensure transparency, it becomes clear that such a network can be a confusing issue.

More neurons would result in a high amount of parameters if the image has too many dimensions [Goodfellow et al., 2016, p. 324]. An alternative approach would be to use the

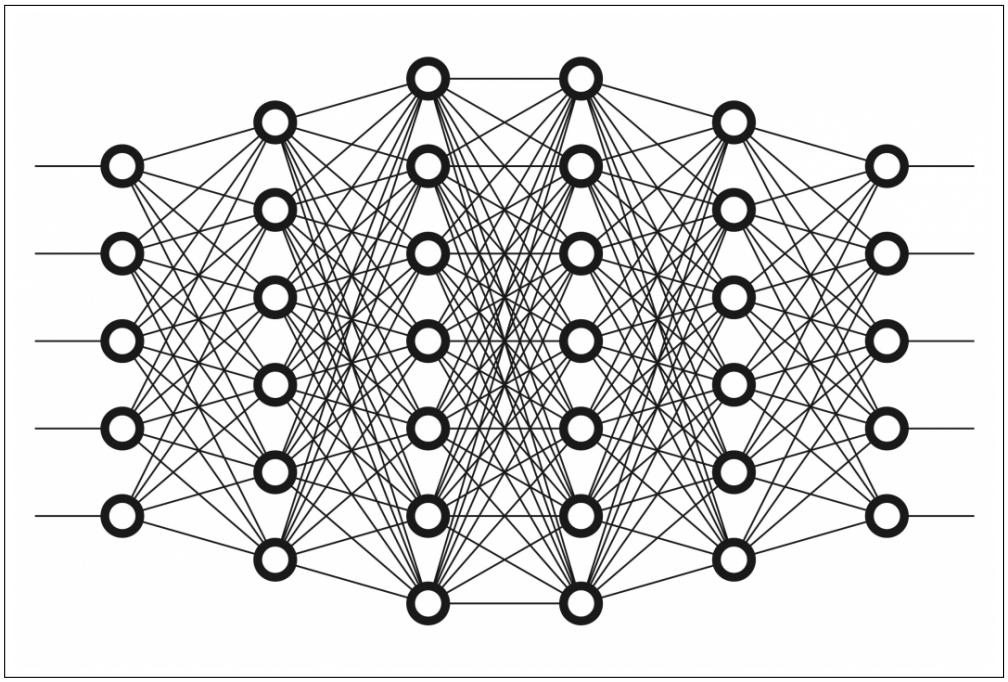


Figure (2.9): Examples of a fully connected neuronal network

mathematical operation of convolution [3]. The convolution makes it possible to detect patterns like edges which could be used to classify an image. If the images have edges which are similar to the kind of edges from another image, the probability is high that it is in the same class. For example, if the algorithms detect a set of edges which are typically for cat ears, the algorithm will probably consider this image as a cat. Other possibilities of extracted patterns are shown in Figure 2.11.

Convolution operations are widely spread in computer vision algorithms, so it is not unique for convolutional neural networks. It is a mathematical operation where a small matrix of numbers is passed through the matrix image representation. This matrix is a so-called filter or kernel. Every colour channel would need its filter. The name filter comes from the fact that it filters specific features from the input image, e.g. horizontal and vertical images. In Figure 2.11, where

- ***** is the mathematical operation of convolution
- **X** represents an image with a horizontal edge in the middle of the image
- **A** = the output matrix (feature map) which shows where the edge is[4]

[3] Technically I skip the narrowing operation, so this would be a cross-correlation instead of convolution. Still, as in literature and by convention, I call this a convolutional operation anyway[Ng, 2008]

[4] because this images are very small the dimensions of the edge which is shown in A are not accurate and unprofessionally. While using matrices of higher dimensions the proportions would be more accurate

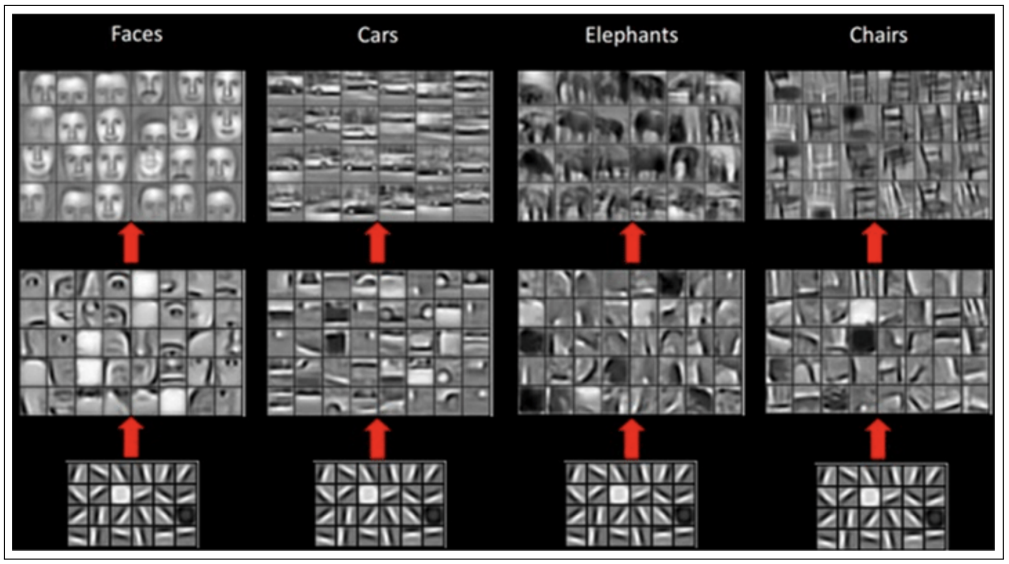


Figure (2.10): Patterns identified while using convolutional neural networks [Stewart, 2019]

- $\mathbf{f}$ = a filter for detecting horizontal edges,

you can see how a vertical edge, which is represented by a 6×6 matrix X is filtered and represented by a matrix while using another 3×3 matrix as a filter.

For a convolution operation, a filter would be placed in front of a selected pixel. Afterwards, each value from the filter has to be multiplied with the corresponding values from the image. Finally, the sum would be placed in the right place in the original matrix, as shown in Figure 2.12.

The algorithms use a specific filter matrix, to detect of edges of every kind (e.g. vertical and horizontal edges). Two filters are used to identify at first horizontal edges (Figure 2.13)and second vertical edges (Figure 2.14). Finally, both edges get projected to the original image and colour values were normalized to show the vertical and horizontal edges more clearly (2.15). To figure out which filters shall be used to detect patterns is challenging because there are almost endless opportunities. That is where the neural network comes into part. Instead of setting the filter values manually, they are the parameters of a neural network. These parameters can now be optimized by using the cost function of the network (Definition 2.2.5). That means the total amount of parameters comes no longer from the size of an image it comes now from the filter size [Ng, 2008])

Figure 2.16 shows the architecture of such a network which is used to identify filter values and finally helps to classify more efficiently.

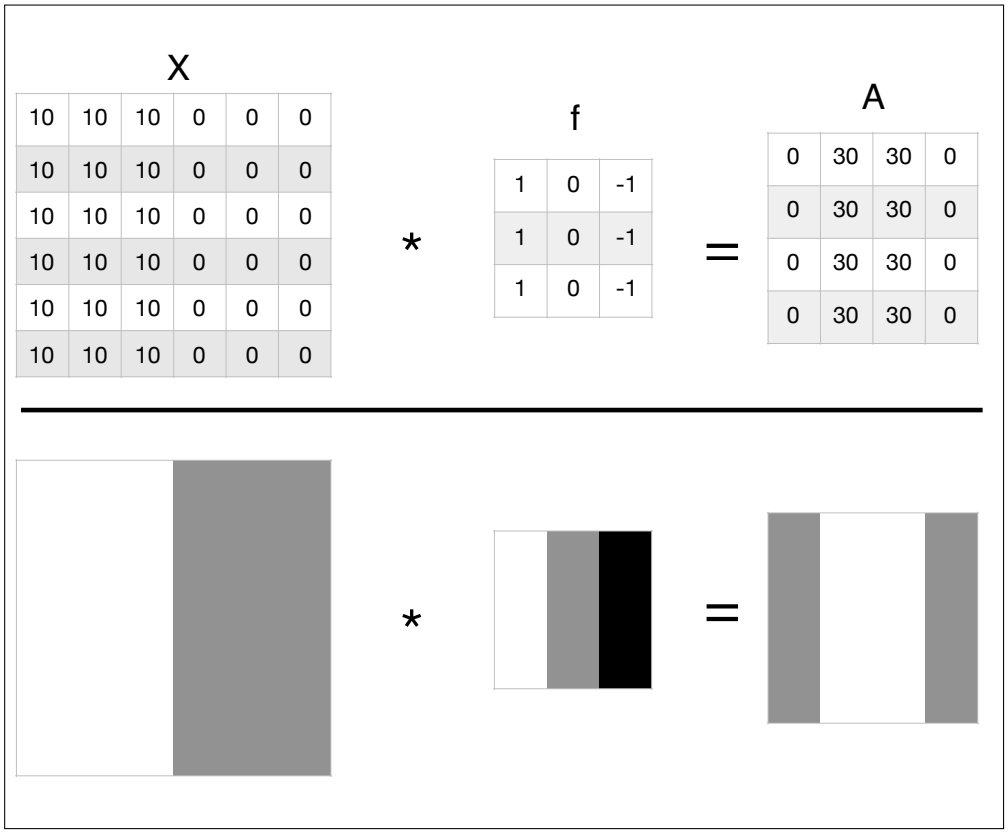


Figure (2.11): Using Convolution for edge detection

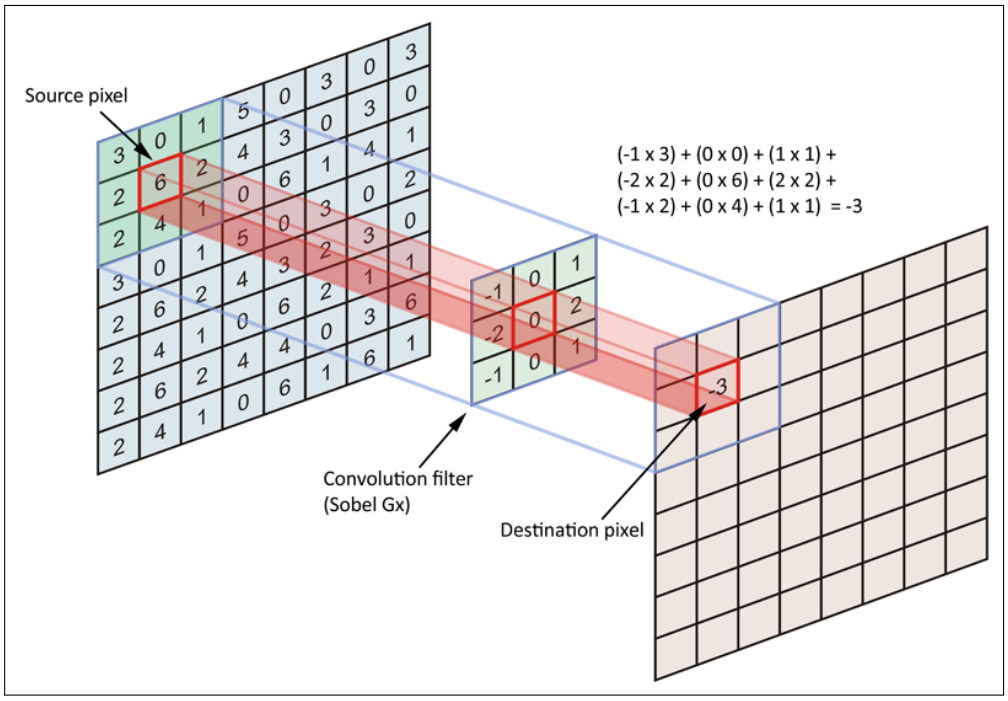


Figure (2.12): Convolution Operation over a matrix

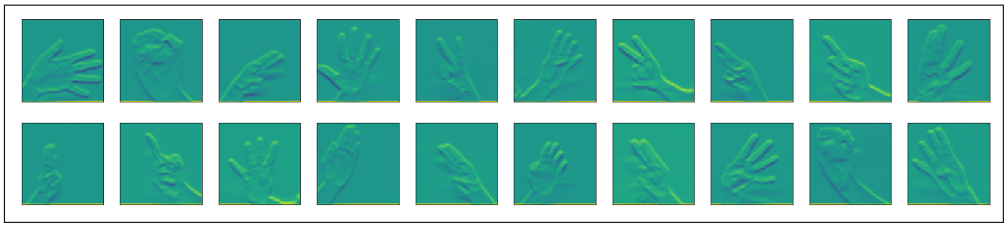


Figure (2.13): Convolution Operation on hand gestures, using a filter for horizontal edges and projecting them on the original image (without changing the size of it)

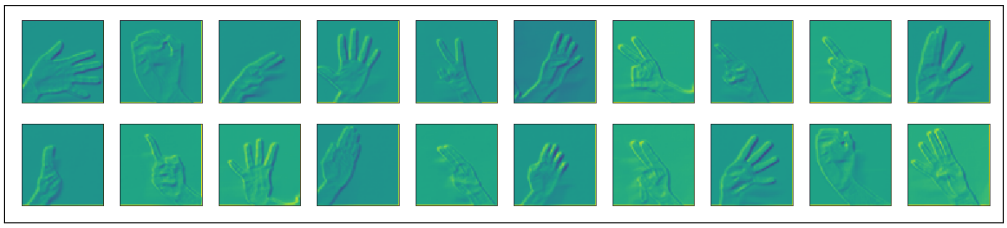


Figure (2.14): Convolution Operation on hand gestures, using a filter for vertical edges and projecting them on the original image (without changing the size of it)

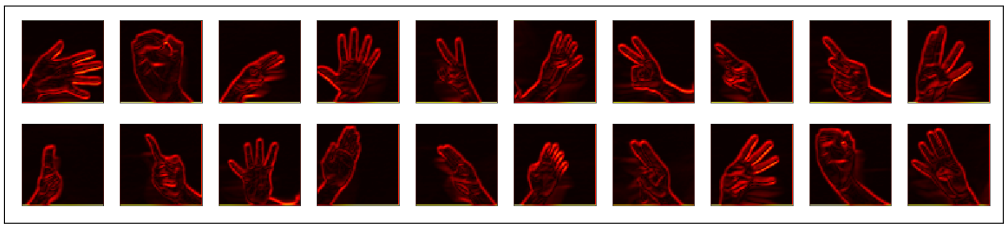


Figure (2.15): Convolution Operation on hand gestures, using a filter for horizontal and vertical edges and projecting them on the original image (without changing the size of it but while using normalization to make the edges more clear)

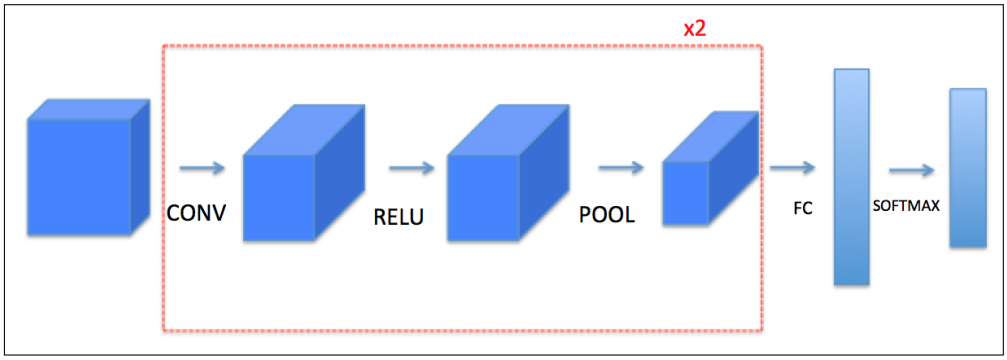


Figure (2.16): A typical architecture of convolutional neural network made from different building blocks

As can be seen in the image, a convolutional neuronal network contains more than just convolution operations. One example is the RelU which is an activation function and is used to standardize values between the layers. The softmax activation function is used to get probabilities for each possible class. Afterwards, a decision boundary can be decided whether the image belongs to a class or not. The pooling layer reduces the height and width of the input. It helps minimize computations, as well as it helps to make feature detectors more invariant to its position in the input data. Typically a pooling layer is one of the two following types:

- Max-pooling uses another matrix which slides over the input and stores the max value of the window in the output.

- Average-pooling uses another matrix which slides over the input and stores the average value in the output.

In the Figure 2.18 2.17 you can see how this would be done, where:

- **Stride** is the value which determines for how many pixels the filter get shifted each time.

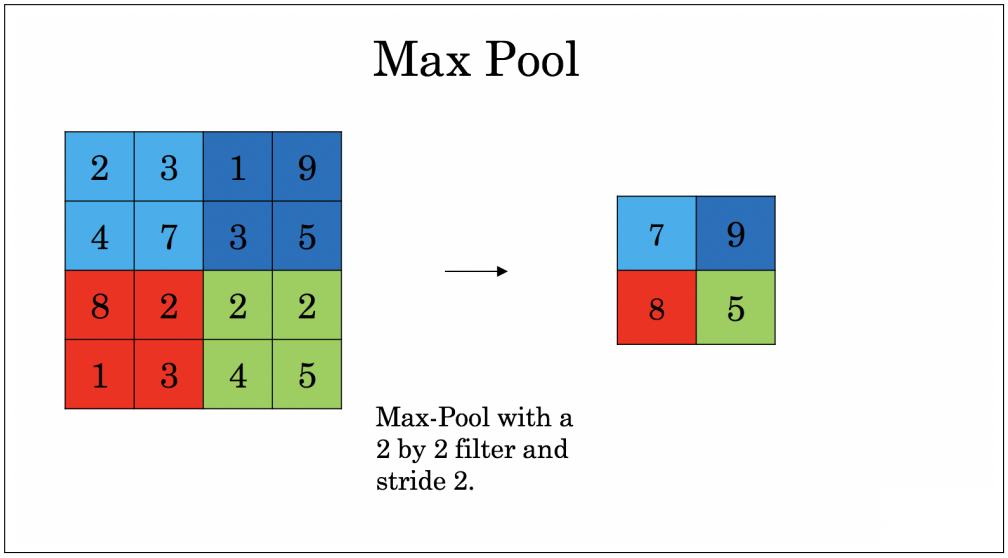


Figure (2.17): Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling [].

This kind of network and a dataset of ten different classes achieve accuracy close to 80%[5].

[5]The result could be even better if more training data and a deeper network would be used

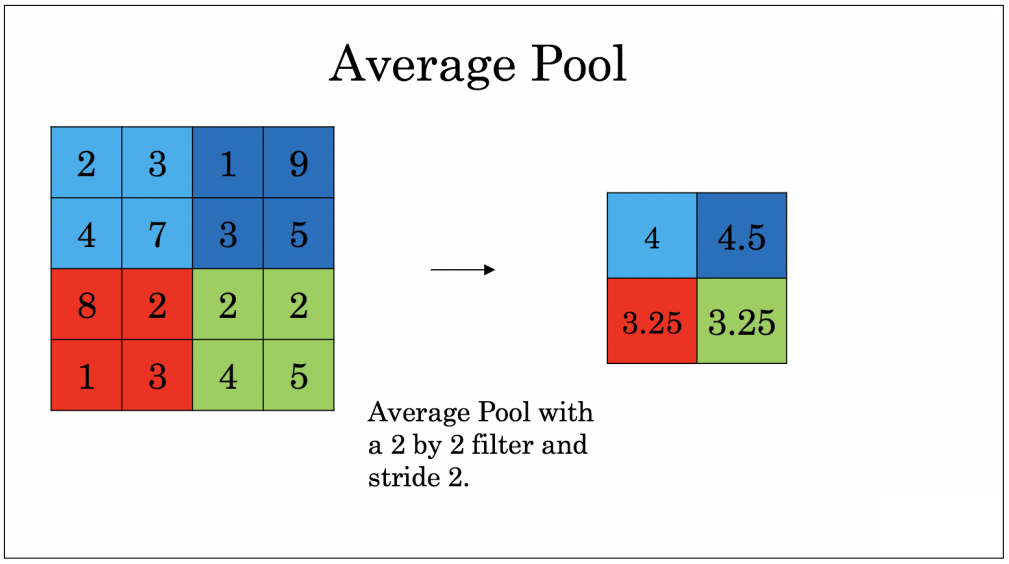


Figure (2.18): Average pooling involves calculating the average for each patch of the feature map. This means that each 2×2 square of the feature map is down sampled to the average value in the square.

```
1 Cost after epoch 0 =      1.917929
2 Cost after epoch 5 =      1.506757
3 Train Accuracy =      0.940741
4 Test Accuracy =      0.783333
```

Code Listing (2.3): Test accuracy is 80% after iterations 2400 times and using 209 examples with 12288 features (64 × 64 pixels). That accuracy is not state of the art but very good if considering that this is an algorithm which is not specialized to recognize images.

The state of the art results in image recognition, e.g. on the cifar10[6] dataset, are impressive examples of how this technique has increased the performance of machine learning in the last years. They achieve results of 90% and can be computed within a pleasant amount of time.

However, because of their non-linear structure, convolutional neural network algorithms with outstanding overall performance are usually seen as a black box. That means no information is provided about what exactly lead the networks to their conclusion. Transferred to the Instagram upload filter example, this would imply that 10% of the users are blocked more falsely during the uploading. The user will not be informed about why the algorithm came to its decision.

[6] CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32 × 32 colour images containing one of 10 object classes, with 6000 images per class. Alex Krizhevsky, Vinod Nair collected it, and Geoffrey Hinton.

The goal of the next section is to present the theory of different methods of how a convolutional neural network behave more like a white-box model. In other words, a decision support system from which the user can be informed of why the algorithm came to a specific decision.

2.3. Explainable Artificial Intelligence

Most deep learning models today are black-box. That means no counteraction is set to make them transparent. Within the last years, techniques have arisen, which help to make those models more transparent. That means for a given prediction, how important is each input feature value to that prediction? Before two of these techniques are get explained in detail, it will be demonstrated one more time why this is so important.

Since deep learning has become more and more successful, science is also increasingly concerned with how to make deep learning models more transparent. For business leaders, machine learning engineers and the users who are confronted by the results, it is essential to know why a decision was made. Unless we were not able to learn from these models and they will still be a black box [Michael Jordan, 2018] [Kuang, 2017].

Figure 2.19 shows the difference between a traditional machine learning model and an explainable artificial intelligence model. Whereas the traditional approach focuses on the output itself (e.g. this is a cat) the explainable approach combines some features to an explanation which can be understood by humans (e.g. has fur, whiskers and claws).

Managers with a high responsibility for there department find it challenging to use these complex algorithms. Although they are aware that the results could increase the success of their company, they are afraid of using these models, because they are not transparent. If something goes wrong, they can hardly explain why the algorithm made the wrong decision. The results created by deep learning in image recognition could be beneficial for the healthcare sector as well, but if it is unclear why decisions were made, doctors will not give these approaches any chance. As well as the manager's doctors are responsible for their decisions, and if they get supported by an algorithm, they must understand the algorithm in the first place. Otherwise, they could not trust them, because they would not be able to distinguish between a right and a wrong decision.

A lack of transparency is responsible for the rejection of deep learning models in the finance industry as well. Especially since the last big financial crisis in 2009, banks and financial

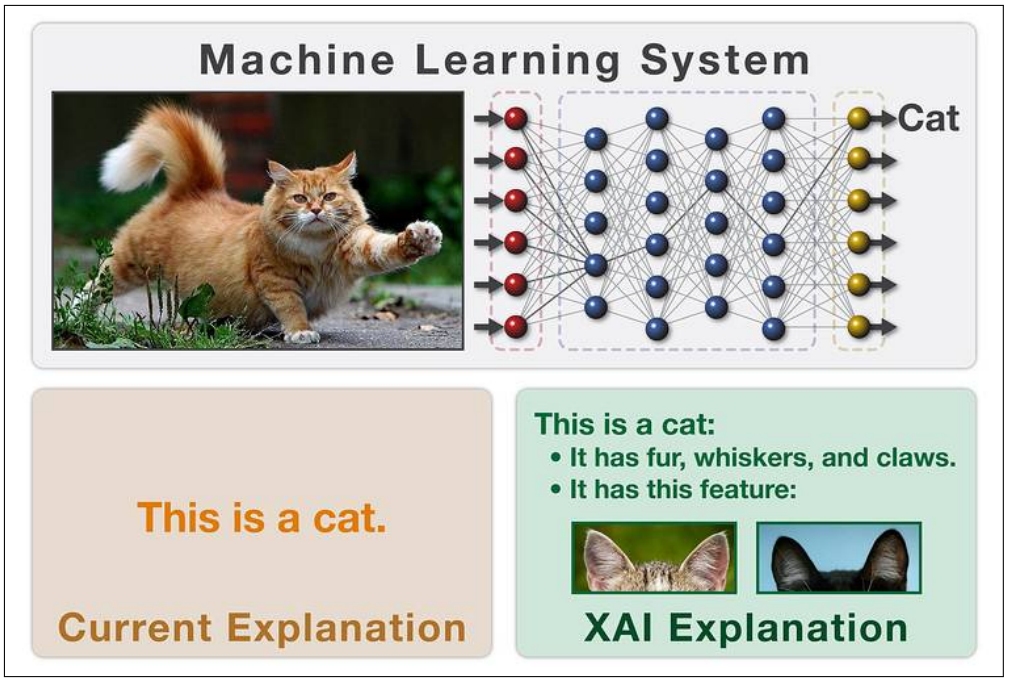


Figure (2.19): This is a cat, but how do you know? Explainable artificial intelligence seeks to develop systems that can translate complex algorithmic decision-making into language humans can understand.[Robinson, 2017]

companies must make their decisions transparent to different stakeholders. And as already introduced in the introduction, small business owners could be affected by semitransparent models as well. If they run their business with the help of social media, it would harm them if their uploads get rejected automatically, without even knowing why this decision was made.

More specifically, a small shop owner wants to upload an image which shows him, introducing a new product. The algorithm was trained to detect the "Hitlergruß", and so he predicted the user performs this gesture. So, explainable artificial intelligence would help to explain automatically why this prediction would be made. Thus, the shop owner could change the crucial factors immediately.

The following sections will focus on two techniques which could be used in an upload filter. This could help the shop owner from the example given in the paragraph above. These two techniques are [Kuang, 2017]:

1. Shapely Additive exPlanations (SHAP) and

2. Integrated Gradients (IG)

The Shapley Additive exPlanations and the Integrated Gradients (IG) belong to two different categories for explainable artificial intelligence.

1. Shapley-value-based algorithms and
2. gradient-based algorithms

Two factors can easily distinguish these models. Firstly, the assumptions and secondly, the underlying mathematical principles used in the models. Before it is explained how they differ, the fundamentals of each category will be explained.

The two fundamentals of the shapely-value-based algorithms and gradient-based algorithms are [Michael Jordan, 2018] [Kuang, 2017]:

1. Shapley Values
2. The Gradient

Shapley Values

Let assume that the algorithm behind an upload filter predicts the price for a painting. For a particular painting, it predicts 300000€, and the prediction will be explained through shapely values. The art has an age of 50 years, is part of a private collection, is not mentioned in literature and is in good shape (Figure 2.21).

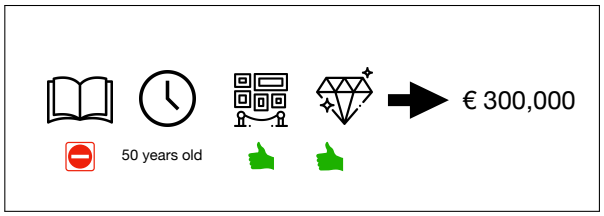


Figure (2.20): A concrete visualization of the features from painting for which the hapley values will be calculated.The painting has an age of 50 years, is part of a private collection, is not mentioned in literature and and is in a good shape.

The average prediction for all paintings is 310000€. Now the question is how much has each feature value contributed to the prediction compared to the average prediction? For that, quantitative criteria will be used.

For a linear regression model, the answer is quite simple. The attribute of each feature is the weight of the feature times the feature value. It works because linear regression is a linear model. So, to make more complex models (e.g. deep learning, convolutional neural networks) transparent, a different solution is required. The Shapley value, coined by Shapley (1953),

is a method for assigning payouts to players depending on their contribution to the total payout. Players cooperate in a coalition and receive a certain profit from this cooperation [Aas et al., 2019] [Lundberg and Lee, 2017] [Lundberg et al., 2018].

Because this is not a real game, the game is the prediction task for a single instance of the dataset. Therefore the gain is the actual prediction for this instance, minus the average prediction for all instances. The players are the feature values of the instance that collaborate to receive the gain (predict the price for the painting). In the example, the feature values "Not mentioned in the literature, "painting is in good shape", "painting is 50 years old" and "painting is part of a private collection" worked together to achieve the prediction of 300000€. Our goal is to explain the difference between the actual prediction (300000€) and the average prediction (310000€): a difference of -10000€.

An answer looks like the following:

- "Not mentioned in literature" feature contributed 30000€
- "50 years old" feature contributed 10000€
- "Part of a private collection" feature contributed 0€
- "Paining is in good shape" feature contributed 50000€

The contributions add up to 10000€, the final prediction minus the average predicted apartment price.

How do we calculate The Shapley value for one feature gets calculated the follows. The Shapley value is the average minimal contribution of a feature value among all possible coalitions.

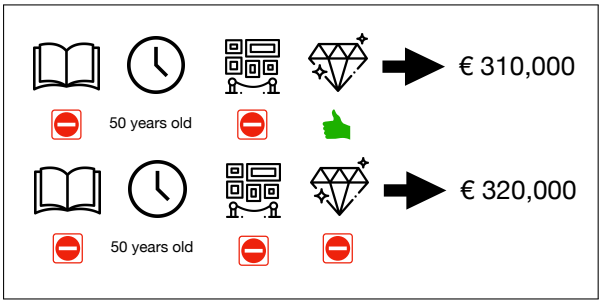


Figure (2.21): One sample in order to calculate the contribution of "is in a good shape" feature value to the prediction when added to the coalition of age, mentioned in literature and within a private collection

In the following we evaluate the contribution of the "Painting is in good shape" feature value when it is added to a coalition of "Not mentioned in literature" and "50 years old"."Not mentioned in literature", "Painting is in good shape", and 50-years old" feature values are from the given predicted instance (painting with a worth of 300000€). The feature value for

whether a painting is or is not in a private collection came from a random choice among all possible paintings which are in the same distribution. So, the value "in private collection" was replaced from yes by no. If the price of the painting gets now predicted again (with this combination), the predicted price is 310000€. In a second step, the "in good shape" feature value get doped from the coalition by replacing it with a random value among all other possible paintings which can be predicted. In the example, it was not in good shape, but it could have been good shape again. The prediction of the painting price for the coalition of "mentioned in literature" and "50 years old" is 320000€. The contribution of "In Good Shape" was:

$$310{,}000€ - 320000€ = -10000€$$

This estimate depends on the values of the randomly drawn painting that served as a “donor” for the "shape" and "private" feature values. We will get better estimates if we repeat this sampling step and average the contributions. The result is even better if the sampling step gets repeated and calculate and calculate the average of all contributions. Furthermore, it is done repeatedly computing all coalitions. Because of the repeatedly computing for every combination, the calculation time increases exponentially. One solution to keep the computation time small is to calculate contributions for only a few samples of the possible coalitions.

If we estimate the Shapley values for all feature values, we get the complete distribution of the prediction (minus the average) among the feature values.

SHalpey Additive exPlanations (shap-software-library)

If Shapley values are a method which maps a contribution to each player, a Shapley value-based explanation method will aim to get estimated shapely values which are close to the precisely calculable ones. It is achieved because the Shapley Additive Explanations are randomly dropping out some of the features. Potentially any combination of features can be left out. As long as you look at each feature individually, there is an almost infinite number of combinations. A calculation would take too long. As a counteraction, pairs of features are put together and in the following considered as one player. The individual success of a player is then estimated. Let us look again at the example of the shop owner. If a group of pixels does not contribute to the overall result of the game, it will not change even if you drop out the whole group [Lundberg et al., 2018] [Molnar, 2019] [Lundberg and Lee, 2017].

Figure 2.22 shows an easy example of how the shape library explains the feature importance of a machine learning model. The colours help to get an intuitive understanding of

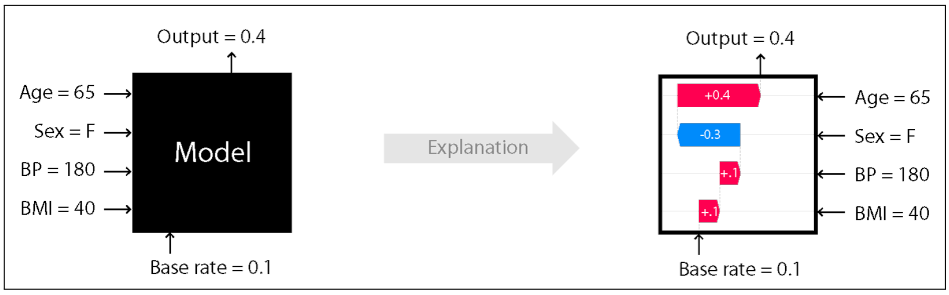


Figure (2.22): The header of the SHAP Github Repoitory shows an example of a Machine Learning Algorithm is explained through the library. It takes a model with age, sex, bp and bmi as feature examples and maps an attribute to this values. Each values showsthe average marginal contribution of a feature value across all possible coalitions of features.

which features are more important, so red stands for significant contribution to the overall prediction, and blue is less important to it.

Gradient and Gradient Decent

I have to give some theory about the gradient and the gradient descent if it comes to an explainable artificial intelligence method which is built on top of the gradient and the gradient descent.

Gradient descent is possibly the most used optimization algorithm in the field of deep learning and machine learning. If there would be a cost function as introduced in the section about machine learning, the goal is to find the optimum for every weight in the model. So, the cost value would be as small as possible as long as the value is not adapting to a specific set of examples. The gradient descent is an algorithm that optimizes the cost value by making changes to the weights. The difference is determined by the gradient, which shows the direction of the deepest decent. In other words, it is a value which determines a direction which minimizes the cost in as few backpropagation steps as possible. Whereas in the so-called forward propagation the actual cost is calculated the backpropagation uses the chain rule to get the gradient at first and at second adjusts the weights by subtracting the gradient multiplied by the learning rate from them [Goodfellow et al., 2016].

This method is quite old, and one of the reasons why this method got popular in the last years was the increase in computation power. Until a few years, a new generation of computational processing units and graphical procession units are fast enough to work on such tasks efficiently.

Figure shows2.23, the graph of a cost function in three-dimensional space. The line shows the line from a starting point by going down to a local minimum. It is crucial to know that gradient descent can not distinguish between a local and a global minimum. Therefore it would need different steps of gradient descent. In practice, this is not important because it works for most cases just fine enough. For a field where very high accuracy is a must, this kind of machine learning would be the wrong approach anyway. Therefore a logic-based attempt would lead to a state of the art result [Russell and Norvig, 2009] [Kohlhase et al., 2017].

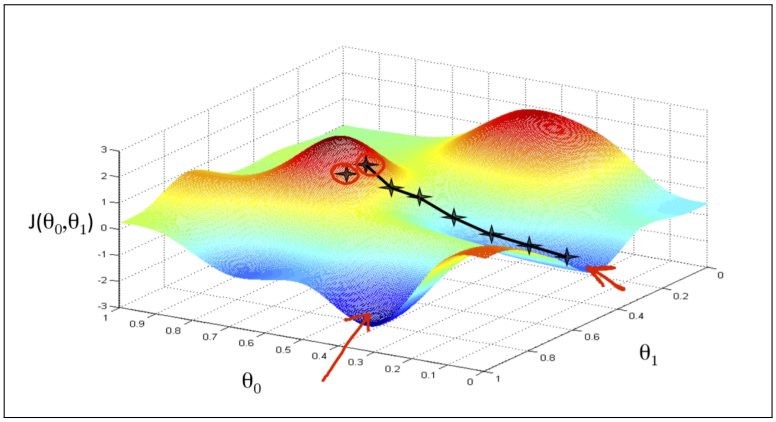


Figure (2.23): Shows the path to a local minimum while using the parameters gradients to compute the path [Wikipedia, 2020]

2.3.1. Integated Gradient

A gradient-based explanation method tries to explain a given prediction by using the gradient of the output concerning the input features. The gradient is not only used here to optimize the weights of the parameters but also to see which features have been adjusted the most and are therefore most important.

The goal is to know how a decision was made. So it can be concluded afterwards why this decision was made, so the gradient is used again. As already mentioned, the gradient points in the direction of the next local minimum for each parameter. If it is observable which parameters are adjusted more than others, a statement which parameters are more important than others can be made. Since each parameter is assigned to an input value, it can be concluded how important this value is to reach the desired minimum as fast as possible. An efficient implementation of a deep learning approach is essential to save computing power. An exact calculation costs an enormous amount of resources, which is why the gradients are not calculated in this step but estimated [Molnar, 2019] [Tjoa and Guan, 2019] [Sundararajan et al., 2017].

Figure 2.24 shows three paths between a baseline (r1 , r2) and an input (s1, s2). Path P2, used by integrated gradients, simultaneously moves all features from off to on. Path P1 moves along the edges, turning features on in sequence. Other paths like P1 along different edges correspond to different sequences. SHAP computes the expected attribution over all such edge paths like P1.

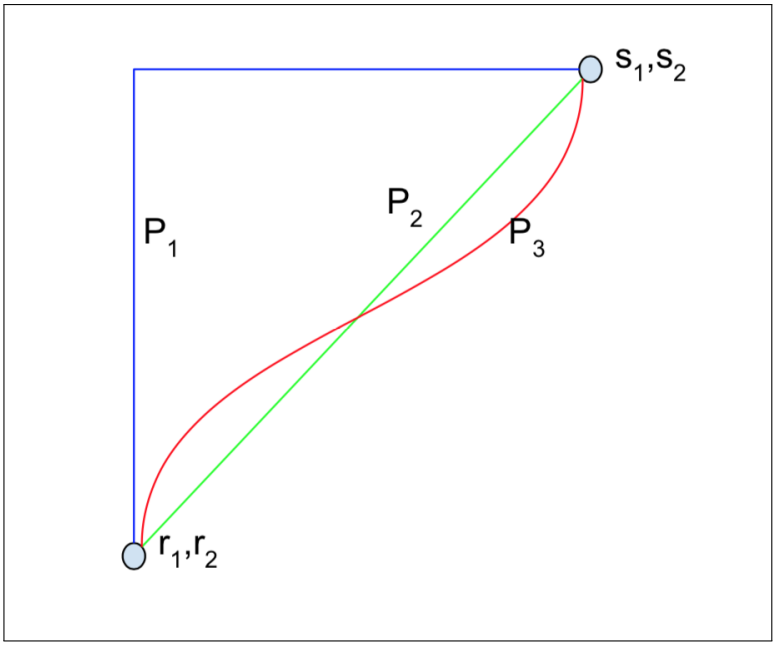


Figure (2.24): Shows different paths between a baseline and an input to compare it the Integrated Gradient (P2) with the SHalpey Additive exPlanations[Tjoa and Guan, 2019]

The integrated gradient approach tries to estimate Aumann-Shapley values which are as close as possible to the calculated values. The integrated gradient works by assuming there is a straight line, from the actual input (e.g. the image of a shop owner which tires of uploading it to Instagram) to a specific baseline input (e.g. a black image). The gradient of the prediction concerning input features would be integrated along this path [Sundararajan et al., 2017].
The integrated gradient is also a method where the input varies along a straight line between the baseline and the input. At the same time, the prediction moves from uncertainty to certainty (the final result, e.g. for 98% the image shows the "Hiterlergruß". At each point on this path, the gradient is used to attribute the change in the prediction probability back to the input features. So, integrated gradient aggregates these gradients along the paths integral [Mudrakarta et al., 2018]. Figure 2.28 visualizes who this method would look like after applying it to a few examples.

1. Choose any image as a baseline (e.g. black image with each pixel 0)

2. Make images brighter as long as they become the input again (Figure 2.25)

3. Compare the final output (as moving from certainty to uncertainty) to the path of the images (from the black baseline to the input, Figure 2.26)

4. We want to know then the slope of the score vs intensity graph doesn't remain stagnant (Interesting Gradients)

5. The Input images get changed such that the interesting gradients can be seen in there (Figure 2.27)

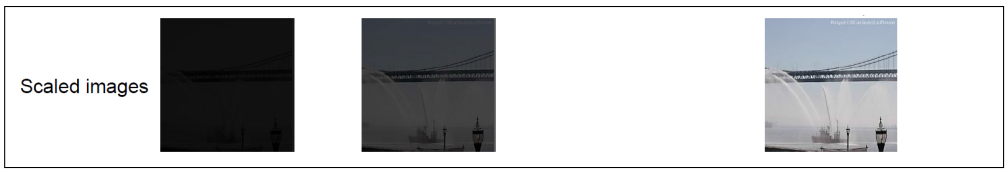


Figure (2.25): Three different steps along the path from the baseline to the input

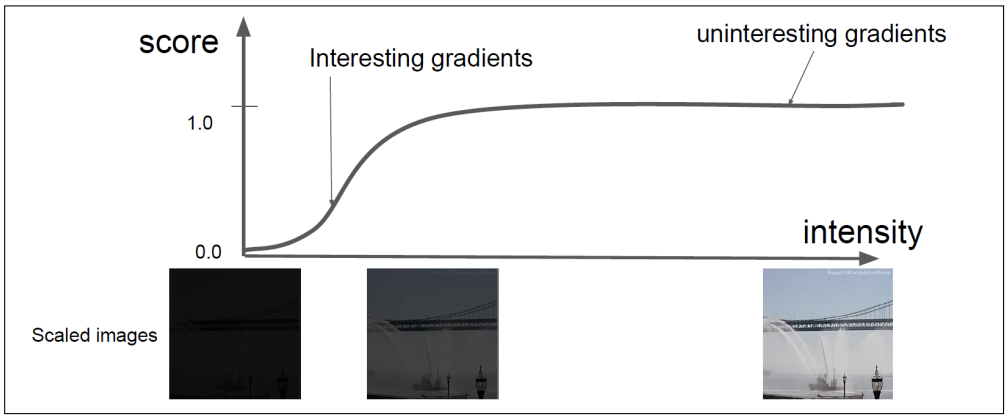


Figure (2.26): compression from the two paths (along the result and along the baseline to output)

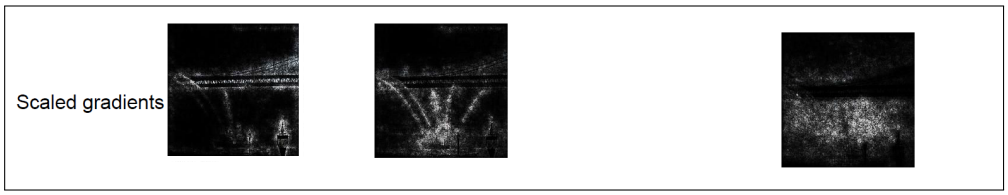


Figure (2.27): result of the Integrated Gradient

As the Integrated Gradient approximates Aumann-Sapley values the function which estimates the values must be a piecewise differentiable function of the input features.

Because the method should make sensible feature attributions a sufficient baseline is crucial. For example, if a black image is selected as a baseline, the integrated gradient would not choose attribute importance to a completely black pixel in an actual image. If black pixels are not necessary, e.g. because just the frame of the shop owners image is black and the rest of the image is only bright, this will work perfectly fine. But in general, the baseline value

should both have a near-zero prediction, and also faithfully represent a complete absence of signal.

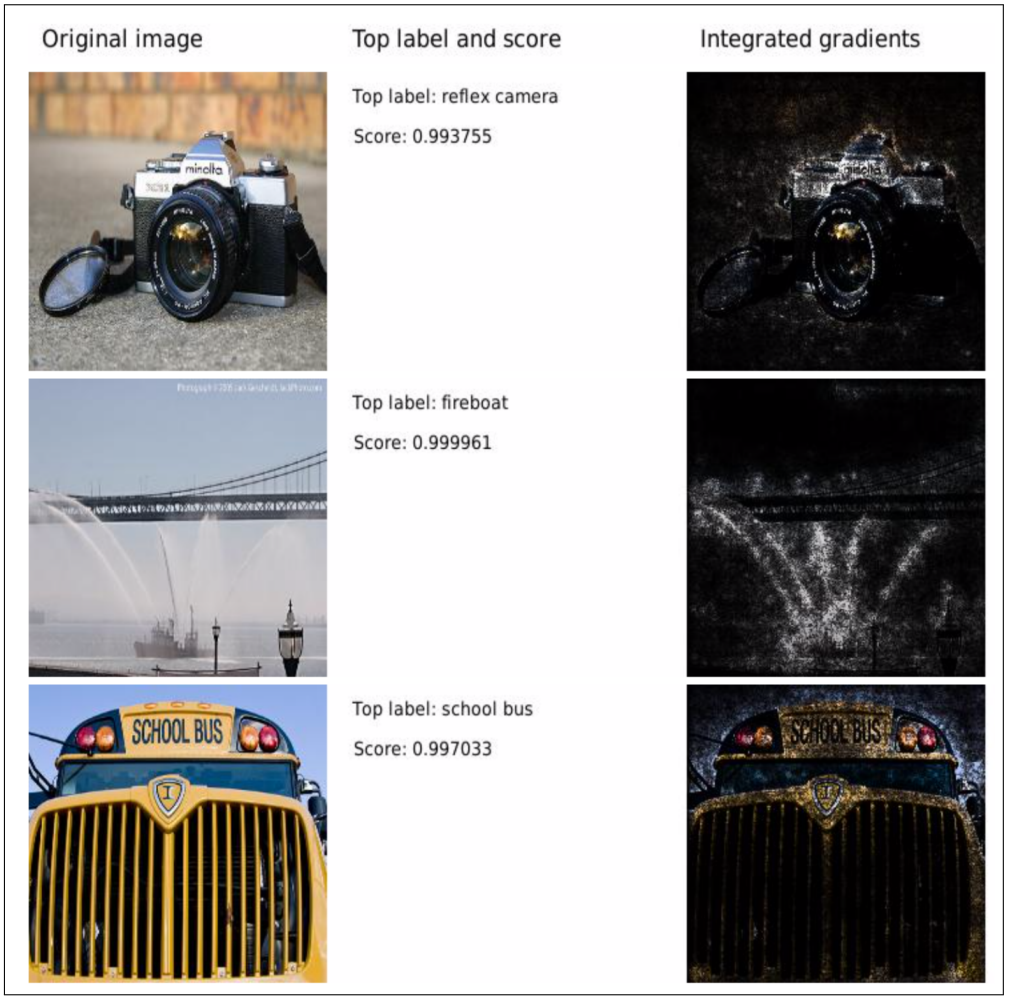


Figure (2.28): The integrated gradient applied to sample of images and visualized to demonstrate which pixels are important (brighter) and which pixels are less important (darker)

2.3.2. Expected Gradient Approach

The expected Gradient Approach combines the two methods which were described above. It is useful because the two approaches above generate noisy data. While in the two methods above, one single example is used as a reference value. This approach allows using the whole dataset. It makes the resulting values easier to interpret.

It tries to combine a multitude of ideas from integrated Gradients, SHapley Additive exPlanations (SHAP) and libraries such as SHAP-Library implement some brilliant approximations and samplings. To cover up all their work would fill an entire mater thesis with ease

so this part is left out. Important to understand is the result which is provided by software library such that it can be interpreted later on the course of this thesis.

An intuitive way to understand the expected gradient values is following illustration: The feature values are chefs who are entering a hotel kitchen step by step in random order. All chefs (feature values) in the room contributing to the lunch (= contribute to the prediction, which is predicted by a machine learning model). The expected gradient values are equal in the same grade given for the meal which got prepared in the kitchen. This value is the average change in the grading. That mentioned change is defined as follow: Grade revived by a combination of cooks which are already in the kitchen if the actual chef is entering. More precisely, the average of all different combinations of chefs which are in the kitchen, while the specific chef comes into the kitchen.

Figure 2.29 shows how this can be visualized on an overlay on image predictions. Especially, it can be seen which regions on the image are responsible for the predicted probability of being part of a class (red) or not (blue).

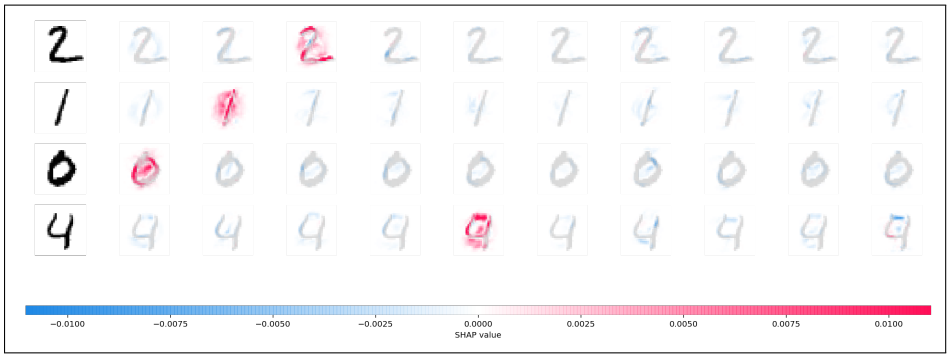


Figure (2.29): The shap applied to sample of images and visualized to demonstrate which pixels are important (red) and which pixels are less important (blue) [Slundberg, 01]

Chapter 3.

Requirements

"Big shots are only little shots who keep shooting."

- Christopher Morley

Explainability is the primary cause of this thesis, and until now, the development of explainable deep neural networks has been considered as a theoretical issue (Chapter 2). Within the next few chapters, a prototype is set to become an essential part, when it comes to transforming this academic issue into a real-world problem.

The previous chapters focused on systematic literature research around the following topics:

- Supervised Machine Learning (Section 2.2)
- Deep Learning (Section 2.2.1)
- Convolutional Neural Networks (section 2.2.2)
- Explainable Artificial Intelligence (Section 2.3.2)

Therefore my prototype has to be an implementation of a supervised machine learning algorithm. More precisely a deep neural network of a convolutional architecture. Moreover, the implementation of the prototype has to present the results clearly and transparently. If the prototype can not do this, an application will not help to apply the given knowledge into practice. This section can be seen as a preliminary explanation of what a prototype has to accomplish, such that the requirements can be discussed in greater detail.

3.1. Objective

The prototypes objective is to make predictions on given input and present these predictions (output) transparently. The details of what input, output and transparency mean will be described in the following sections.

3.2. Input Data

In her cutting edge master thesis, Katarina Blandina Weitz presented a prototype to decode facial expressions of pain and emotions. For example, images of human's emotions as input were notably valuable. In order to identify the worth of explainable deep learning methods in upload filters, the subject does not matter. Besides, a limitation of a specific subject (e.g. human's emotions) is to use the results for a generalization. This thesis uses a variation of Weitz input. In fact, the subject of the images will change, but the specifications are similar. This means the prototypes input consist of photographs which show a random subject except for human expressions. As given in the preliminary explanation above, the prototype uses deep learning with a convolutional architecture. I have already mentioned the technical details of input data in the theory part of convolutional neural networks (Section 2.2.2). Within this chapter, the Definition 2.2.2 specified the input data when it comes to training a machine learning algorithm as

> *a vector with probabilities for each image which gots predicted. Every probability stands for the likelihood to belong to a specific class.*

In addition to Definition 2.2.2, another input has to be defined. The objective of the prototype is to make predictions. Some information has to be given to enable an algorithm to do this. The need of a second input was also explained in (Chapter 2).

A problem with input data is the computation time. In a nutshell, if more data and classes are given, the computation time increases exponentially. So, it makes sense to keep the input dataset and its possible outcomes as small as possible. This relation is explained in Chapter 2). As a summary, the prototypes input data will fulfil the following requirements:

- A dataset which can be divided into a training and test set
- A dataset which consists of images with any subject except for human facial expressions
- The size of the input dataset shall not be too big. Otherwise, it can not be trained on a personal computer

3.3. Data processing

As mentioned in the introduction of this chapter, the prototype is useful to turn the theory into applied knowledge. The prototype exploits the introduced methods to make predictions on unseen data (test data), in particular convolutional neural networks. The principal

characteristic of the prototype is its transparency, which will be achieved by using Shapley values and the integrated gradient. So, the required process looks as follows:

1. The images (input) will be loaded
2. The input must be divided into a training and test set
3. Parameters of a convolutional neural network got trained while using the test set
4. The model will be evaluated during the test set
5. A few examples of the test set are used to show a transparent decision

This process is required to implement my prototype successfully. Figure 3.1 visualizes the processes within a circle. It means that the processes can be repeated several times, as long as the output fulfils the required form. The symbols shall provide a better understanding of the process. For example, step four aims for high accuracy. In other words, it is import that the prediction hits the right label of the image. It can be illustrated with an arrow that is supposed to hit a target. The requirements of step five, on the other hand, are less clear. It is expected that results are transparent, but everyone can have a unique understanding of what transparency means. For example, in the eyes of a non-technical user, transparency is something different, compared to the technical user. So, this step is visualized through an eye.

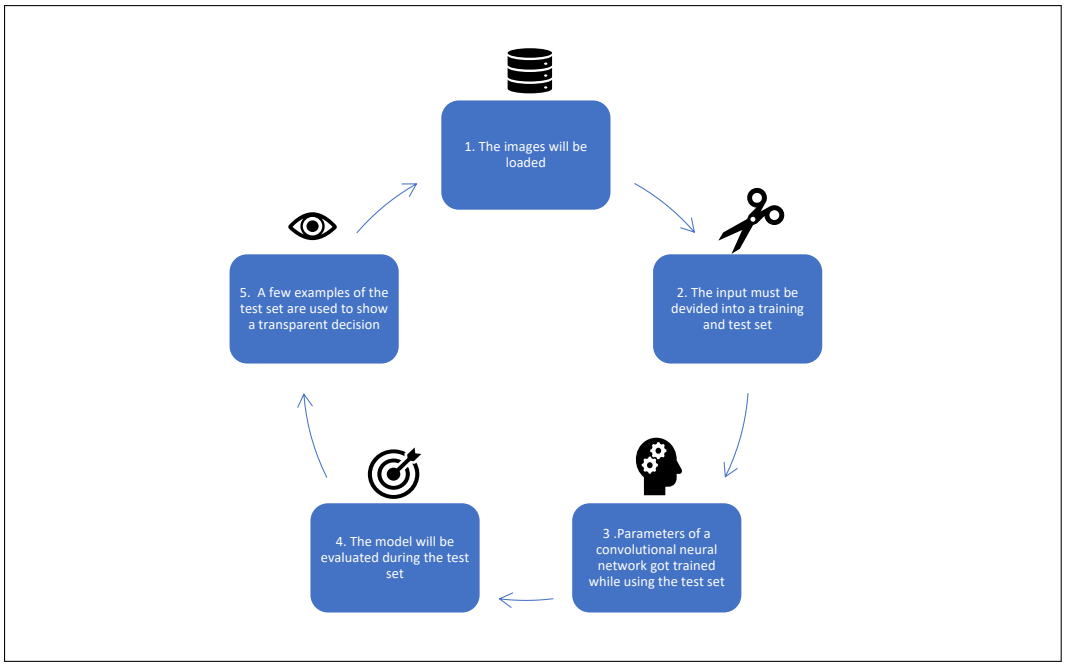


Figure (3.1): The Figure shows the required process of my prototype. Every symbol visualizes the main objective and the focus of each step. Step five points out that it is expected that the results are transparent. Everyone has a unique understanding of what transparency means. For example, in the eyes of non-technical users, transparency is different, compared to technical users. So, this step is visualized through an eye.

3.4. Output Data

The predictions are gonna made as part of an explainable deep learning model. A classifier to make predictions on unseen data (step four of the previous section), is the first step along the road to transparent decisions. This step of making predictions on unseen data has already been achieved through the use of a convolutional neural network. How this works in general, is also described in Section 2.2.2. It defines the output as follows:

> *A vector with probabilities for each image which gots predicted. Every probability stands for the likelihood to belong to a specific class.*

Transparency is a crucial aspect of the prototype. That is why everything that explains how the algorithm came to the probabilities will be seen as a second output. The best method for this investigation is to use the expected gradient approach. Understanding the output of this method is absolutely crucial if it comes to make transparent decisions. An intuitive way to understand the output of the expected gradient approach is the following illustration:

> *The input values enter a room in random order. All input values in the room contribute to the final prediction. The Shapley value of a feature value is the average change in the prediction that the coalition which is already in the room receives, when the feature value joins them.*

Nevertheless, this intuition is not appropriate for people without an information technology background. For those people, it is beneficial to visualize these values on the given input. This leads to the following requirements:

1. The prototype returns a prediction for each image in the test data set
2. The prototype returns additional values for a subset of the predictions. More precisely, a value is mapped to each input variable of the examined image. This mapping reflects its contribution to the overall result
3. The final output is an image. Therefore the output of Step 2 gets projected on the examined image. The pixels in the image should be coloured in such a way that it is clearly visible how valuable the surrounding image area was for a prediction.

3.5. Evaluation

Transparency is a qualitative value. Therefore it is hard to evaluate the results. Instead, it is possible to make assumptions about how transparent and clear the results are by a so-called visual examination. But without a well-designed survey, such an unjustified assumption can

lead to wrong findings and a false conclusion. To avoid this scenario, only the output, as explained before gets evaluated. If the output is equal to this explanation, the transparency will be regarded as useful.

Nevertheless, there is also a quantifiable characteristic that has to be evaluated. If the accuracy (Section 2.2) on the test set is below 90% the prototype will not be useful for making predictions. Instead, the expected gradient approach can be used to investigate why the classifier fails.

Chapter 4.

Functional Specifications

> "Details matter. It's worth waiting to get it right."
>
> - Steve Jobs

This chapter is divided into five sections.

1. The first section is a brief overview of the data set, which will be used within the prototype implementation. In this section, the decision for a particular dataset is clearly explained. Furthermore, the flow of the data will be introduced.

2. The purpose of the second section is to describe different approaches and alternatives when it comes to split a dataset. The prototypes dataset is divided into separate parts for training, validation and testing.

3. Within the third section, the so-called VGG16 architecture will be explained in greater detail.

4. How a model came to its decisions is related to the topic loss functions and the gradient. Because this was already addressed in the associated theory section, a detailed explanation will be left out. Instead, this section elaborates details for pre-trained model implementations. The prototype uses pre-trained networks to make its predictions, and this section explains how exactly this can be done.

5. To make decisions transparent, the prototype uses the SHAP library, which is introduced in the theory section as well. This chapter summarizes the theory part. It also introduces other methods and explains why SHAP is used to get the defined requirements of a transparent decision.

The five sections are related to the five steps, which are introduced at the end of the requirements chapter. Moreover, it is strongly related to the theoretical part of this thesis. The overall intention is to apply the given knowledge practically, by using the prototype implementation.

4.1. Input Data

The dataset "cats vs dogs" is used as an input dataset because it is one of the most rapid ways to enter the field of image recognition. Furthermore, the decision is based on a quantitative approach. The CIFAR-10, CIFAR-100 and the "cats vs dogs" datasets are compared on their size:

- The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- The CIFAR-100 dataset is just like the CIFAR-10, except it has 100 classes instead of 10. The classes are containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" and the superclass to which it belongs.
- The training archive contains 25000 images of dogs and cats. The resolution of the images is quite different.

Because the total amount of images and their resolution is different, total download size and the total amount of possible results are compared. Because the bullet points are not very clear, the following illustration shall present the information precisely and allows to apply a quantitative approach on the given datasets:

As mentioned in the theory chapter and the requirements chapter, the total size and the number of classes matters, if it comes to choosing a handy dataset. Usually, the number of classes is way more important as the total size of examples. This is true as long as this size is not too big. This is why the "cats vs dogs" dataset is the best fit for the prototype. Another reason to choose the "cats vs dogs" dataset is the course of dimensionality. In a Nutshell, if the feature space of an image recognition task is quite big, it leads to better results if more input data is available. As can be seen on the right plot in Figure 4.1, the "cats vs dogs" dataset has more data per class (measured in consumed storage space per class). A few samples of the prototypes input dataset can be seen in Figure 4.2. The images are not in high definition. This is good because otherwise the total amount of features (pixels) is just too big and must be scaled anyways in order to ensure a suitable training time on a personal computer.

Last but not least, Figure 4.3 visualizes the data flow within the prototype implementation. It shows the flow from a non-technical point of view. At first, the data gets loaded from a web host (1). At second, the dataset will be divided into different subsets (2). Each for testing, training and validation of the prototypes deep learning model. The input data comes in as "images" represented by a matrix as already explained in the section about convolutional neural networks (Section 2.2.2). The validation is everyday praxis among the

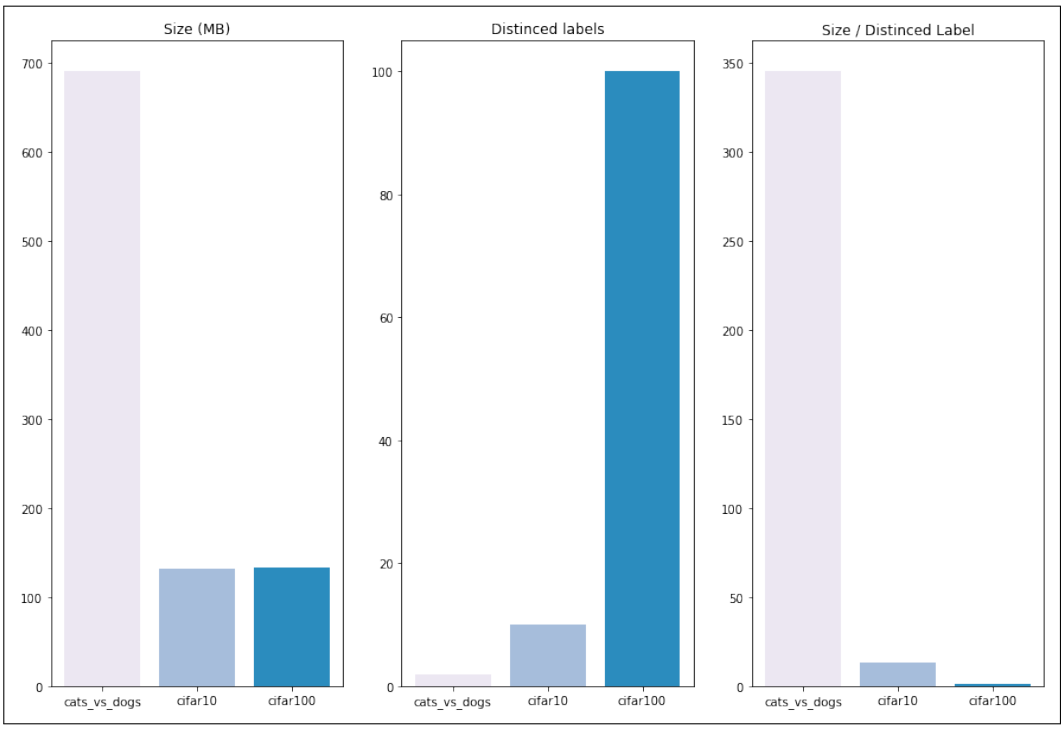


Figure (4.1): The "cats vs dogs" dataset is much bigger than the CIFAR-10 and CIFAR-100 dataset (left). The possible results for the CIFAR-10 and CIFAR-100 are much more as in "cats vs dogs" (mid). If the size and the potential outcomes are put in relation to each other, it can be seen that "cats vs dogs" dataset offer the most substantial amount of sample data per class. According to the course of dimensionality, is the "cats vs dogs" dataset the right choice (right).

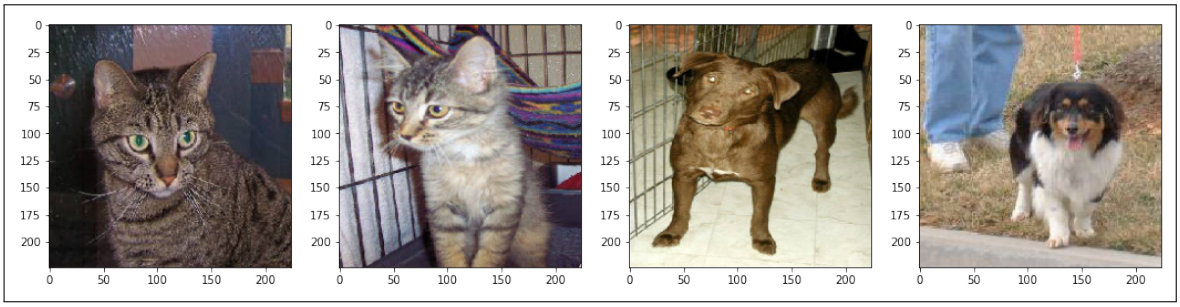


Figure (4.2): Example records from the "cats vs dogs" dataset. The pictures have already been scaled (compare IT concept).

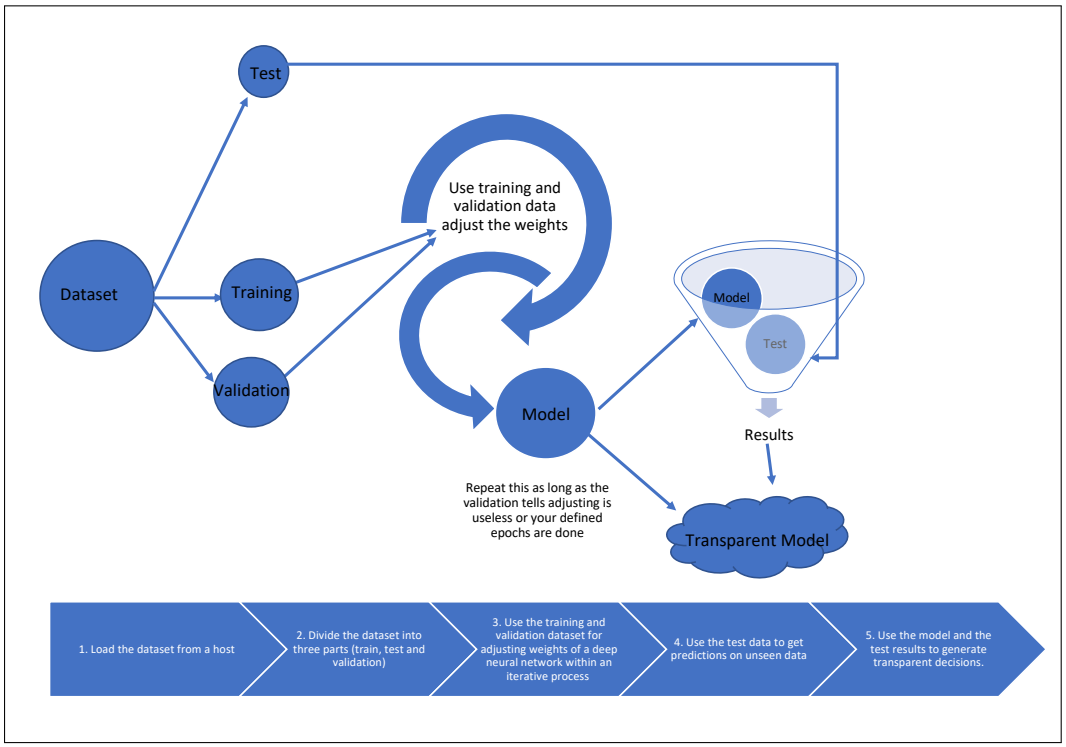


Figure (4.3): If the data is loaded (1) it will be decided into subsets afterwards (2). With the help of the training and validation subset, a deep neural network gets adjusted as long as it fits the training and testing data (3). Afterwards, the model gets tested of its ability to make predictions on the test subset (4). Finally, the model and the results will be used to make transparent decisions (5).

implementation of deep learning models. It turns out it is better to test a network twice. Once on unseen data after the model is trained (4). And twice with the validation subset, the model will be tested after each iteration (3). This allows stopping the iteration process of adjusting the wights before it comes to an overfit. The validation dataset influences the model. It is essential to know that validation can not be seen as a real test.

4.2. Scale the Data

As mentioned in the theory chapter, usually the data will be split into two subsets: a test set and trainset and sometimes to three: training set, validate set and test set. The model's weights are adjusted on the training data. The test set is used to make predictions on unseen data. When this is done, one of two things might happen: the model is overfitted or underfitted. Within the prototype implementation, it is important to avoid over and underfitting. In order to achieve that, the dataset is divided into three subsets. As can be seen in Figure 4.4, the training set is by far the largest one. The validation and training set is relatively small.

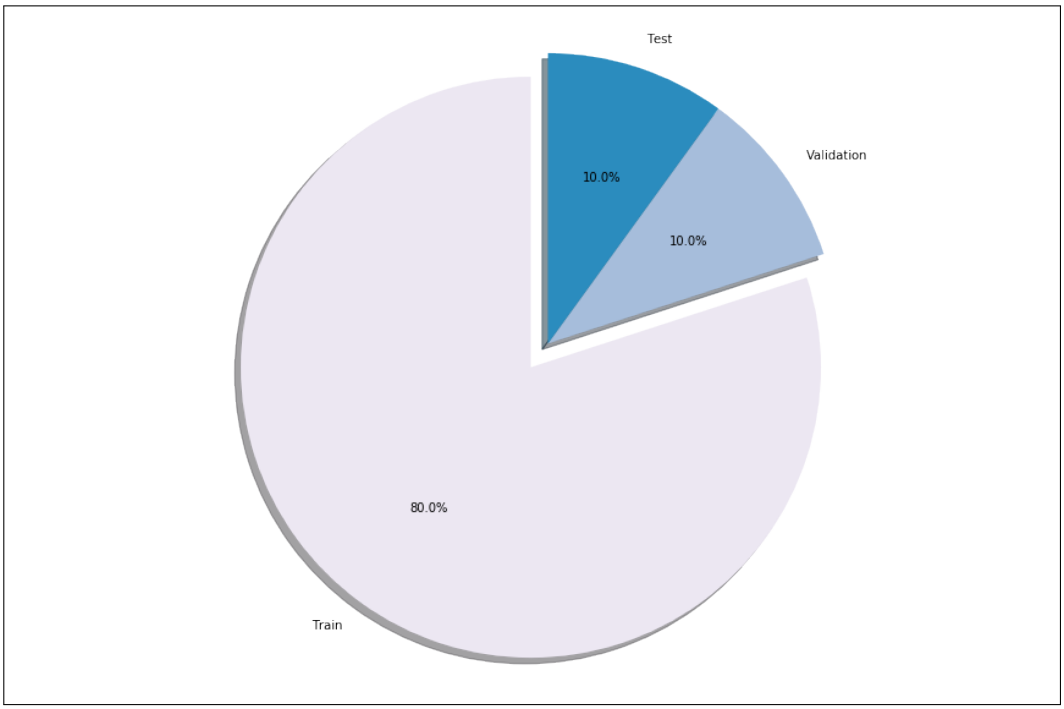


Figure (4.4): The Figure visualizes how the data for my prototype implementation is divided into different parts. The training set is by far the most enormous one. The validation and training set is relatively small.

Alternative approaches or a different size for each part can work, as well. With the help of an iterative process, the perfect size of each piece can be determined. But as long as the results are good (defined in the requirements section), this works fine for the prototype implementation.

4.3. Fit a Model

The prototype uses a convolutional deep learning approach with a so-called VGG16 architecture.

This architecture can be seen in Figure 4.5. The name is determined by its 16 different layers which consist of convolutional layers, max-pooling layers, activation layers and fully connected layers. VGG16 is a convolution neural network (CNN) architecture which was used to win the ILSVR (ImageNet) competition in 2014. It is considered to be one of the most relevant architectures than it comes to explain these kinds of models. A unique thing about VGG16 is that instead of having a large number of hyper-parameters, the founders of this architecture focused on having convolution layers with a 3×3 filter (stride = 1) and the same padding. Furthermore, it has a max-pooling layer with a 2×2 filter (stride = 2). It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. The following arrangement of the same building blocks over and over again makes it easy to explain. This is why the architecture is perfect if its decision shall be made transparent.

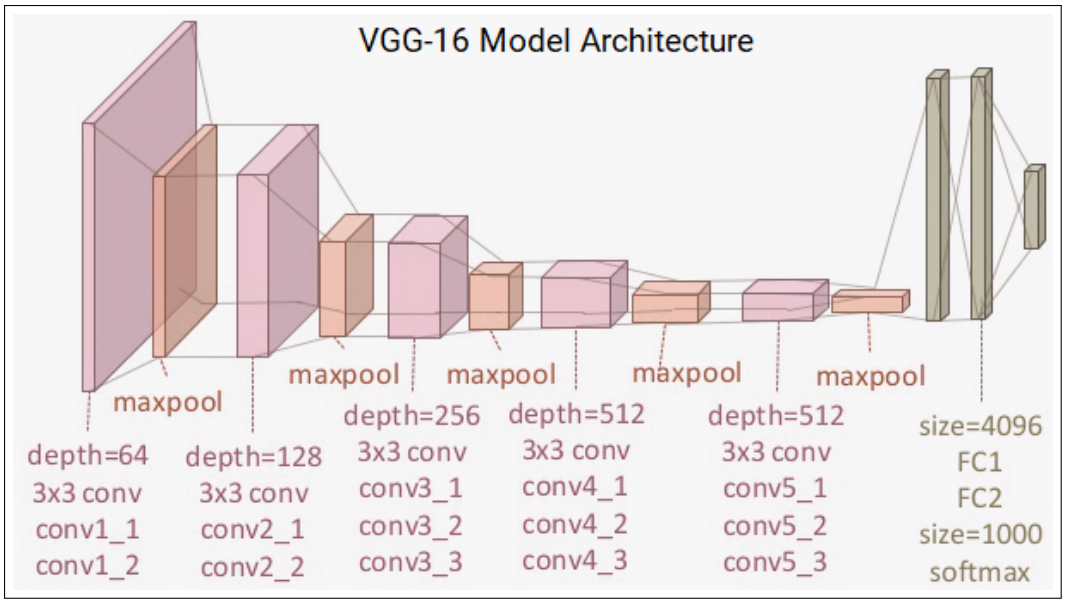


Figure (4.5): There are 13 convolutional layers, five Max Pooling layers and three Dense layers which sum up to 21 layers but only 16 weight layers. Convolutional layer 1 has 64 filters, while convolutional layer 2 has 128 filters, convolutional layer 3 has 256 filters while convolutional layer four and convolutional layer 5 has 512 filters. VGG-16 network is trained on the ImageNet dataset, which has over 14 million images and 1000 classes and archives 92.7% top-5 accuracy. It exceeds AlexNet (another convolutional neural networks architecture) by replacing large filters of size 11 and five in the first and second convolution layers with small size 3×3 filters.

In the following, I will explain the key factors around the VGG16 convolutional neural network architecture. The building blocks are described in Section 2.2.2.

- There are 13 convolutional layers, five max-pooling layers and three Dense layers which sum up to 21 layers, but only 16 weight layers
- The first convolutional layer has 64 filters
- The second convolutional layer has 128 filters
- The third convolutional layer has 256 filters
- the convolution layers four and five have 512 filters each

The VGG-16 network is pre-trained on the ImageNet dataset, which has over 14 million images and 1000 classes and archives 92.7% top-five accuracy. It exceeds AlexNet (another convolutional neural networks architecture) by replacing large filters of size 11 and 5 in the first and second convolution layers with small size 3×3 filters. VGG16 achieved great results. It is used even there are more advanced architectures today because it is easy to explain.

Even though the VGG16 is the right fit for the prototype, the architecture has some disadvantages. They are mentioned in the following because they affect the implementation process:

- It has such numerous weight parameters, that the model is very "heavy". Normally, the model is bigger than 500 MBs.
- Furthermore, the enormous amount of parameters lead to long computation time.

4.4. Make Predictions

A highly effective approach in the deep learning field is to use someone else's work. While this approach might be a no-go in other fields, it is most welcomed in the deep learning area. This works because a pre-trained network is usually trained on a big dataset for large scale image-classification tasks. Consequently, the hierarchy of features learnt by such networks can very effectively act as generic models for a lot of computer vision problems. Even if new issues involve completely different classes, than those of the original task for which the model was trained. This shift of learnt features is important for the success of pre-trained networks. There are two ways to use a pre-trained network:

1. Feature Extraction
2. Fine Tuning

In general convolutional neural networks involve two parts. First, they start with a series of convolutional and pooling layers as we had seen before in the section where the VGG16 model is described in detail. Usually, the architecture ends with a flatten layer. This part is called the convolutional base. The base is responsible for learning all the features of the images. On top the convolutional base sits the prediction layers, which consists of several dense layers, ending in the output layer with a softmax activation to predict some output classes (e.g. the pre-trained VGG16 ends with a softmax layer to predict 1000 classes). This technique involves reusing the convolutional base, which has already learnt feature representations from a large image set (like ImageNet). Then a custom prediction layer gets slapped atop the pre-trained convolutional base. During the training process, the convolutional base is frozen so that its weights are not adjusted. Just the weights of the custom prediction layer gets updated (Figure 4.6).

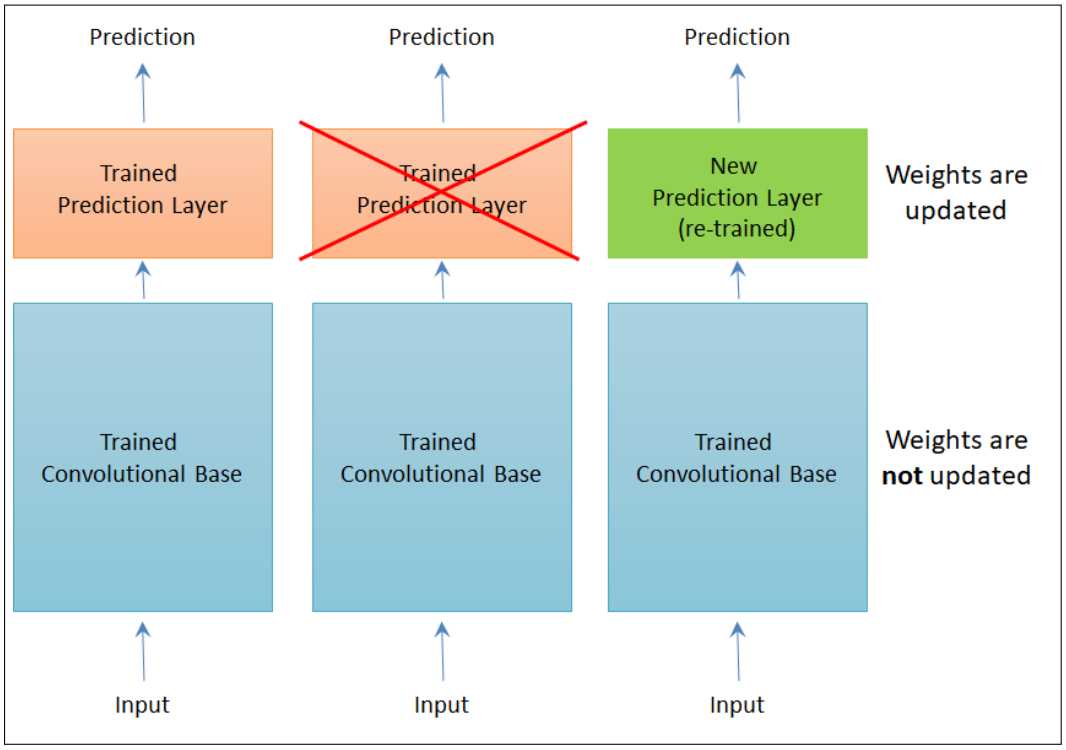


Figure (4.6): During the training process, the convolutional base is frozen (so that its weights are not updated). Weights of only the custom prediction layer get updated

The second approach is called fine-tuning. It is not used within the prototype implementation. This is why it is not explained in every detail. The idea behind is to unfreeze a few of the top layers of the convolutional base and jointly train it with the custom prediction layer which is put atop on the convolutional base (Figure 4.7).

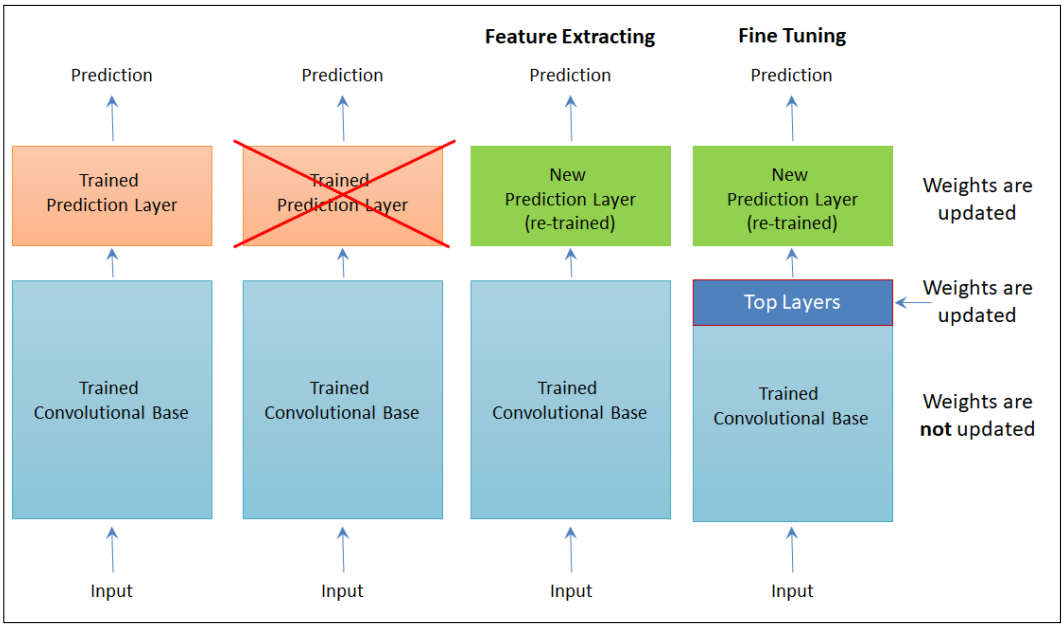


Figure (4.7): Fine Tuning means, freezing a few of the top layers of the convolutional base as in the feature extraction, and jointly train it with the custom prediction layer which we slapped atop the convolutional base.

4.5. Make Transparent Decisions

SHAP Gradient Explainer was fully covered in the corresponding theory section. It is briefly summarized, and a few alternative approaches are introduced in the following. Moreover, a short statement is given why the SHAP is the right fit for the implementation of transparent decisions.

Integrated gradients values are a bit different from SHAP values and require a single reference value to integrate from. However, in SHAP Gradient Explainer, expected gradients reformulate the integral as an expectation and combine that expectation with sampling reference values from the background dataset. The technique uses an entire dataset as the background distribution versus just a single reference value.

There are a wide variety of techniques and tools for interpreting decisions made by deep learning models. Except for the SHAP Gradient Explainer, the significant methods are the following:

- Visualizing activation layers visualize how a given input comes out of specific activation layers. The approach explores which feature maps are getting activated in the model.

- Occlusion sensitivity visualizes how parts of the image affect deep neural networks confidence by hiding parts of the image iteratively.

- Grad-Cam visualizes how parts of the image affect neural networks output by looking into the gradients backpropagated to the class activation maps.
- SmoothGrad averages the so-called sensitivity maps for an input image to identify pixels of interest.

This gives an overview of what the neural network models do. The list of techniques here is not exhaustive. It just covers some of the most popular and widely used methods to interpret convolutional neural network models.

The decision for the SHAP is based on the following two reasons:

1. It is no longer the state of the art than it comes to make transparent decisions, but it is easy to understand.
2. It can be used in other machine learning fields as well. If it comes to generalization and to find a conclusion, it makes sense to have an implementation which fits a broad range of issues.

This chapter was very extensive because it covers a lot of necessary aspects for the implementation of the prototype. As mentioned before, detailed explanations for almost all points mentioned in this chapter are in the Theory Chapter (2). Unfortunately not every detail could be considered, because otherwise, the work would be too large. Anyway, the next chapter is about the technical specifications of the prototype. More precisely, which libraries possibly exist to implement the prototype and which are finally used.

Chapter 5.

IT-Specifications and Implementation

"What would life be if we had no courage to attempt anything?"

- Vincent Van Gogh

The previous two chapters focus on the theory of the prototype. In contrast, this chapter goes into detail of the implementation.

5.1. The Prototypes Structure

The structure of the prototype implementation is more or less identical to the five steps which are introduced in the requirements of the prototype (Chapter 3). In Figure 5.1, the steps are visualized again. Additionally, Figure 5.1 contains information about which tools are mainly used in each process step.

Python and the so-called Matplotlib library are used in the overall implementation of the prototype. Whereas Python is the primary programming language, Matplotlib is used to make visualizations. For example, if a dataset is successfully loaded (Section 4.1), the images are visualized with Matplotlib. Furthermore, Matplotlib is used to create diagrams for the thesis in order to support statements as can be seen in the following listing:

```
1  colors = ( '#ece7f2','#a6bddb','#2b8cbe')
2  distincted_lables = np.array([2,10,100])
3  fig, (ax1,ax2,ax3) = plt.subplots(1, 3,figsize=(15,10))
4  ax1.bar(data_sets, data_sets_sizes, color=colors)
5  ax1.set_title("Size (MB)")
6  ax2.bar(data_sets, distincted_lables, color=colors)
7  ax2.set_title("Distinced labels")
8  ax3.bar(data_sets, (data_sets_sizes/distincted_lables), color=colors)
9  ax3.set_title("Size / Distinced Label")
10 plt.show()
```

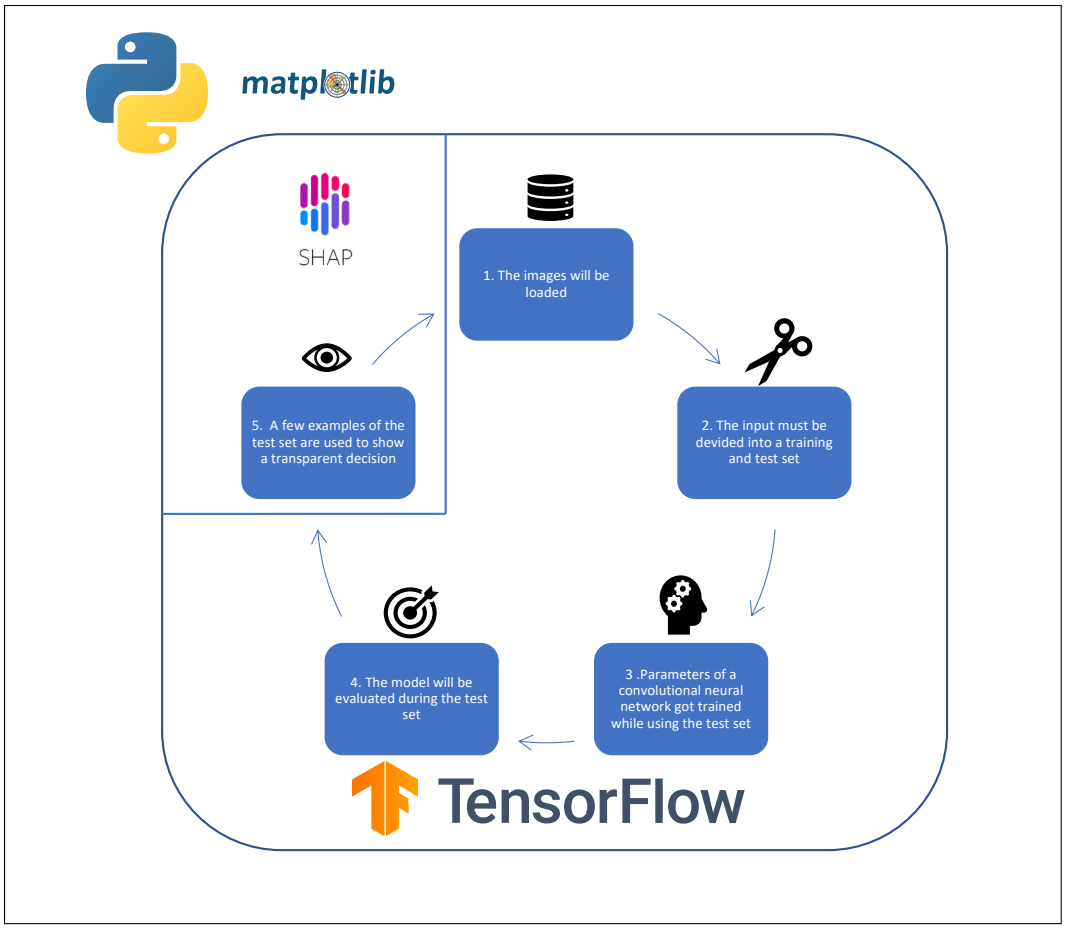


Figure (5.1): Python and the so-called Matplotlib library are used in the overall implementation of the prototype. Whereas Python is the primary programming language, Matplotlib is used to make visualisations. Also, it is used to create diagrams for this thesis to support statements like Figure 4.1 The data engineering, the preprocessing steps, the VGG16 model and the evaluation criteria are implemented with the help of the TensorFlow library. The SHAP library is used within the last step, to make transparent decisions.

The result of the code can be seen in Figure 4.1 and is used in the section about the input data for the prototype (Section 4.1).

The data, the preprocessing steps, the VGG16 model and the evaluation criteria are implemented with the help of the TensorFlow library (Sections 4.1 - 4.4). The SHAP is used within the last step, to make transparent decisions (Section 4.5).

In the following, it is described why the tools are used within the prototype implementation. Besides, the tools are compared to alternatives.

5.2. Python as programming Language for the prototype

Python is the primary programming language for the prototype implementation because of the following reasons:

1. Python is the most popular language when it comes to machine learning and deep learning. This can be observed in Figure 5.2. It shows the percentage of matching job postings and is created by the job portal Indeed. The Figure compares the languages Python, R, Java, Javascript, C, C++, Julia and Scala [Puget, 2016].
2. Pythons excellent libraries, which allows implementing the required steps straight forward
3. Because of its popularity and because the language is open source, there is a vast community to help with problems. Furthermore, a lot of documentation for Python is available online.
4. As already named as a reason above, Python offers a variety of libraries, and some of them are excellent visualisation tools as well. However, for the prototype implementation, it is necessary to be able to represent data in a human-readable format.
5. Libraries like Matplotlib allow building plots for better data understanding and visualisation. Different application programming interfaces also simplify the visualisation process and make it easier to generate figures for transparent decisions.

A few alternatives to Python can be seen in Figure 5.2. Every programming language has its downsides, and none of these languages is perfect. For example, other programming languages are maybe faster and more efficient as Python. In comparison, Python is considered as the languages with better packages and the greater community. These two points there mentioned as reasons why Python is the choice for the prototype implementation. If the community and package support would increase in other languages, the decision could be different.

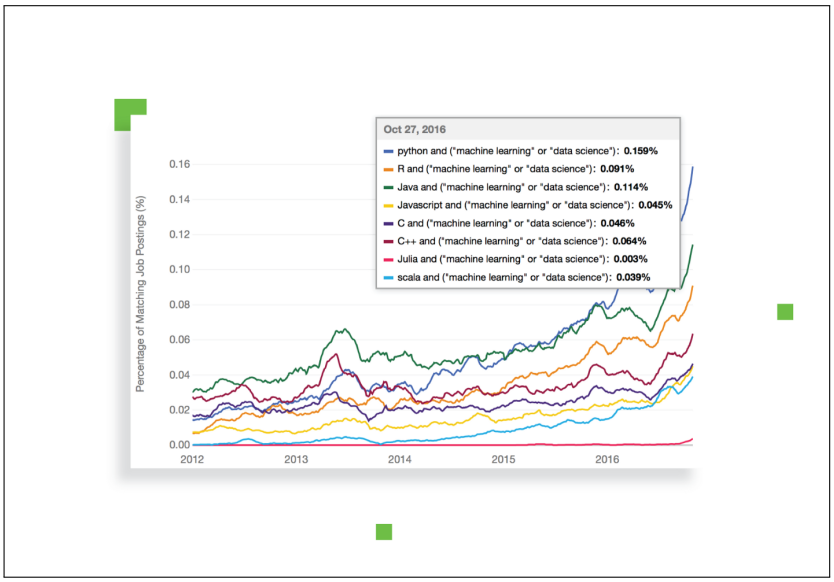


Figure (5.2): Python is the most popular language when it comes to machine learning and deep learning. The Figure shows the percentage of matching job postings and is created by the job portal Indeed. The Figure compares the languages Python, R, Jave, Javascript, C, C++, Julia and Scala [Puget, 2016].

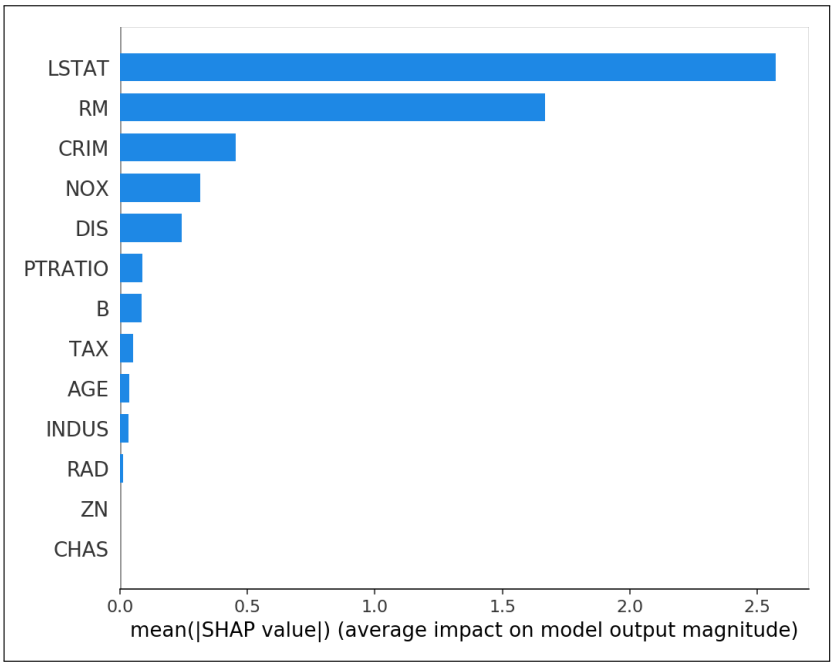


Figure (5.3): The Figure shows a build-in visualization from the SHAP library. It can be used to plot a visualization of the SHAP values for non-image data.

5.3. Libraries within the Prototype implementation

5.3.1. Matplotlib

Matplotlib is already mentioned in the section about the structure of the prototype as a tool for visualisation. Furthermore, it is said as one of the reasons why Python is used as the primary programming language in the first place.

Matplotlib is quite easy. Usually, while plotting, the same structure is used in every new plot. For example, for questions that arose during this thesis, Matplotlib can be used to visualise the answers in an easy way. As can be seen in the following listing, the code which is used to create two different diagrams is quite similar:

```
1  # Figure 1
2
3  colors = ( '#ece7f2','#a6bddb','#2b8cbe')
4  distincted_lables = np.array([2,10,100])
5  fig, (ax1,ax2,ax3) = plt.subplots(1, 3,figsize=(15,10))
6  ax1.bar(data_sets, data_sets_sizes, color=colors)
7  ax1.set_title("Size (MB)")
8  ax2.bar(data_sets, distincted_lables, color=colors)
9  ax2.set_title("Distinced labels")
10 ax3.bar(data_sets, (data_sets_sizes/distincted_lables), color=colors)
11 ax3.set_title("Size / Distinced Label")
12 plt.show()
13
14 # Figure 2
15
16 labels = 'Train', 'Validation', 'Test',
17 sizes = [80,10,10]
18 explode = (0.1, 0, 0)
19 plt.rcParams['font.size'] = 20
20 fig1, ax1 = plt.subplots(figsize=(15,10))
21 ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
22 shadow=True, startangle=90, colors = colors)
23 ax1.axis('equal')
24 plt.show()
```

The name comes from the fact that Matlab is emulated. Matlab is an enterprise software environment and programming language which is traditionally used by scientists and companies to work on machine learning tasks. It is expensive, difficult to scale and as a programming language, it is slow in comparison with other programming languages which are introduced in Figure 5.2. However, Matlab offers excellent ways to implement data visualisation. So, Matplotlib in Python is used as a robust, free and easy library for data visualisation instead.

Seaborn is an alternative approach than it comes to visualise data in Python. It is almost the same with libraries for visualisation and programming languages. Like described at the end of the python section, visualisation libraries have differences in use cases, scalability and many other things. So, based on this statement, the best visualisation tool for the particular example of the prototype implementation is Matplotlib. If the prototype would contain more statistics instead of image recognition, then Seaborn is the right choice because it has a lot of things suitable for mathematical tasks, built-in. The prototype has to deal with image data, and Matplotlib offers excellent opportunities to visualise them. For example, to get a first impression of the downloaded data, it takes just a few lines of code to visualise them:

```
fig = plt.figure(figsize=(20, 4))

for i in np.arange(20):
ax = fig.add_subplot(2, 10, i+1, xticks=[], yticks=[])
plt.imshow(x_train[:20][i], cmap='gray')
ax.set_title(cats_vs_dogs_labels[y_train[:20][i].item()])
```

X_train contains the image data and y_train the corresponding labels as described in the Theory Chapter (Chapter 2).

5.3.2. TensorFlow

TensorFlow is used as a full-stack library, for four out of five steps of the prototype implementation. That means it offers ways to implement data loading, data preprocessing, creating models and make predictions (Sections 4.1 - 4.4).

TensorFlow offers different levels of abstraction. So, for the prototype implementation, a level of abstraction is used, which provides two things. First, a syntax which makes it easy to implement ideas and second enough freedom to adjust the code. To adjust the code is important because as described in the functional-specifications, the base of the pre-trained model has to be changed. If the syntax is overwhelming, it can make things worse. Mainly because the goal is to make transparent decisions, it is not good to use an abstraction level that confuses even more. This is why Keras, as a high-level application programming interface, is used as a sufficient level of abstraction.

For example, if it comes to implement a VGG16 architecture as described in the functional specifics, this can help a lot. In the following listings, two things can be observed. First, the code which is used in the prototype implementation needs just a few lines of code to load a pre-trained VGG16 model. That means the code offers two things as well. At first, the model is written as executable code, and then the pre-adjusted weights get loaded:

```
from keras.applications.vgg16 import VGG16

```

```
3  model = VGG16(weights='imagenet', include_top=True)
4  model.summary()
```

The following listing shows an implementation from scratch, of a simple neural network with just one neuron:

```
1  class NeuralNetwork:
2  def __init__(self, x, y):
3  self.input      = x
4  self.weights1   = np.random.rand(self.input.shape[1],4)
5  self.weights2   = np.random.rand(4,1)
6  self.y          = y
7  self.output     = np.zeros(self.y.shape)
8
9  def feedforward(self):
10 self.layer1 = sigmoid(np.dot(self.input, self.weights1))
11 self.output = sigmoid(np.dot(self.layer1, self.weights2))
12
13 def backprop(self):
14 # application of the chain rule to find derivative of the loss function
                                        with respect to weights2 and weights1
15 d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) *
                                        sigmoid_derivative(self.output)))
16 d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) *
                                        sigmoid_derivative(self.output), self.
                                        weights2.T) * sigmoid_derivative(self.
                                        layer1)))
17
18 # update the weights with the derivative (slope) of the loss function
19 self.weights1 += d_weights1
20 self.weights2 += d_weights2
```

It has more lines of code, even though the model is so simple. This points out the importance of a library than it comes to the implementation of a deep learning model.

Furthermore, the library is used to get the data and adjust it such it fits to the use case of my prototype. Instead of download the data by hand and use multiple libraries to get the result, TensorFlow provides a very handy end-to-end solution for the first two steps. With just one function call it downloads the images, scales them and split them into different parts:

```
1  (raw_train, raw_validation, raw_test), metadata = this.load(
2  'cats_vs_dogs',
3  split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
4  with_info=True,
5  as_supervised=True,
6  )
```

Even the alternative library PyThorch is faster, and its syntax is even more apparent, the opportunities to handle two out of five steps with just one function call made the decision clear. TensorFlow is the best fit for the prototype implementation because it allows implementing the whole prototype with just a few lines of code.

5.3.3. SHAP Library for transparent decision

Based on the work of Luneberg and Lee, the SHAP library offers a method which allows computing the SHAP values after each iteration from each layer. For a visualization, as described in the requirements (Chapter 3) and the functional specifics (Chapter 4), a separate implemented function is required. The SHAP software library others different classes and methods which optimise the calculation of SHAP values for various machine learning approaches. The prototype uses deep explainer, which is optimised for deep learning models. It is not only for convolutional neural networks, but at least, the implementation works fine and comes to a result in a sufficed amount of time.

There is no approach which provides similar functionalities and classes as the SHAP Library. This is why the library can not be compared with others and has to be used for the calculations. Otherwise, the implementation has to be done from scratch.

5.3.4. Implementation

Data visualisation and model explainability are two integral aspects of the prototype implementation. They are the binoculars helping to see the patterns in the data, which lead to prototypes decisions. That is why the prototype uses a function which combines the original image with another layer that shows the calculated values on top. The visualisation is based on the visualisation, which is already offered by the SHAP library for non-image data. Figure 5.3 shows such a kind of visualization from the SHAP library.

It can just be implemented by calling the shap_plot() Method. As a parameter, it gets the features, the calculated SHAP values and the corresponding machine learning model. The function which is implemented in the prototype works similar. However, it allows - by the following function call - to get visualizations of the SHAP values for image data:

```
1  visualize_model_decisions(shap_values=shap_values, x=to_predict,
2  labels=index_names, figsize=(20, 40))
```

The output of the function can be seen in Figure 5.4. More precisely, the images show how the function was used on the dataset after predictions were made.

Note that not every aspect of the shown listings is explained in any detail. They provide a better understanding of the mentioned points in each section. The theoretical background

Figure (5.4): The output of the visulaize_model_deciions function. More precisely, the images show how the function was used on the dataset after predictions were made to visualize the SHAP values of the 7th layer in a convolutional neural network.

and the functional usage towards the listings is described in the corresponding section at the Functional Specifications (Chapter 4) and Theory Chapter (2).

Chapter 6.

Evaluation

"Small daily improvements over time lead to stunning results?"

- Robin Sharma

The evaluation goes through the points which are mentioned in the requirements chapter (Chapter 3):

1. The Prototypes ability to successfully recognize cats and dogs (greater than 90% accuracy on unseen images)
2. The Prototypes ability to calculate SHAP values
3. The Prototypes ability to visualize these values on the images which are successfully recognized by the model.

Even more, it was mentioned that the second and third ability is hard to evaluate because these abilities are qualitative capabilities. This is why these requirements can have just to status: Fulfilled or not fulfilled without concerning the levels of fulfilment.

6.1. Ability to predict

As in Figure 5.1 can be seen that the prototype can be evolved within an iterative processes. The visualization in Figure **??** visualizes the result of a very simple convolutional neural network. The visualization shows characteristics of overfitting. The training accuracy increases linearly over time, until it reaches nearly 100%, while the validation accuracy is at 70-72%. The validation loss reaches its minimum after only five epochs, then it does not change anymore, while the training loss keeps decreasing until it reaches nearly zero.

This is characteristic for so-called overfitting which was described in the section about convolutional neural networks.

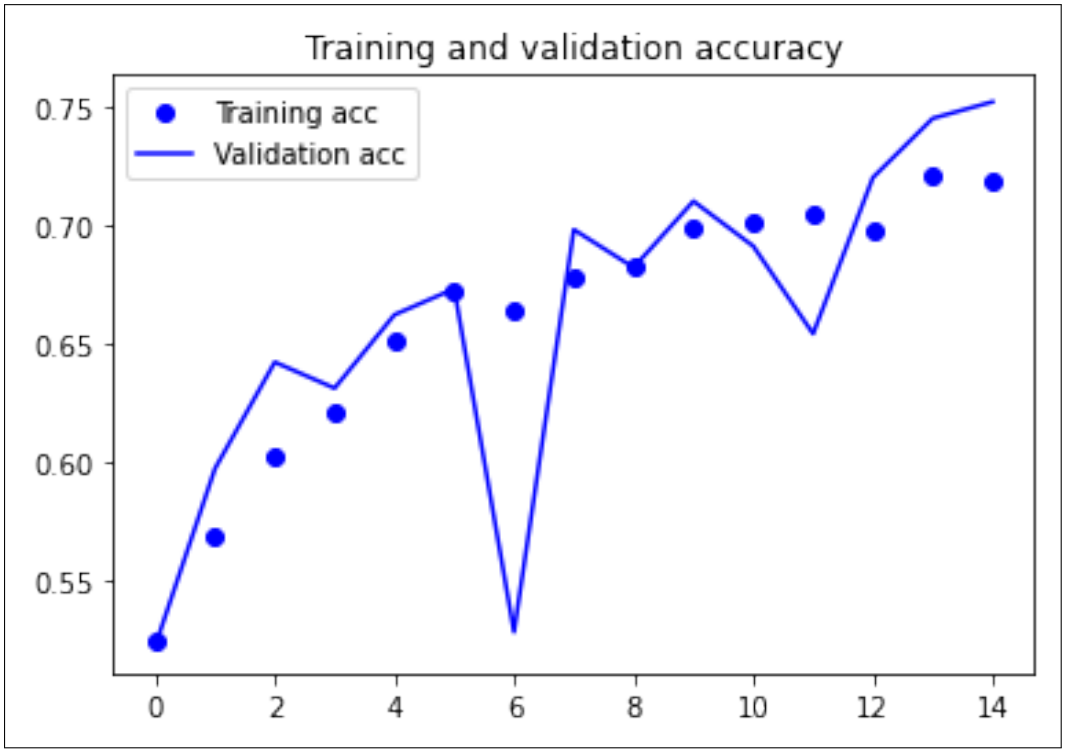


Figure (6.1): The visualization shows characteristics of overfitting. The training accuracy increases linearly over time, until it reaches nearly 100%, while the validation accuracy is at 70-72%. The validation loss reaches its minimum after only five epochs, then it does not change anymore, while the training loss keeps decreasing until it reaches nearly zero

Within the next step, the requirement form over 90% will be achieved. With the help of a pre-trained model, the variance bias trade was optimized as can be seen in Figure 11.

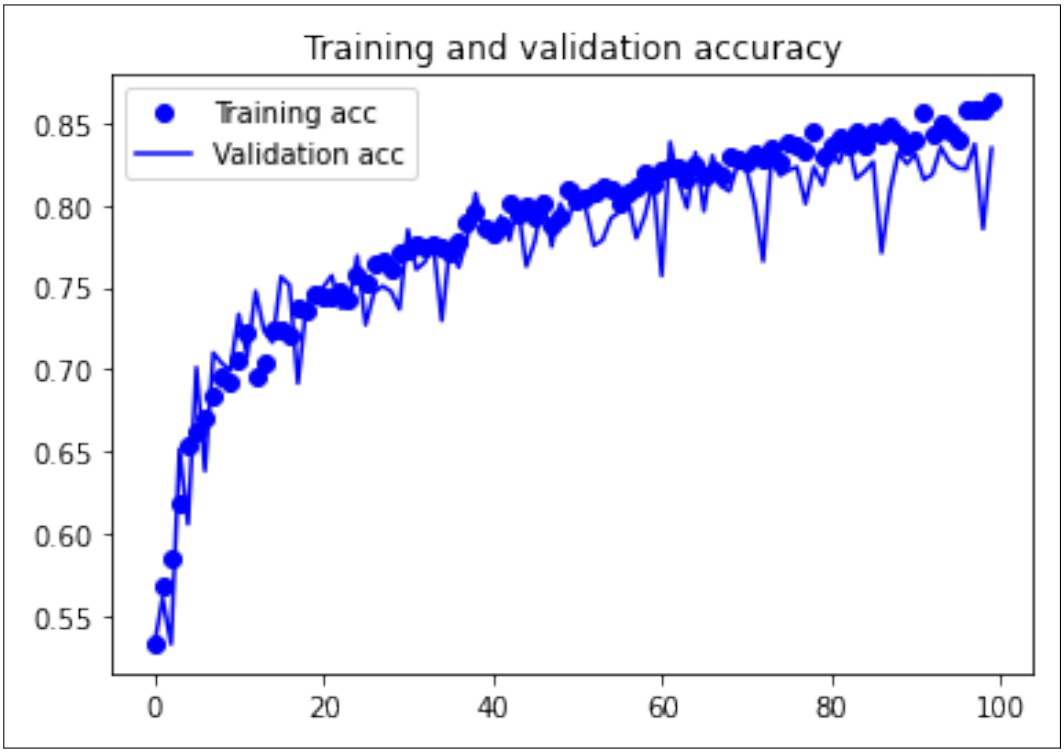


Figure (6.2): The visualization shows how with the help of a pre-trained model, the variance bias trade was optimized.

6.2. Ability to calculate SHAP values

As described in the IT-Concept, the calculation of the SHAP is done by the SHAP library. The function form the SHAP library uses three things form the prototypes implementation:

1. the deep learning model (model),
2. a subset of the test set (to_explain)
3. two batches to predicted labels (X).

The model is used to get the inputs and outputs of the defined layers. For the example of the prototype, it uses the following layers to get the SHAP values:

- The 7th
- The 14th

The following listing shows how the calculation is implemented, with the help of the SHAP library:

```
e = shap.GradientExplainer(
(model.layers[2].input, model.layers[-1].output), map2layer(X, 2),)
shap_values,indexes = e.shap_values(map2layer(toExplain,2),rankedOutputs=1)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, to_explain, index_names)
```

How this is done is partly described in the corresponding SHAP section. But as a brief reminder:

> *What it does it, it uses two batches as another training set and simulates a training for which it estimates the SHAP values of the subset. A more detailed explanation can not be given because of its complexity. It can fill another thesis with ease.*

So, the prototype fulfils actually two out of the three defined requirements. First, the implementation recognizes cats and dogs on unseen images with an accuracy of over 90%. Second, it calculates SHAP values for every model in the implementation.

6.3. Ability to visualize SHAP values

The function which is described in the IT-Specifications can visualize the SHAP values on top on images. Figure 5.4 shows how this looks for the 7th layer of the prototypes model. The intuition about it is:

> *The visualization put another layer on an image which was recognized from the prototype. It shows the labels and it highlights important parts of the images red. In this case important means important with respect to the absence of other parts in the image as described in the theory chapter (Chapter 2)*

Finally, the prototype fulfils every requirement which is defined in the Requirements (Chapter 3). This means it can recognize cats and dogs, calculate SHAP values and visualizes them to make the decisions more transparent.

6.4. Evaluation of the implementation process

Even though it was not mentioned before the implementation process shall be evaluated as well, to ensure the reader is not wondering why the append code differs from the IT-Concept. The implementation is slightly different, as described in the IT-Specification because the SHAP, TensorFlow Datasets and the TensorFlow libraries had some major changes while the prototype was implemented. This leads to a couple of problems when it comes to the implementation. For the evaluation this means the IT-Specification was unsuccessful. Nevertheless, to present a result which can be evaluated, the implementation was changed. For example, the prototype works know with the original dataset from the website kaggl.com. The changes are not presented in the process because the basic structure is still the same. So, the reader shall be able to understand and run the append code anyways.

Chapter 7.

Generalization

"The big picture doesn't just come
from distance; it also comes from time."

- Simon Sinek

The prototype was successful due to the evaluation process within the last section. It can be seen how the decisions of an image recognition task can be visualized. This chapter focus on the question of what this means in a more general sense.

The Introduction presented a couple of reasons why explainable artificial intelligence is more and more critical in the next few years, especially when it comes to the use of upload filters in social media.

In the following, this statement will be combined with the evaluated results of each category to draw a complete picture:

7.1. Ability to predict

As can be seen, it is possible to recognize patterns on images. The problem is that real-world use cases are not binary as the prototype assumes (cats and dogs). Moreover, it is a real challenge to get the right amount of suitable training examples. Research points out that there is still a lot of work to do if it comes to the use of convolutional neural networks in praxis.

At the same time, research points out that convolutional neural networks are used in praxis and that around 90% is enough accuracy if the application is not critical as in social media. Besides, even then it is crucial to have a high accuracy as in the financial sector, those networks already used in practice.

The statement above leads to the assumption that it is still essential to have explainable models, as shown in the Introduction, even though it will take a few more years as those networks become best practice.

7.2. Ability to predict Shapley values

One of the drawbacks of SHAP values is its calculation time. Because of the popularity of mobile devices, it gets more and more essential to use efficient methods. This is also true when it comes to calculating transparent models, for example, for the use within an upload filter. So, in general, this leads to the statement that SHAP values are not very good, then it comes to upload filters because it needs vast hardware resources and takes long for computation.

It would be better to have methods which can be computed within seconds such that users had a transparent decision as described in the Introduction immediately.

There are a few methods which are already fast in comparison with the SHAP values. They could be the right choice in the upcoming future.

7.3. Ability to visualize SHAP values

Figure 5.4 shows that it is possible to visualize SHAP values to make a model more transparent. It is at least questionable if a non-technical user can understand the intuition about it.

The discrepancy between technical and non-technical users is typical for explainable artificial intelligence. A user who is in the field in computer science or something related to it can understand visualization of SHAP values. Still, it is hard to say if the visualization is transparent enough for every user. For example, it is not clear if an Instagram user will have any idea why its contend get blocked after he or she had a look on the SHAP value visualization.

I think in the upcoming future visualizations are the key to success then it comes to explainable artificial intelligence. The problem is that this is not just a field of machine learning engineers and data scientists. Instead, it is a cross-functional filed for user experience designers, business analysts, computer scientists and machine learning engineers. For example, for a transparent decision of an upload filter, it is essential to understand the interaction between the user and the visualization (user experience design). Furthermore, the model shall be in the interest of the companies interests (business analysts), and the ideas must be computable with as few resources as possible (computer scientists). Last but not least, the visualization shall present the correct results (data scientist).

Chapter 8.

Conclusion

> "It is beyond a doubt that all our knowledge begins with experience."
>
> - Immanuel Kant

I think the rise of deep learning has been defined by a shift away from transparent and understandable human-written code towards complicated black boxes. Their creators have still little understanding of how or even why the algorithms came to a specific decision. The use of these complicated models in real-world situations like upload filters leads to a need for transparent models.

8.1. Metaphoric interpretation of the evaluation and the generalization

I think it can be compared with the development of programming languages. In the beginning, these were difficult for people to understand. It took experts to use them for programming applications. Even today an excellent training is vital to use these tools professionally. But the handling has become more accessible and more intuitive for humans. Today it is possible that even kids can have fun with simple programming tasks and learn to use them intuitively. In this metaphor, the beginning of the programming languages stands for the theory chapter. It can be seen how complicated the connection between the recognition of patterns on images and the associated decision making is. In the following, the prototype showed how a simple visualization could lead to an improved intuition of a deep learning model. I believe that as shown in the generalization chapter, a lot of work in different areas is necessary to get to the same level as simplicity as in some "toy" programming languages like Scratch today.

I hope that this development will take place and that in a few years it will be possible for everyone to use these programs intuitively. Even if it still needs some experience in this field, it should be possible for everyone who is affected by an algorithmic decision to

understand that decision. Th upload filters are an example of why this is important. Also, the corresponding discussion in the media and protests around the upload filter discussion shows that at least in Germany, there is an interest in the understanding of how algorithms make decisions.

8.2. Personal Statement

I hope to be able to meet this interest and responsibility in my master's degree, which computer scientists have when it comes to making deep learning models more transparent.

Appendix A.

Supplemental Information

Appendix B.

Supplemental Information

I provided my Jupyter Notebook as a PDF, since I wrote my entire prototype in this Jupyter Notebook.

Automatic image recognition in upload filters

June 10, 2020

The following Notebooks represents the Prototype which is introduced in the Bachelorthises "Automatic image recognition in upload filters - computing of transparent decisions (XAI), with the help of Deep Learning methods". The Prototype Implementation was created to show how the SHAP library can be used to make convolutional networks decisions transparent.

Therefore it uses a VGG16 architecture and transfer learning as can be seen in the corresponding thesis.

[3]:
```
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.models import load_model
from matplotlib import pyplot
import keras.backend as K
import numpy as np
import json
import shap
import os
import shutil
import sys

#print(keras.__version__)
#print(shap.__version__)
```

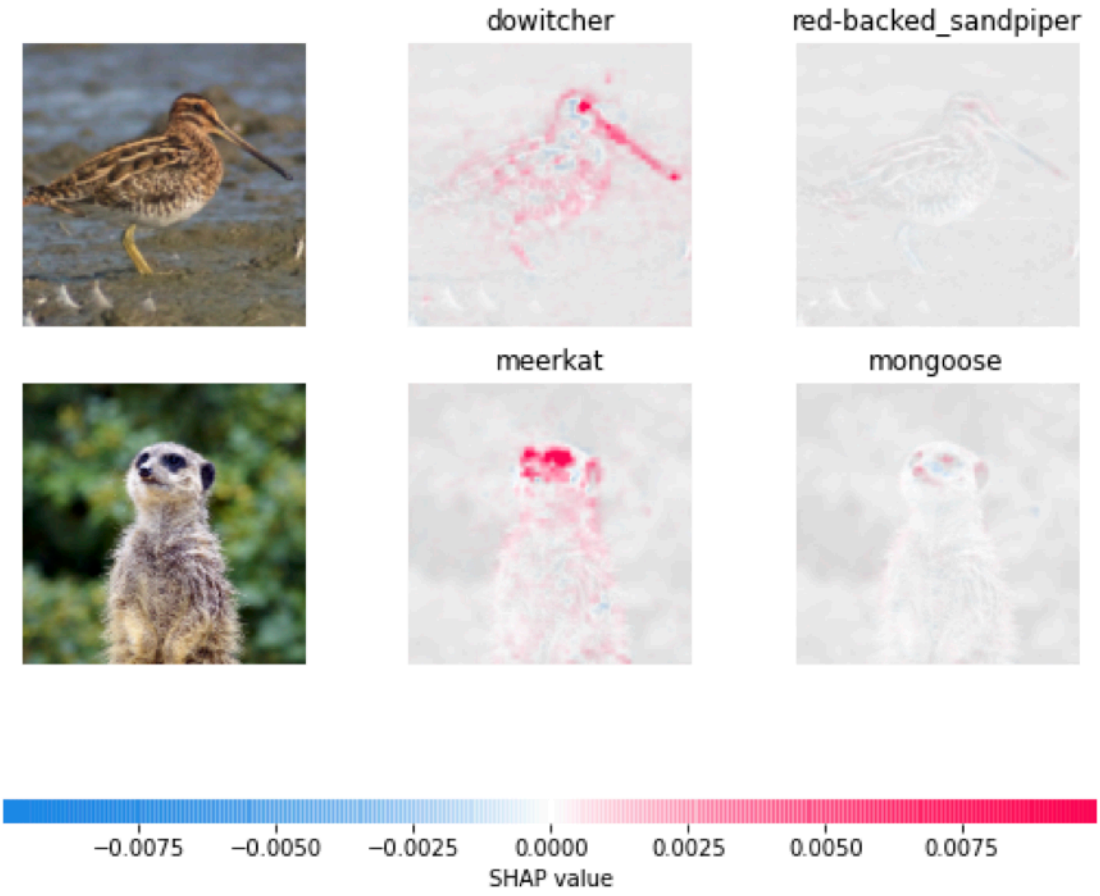
1 Test of the SHAP Library

1.1 Use the example case from the github page

[4]:
```
# load pre-trained model and choose two images to explain
model = VGG16(weights='imagenet', include_top=True)
X,y = shap.datasets.imagenet50()
to_explain = X[[39,41]]
```

```
# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/
 ↪imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)
```

[5]:
```
# explain how the input to the 7th layer of the model explains the top two␣
 ↪classes
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[7].input, model.layers[-1].output),
    map2layer(X, 7),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(to_explain, 7), ranked_outputs=2)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, to_explain, index_names)
```



2 Test the SHAP with a different data set

2.1 Unzip the dataset and put it into the right directories

[11]:
```
# organize dataset into a useful structure
from os import makedirs
from os import listdir
from shutil import copyfile
from random import seed
from random import random

# create directories
dataset_home = 'dataset_dogs_vs_cats/'
subdirs = ['train/', 'test/']
for subdir in subdirs:
    # create label subdirectories
    labeldirs = ['dogs/', 'cats/']
    for labldir in labeldirs:
```

```
        newdir = dataset_home + subdir + labldir
        makedirs(newdir, exist_ok=True)
# seed random number generator
seed(1)
# define ratio of pictures to use for validation
val_ratio = 0.25
# copy training dataset images into subdirectories
src_directory = 'datalab/train/'
for file in listdir(src_directory):
    src = src_directory + '/' + file
    dst_dir = 'train/'
    if random() < val_ratio:
        dst_dir = 'test/'
    if file.startswith('cat'):
        dst = dataset_home + dst_dir + 'cats/'  + file
        copyfile(src, dst)
    elif file.startswith('dog'):
        dst = dataset_home + dst_dir + 'dogs/'  + file
        copyfile(src, dst)
```

[16]:
```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_dir = 'dataset_dogs_vs_cats/train/'
validation_dir = 'dataset_dogs_vs_cats/test/'


train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(224, 224),
        batch_size=50,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(224, 224),
        batch_size=50,
        class_mode='binary')
```

```
Found 18697 images belonging to 2 classes.
Found 6303 images belonging to 2 classes.
```

2.2 Choose a subset which got preditced with help of the convolutional neural network

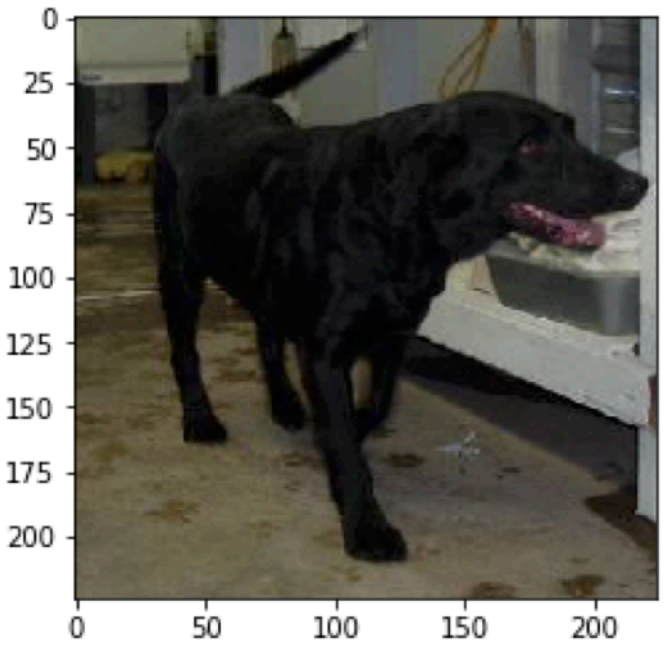
[41]:
```
for data_batch, labels_batch in train_generator:
    X = data_batch
    y = labels_batch
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break

to_explain = X[8:10]
to_explain.shape
```

```
data batch shape: (50, 224, 224, 3)
labels batch shape: (50,)
```

[41]:
```
(2, 224, 224, 3)
```

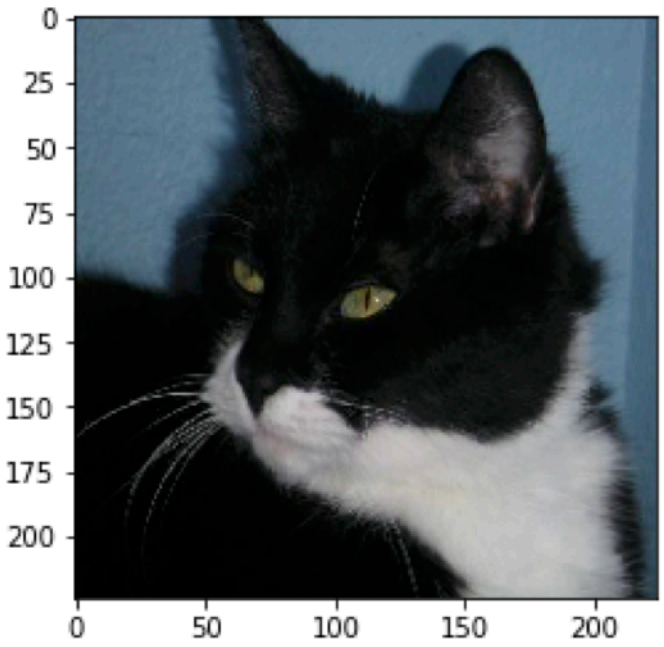
[42]:
```
print(pyplot.imshow(to_explain[0]))
```

```
AxesImage(54,36;334.8x217.44)
```



[43]:
```
print(pyplot.imshow(to_explain[1]))
```

```
AxesImage(54,36;334.8x217.44)
```

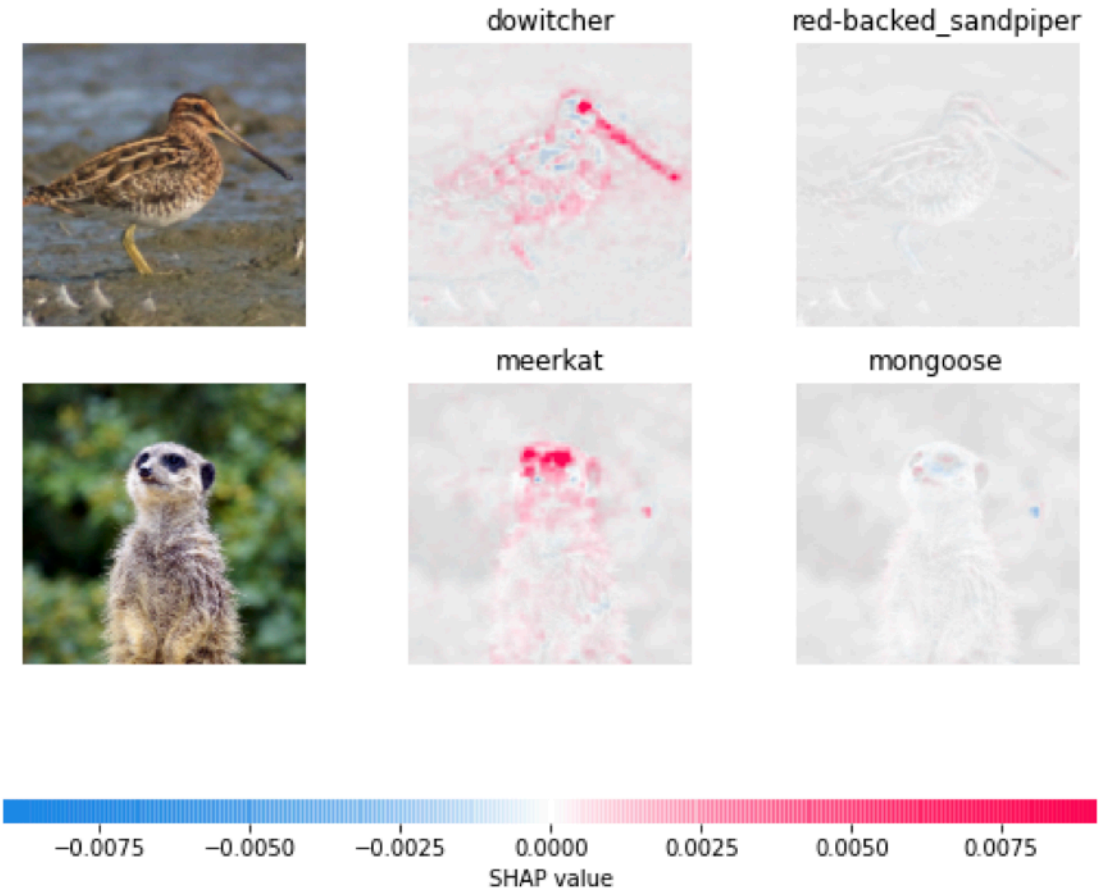


2.3 Calculate the SHAP Values for my own Dataset

2.3.1 Show the output of the 7th layer

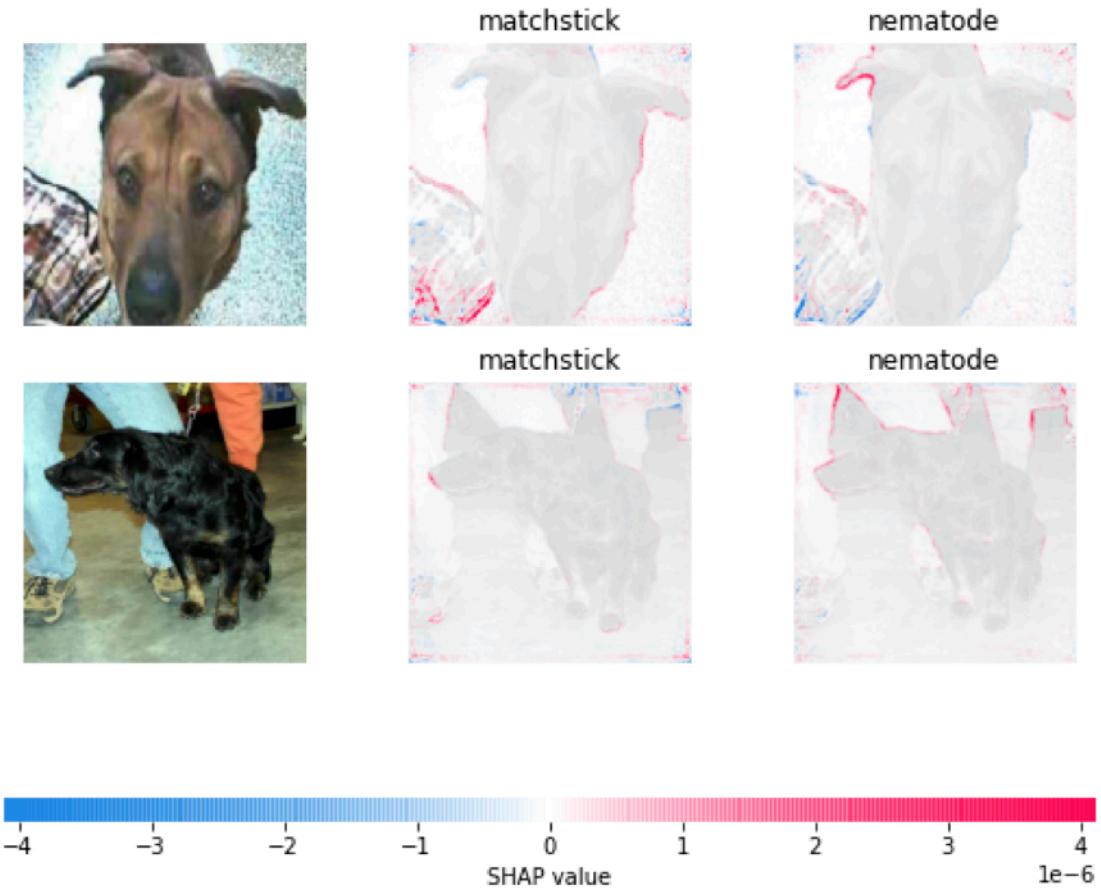
[13]:
```
# explain how the input to the 7th layer of the model explains the top two␣
↪classes
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[7].input, model.layers[-1].output),
    map2layer(X, 7),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(to_explain, 7), ranked_outputs=2)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)
```

[14]:
```
# plot the explanations
shap.image_plot(shap_values, to_explain, index_names)
```



[15]:
```
### Show the output of the 14th layer
```

[21]:
```
# explain how the input to the 7th layer of the model explains the top two␣
↪classes
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[3].input, model.layers[-1].output),
    map2layer(X, 3),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(to_explain, 3), ranked_outputs=2)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, to_explain, index_names)
```



[16]:
```
# Train a model from sratch and visualize it
```

[9]:
```
# baseline model for the dogs vs cats dataset
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD

# define cnn model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_uniform', padding='same', input_shape=(224, 224, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
```

```
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(1, activation='sigmoid'))
# compile model
opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])


# plot diagnostic learning curves
def summarize_diagnostics(history):
    # plot loss
    pyplot.subplot(211)
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(history.history['loss'], color='blue', label='train')
    pyplot.plot(history.history['val_loss'], color='orange', label='test')
    # plot accuracy
    pyplot.subplot(212)
    pyplot.title('Classification Accuracy')
    pyplot.plot(history.history['accuracy'], color='blue', label='train')
    pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
    # save plot to file
    filename = sys.argv[0].split('/')[-1]
    pyplot.savefig(filename + '_plot.png')
    pyplot.show()
    pyplot.close()

# run the test harness for evaluating a model

# create data generator
datagen = ImageDataGenerator(rescale=1.0/255.0)
# prepare iterators
train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
class_mode='binary', batch_size=64, target_size=(224, 224))
test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
class_mode='binary', batch_size=64, target_size=(224, 224))
# fit model
history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
validation_data=test_it, validation_steps=len(test_it), epochs=30, verbose=1)
# evaluate model
_, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=1)
print('> %.3f' % (acc * 100.0))

model.save('self_made_model.h5')
```

```
Found 18697 images belonging to 2 classes.
Found 6303 images belonging to 2 classes.
Epoch 1/30
293/293 [==============================] - 287s 981ms/step - loss: 0.7345 -
```

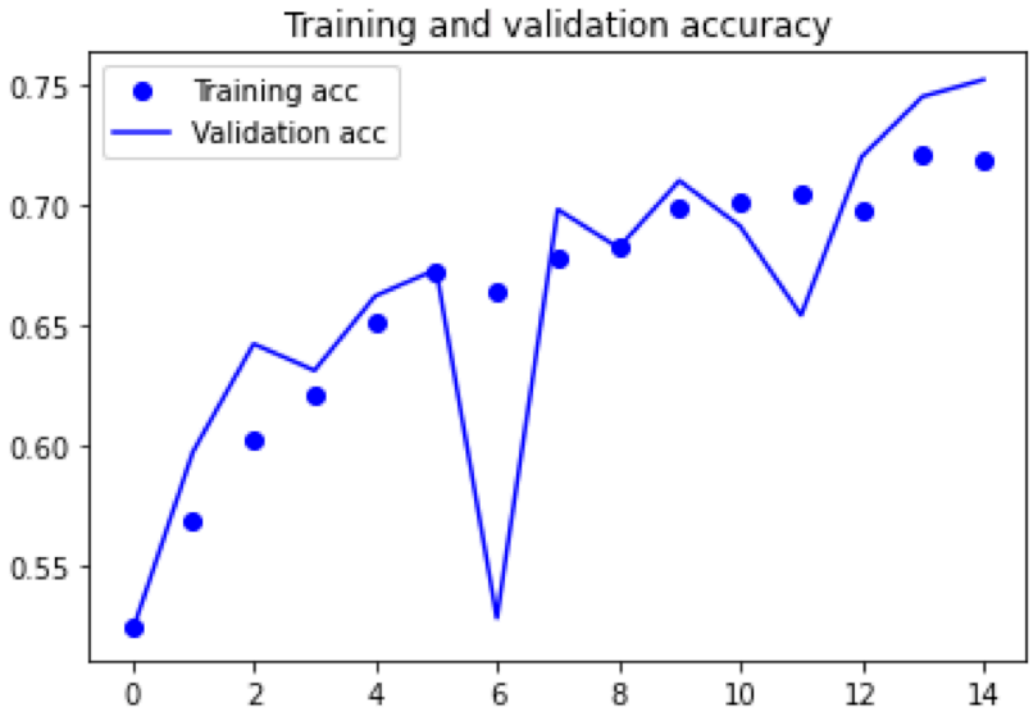
```
accuracy: 0.5450 - val_loss: 0.6505 - val_accuracy: 0.5420
Epoch 2/30
293/293 [==============================] - 643s 2s/step - loss: 0.6738 -
accuracy: 0.5722 - val_loss: 0.7355 - val_accuracy: 0.5680
Epoch 3/30
293/293 [==============================] - 284s 969ms/step - loss: 0.6756 -
accuracy: 0.5802 - val_loss: 0.7070 - val_accuracy: 0.5769
Epoch 4/30
293/293 [==============================] - 289s 985ms/step - loss: 0.6743 -
accuracy: 0.5781 - val_loss: 0.6954 - val_accuracy: 0.5167
Epoch 5/30
293/293 [==============================] - 287s 978ms/step - loss: 0.6696 -
accuracy: 0.5880 - val_loss: 0.7421 - val_accuracy: 0.5864
Epoch 6/30
293/293 [==============================] - 284s 970ms/step - loss: 0.6649 -
accuracy: 0.5972 - val_loss: 0.6228 - val_accuracy: 0.5988
Epoch 7/30
293/293 [==============================] - 291s 994ms/step - loss: 0.6754 -
accuracy: 0.5766 - val_loss: 0.6467 - val_accuracy: 0.6002
Epoch 8/30
293/293 [==============================] - 287s 981ms/step - loss: 0.6610 -
accuracy: 0.6073 - val_loss: 0.6745 - val_accuracy: 0.5610
Epoch 9/30
293/293 [==============================] - 291s 993ms/step - loss: 0.6619 -
accuracy: 0.6033 - val_loss: 0.6416 - val_accuracy: 0.6129
Epoch 10/30
293/293 [==============================] - 289s 986ms/step - loss: 0.6646 -
accuracy: 0.5978 - val_loss: 0.6732 - val_accuracy: 0.5994
Epoch 11/30
293/293 [==============================] - 291s 995ms/step - loss: 0.6552 -
accuracy: 0.6185 - val_loss: 0.6610 - val_accuracy: 0.6183
Epoch 12/30
293/293 [==============================] - 291s 992ms/step - loss: 0.6494 -
accuracy: 0.6258 - val_loss: 0.6502 - val_accuracy: 0.6146
Epoch 13/30
293/293 [==============================] - 290s 990ms/step - loss: 0.6505 -
accuracy: 0.6215 - val_loss: 0.6551 - val_accuracy: 0.6275
Epoch 14/30
293/293 [==============================] - 285s 971ms/step - loss: 0.6434 -
accuracy: 0.6360 - val_loss: 0.7527 - val_accuracy: 0.6243
Epoch 15/30
293/293 [==============================] - 285s 973ms/step - loss: 0.6387 -
accuracy: 0.6400 - val_loss: 0.7623 - val_accuracy: 0.6303
Epoch 16/30
293/293 [==============================] - 286s 977ms/step - loss: 0.6359 -
accuracy: 0.6424 - val_loss: 0.6451 - val_accuracy: 0.6359
Epoch 17/30
293/293 [==============================] - 290s 990ms/step - loss: 0.6307 -
```

```
accuracy: 0.6514 - val_loss: 0.6141 - val_accuracy: 0.6470
Epoch 18/30
293/293 [==============================] - 288s 984ms/step - loss: 0.6216 -
accuracy: 0.6601 - val_loss: 0.6358 - val_accuracy: 0.6338
Epoch 19/30
293/293 [==============================] - 291s 994ms/step - loss: 0.6109 -
accuracy: 0.6771 - val_loss: 0.6554 - val_accuracy: 0.6687
Epoch 20/30
293/293 [==============================] - 290s 990ms/step - loss: 0.6065 -
accuracy: 0.6763 - val_loss: 0.6444 - val_accuracy: 0.5777
Epoch 21/30
293/293 [==============================] - 296s 1s/step - loss: 0.6006 -
accuracy: 0.6844 - val_loss: 0.6881 - val_accuracy: 0.6722
Epoch 22/30
293/293 [==============================] - 291s 993ms/step - loss: 0.5893 -
accuracy: 0.6927 - val_loss: 0.6378 - val_accuracy: 0.6735
Epoch 23/30
293/293 [==============================] - 288s 984ms/step - loss: 0.5802 -
accuracy: 0.7018 - val_loss: 0.5984 - val_accuracy: 0.6695
Epoch 24/30
293/293 [==============================] - 287s 978ms/step - loss: 0.5673 -
accuracy: 0.7140 - val_loss: 0.5397 - val_accuracy: 0.6873
Epoch 25/30
293/293 [==============================] - 288s 983ms/step - loss: 0.5531 -
accuracy: 0.7250 - val_loss: 0.6008 - val_accuracy: 0.6911
Epoch 26/30
293/293 [==============================] - 288s 984ms/step - loss: 0.5384 -
accuracy: 0.7388 - val_loss: 0.5418 - val_accuracy: 0.6940
Epoch 27/30
293/293 [==============================] - 296s 1s/step - loss: 0.5222 -
accuracy: 0.7480 - val_loss: 0.5140 - val_accuracy: 0.7044
Epoch 28/30
293/293 [==============================] - 300s 1s/step - loss: 0.4984 -
accuracy: 0.7679 - val_loss: 0.6801 - val_accuracy: 0.7174
Epoch 29/30
293/293 [==============================] - 289s 988ms/step - loss: 0.4744 -
accuracy: 0.7759 - val_loss: 0.6785 - val_accuracy: 0.7181
Epoch 30/30
293/293 [==============================] - 288s 983ms/step - loss: 0.4396 -
accuracy: 0.8002 - val_loss: 0.6203 - val_accuracy: 0.7252
99/99 [==============================] - 25s 248ms/step
> 72.521
```

[10]:
```python
# plot diagnostic learning curves
def summarize_diagnostics(history):
    # plot loss
    pyplot.subplot(211)
```

```
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(history.history['loss'], color='blue', label='train')
    pyplot.plot(history.history['val_loss'], color='orange', label='test')
    # plot accuracy
    pyplot.subplot(212)
    pyplot.title('Classification Accuracy')
    pyplot.plot(history.history['accuracy'], color='blue', label='train')
    pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
    # save plot to file
    filename = sys.argv[0].split('/')[-1]
    pyplot.savefig(filename + '_plot.png')
    pyplot.show()
    pyplot.close()
```

[11]:
```
summarize_diagnostics(history)
```



[44]:
```
#model = VGG16(weights='imagenet', include_top=True)
X,y = shap.datasets.imagenet50()
model = load_model('self_made_model.h5')
```

[45]:
```
# explain how the input to the 7th layer of the model explains the top two␣
 ↪classes
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
```

```
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[2].input, model.layers[-1].output),
    map2layer(X, 2),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(to_explain, 2), ranked_outputs=2)
```

[46]:
```
# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, to_explain, index_names)
```

3 Use a pretrained model

[48]:
```
# vgg16 model used for transfer learning on the dogs and cats dataset

# load model
model = VGG16(include_top=False, input_shape=(224, 224, 3))
# mark loaded layers as not trainable
for layer in model.layers:
    layer.trainable = False
# add new classifier layers
flat1 = Flatten()(model.layers[-1].output)
class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
output = Dense(1, activation='sigmoid')(class1)
# define new model
model = Model(inputs=model.inputs, outputs=output)
# compile model
opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

# plot diagnostic learning curves
def summarize_diagnostics(history):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='orange', label='test')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Classification Accuracy')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='orange',␣
 ↪label='test')
        # save plot to file
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_plot.png')
        pyplot.show()

# run the test harness for evaluating a model

# define model
```

```
# create data generator
datagen = ImageDataGenerator(featurewise_center=True)
# specify imagenet mean values for centering
datagen.mean = [123.68, 116.779, 103.939]
# prepare iterator
train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
    class_mode='binary', batch_size=64, target_size=(224, 224))
test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
    class_mode='binary', batch_size=64, target_size=(224, 224))
# fit model
history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
    validation_data=test_it, validation_steps=len(test_it), epochs=1, verbose=1)
# evaluate model
_, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
print('> %.3f' % (acc * 100.0))

model.save('final_model.h5')

# learning curves

summarize_diagnostics(history)
model.save('final_model.h5')
```

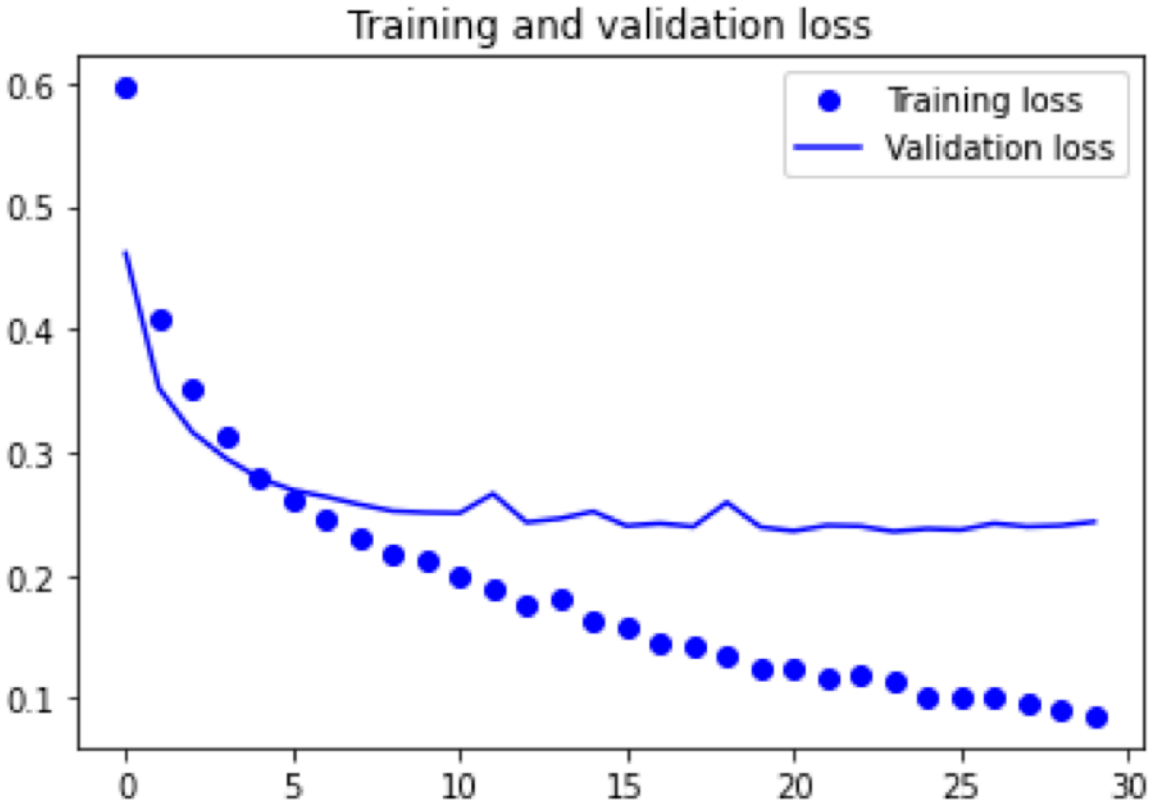
```
Found 18697 images belonging to 2 classes.
Found 6303 images belonging to 2 classes.
Epoch 1/1
293/293 [==============================] - 2980s 10s/step - loss: 0.5588 -
accuracy: 0.9592 - val_loss: 0.9988 - val_accuracy: 0.9703
> 97.033
```

[49]:
```
# explain how the input to the 7th layer of the model explains the top two␣
 ↪classes
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[7].input, model.layers[-1].output),
    map2layer(X, 7),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(to_explain, 7), ranked_outputs=2)


# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
```

```
shap.image_plot(shap_values, to_explain, index_names)
```

[53]:
```
# plot loss
pyplot.subplot(211)
pyplot.title('Cross Entropy Loss')
pyplot.plot(history.history['loss'], color='blue', label='train')
pyplot.plot(history.history['val_loss'], color='orange', label='test')
```

```
# plot accuracy
pyplot.subplot(212)
pyplot.title('Classification Accuracy')
pyplot.plot(history.history['accuracy'], color='blue', label='train')
pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
# save plot to file
filename = sys.argv[0].split('/')[-1]
pyplot.savefig(filename + '_plot.png')
pyplot.show()
```



List of Figures

List of Listings

References

[Aas et al., 2019] Aas, K., Jullum, M., and Løland, A. (2019). Explaining individual predictions when features are dependent: More accurate approximations to shapley values. cite arxiv:1903.10464.

[Bohannon, 2016] Bohannon, J. (2016). Who's the Michael Jordan of computer science? New tool ranks researchers' influence. *Science*.

[Britz, 2015] Britz, D. (2015). Implementing a Neural Network from Scratch in Python – An Introduction – WildML.

[Brownlee, 2019] Brownlee, J. (2019). *Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python*. Machine Learning Mastery.

[Chollet, 2016] Chollet, F. (2016). Building powerful image classification models using very little data. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html. (Accessed on 05/01/2020).

[COPELAND, 2017] COPELAND, M. V. (2017). DEEP LEARNING EXPLAINED WHAT IT IS, AND HOW IT CAN DELIVER BUSINESS VALUE TO YOUR ORGANIZATION CHAPTER 1 |. Available at https://www.nvidia.com/content/dam/en-zz/Solutions/deep-learning/home/DeepLearning_eBook_FINAL.pdf.

[Europarl, 2017] Europarl, M. (2017). Copyright directive: statement by Axel VOSS (EPP,DE), rapporteur - Multimedia Centre. Available at https://multimedia.europarl.europa.eu/en/copyright-directive-statement-by-axel-voss-eppde-rapporteur-{_}I158298-V{_}v.

[Gallwitz, 2019] Gallwitz, F. (2019). EU-Urheberrechtsreform: Experten zu Upload-Filtern | Science Media Center Germany. Available at https://www.sciencemediacenter.de/alle-angebote/rapid-reaction/details/news/eu-urheberrechtsreform-experten-zu-upload-filtern/?fbclid=IwAR1E0AqgLK8566U8VkGBo6e6YcV2QBZn5L4d3rb1dib{_}ik9T7iJk0augXzs.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Gross, 2017] Gross, D. (2017). 2. Was leisten die Domänen der Künstlichen Intelligenz? Available at https://www.sigs-datacom.de/ots/2017/ki/2-was-leisten-die-domaenen-der-kuenstlichen-intelligenz.html.

[Gunning, 2019] Gunning, D. (2019). Darpa's explainable artificial intelligence program. *AI Magazine*, 40(2):44–58.

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

[Knight, 2019] Knight, W. (2019). The Dark Secret at the Heart of AI - MIT Technology Review. Available at https://www.technologyreview.com/s/604087/the-dark-secret-at-the-heart-of-ai/.

[Kohlhase et al., 2017] Kohlhase, M., Koprucki, T., Müller, D., and Tabelow, K. (2017). Mathematical models as research data via flexiformal theory graphs. In Geuvers, H., England, M., Hasan, O., Rabe, F., and Teschke, O., editors, *CICM*, volume 10383 of *Lecture Notes in Computer Science*, pages 224–238. Springer.

[Kriesel, 2007] Kriesel, D. (2007). *A Brief Introduction to Neural Networks*. Available at http://www.dkriesel.co.

[Kuang, 2017] Kuang, C. (2017). Can A.I. Be Taught to Explain Itself? - The New York Times. Available at https://www.nytimes.com/2017/11/21/magazine/can-ai-be-taught-to-explain-itself.html.

[Lundberg and Lee, 2017] Lundberg, S. and Lee, S. (2017). A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874.

[Lundberg et al., 2018] Lundberg, S. M., Erion, G. G., and Lee, S. (2018). Consistent individualized feature attribution for tree ensembles. *CoRR*, abs/1802.03888.

[Michael Jordan, 2018] Michael Jordan (2018). Michael I. Jordan Interview: Clarity of Thought on AI. Available at https://medium.com/syncedreview/michael-i-jordan-interview-clarity-of-thought-on-ai-ed936d0dc421.

[Microsoft, 2013] Microsoft (2013). PhotoDNA | Microsoft. Available at https://www.microsoft.com/en-us/photodna.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

[Molnar, 2019] Molnar, C. (2019). *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/.

[Mudrakarta et al., 2018] Mudrakarta, P. K., Taly, A., Sundararajan, M., and Dhamdhere, K. (2018). Did the model understand the question? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1896–1906, Melbourne, Australia. Association for Computational Linguistics.

[Murphy, 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[Ng, 2008] Ng, A. (2008). Stanford cs229 - machine learning - andrew ng.

[Puget, 2016] Puget, J.-F. (2016). Legacy communities - ibm community. https://community.ibm.com/community/user/legacy?lang=en. (Accessed on 06/03/2020).

[Reda, 2019] Reda, J. (2019). Julia Reda – Upload filters. Available at https://juliareda.eu/eu-copyright-reform/censorship-machines/.

[Robinson, 2017] Robinson, D. (2017). You better explain yourself, mister: DARPA's mission to make an accountable AI • The Register. Available at https://www.theregister.co.uk/2017/09/28/inside{_}explainable{_}ai/.

[Rocke, 2019] Rocke, A. (2019). The true cost of alphago zero. https://keplerlounge.com/artificial/intelligence/2019/03/24/alpha-go-zero.html. (Accessed on 05/01/2020).

[Russell and Norvig, 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition.

[Samek et al., 2017] Samek, W., Wiegand, T., and Müller, K. (2017). Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *CoRR*, abs/1708.08296.

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229.

[Slundberg, 01] Slundberg (01). shaoshanglqy/shap-shapley. https://github.com/shaoshanglqy/shap-shapley. (Accessed on 05/01/2020).

[Spoerri, 2019] Spoerri, T. (2019). On Upload-Filters and other Competitive Advantages for Big Tech Companies under Article 17 of the Directive on Copyright in the Digital Single Market — jipitec. Available at https://www.jipitec.eu/issues/jipitec-10-2-2019/4914.

[Stewart, 2019] Stewart, M. (2019). Simple introduction to convolutional neural networks. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac. (Accessed on 05/01/2020).

[Sundararajan et al., 2017] Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365.

[Tagesspiegel, 2019] Tagesspiegel, D. (2019). Protest gegen upload-filter: Aufstand der generation youtube - medien - gesellschaft - tagesspiegel. https://www.tagesspiegel.de/gesellschaft/medien/protest-gegen-uploadfilter-aufstand-der-generation-youtube/24082412.html. (Accessed on 05/01/2020).

[Tjoa and Guan, 2019] Tjoa, E. and Guan, C. (2019). A survey on explainable artificial intelligence (XAI): towards medical XAI. *CoRR*, abs/1907.07374.

[Trask et al., 2015] Trask, A., Gilmore, D., and Russell, M. (2015). Modeling order in neural word embeddings at scale. *CoRR*, abs/1506.02338.

[Vasudev, 2019] Vasudev, R. (2019). Understanding and Calculating the number of Parameters in Convolution Neural Networks (CNNs).

[Versloot, 2019] Versloot, C. (2019). Visualizing keras cnn attention: Grad-cam class activation maps – machinecurve. https://www.machinecurve.com/index.php/2019/11/28/visualizing-keras-cnn-attention-grad-cam-class-activation-maps/#. (Accessed on 05/01/2020).

[Wagner, 1983] Wagner, N. R. (1983). Fingerprinting. In *1983 IEEE Symposium on Security and Privacy*, pages 18–18.

[Waltermann and Hess, 2019] Waltermann, H.-M. and Hess, T. (2019). Upload-filter für content. *MedienWirtschaft*, 16(2):16–21.

[Waters, 2017] Waters, R. (2017). Intelligent machines are asked to explain how their minds work | Financial Times. Available at https://www.ft.com/content/92e3f296-646c-11e7-8526-7b38dcaef614.

[Weitz, 2018] Weitz, K. (2018). Available at http://www.cogsys.wiai.uni-bamberg.de/theses/weitz/Masterarbeit_Weitz.pdf.

[Wikipedia, 2020] Wikipedia (2020). Gradient descent - wikipedia. https://en.wikipedia.org/wiki/Gradient_descent. (Accessed on 05/01/2020).

[Woollacott, 2019] Woollacott, E. (2019). EU Copyright Directive Passed - Upload Filters And All. Available at https://www.forbes.com/sites/emmawoollacott/2019/03/26/eu-copyright-directive-passed-upload-filters-and-all.

[Yao et al., 2017] Yao, Y., Xiao, Z., Wang, B., Viswanath, B., Zheng, H., and Zhao, B. (2017). Complexity vs. performance: empirical analysis of machine learning as a service. pages 384–397.

[YouTube, 2010] YouTube (2010). How Content ID works - YouTube Help. Available at https://support.google.com/youtube/answer/2797370?hl=en.

List of Definitions