Enriching LIME with Inductive Logic Programming: Explaining Deep Learning Classifiers with Logic Rules in a Companion System Framework

Masterarbeit

im Studiengang Angewandte Informatik
der Fakultät Wirtschaftsinformatik
und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Verfasser:
Johannes Rabold

Gutachter:
Prof. Ute Schmid

Bamberg, 08.10.2018

Bamberg, 2023

URN: urn:nbn:de:bvb:473-irb-465273
DOI: https://doi.org/10.20378/irb-46527

Abstract

With the rise of black-box classifiers like Deep Learning networks, the need for interpretable and complete explanations for them becomes apparent. Users need to have the possibility to ask why a classifier inferred a particular result. Logic clauses induced by Inductive Logic Programming systems are superior in expressibility over visual explanations alone. This thesis uses the ideas of LIME, a visual explanation framework, and enriches it with an ILP component to get comprehensible and powerful explanations for the inference results of Deep Learning Networks for images. The background knowledge for the predicates is obtained both automatically and by an annotation system that lets humans annotate labels and relations. The human labeling system and the explanation component form a Companion System where not only AI helps the user but also the other way round.

Keywords: Explainable AI, Deep Learning, Inductive Logic Programming, LIME, Companion System.

Contents

List of Figures

List of Tables

List of Abbreviations

ML Machine Learning

DNN Deep Neural Network

CNN Convolutional Neural Network

LIME Local Interpretable Model-Agnostic Explanations

ILP Inductive Logic Programming

FOL First Order Logic

XAI Explainable Artificial Intelligence

Aleph A Learning Engine for Proposing Hypotheses

1 Introduction

1.1 Motivation

Imagine the following: You go to a doctor because you suffered from repeating headaches and you want to get a quick check-up. The doctor examines you but he/she can not find any indicators for a disease. Because the headaches came very frequently and strong, a CT scan from your brain is run. But still no signs for something physically wrong. Then the doctor tells you there is a novel system which is still in the experimental phase. It can find slight irregularities in the brain structure and based on them can give a probability for having typical diseases that affect the brain. Long story short, you decide on trying it out and the system tells that you have a certain kind of brain cancer. In fact the system estimated a probability of 83 percent for cancer. The question is: Would you trust the system? Would you decide on a potentially life-threatening surgery to get rid of a tumor solely based on a percentage number? In other words: Which additional information is enough for you or the doctor to establish trust in a computer program?

The scenario above seems to draw a rather pessimistic future where artificial intelligence is on the rise of depriving us from every decision we have to make, without us questioning the output up front. But we are already living in a time where automatic decision making is performed in the everyday life in order to categorize people into classes of credit- or insurance-worthiness or deciding on a potentially dangerous situation in a crowded place [Fag18]. These systems need to obey our moral and ethical standards and not include some kind of bias (e.g. racially).

Modern systems that perform such profiling based on a mix of visual clues and a pool of heterogeneous background knowledge are often based on technologies from the Machine Learning field of Deep Learning. This relatively new approach can find regularities in input data and weigh them against each other to decide on a quantitative estimator for the classes that the system was trained for. The output can now either be the raw estimators along with the class names or - as in a concept learning task - the simple statement if the output belongs to the class or not.

Deep Learning models are trained like "traditional" Machine Learning models by presenting the model with pairs of input data and the corresponding true class label. However, the training data does not have to be preprocessed to contain human understandable features. As long as it is structured in a certain way that the model can read in, you can feed it as input. The model itself is able to extract the features that are relevant for deriving the class of the input. In order to do that, it takes extremely many correct input-output pairs. Getting that many training examples is tedious. Getting correctly labeled training examples even more. But this correctness is crucial since every bias that a human labeler might have can influence the learned model and the bias is transferred to the generalization task when classifying new examples.

The learned Deep Learning models typically consist of layers of matrices which are not inherently understandable by humans. A practitioner that uses such a model in order to get a "second opinion" before he/she decides on an action might simply get the raw class output without a proper explanation on how the result might have come about. This is not enough to build trust in a system. So one solution would be to use another Machine Learning approach whose learned hypotheses are inherently comprehensible. This would however mean giving up on the advantage of Deep Learning to be able to extract its own features from heterogeneous data.

1.2 Objectives

The second solution would be to keep the Deep Learning approach with all its advantages and to extend it with an explanation interface that can explain the complex, incomprehensible model by a more human understandable one. This is the way I want to go in this master's thesis. Such an explanation has to be both interpretable and accurate to the underlying classifier.

The Machine Learning field of Inductive Logic Programming (ILP) offers an inherent interpretability since the hypotheses that can be induced with it consist of logic rules. It also provides the complete expressibility of First Order Logic (FOL) which makes it powerful. I implemented a system that makes it possible to explain the output of a - possibly complex - classifier in terms of logic rules.

For the logic rules to be understandable I have to make use of human predicates. Therefore the input, which will be images, has to be annotated up front by a human labeler. I have implemented an annotation system that supports the user in inputting labels as well as relations for image input. Additionally, automatically inferred labels and relations are generated that might be relevant for the system.

With the human annotation, it will be possible to infer rules with the ILP system Aleph. Aleph needs a pool of positive and negative training examples as well as background knowledge. This background knowledge will be taken from the human annotation. The example pool is generated with an approach similar to the explanation framework LIME [RSG16]. LIME uses perturbed versions of the input along with the classifier output for these versions to fit a linear model to the non-linear decision boundary of a black-box classifier and outputs a visualization of the parts in the image that are relevant for the inference result.

In this master's thesis I want to answer the following research questions:

- How can inference results of black-box classifiers be explained by logic rules?
- How do logic rules add expressiveness to an explanation compared to a visualization approach?

1.3 Structure of this work

The thesis is structured as follows:

- **Background and Related Work**

  I will present the related work that is important for this thesis. First there will be a chapter on black-box learning, especially Deep Neural Networks. I will elaborate the up- and downsides and will present one of the most important model types, Convolutional Neural Networks. A chapter about Explainable AI will give some insights in the field as well as approaches for interpretable and complete explanations for black-box classifiers. One important explanation framework, LIME, will be introduced in the next chapter. We will see the shortcomings of the framework and see that the field of Inductive Logic Programming can help to overcome these by exploiting the power of First Order Logic.

- **Methods**

  This chapter will show my workings on a system that is concerned with providing the user of a black-box model three levels of explanations with increasing expressiveness. An annotator is presented that gives the ability for a human labeler to provide background knowledge for a logic rule induction system.

- **Experiments and Results**

  This chapter contains the experiments I conducted in order to prove the expressiveness of my explanation framework. A baseline experiment shows that even with automatically generated information, an explanation based on First Order Logic rules provides a more powerful insight in the inner workings of the black-box than a visualization approach alone. When human labels and relations are added, this rules become more comprehensible.

- **Conclusion and Future Work**

  This chapter concludes the thesis and summarizes the topics that are subject to future work.

2 Background and Related Work

This section sheds light on important related work that this master's thesis is built upon. While the focus is on the different methods for presenting explanations, I will first take a look in the details of black-box classifiers, especially Deep Learning models. I will explain where their advantages and disadvantages lie and explain why the learned hypothesis is not inherently comprehensible for humans. I will then continue by giving an overview of the different methods on how to establish trust in black-box classifiers by using methods that emerged with Explainable AI. The next section will consist of a presentation of LIME; a method for generating visual explanations for black-box classifier outputs. While this framework will already give many more information on the internals of the black-box, it is still limited in its expressibility. This is where the next section about Inductive Logic Programming (ILP) comes into play. It adds a level of expressibility to the explanation by exploiting the complete power of First Order Logic.

2.1 Black-box Learning

The term black-box learning is a general term that refers to a certain group of machine learning algorithms whose internals are to some extent too complex for humans to comprehend. They work like most machine learning algorithms by processing training examples and inducing a hypothesis that generalizes over the input. However the induced hypothesis consists of a model that makes sense for the algorithm but is normally not interpretable by humans.

Due to recent successes in a variety of fields (like image-, text-, and sound-recognition as well as machine translation, etc.), the most popular type of black-box models are Neural Networks and especially Deep Neural Networks (DNN). The following sections will consist of an introduction to DNNs.

Deep Learning - it has become a buzzword in the media not only since its successful implementation in tasks like winning games (Chess, Go, ...) [Lai15] [SHM$^+$16]. Early artificial intelligence systems tackled problems difficult to humans but easily solvable by computers. Such problems could be easily expressed formally and thus were understandable by computers. Deep Learning is often being attributed as being a huge step towards "true" artificial intelligence. The tasks that they are able to solve (for example face recognition) are problems that can be easily done by a human in a matter of seconds but were thought of being too complicated for machines [GBC16].

The secret to the power of Deep Learning can be attributed to the way they are trained from example data. They are able to find patterns in the data that are special for the particular concept or classes they are being trained for. The data does not have to be preprocessed to contain handcrafted features. Deep Learning models are able to select these features by itself. Layers of *Feature*

Extractors can be trained to recognize hierarchies of features that become more and more abstract in the input data and output an estimator for the probability that the input belongs to a certain concept/class.

However with more powerful models, the networks can become quite large and complex. Deep Learning Networks emerged from Neural Networks and share with them the hypothesis language of layers of floating point matrices. These networks are not inherently understandable for humans. It is not always possible how such a system inferred a decision and which internal components lead to it.

Convolutional Neural Networks (or CNNs for short) are probably the most successful and best studied models in the field of Deep Learning. That is due to their wide applicability for a number of typical Machine Learning tasks that include classification and segmentation of a variety of input formats. The first research and application dates back to a paper from Yann LeCun et al. in 1989 [LBD+89] where they introduced a new kind of neural network architecture used for recognizing ZIP codes. This new architecture did not work with vectors of input but with two dimensional data. This provided the possibility to efficiently process input that is organized in a grid-like structure like images or sound. While the pixels in an image are arranged in a 2D grid, sound can also be seen as multi-dimensional, since it contains a time axis as well as an intensity- or frequency axis. They also share the property that their components are dependent on their spatial or temporal vicinity. Pixels that are closer together usually share similar properties. This spatial dependency in images was not taken care of in traditional neural networks. In the following I show the different parts of a typical CNN and their function. The knowledge was taken mainly from Goodfellow et al [GBC16].

Convolutional Neural Networks for classification of images usually consist of two main parts: A part that extracts features and a traditional Neural Network as classificator (see Figure 1).

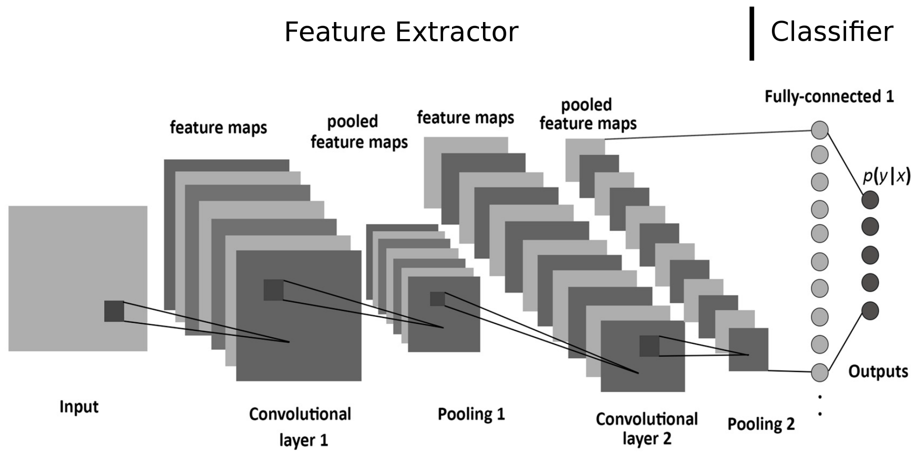


Figure 1: An abstract overview of the components of a Convolutional Neural Network (adapted from [AM17]).

The feature extraction part is capable of finding the important features of the image. It consists of a cascade of layers; each layer with a distinct task. As typical for Deep Learning, the deeper we dive into the layer structure, the more abstract the features become that each layer can learn. On the very first layer, the features are normally just edges, corners or color blobs. But advancing further, the features are geometrical forms, parts of objects or even complete objects like eyes, ears, noses, etc. On the last layer of the Feature Extractor, the CNN can already tell which objects appear in the image and where it is located.

The Classificator is a traditional Multi-Layer Neural Network. It takes the information from the Feature Extractor as input and outputs an array of estimators for the classes of the classification problem.

The Feature Extractor contains a variety of layers, the most important of which are the Convolution Layer (hence the name of the network) and the Pooling Layer. The most used architecture consists of a stack of multiple Convolution Layers followed by one Pooling Layer. This pattern continues, forming the structure of the Feature Extractor. The Convolution Layers take the input of the last layer (at the beginning this is the 2D grid of the image) and produce so called Feature Maps. These contain the location and the intensity of a feature in the input. The Feature Maps are calculated by a mathematical operation called convolution. You take an input image and let an array of weights shift over the complete image. For every pixel and its vicinity you calculate the linear combination. This is the output for one pixel in the Feature Map. Given a suitable array of weights, arbitrary features can be detected in the image and highlighted. The array of weights is exactly what the network has to learn during its training phase. Every Convolution Layer contains multiple weight-arrays for detecting different features. When a Convolution Layer receives a Feature Map as input, it is simply handled like an image.

After every few Convolution Layers a Pooling Layer is used. These layers aggregate the feature information of the Feature Maps and downsample it to half its size in every dimension. The most used method is the Max-Pooling. The input gets segmented into blocks of four values and just the highest value is kept.

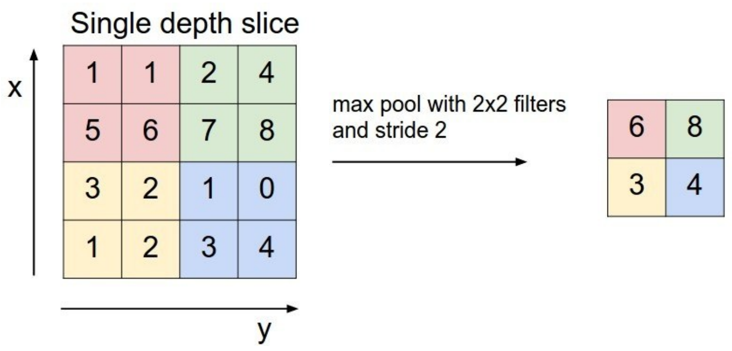


Figure 2: The procedure of maxpooling (from [Karen]).

With this method, the important information in an image is aggregated until almost any information on the location of the objects is gone. What stays is the information about which objects the image contained. The last layer of the Feature Extractor is followed by a Flatten Layer. The two dimensional Feature Map is transformed to a one dimensional vector. This vector is the input for the Classifier. This is just a simple traditional neural network with multiple fully connected layers. The value of the features from the Feature Extractor are fed forward in the network until the last layer. This output layer contains as many output neurons as there are classes in your classification task. The result of a CNN can be defined as the class that is associated with the neuron whose output value is the highest.

Besides from CNNs for classifying two dimensional input, there also exist Deep Learning Networks like LSTMs (Long Short-Term Memory) [HS97] suitable for processing time series and to predict the next value in the series. But explaining these would go beyond the scope of this thesis since we want to focus on explaining the classification result for images.

2.2 Explainable AI: Establishing Trust

Explainable AI (XAI) is a movement in the field of Artificial Intelligence that is concerned with designing AI systems that, apart from their inherent task, also can be audited by users in order to make clear why they behave in a certain way. Ras et al. [RHvG18] describe four components that XAI should address:

- **Users**

  AI should be made to help humans. We as a society have certain desires and ethical standards. In order to be able to trust an AI system it should not violate these. Ras et al. distinguish between expert users and lay users of AI [RHvG18]. An expert could be concerned with how the different hyperparameters of the model affect the behavior of it. A lay user may want to know what the system is able to or wants to have a justification for the decision result.

- **Laws and Regulation**

  With data protection laws there comes the important right to ask an authority which information they have of oneself. An in-transparent data processor like a Neural Network makes it difficult to access stored knowledge.

- **Explanation**

  With this demand, there comes the need for Explainable AI to provide transparent models or at least give a method to extract comprehensible knowledge from a black-box. A user should know about what is going on inside a black-box. Note however that transparency also can come with the risk of misuse and manipulation (for example to infiltrate a system with adversarial training data that alters the intended outcome).

- **Algorithms**

  The last component is concerned with the inherent risks of different algorithms used in AI and especially in Machine Learning. The training phase can be infiltrated to cause unwanted side effect. Or the data set could simply be of bad annotation quality.

The main component of XAI is the explanation itself. What makes an explanation a *good* explanation? Gilpin et al. [GBY$^+$18] answer this question by giving an explanation two dimensions that have to be maximized for it to be *good*: The first one is *interpretability*. An interpretable explanation describes the internal operation of a classifier such that it is comprehensible to humans. To decide if this goal is fulfilled depends on the user, because he has to be able to understand the explanation vocabulary. The second dimension is *completeness*. An explanation is said to be complete if it perfectly resembles the decision process of a system with all its possibly complex sub-decisions. In general, completeness is about accurately anticipating the behavior of a system.

Without doubt these goals are in competition with each other. A perfectly complete explanation of a Deep Neural Network can only be achieved by stating all the layers with the corresponding matrices. Such an "explanation" would perfectly "explain" the behavior of the network. However it would also have the highest level of uninterpretability. Also, a very simple human understandable explanation would most likely not resemble the complicated decision processes a Deep Neural Network would attempt. As Herman et al. [Her17] state, it can even become a problem if we base the fulfillment of the interpretability goal solely on the human understanding of interpretability as humans tend towards the simplest explanation. We have to be careful not to sacrifice transparency of the internal operations in favor of interpretability as the simplification of models can easily empower a bias towards *persuasive* explanations rather than *descriptive* ones.

The establishment of trust in a result outputted by a Machine Learning algorithm becomes ultimately important as soon as it is used to aid decisions with a potentially high impact. Papernot et al. [PMJ$^+$16] point out how even simple perturbations in the input can be amplified during the feed-forward phase of the network and produce a very different output. Such attacks can be executed without retraining the network. So an adversary just has to know how to forge the input such that the network completely misclassifies it. With an explanation on which parts of the input lead to the classification, such attacks can be made more difficult.

Two main ways on how to approach explainability in Machine Learning can be identified:

- Use inherently interpretable Machine Learning approaches like ILP (discussed in chapter 2.4) or Decision Trees
- Extend black-box classifiers by an explanation component

The first approach, which has been successfully applied to a variety of problems, is often outperformed by Deep Learning classifiers in terms of accuracy. The second method gives the possibility of generating expressive explanations by keeping the power of Deep Learning models. I will give a brief overview on different approaches to that:

Ras et al. [RHvG18] propose a taxonomy for explanation methods. It consists of rule-extraction methods, attribution methods and intrinsic methods. Rule-extraction methods, as the name implies, try to extract human-readable rules of how the system behaves, e.g. by creating Decision Trees [ZMJ16]. These rules should approximate the internal decision process relatively accurate by staying comprehensible. An important sub-field are pedagogical approaches. These work in a model-agnostic way and try to fit rules by viewing the rule-extraction itself as learning task where the ground truth is the output of the system to be explained [CS94] [RSG16].

Attribution methods work with perturbing the input in a certain way and measuring how much the results of the systems change [ACOG18]. Also the components of the system can be altered and the change measured. Intrinsic methods already start during training of the systems. Especially designed adversarial training data can give insights on what potential error sources might be in the model [DSZB17].

2.3 LIME

A promising pedagogical approach is LIME. The LIME framework (short for Local Interpretable Model Model-Agnostic Explanations) was introduced by Ribeiro et al. [RSG16]. The framework can be used to obtain explanations for inference results of arbitrary classifiers, thus making it model-agnostic. The generated explanations depend on the type of input - LIME is currently capable of dealing with tabular data, text and images.

The way LIME works in its original implementation is that it fits a linear model to approximate the complex, non-linear decision boundary of the classifier in the locality of the instance to be explained. The weights of this simplified model can then be interpreted as indicators of how strong certain features of the input instance influence the overall classification result. It therefore generates multiple perturbed versions of the input instance that, along with their classifier output, act as training examples for the linear model. That way it does not need to get insight in the (potentially incomprehensible) hypothesis of the classifier but learns a linear mapping function solely from input/output pairs.

In the following, the underlying algorithm of LIME will be explained in more detail. We will focus on the use case of explaining the classification result of images as this is also the focus of this thesis.

Algorithm 1 Linear Model Generation with LIME (adapted from [RSG16])

```
1: **Require:** Classifier $f$, Number of samples $N$
2: **Require:** Instance $x$ and its interpretable version $x'$
3: **Require:** Similarity kernel $\pi_x$, Length of explanation $K$
4:    $\mathcal{Z} \leftarrow \{\}$
5:    **for** $i \in \{1, 2, 3, \ldots, N\}$ **do**
6:       $z'_i \leftarrow$ sample_around($x'$)
7:       $\mathcal{Z} \leftarrow \mathcal{Z} \cup (z'_i, f(z_i), \pi_x(z_i))$
8:    **end for**
9: $w \leftarrow$ K-Lasso($\mathcal{Z}, \mathcal{K}$)$\triangleright$ with $z'_i$ as features, $f(z)$ as target
10: **return** $w$
```

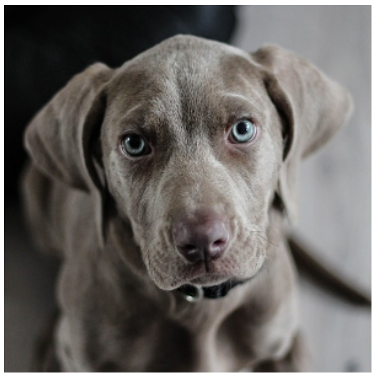
Suppose you have an image x that you want to explain. LIME will find you the parts of the image that influence the classification result in an either positive or negative way. The domain of the explanation will be a weight vector $\vec{e_x}$ with an adaptable length K that holds the calculated positive and negative weights for the parts of image x. The length K of the explanation can be set up front from the user and is the number of parts the image should be divided into. A higher K thus means a more fine-grained explanation, but also a higher computational overhead. The image x has to be divided into the K parts up front. This is done via superpixel segmentation algorithms which I will talk about in the next chapter.

We call the partitioned representation of x x'. Once we obtained the parts of the image, we now need a pool of training examples for the linear model that is capable of describing a locally faithful approximation for the classifier decision boundary. LIME therefore generates N perturbed versions z' of x'. To obtain one z', the algorithm uniformly chooses a certain amount of image parts and "turns them off" by turning all the pixels in these parts to the mean color of the part. Nothing else changes; the images do not get altered in another way.

For every z', a tuple $\langle z', f(z), \pi_x(z)\rangle$ is calculated. Apart from the interpretable representation z' it contains the output of the classifier for z and a distance measure π_x. This measure calculates the distance of z from the image x in the Euclidean space. It simply calculates a pixel-wise Euclidean distance for the two images. This measure is an indicator for the model of how un-similar a version is to the original image. For the linear model it is important that it is fitted to be locally accurate to the original image. Therefore the image versions that have a low distance are preferred for the approximation.

Apart from local fidelity to a complex classifier function, LIME has a mechanism that prevents the linear model from incorporating too many features. A number K bounds the features that will influence the linear function. Now, to obtain a linear model that fits the classifier function locally and remains interpretable, LIME selects K features with a statistical technique called LASSO [Tib96]. This is a technique for fitting an interpretable regression model by selecting only a few variables that are most important for the final model. After the selection process, the weights for the linear model can be learned via least squares. In the paper, this procedure is called K-LASSO.

The obtained weights ultimately can be interpreted as indicator of how much the corresponding part of the image is important towards the class of the image. The LIME implementation comes with a tool to visualize these weights directly on the image to be explained. In Figure 3, parts that correspond positively to the correct class are highlighted.


(a) Original image of a weimaraner dog (from: `https://pxhere.com/en/photo/626898`).

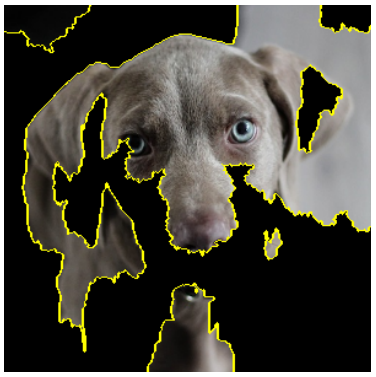

(b) The output of LIME.

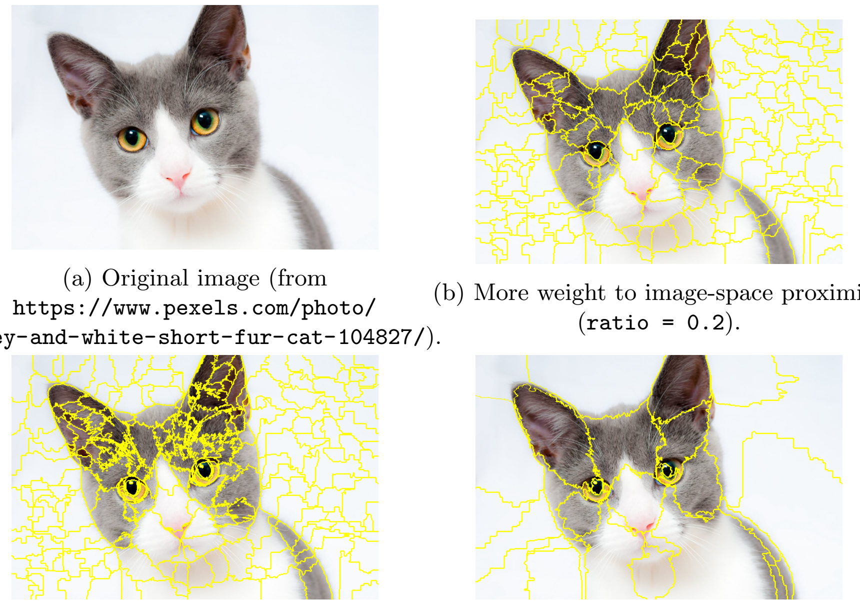
Figure 3: Visualization of the LIME result. The highlighted image parts represent positive correlation towards the true class.

2.3.1 Intermezzo: Superpixel Segmentation

LIME uses various algorithms that divide the input image into processable units. These units are called superpixels. The term "superpixel" can first be found in a 2003 paper by Xiaofeng Ren and Jitendra Malik [RM03]. As a pre-processing step for a grouping task, they divided an image into many small coherent pixel patches. But methods for deriving a segmentation of an image can also be found earlier, e.g. in Felzenszwalb et al. [FH04]. The criteria for which pixels form a unit are largely derived from Gestalt psychology. The guiding laws are proximity, similarity and good continuation [RM03].

Advantages of superpixels can be seen especially in connection with tasks from Computer Vision: Superpixels are a much more natural subdivision of an image for humans. The division into pixels is merely a product of digital scanning during the generation of the image. A further advantage is the reduction of the units to be managed for further tasks. This leads to a general reduction of the computational effort [RM03].

In the following, the standard method in LIME - quickshift [VS08] - and the result for the segmentation will be presented. I will not go into the details of the algorithm used but will rather show how the segmented image looks like, given particular input parameters.



(a) Original image (from https://www.pexels.com/photo/grey-and-white-short-fur-cat-104827/).

(b) More weight to image-space proximity (`ratio = 0.2`).

(c) More weight to color-space proximity (`ratio = 1.0`).

(d) Higher Gaussian kernel size (`kernel_size = 20`).

Figure 4: Result of different settings for quickshift.

As you can see in Figure 4, you can drastically alter the number and granularity of the superpixels detected. 4a shows the original image. There is a parameter called `ratio` that handles how the underlying clustering algorithm derives proximity [sidten]. Values closer to 0 give more weight to image-space proximity; values closer to 1 give more weight to color-space proximity. The effects on the image for different ratio settings can be seen in 4b and 4c.

Another setting is the `kernel_size` [sidten]. It regulates how big the Gaussian kernel for smoothing will be. When you set it to higher values that means fewer superpixels generated. The effects of a high setting for this parameter can be seen in Figure 4d.

LIME uses the standard setting of `ratio = 0.2` and `kernel_size = 4` which comes close to the image in Figure 4b. That way the superpixel are small enough to grasp small objects but big enough to not get lost in irrelevant details and to not generate too many superpixels.

2.3.2 LIME downsides

Although being a very powerful toolset for explaining all sorts of classifiers and a variety of input in a comprehensible way, LIME has certain shortcomings that shall be described in the following.

The first rather obvious downside is the restriction to fitting only linear models for explanations. The paper [RSG16] already mentions this and indicates that problems in the local fidelity can arise when the user has to deal with classifiers that have a highly non-linear decision boundary. However, the faithfulness of the explanation on the chosen sample can be measured and at least given to the user so that he can react to it.

The second downside refers to a lack of expressiveness that is closely linked to the chosen method of gaining the interpretable representation x' of x. The paper describes a use case where a sepia-toned image was labeled as *retro* by a classifier. Since LIME gains x' by segmenting the image into superpixels and assigning a weight to every superpixel, the visualization of the explanation could become rather trivial: Every superpixel is important to the classification, so label everyone as positive and color it green. Obviously this gives close to no beneficial insight into the internals of the classifier. One could rather guess that a property that is present in every superpixel is important for the decision. But this can easily lead to a search for a "hidden object" in every superpixel.

It becomes clear that a simple visualization of parts of the image is sometimes simply not expressive enough to explain a classifier. When we are dealing with attributes, that describe an image in its entirety, we would be better off being handed a simple predicate that describes in an almost natural language what the image is about. Another "attribute" that is not easily expressed with a visualization is any kind of relation between different parts in the image. Think of a classifier for bicycles. It is important for the classifier that the image contains two tires. Otherwise it would be a unicycle. A LIME visualization would merely highlight the two tires as being important. An auditor of the classifier can not see if the classifier would also classify it as a positive example if only one tire would be visible. The relation between them (for example `left_of(tire1, tire2)`) or the statement `contains(tire1) AND contains(tire2)` is important for the class. The expression of such logic terms and especially the learning of theories in logic from examples is the strength of Inductive Logic Programming. This field will be explained in the next chapter.

2.4 Inductive Logic Programming

Inductive Logic Programming (ILP) is one of the newer disciplines in the field of Machine Learning. Its origins date back to the beginning of the 90s. One of the forefathers of ILP, Stephen Muggleton, sums up the discipline as the intersection of ideas from Machine Learning and Logic Programming [Mug91]. The application of induction is the fundamental inference method of Machine Learning. The hypotheses induced during the learning process in ILP are represented as logic rules in a logic programming language like Prolog.

The driving force behind research in ILP can be seen by looking at the disadvantages of "classical" machine learning that ILP wants to overcome. When we look at a standard approach like the generation of a Decision Tree, there are limitations in expressiveness that can be overcome by ILP. Standard Decision Tree algorithms work with propositional attributes only. But this is not enough for many tasks that require the usage of First Order Logic (FOL) to express the problem domain and therefore the induced hypotheses (see [MDR94]). As an example, a relation or a recursion can be written explicitly in the form of first order predicates.

Inductive Logic Programming tackles this by exploiting the full expressiveness of FOL. This is accomplished by the usage of a logic programming language. Another problem of the traditional Machine Learning approaches is the lack of usage of domain knowledge. It is a core finding of Artificial Intelligence research that the usage of domain knowledge is crucial for the achievement of intelligent behavior [MDR94]. FOL formalism are predestined to express background knowledge in a suitable, human-understandable way.

ILP as a Machine Learning discipline needs the typical building blocks of an inductive learning system: A hypothesis language, a hypothesis space and positive and negative examples. ILP also adds the use of background knowledge to these components. The hypothesis language are first order clausal theories. When you have a logic clause of the form $A \vee \neg B \vee \neg C$ where A, B and C are first order predicates, the clause is also called a Horn clause. A Horn clause is a logic clause with at most one positive literal. It can be transformed to the material implication $B \wedge C \implies A$ or, in other words, a logic rule. In the framing of Machine Learning, A is the predicate that we want to learn a rule for - the target predicate - and B and C are the preconditions that have to hold in order to derive this target predicate [Mug91].

Being a Machine Learning approach, ILP inductively learns its hypotheses by training with positive and negative examples. Positive examples are examples for which the target predicate is true. Negative ones are the opposite. Background knowledge is given by logic facts and rules that further describe the examples and the overall problem domain. The task is to find a hypothesis H from positive (E^+) and negative (E^-) examples together with some background knowledge B. Further, the following conditions (among others) must hold [MDR94]:

- **Posterior Sufficiency** $B \wedge H \models E^+$
- **Posterior Satisfiability** $B \wedge H \wedge E^- \not\models \Box$

This means that background knowledge B together with a hypothesis H must model the positive examples E^+ by also being consistent with the negative examples E^-.

I will now give a typical example of how the different components might look like in a well studied task: Deriving a rule for the target predicate `grandfather(X, Y)` which expresses that `X` is the grandfather of `Y`. We want to find a general logic rule for this target predicate. We have the following positive and negative examples:

$E^+ = \{$ `grandfather(rob, emily)`, `grandfather(rob, bert)`$\}$

$E^- = \{$ `grandfather(rob, peter)`, `grandfather(susan, emily)`, `grandfather(bert, bert)`$\}$

Further, the following background knowledge B is given:

`parent(rob, peter)`, `parent(susan, peter)`, `parent(peter, emily)`,
`parent(peter, bert)`,
`male(rob)`, `male(peter)`, `male(bert)`

With this information, a hypothesis for the target predicate has to be found. For a human, it quickly becomes clear that the most plausible rule is

$$\texttt{grandfather(X, Y)} \leftarrow \texttt{parent(Z, Y)} \wedge \texttt{parent(X, Z)} \wedge \texttt{male(X)}.$$

An algorithm has to be designed very carefully in order to induce a rule that is both correct and sufficiently general to be able to also cover new data.

A lot of research has been done in designing algorithms tailored to induce general hypotheses from examples. These algorithms are in principle based on methods of inverting deductive inference. Besides the approach of inverting resolution [MB88] as well as implication [Mug92], there has been extensible effort in inverting deduction in general. It has been shown that approaching induction from a model theoretic viewpoint rather than a resolution proof-theory is much simpler and makes the problem domain more clearly defined [Mug95]:

Remember again the fundamental problem statement of ILP where a hypothesis H has to be found that, together with background knowledge B models the set of examples E:

$$B \wedge H \models E$$

This statement can be proved to be equivalent to

$$B \wedge \overline{E} \models \overline{H}.$$

Assuming that E consists of a single positive example, we can add the negation of it to the background knowledge and deductively derive a finite conjunction of ground facts $\overline{\bot(B, E)}$. From this *bottom clause* we can construct hypotheses H which subsume $\bot(B, E)$ [MB00]. A system that makes use of this idea of inverting entailment and finding a "best" hypothesis H for examples and background knowledge is Aleph.

Aleph [Sri07] is a very popular ILP framework. The name is an apronym for **A** **L**earning **E**ngine for **P**roposing **H**ypotheses. It originated under the name P-Progol in 1993 as "part of a fun project [...] at Oxford University" [Sri07]. The original authors were Ashwin Srinivasan and Rui Camacho but the official website for the manual of Aleph mentions multiple researchers whose findings actively influenced the implementation. Aleph is intended to be a framework for experimenting with various theoretical questions related to ILP. Its high adaptability makes it especially suitable for trying out new ideas and approaches and therefore will be the framework of choice in this thesis.

In the following section I will explain the basic underlying algorithm of Aleph and show how it induces hypotheses based on positive and negative examples as well as background knowledge.

The algorithm consists of four main steps which are:

1. Select example
2. Build most-specific-clause
3. Search
4. Remove redundant

These steps repeat until a stopping criterion is met.

1. First, the algorithm selects one example from the pool of positive and negative examples. This example will be generalized in the next steps. If no examples exist anymore, the algorithm terminates.
2. This step finds the most-specific clause that entails the selected example (and is in the language restrictions (see later)). This is sometimes called the *bottom clause* (see above) and typically contains many literals.

3. From the bottom clause, Aleph then tries to find a clause that is more general. It generates clauses with a subset of the current literals and calculates a score for each clause. The one with the "best" score will be selected. The nature of this score will be explained later.

4. Cover Removal: The clause that was found in the previous step is added to the pool of clauses (the theory). All examples that are now covered by this clause will be removed. The algorithm repeats from step 1.

Aleph is entirely written in Prolog. Everything that you give Aleph as input has to be some kind of Prolog code. Also the output of Aleph (the hypotheses forming the theory) is expressed as Prolog rules. The following section will give an overview of the typical Prolog input for Aleph.

Aleph, as an ILP framework, needs positive and negative examples as well as background knowledge in order to induce hypotheses. The examples must be provided as Prolog facts. Positive and negative examples go in different files. A typical positive examples file for a classification task (where the target predicate is `cat`) could look like this:

```
cat(mrSnuggles).
cat(meowingtons).
cat(captainCuddle).
```

Note that the parameters for the target predicate are starting in lower case, which indicates that the provided examples have to be constants. The negative examples file will look similar, except that it states all examples for which the target predicate yields *false*.

The background knowledge is again a different file and states everything that is further known (e.g. of the examples). Sticking with our example about cats, we could also want to provide Aleph with some background information on certain attributes of our examples. So a typical background knowledge file could look like this:

```
animal(mrSnuggles).
furry(mrSnuggles).
has(blueCollar, mrSnuggles).

animal(fluffy).
has(leftEar, fluffy).
has(rightEar, fluffy).
larger(leftEar, rightEar).
```

Since the most specific clause for an example can become very large with many ground literals, the search for a more general clause that can be used as hypothesis has to be constrained by construction rules for these clauses. Aleph therefore uses mode declarations that guide the introduction and form of new literals into the clause. There are two types of modes; `modeh` for the head of a rule (the positive literal in a Horn clause) or `modeb` for the body literals. The modes consist of a recall n that states how many versions of the literal are allowed in a rule and an atom with place-markers that state the literal together with annotations on input- and output-variables as well as constants (see [Mug95]).

To make this more clear, here is an example for the mode declarations for the grandfather task from above:

```
:- modeh(1, grandfather(+human, +human)).
:- modeb(*, parent(-human, +human)).
:- modeb(*, male(+human)).
```

The first mode states that the head of the rule (and therefore the target predicate) will be the atom `grandfather`. Its parameters have to be of the type `human`. The $+$ annotation says that the rule head needs two variables. The second mode declaration states the `parent` atom and declares again that the parameters have to be of type human. Here, the $+$ at the second parameter tells, that the system is only allowed to introduce the atom `parent` in the clause if it already contains a variable of type human. The $-$ at the first attribute introduces a new variable into the clause. With this mode declaration we made sure that the `parent` atom will generate a new variable that can act as the parent of the human already in this atom. The `male` mode declaration needs as input just one human. It does not need to generate a new variable for the clause but can only be used if at least one variable of type human already exists.

One of the big advantages of Aleph is its adaptability. I will use Aleph because it is perfectly suited to experiment with different settings. The search for a good clause in Aleph is mainly driven by an evaluation function for the clause. The policy is to take the clause with the highest evaluation rating. The default evaluation function prefers clauses that have a high number of positive examples that they cover compared to the negative ones. But this function is adaptable (which we do in our system).

3 Methods

This chapter utilizes all of the previously described ideas, theories, methods, frameworks, etc. to build a Companion System for explaining black-box classifiers with Inductive Logic Programming. In a first step, I will describe the different levels that an explanation can have. Next, the overall structure and workflow of the proposed system will be described. The modules in this step will then be explained in detail and the concrete implementation is progressively developed.

3.1 Three Levels of Explanation

With my proposed system it will be possible to choose at which level of expressiveness the inference result of a classifier shall be explained. This will help practitioners that make heavy use of black-box classifiers to ponder the necessity of getting an exhaustive explanation against possible time constraints.

The three levels of expressiveness for explanations in the system are:

1. Raw inference result
2. Visualization of important parts in the input (for the inference result)
3. Logic rules for the inference result with predicates understandable for humans

There are some use cases in which a practitioner only wants to know the raw inference result. That means the system will only tell the top-5 classes with the highest black-box output along with the estimators. [1] This sparse information can be relevant if the user already got an explanation earlier and just wants to know the top classes again. Or the user is absolutely sure of a decision and wants to get a confirmation. However in critical application contexts this information alone should never be used as the ultimate decision-making power.

The second level of expressiveness is the visual explanation of the inference result. In this level the user can see the input image along with a highlighting of parts of the image that are positively and negatively correlated with the class. This explanation is the result of the application of the standard LIME algorithm.

In many cases the visual explanation is not enough to explain all complex decisions the black-box classifier made during inference (see 2.3). Therefore the third and most expressive level of explanation consists of the statement of logic

[1] Note that the term estimator is used and not probability. Although the output of a network is often normalized to be between 0 and 1 and the value can be interpreted as "How probable is the class compared to another class?", it is no probability in the formally correct sense.

rules. These rules describe the decision process with predicates understandable by humans. A major problem of this requirement is the lack of predicates available to systems that could infer such logic rules. In other words, the building blocks for the background knowledge that Aleph would need are simply not present up front.

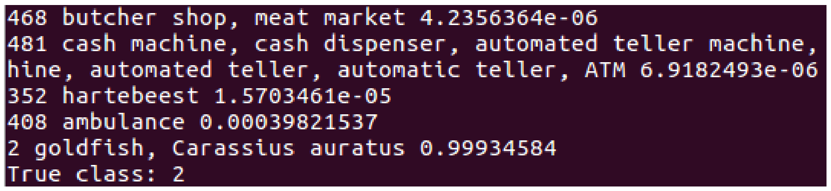
Let us restate and summarize the problem: You have an image for which a black-box classifier inferred a class. You want to get an explanation for the decision in the form of a logic rule. What is available to you is a representation of the input image where it is segmented in superpixels. These superpixels can be used as background knowledge for an ILP system (We will use Aleph). The only downside is that we do not have predicates yet for the parts of the image. And even further, we also want to exploit the full expressibility of first order logic. This means we would also need the possibility to get relations that are embedded in the image parts. How can one approach this problem?

One early thought was the usage of the very same classifier not only to classify the whole image but also the parts of it. But this has a few flaws: First of all it would mean to apply a potentially bad classifier again in order to get an explanation for itself. So the question would arise: Why should I trust this classifier? This could be however mitigated when the domain is known and a classifier especially for this field of use has been thoroughly tested. There is however another rather technical difficulty: The image parts to be labeled by a classifier often come in very irregular shapes. However traditional Convolutional Neural Networks require a fixed, rectangular input shape. These requirements could be fulfilled by resizing the input to the required shape and filling the missing values with reasonable numbers but I let these considerations be subject of future work.

Coming back to the levels of explanation, the very first step is the inference of the "true class" as given as inference result by the black-box classifier. At the same time this is the result of the very first level of explanation. This is the decision result, which should now be challenged. In the following levels of the system, this is the result that shall be explained.

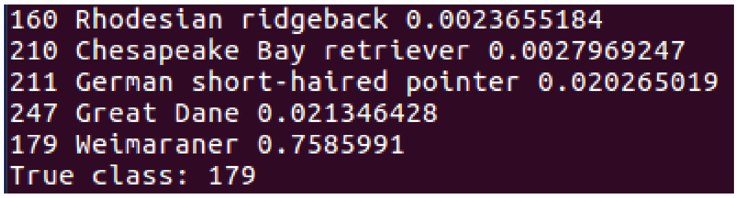
We start with an image which we do not know the class of. The only preprocessing that we perform before we feed it into the black-box is a resizing to the input needed by the classifier. Every further preprocessing steps that the black-box performs will be considered unknown. We do this in order to be sure that our explanations cover all the - potentially complex or error-prone - behavior of the black-box. It can not be ruled out that the preprocessing alters the image already in a way that would explain weird output.

Now the system gives the user the top-5 inference results along with an estimator number for each class. The top-5 results are the five classes that the classifier finds being most probable for the image. By looking at the estimators the user also already can see how confident the classifier was with its decision.



```
468 butcher shop, meat market 4.2356364e-06
481 cash machine, cash dispenser, automated teller machine,
hine, automated teller, automatic teller, ATM 6.9182493e-06
352 hartebeest 1.5703461e-05
408 ambulance 0.00039821537
2 goldfish, Carassius auratus 0.99934584
True class: 2
```

(a) Inference result for an image of a goldfish.



```
160 Rhodesian ridgeback 0.0023655184
210 Chesapeake Bay retriever 0.0027969247
211 German short-haired pointer 0.020265019
247 Great Dane 0.021346428
179 Weimaraner 0.7585991
True class: 179
```

(b) Inference result for an image of a weimaraner dog.

Figure 5: The top-5 estimators for the inference result of an Inception V3 network [SLJ+15] for two images.

If the estimator for a class is close to 100 percent, the classifier was pretty sure. If we have multiple results that have very close estimators, chances are that the classifier had trouble telling them apart. Figure 5 shows example output for two images with different classes.

The next level of explanations is the highlighting of relevant parts in the image. This is done by LIME. It outputs an image that shows exactly which parts positively and negatively correspond to a selected inference result that we want to explain. With LIME it is not only possible to explain the top-1 class but any class and get the weights for every image part out of it. This is helpful since sometimes the user does not only want to know an explanation for the class with the highest estimator but also for the second- or third-best class. This also gives more insight on why certain image parts could fool a system into preferring wrong classes.

The most expressive level of explanations is the third, logic-rule-layer. It is able to also present information to the user that can not be expressed by highlighting pixels. This advantage has been thoroughly described earlier. The most important gains are the possibility of expressing not only facts about image parts but also their relations with each other. As described earlier an ILP system to induce the logic rules for this level needs background knowledge in the form of predicates. This is why in the next chapter, I will present a system that lets a user annotate an image in order to get human understandable predicates.

3.2 An Annotation System

The simplest and obviously most "human-like" way to get semantic information for the image parts is to let them be labeled by humans. This could be done by any person but at best the user that has to make a decision based on the classifier result. The question is, how such an annotation can take place. We always have to keep in mind, that the end result has to be a set of logic rules that is more expressive than the visualization alone. The user has to have an advantage over the visualization that justifies the additional work overhead that comes with the annotation.

3.2.1 Which parts to annotate?

Every design decision regarding the annotation workflow will have some kind of bias. This starts with the consideration of which information in the image should be annotated at which layer of granularity. The straight forward way that I took is to just take the same approach as LIME and use a superpixel segmentation algorithm that provides me with distinct, non-overlapping image parts. We now have a representation of the original image that makes it possible to talk about its contents; at best, distinct objects. Note that this approach is not perfect in terms of segmenting the image in a way that is semantically plausible. As described earlier, the superpixel segmentations are normally based on finding patches of similar pixels that extend until an edge is found. But we accept this bias for this thesis and postpone other approaches to future work.

3.2.2 System Overview

Annotating an image with potentially many parts along with their relations seems very tedious, especially when the user already saw a visual explanation in a previous step. It might be tempting to just rely on the visualization when making a decision with potentially high impact. Therefore the user of the system has to be supported in the best way possible in order to give an incentive to execute the annotation process. I therefore set a few high-level requirements for my system:

- Maximize the ease of use
- Maximize flexibility
- Support user by giving automated suggestions
- Present an appealing layout

I will now present the ideas and design decisions that went into the implementation of an annotation system that fulfills the given requirements and guides the annotation process.

The superpixels obtained from the previous segmentation step need to be presented to the user in a natural way. That means we can not just show them one by one without context and ask the user to label them. One disadvantage of this approach would be the missing interpretability of what can be seen on the superpixel. A small part of a dog's all grey fur can not easily be interpreted as that. The other problem would be the inability to annotate relations. By just showing the superpixels one by one, the user is not able to easily link two superpixels semantically together. The most natural way of presenting the superpixels is to keep them in their original spatial structure.

I decided on a simple Webpage for the system because it is set up quickly and does not require any external frameworks. In addition, most users should be familiar with the use of a browser and elements of a typical website.

The main window has the following components: One figure that shows the superpixels to be annotated and another the original image as a reference. Below is a button that lets the user download an XML file containing the annotation information for further steps. Figure 6 shows this main window.

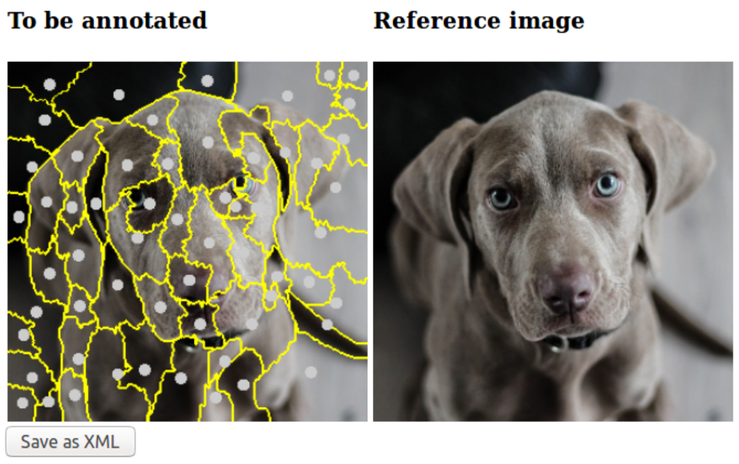
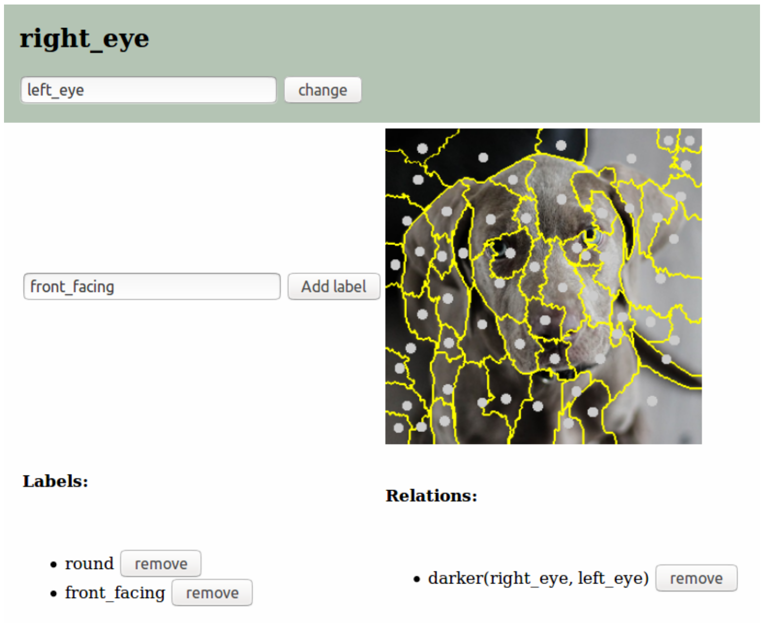


Figure 6: The main window of the Annotator website.

The figure that shows the superpixel segmented image is clickable at specific hotspots; one hotspot for every superpixel. When the user clicks on it, another window pops up that gives the user the possibility to fill in information for the superpixel. For every superpixel it is possible to give multiple (or no) labels and multiple (or no) relations to other superpixels. It is always possible to remove them later. When you try to insert a label with a name that is already present as a label, the insertion is blocked. Also for relations with the same name leading to the same superpixel, an insertion is not possible. However two relations from one superpixel can have the same name when one links to a different superpixel (see Figure 7).


right_eye

left_eye change

front_facing Add label

Labels:

- round remove
- front_facing remove

Relations:

- darker(right_eye, left_eye) remove

Figure 7: The annotation window for one superpixel.

The relations for one superpixel can be created by clicking on a superpixel in an exact copy of the original annotation figure. When you do this, a prompt pops up asking you for a name for the relation. As soon as you press OK, the relation is inserted. It is important to mention that the relations for a superpixel are just unidirectional. When you want to assign bidirectional relations, you have to go to the annotation window of the other superpixel and assign a relation with the same name leading to the first superpixel.

Once the user has finished annotating, it is possible to download and save the generated XML document. This is done by clicking on the button "Save as XML". The resulting document can now be used as input for the next step: Inducing logical rules using ILP.

3.2.3 Automatically generated annotations

There is a number of annotations that the system is able to derive automatically and therefore taking some work away from the user. The most obvious one is just the unique identifier of a superpixel. It looks like this: `sp_N` where N is a unique number. The next annotation is the size in pixels for a superpixel. A classifier might find it relevant how big a pixels is. The last label is the mean color of the superpixel. I derived a human readable name by seeking for the least distance in the Euclidean Color Space to a known color. So in the background knowledge you can find colors like `darkslategrey`, `olive` or `gold`.

There is also two spatial relations I let the system derive automatically, `top_of` and `left_of`. The system looks at pairs of superpixels and derives the relation from the coordinates of the center point of each superpixel. I only used relevant superpixels for the finding of the relations, because otherwise the background knowledge could become too large.

What does relevancy mean in this context? Obviously superpixels that just represent background do not need to be linked by a relation. The stone wall in the background of the "Mona Lisa" painting does not add any information to a classifier. But the relation to other exhibits from the "Louvre" that are visible in the image focus could be extremely important for a classifier to infer that you can see in fact the original "Mona Lisa" in the image and not a replica in another atelier.

Other examples for "irrelevant" superpixels normally are big patches of basically the same color inside the object in question. They do not add any extra value for a classifier and therefore for an explanation. Suppose the segmentation algorithm decided to sub-segment the black TV screen surface into multiple patches. Most of these patches are redundant.

So how can we decide on which superpixels are just redundant and do not give any extra value? The answer lies in the exploitation of the output from the previous level. LIME provided us with everything that we need in order to decide on relevant information. The information is coded into the weights for the superpixels. These weights give us a measurement for relevancy. Remember that the weights could be positive or negative, either describing a strong positive or negative correlation towards the class in question. Superpixels with a high absolute value therefore can be interpreted as being relatively important for the black-box classifier. Note that we use this technique to also include weights with a high negative importance for the class.

My straight forward solution on deciding which superpixels to take for the automatic relation annotation is to just keep a user defined keep percentage θ of the superpixels with the highest absolute weight.

3.2.4 Technical Details

The technologies that went into the implementation of the Website are:

- HTML
- CSS

- Javascript
- XML

I will now give a description of the XML file structure that is created in the background to hold the information given by the annotator.

The root element is always `image`. It has an attribute `true-class` that has the index of the class that the classifier calculated. The child elements of `image` are tags with the tagname `superpixel`. They contain automatically calculated attributes for the color, the coordinates of their middle point, a unique identifier, the size in pixels and the calculated LIME-weight. An additional attribute is `name`. When nothing was inputted by the user, the name is identical to the identifier.

When a superpixel has human annotated labels, the superpixel tag contains further child elements with the tagname `label` containing the label name as content. For all human annotated relations there is another type of child element with the tag name `relation` that contains the relation name as text. It further has the attribute `with` that has as value the identifier of the superpixel the relation is formed with. Note here, that this relation is unidirectional. No corresponding counterpart is created for the partner superpixel. This is, because the system can not infer what the reverse relation should be named.

3.3 LIME-Aleph 2.0

This section presents the ideas and implementation of a system that outputs logic rule explanations for the inference result of black-box classifiers. Since the foundation of this work was set in an earlier paper (see [RSS18]), I named the resulting system "LIME-Aleph 2.0" How can one induce one (or more) logical theories for the occurrence of an inference result with the help of structured information about an image that is comprehensible to humans?

Remember again the building blocks that an ILP system needs: On the one hand there are several positive and negative examples. Aleph can use them to generalize beyond them by inverse entailment to such an extent that a set of general rules (= the theory) emerge. But this is not enough. The algorithm also requires background knowledge that further describes the examples. This background knowledge is the pool from which predicates are drawn that the induced theory uses.

The good news is we already have the background knowledge. Through the preceding annotation step we have structured information about the individual parts of the image as well as the relations of the parts to each other. The problem is: We only have one example. This is not enough for generalization. So how do we manage to create several from one example so that they don't deviate too much from the original image, but at the same time offer an added value for the generalization? It is important that the changed images are not too different from the original image, because we need a local explanation for the image.

3.3.1 The Idea

The answer is an adaptation of LIME. During the algorithm, LIME generates many versions of an input image by randomly switching off superpixel. In the next step, LIME infers the classes of these modified versions and assigns them a similarity value with the original image. With this information it can create a linear model that locally approximates the black-box classifier.

My approach makes use of this mechanics and changes the original algorithm as follows:

1. Create the perturbed versions with deactivated superpixels as in the original LIME code. These will be the examples for Aleph.
2. If a superpixel is switched off, its information is not included in the background knowledge for this image version.
3. Infer for each image version whether the black-box classifier thinks it belongs to the true class or not. If it belongs to the class, the image version becomes a positive example, otherwise a negative one.

4. Infer a logic theory with Aleph to get rules for what the black-box classifier finds important in the original input image

Figure 8 shows the overall workflow of the system.

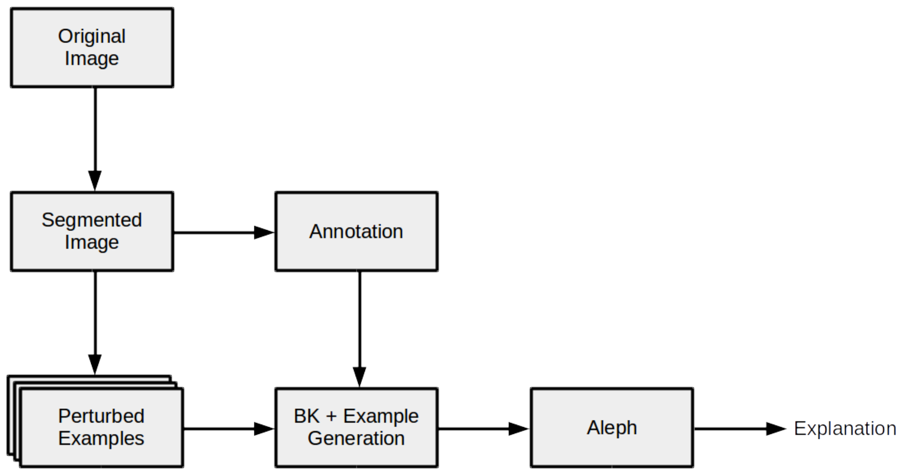


Figure 8: The overview of LIME-Aleph.

The way I decide that annotated information belongs to the background knowledge is straight forward: If a superpixel is switched on in a perturbed version, it receives a predicate that indicates that the example contains the superpixel. All the information that is linked to it (like labels or being one end of a relation) will only be included if it can be derived from the superpixels that an example contains. The case for labels is simple: If a superpixel is contained in an example, then all the associated labels are also contained. If and only if two superpixels are contained that have a relation between them, then the relation is contained in the background knowledge.

3.3.2 Generating the Aleph files

Language Constraints

Since Aleph can be used to flexibly control which hypotheses are to be favored, the following considerations will focus on how the search for a particularly meaningful hypothesis in the hypothesis space is to be structured. The modes of Aleph allow a very straight-forward implementation of a language bias.

Aleph first expects the modes to be specified as described above in the chapter about Aleph. The first is the target predicate. I decided on giving it the abstract name `true_class` for all generations. When you want to evaluate the inferred rules later, you can take the human readable class name from the specification of the classifier. The complete mode specification for the rule head (which is the target predicate) in Aleph is `:- modeh(1,true_class(+example)).` This means that there is only one target predicate. Specifying `+example` means that the input parameter for this predicate must be of the "type" `example`, i.e. it describes a complete example. The examples are the versions of the initial image explained above.

Since an example consists of several superpixels and there are several examples, there must be a two-digit predicate describing the affiliation of a superpixel to a particular example. So there must be a mode that covers this predicate: `:- modeb(*,contains(+example,-superpixel)).`. The `+` and `-` annotations tell Aleph that whenever there is an example being used in the current clause, Aleph can include a `contains` predicate that also introduce a new superpixel into the clause. This is Aleph's version of a language bias in the induction of a hypothesis. The `contains` predicate is some form of gatekeeper, that makes sure that the theory can only contain superpixels if they also are contained in the current example.

Next are the modes for the predicates that hold information about one superpixel in the example. I decided to distinguish between information that can be automatically generated (automatic information) and information that a human labeler gave to the system (manual information). Both automatic and manual information can further be subdivided into labels and relations. Table 1 will give the basic automatic information as well as examples for manual information.

	Labels	**Relations**
Automatic	`color`, `size`	`left_of`, `top_of`
Manual	`furry`, `striped`, `smooth`	`whiter`, `more_transparent`

Table 1: Automatic and examples of manual information that are used by Aleph.

The automatically assigned labels `name`, `color` and `size` are always present. The modes for them are two-ary since they need to be linked to a superpixel. The superpixel is the input variable. The name, color and the size are constants:

```
 :- modeb(*, has_name(+superpixel, #name)).
:- modeb(*, has_color(+superpixel, #color)).
 :- modeb(*, has_size(+superpixel, #size)).
```

Note that the name is either identical to the unique identifier of a superpixel if it was not manually labeled or is the name that the labeler has given the superpixel.

The manually given labels are introduced in the modes like the following:

```
:- modeb(*, has_label(+superpixel, #label)).
```

An additional predicate for superpixels that do not have any labels exists:

```
:- modeb(*, unlabeled(+superpixel)).
```

The modes for the relations are two-ary and look like this (the first two relations are the automatically assigned, the others are examples for manually assigned relations):

- `:- modeb(*, left_of(+superpixel, +superpixel)).`
- `:- modeb(*, top_of(+superpixel, +superpixel)).`
- `:- modeb(*, more_transparent(+superpixel, +superpixel)).`
- `:- modeb(*, furrier(+superpixel, +superpixel)).`

Positive and negative examples, background knowledge

As described earlier, the system generates perturbed versions of the original image by switching off superpixels randomly (setting the complete pixel colors to the mean color of the superpixel). Then the classifier gets a score for the class of the original image. Normally, only if the class reaches the top-1 score, we would consider it a positive example. Because the classifiers can vary and can have a huge number of classes, we do not want the criterion if the example is positive to be that restrictive. So we introduced a user-defined threshold ϕ that the prediction of the "true class" has to reach in order to make the example a positive one. After the perturbed versions are generated and put into the classifier, all the positive examples are written into the positive file for Aleph and the negative ones are written in the negative file. All the examples look similar to this: `true_class(exampleN).` with N being a consecutive number for the examples.

Settings

This section is about the settings that can be tweaked in Aleph [Sri07] to guide the search for a hypothesis:

- **The number of iterations**

  Aleph searches the hypothesis space iteratively by expanding the clauses by a new predicate in each step according to the modes. You can bound the number of steps i Aleph takes. We can think of how big the number should be by considering a typical clause that should be learned. With the modes that we have, the setting `:- set(i, 3).` is sufficient.

- **The maximum clause length**

  Especially for a comprehensible explanation we do not want to have rules that are too long. The maximum number of literals in a rule (including the rule head) can be set with `:- set(clauselength, 20).`

- **The minimum evaluation score for a clause**

  This parameter sets a minimal bound on the evaluation score that has to be reached for a clause to be included in the theory. It can be set with `:- set(minscore, 0).`

- **The number of nodes searched for one example**

  The setting `:- set(nodes, 20000).` bounds the number of clauses (the nodes of the search space) that are evaluated for one positive example. Higher values will mean a longer search but a possible better result in terms of evaluation score.

3.3.3 User-Defined Evaluation Functions

One aspect of Aleph that has been taken for granted until now, is the usage of the standard evaluation function. When you do not set a user-defined evaluation function to guide the search, the *coverage* is used by default. Coverage is defined by $coverage(Clause) = P - N$ where P is the number of positive examples covered by the clause and N the number of negative examples. That means, having many positive examples in comparison to negative examples that get covered, is preferred.

One of the perks of Aleph is its high adaptability. The evaluation function is no exception. You can switch on a user-defined function by stating `:- set(evalfn, user).` in the Aleph files. When you do that you also have to provide a cost function for every clause. They all need to have the form `cost(Clause, ClauseLabel, Cost) :- Body.` where `ClauseLabel` has to be a list of the form `[P, N, L]`. P and N have the same meaning as described above. L is the number of literals in the clause. They all can be used to calculate the cost for a clause. We have to ask ourselves which clauses we want to prefer. The dimensions that we have at our disposal are

- P: The number of positive examples covered by the clause
- N: The number of negative examples covered by the clause
- $d(e)$: The dissimilarity of a perturbed example from the original image

An approach for an evaluation function that makes use of these dimensions can be found at our paper [RSS18]. Here we defined the dissimilarity $d(e)$ simply as the pixel-wise Euclidean distance between the perturbed and the original image like in standard LIME. The idea was to prefer clauses where we have a large P with a low distance $d(e)$ with e in P. In contrast, a large N with a low distance $d(e)$ with e in N should be avoided. Informally that means we want to favor clauses that cover a large amount of positive examples that are near to the query image and want to avoid clauses that (falsely) cover a large amount of negative examples that are near to the query image.

The evaluation function of Aleph should give high values to "better" clauses. When you assign a custom evaluation function, Aleph looks for the cost function of the particular clause and simply negates it. Therefore the cost function has to be higher for unpreferrable clauses. A cost function that implements the above idea could look like this:

$$cost(Clause) =$$

$$\sum_{e \in E^+} \begin{cases} -c(e), & \text{if e gets covered by the clause} \\ 0, & \text{otherwise} \end{cases}$$

$$+$$

$$\sum_{e \in E^-} \begin{cases} 0, & \text{if e does not get covered by the clause} \\ c(e), & \text{otherwise} \end{cases}$$

where $c(e)$ is defined as $\frac{1}{(1+d(e))^2}$ with $d(e)$ being the distance assigned to e as described above. E^+ is the set of all positive examples and E^- the set of all negative examples.

The term for $c(e)$ gets smaller very quickly for large distances $d(e)$. Also the additional 1 in the denominator prevents the term from becoming infinitely large, when the example image matches the original image (meaning the distance becomes zero).

For this thesis I will propose a different idea for a dissimilarity measure $d(e)$ which will actually give even better results and is tailored better to the task of extracting information from the annotations. Dissimilarity $d(e)$ of a perturbed example image is still defined as a distance from the original image. But this time we want to prefer images that have a higher $d(e)$. Why is that so? I defined $d(e)$ in such a way that it has higher values when less superpixels are switched on. When a classifier now still gives a positive result for the perturbed example even when so many superpixels are switched off, chances are high that these superpixels are very important for the classifier and the information that they include should be also included in the induced Aleph theory.

My hypothesis is that this way of determining importance of a superpixel is better suited for inducing a logic theory where we basically want to find the important predicates as well as superpixels that are involved in relations. The standard LIME definition of fidelity to the complex classifier is expressed by preferring perturbed examples that are close to the example in question. This becomes more important when approximating a linear model to the nonlinear decision boundary.

How can we formally define the idea of preferring positive examples that have less superpixels switched on? My approach that not only covers absence of superpixels but also the color information is to first define a measure for dissimilarity that is bound to a single superpixel. I want to quantify how much the pixel values in a single superpixel differ from the mean pixel value over the complete original image. So the dissimilarities d_{sp} for every single superpixel sp in the original image can be calculated up front and it is defined as

$$d_{sp} = \frac{1}{\#total\_pixels} \sum_{p \in sp} \frac{|r_p - \bar{r}| + |g_p - \bar{g}| + |b_p - \bar{b}|}{3}$$

Here, $\#total\_pixels$ is the overall number of pixels in the image. $p \in sp$ means that we take pixels p from the superpixel sp. r_p, g_p and b_p mean the values for the red, green and blue color value for the pixel p. $\bar{r}$, $\bar{g}$ and $\bar{b}$ mean the mean value for the red, green and blue color channels over the complete original image. Note that we do not normalize over the size of the superpixel sp in terms of pixels that it contains. However we normalize over the number of total pixels in the complete image in order to generalize over different image sizes.

Now we have found values that describe for every superpixel in the original image how its values deviate from the mean pixel value (in all three color channels) of the complete original image. This actually helps us in defining a measure of how dissimilar one perturbed example is from the original example while staying in the color space realm. The dissimilarity d_e for one perturbed example e is now defined like this:

$$d(e) = \frac{1}{\#sp} + \sum_{sp \notin e} d_{sp}$$

where $\#sp$ is the number of superpixels in the image and $sp \notin e$ meaning that we sum over all superpixels that are switched off in the perturbed example e. This means that we have now expressed the dissimilarity of a perturbed example as the sum of the deviations of the switched off superpixels. The additive constant is necessary because we keep the original image in our perturbed sample pool. This would mean that without the constant, $d(e)$ would become 0 and the example would not be able to contribute in any way.

Let us now rethink how a user-defined cost function should look like that prefers examples that have less superpixels switched off. For the overall cost function of a clause I will keep the structure of a sum over the covered positive and negative examples. What changes is the calculation of the term $c(e)$. The original idea of having negative examples being covered by a clause having negative effect on the search should remain. On the other hand, the magnitude of this negative effect could be derived from the dissimilarity $d(e)$ of the perturbed example. To make this more clear, Table 2 shows how $c(e_i)$ should look like for different examples e_i when they are covered by a clause.

Example	Dissimilarity $d(e)$	Predicted class $p(e)$	Contribution term $c(e_i)$
e_1	low	+1	weakly negative
e_2	low	-1	weakly positive
e_3	high	+1	strongly negative
e_4	high	-1	strongly positive

Table 2: Desired nature of term $c(e_i)$ for different examples e_i.

One can see that the sign of $c(e)$ only depends on the predicted class $p(e)$ for the example. It can be modeled by the inverse signum function $I(p(e)) = -sgn(p(e))$. Since the dissimilarity $d(e)$ is a continuous function, I will use it as a weight that is multiplied to I. So the final contribution term $c(e)$ will be defined as follows:

$$c(e) = I(p(e)) \cdot d(e)$$

The simplified cost function for a clause that covers examples E_{cov} therefore is

$$cost(Clause) = \sum_{e \in E_{cov}} c(e)$$

When defining it as cost function for a clause in Aleph, the user defined evaluation function will take the term $-cost(Clause)$ as score for searching for the best clause. This achieves that clauses that cover examples with few superpixels and still are classified as positive are preferred more over clauses that cover examples with many superpixels and are classified as negative.

Since with the usage of a user defined evaluation function, we do not have the chance to allow for a bit of noise in our training set, we also multiply the terms for the negative examples in the sum with the arbitrary value of 10. This prevents the theory from covering too many False Positives.

4 Experiments and Results

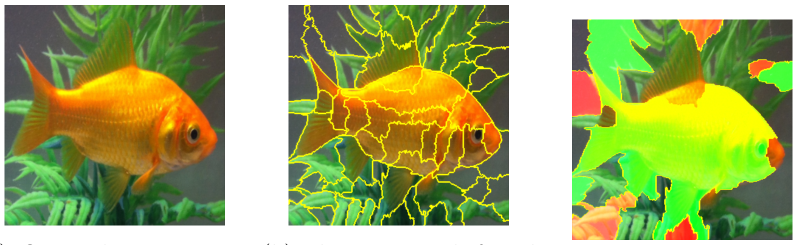
I will now present the experiments that were conducted for this thesis. The basis were always the systems described above. The goal was to show the differences in expressive power of LIME-Aleph 2.0 with different levels of information given to the system and to show that the statement of logic rules in general can give more knowledge about the inference result than visualization alone. The experiments will however not evaluate the usability of the Annotation System since this is not the focus of this work. Note also that the experiments are more of qualitative nature to show which rules were induced by the system.

4.1 Automatic annotation only

The first experiment shall act as a baseline. We will not take any human annotation into account and see how the inferred hypothesis looks like solely based on automatically obtained background knowledge. As the superpixels are addressed by an identifier, we can still incorporate them in the background knowledge as individual objects. But any label associated with it is gone. We will however keep the color information as well as the size information as they are generated automatically. Relations will be considered in a later step.

Several experiments will be conducted with different input images. The different levels of explanations will be shown and compared with each other.

The black-box classifier we will use is the Inception V3 network [SLJ^{+}15]. It was pre-trained with images from the ImageNet database and contains ca. 1000 classes of all sorts of objects and animals [Lab16].



(a) Original input image for the network. (b) The superpixels found with *quickshift*. (c) The LIME output.

Figure 9: The original input of a goldfish[2], the superpixel representation and the LIME output.

[2]Taken from `https://en.wikipedia.org/wiki/Common_goldfish`. Shared and modified under the CreativeCommons Attribution-ShareAlike 3.0 Unported License (Link: `https://creativecommons.org/licenses/by-sa/3.0/legalcode`), Author: Lehava.m, Title of work: specialis hippurus

Let us start with the image of a goldfish. Figure 9a shows the input that was resized to match the input dimensions of the Inception network. First we get an inference result from the classifier by letting the original image run through the network one time. This will give us the following top-5 results:

Class index	**Name(s)**	**Predictions**
468	butcher shop	0.000004
481	cash machine	0.000007
352	hartebeest	0.000016
408	ambulance	0.000398
2	goldfish	0.999346

Table 3: Top-5 estimators for the goldfish image.

The top class is in fact a goldfish. The classifier is very sure about it. Now we want to know the parts of it that the classifier finds important. LIME will provide us with that information. In Figure 9c you can see the LIME output for the original image. It is no surprise that almost all parts of the fish are considered positively relevant for the class. But also parts of the water background seem to be of interest. The patch in the upper left corner of the LIME output has a comparably high positive LIME weight. This could be a first indicator that the network has also learned to relate cloudy water with (gold-)fishes.

Now for the results of the third level of explanation: the logic rules. For this experiment we can circumvent the annotation process since we only use automatically generated predicates. The modes and settings are listed below:

- **Modes**:
  - `:- modeh(1, true_class(+example)).`
  - `:- modeb(*, contains(+example, -superpixel)).`
  - `:- modeb(*, has_name(+superpixel, #name)).`
  - `:- modeb(*, has_color(+superpixel, #color)).`
  - `:- modeb(*, has_size(+superpixel, -size)).`
  - `:- modeb(*, larger(+size, +size)).`
- **Settings**:
  - `:- set(i, 3).`
  - `:- set(clauselength, 20).`
  - `:- set(minscore, 0).`
  - `:- set(nodes, 20000).`

These will be the standard settings for all the experiments. If a particular setting will change, I will indicate this. For the other experiments I will only state the additional settings.

I let the system generate $s = 200$ perturbed examples for the original image. One of these was the original image. For all the examples from this sample, the classifier inferred a class. The threshold ϕ for being a positive example was set to `0.25`. So if the prediction for goldfish was greater than or equal to `0.25` it was classified as positive, otherwise negative. Further, the dissimilarity $d(e)$ was measured.

Aleph inferred the following rules:

1. `true_class(A) :- contains(A,B), has_color(B,olive), contains(A,C), has_color(C,chocolate). [Pos cover = 159 Neg cover = 5]`
2. `true_class(A) :- contains(A,B), has_color(B,chocolate), contains(A,C), has_color(C,saddlebrown). [Pos cover = 120 Neg cover = 3]`
3. `true_class(A) :- contains(A,B), has_color(B,olive), contains(A,C), has_color(C,saddlebrown). [Pos cover = 120 Neg cover = 2]`
4. `true_class(A) :- contains(A,B), has_name(B,sp_15). [Pos cover = 109 Neg cover = 2]`
5. `true_class(A) :- contains(A,B), has_color(B,saddlebrown), contains(A,C), has_name(C,sp_37). [Pos cover = 86 Neg cover = 1]`
6. `true_class(A) :- contains(A,B), has_color(B,orange). [Pos cover = 108 Neg cover = 0]`

The confusion matrix looks like this:

		Actual		
		+	-	
Pred	+	186	10	196
	−	0	4	4
		186	14	200

Table 4: Confusion matrix of the Aleph theory for automatic label annotations only (goldfish).

You can see that Aleph produced quite many rules with very few literals. For some rules, only the superpixel name is important, for some only color. On the first glimpse, the rule that covers the most positive examples (rule 1) tells us, that in order for the image to be a goldfish, the example has to contain a superpixel that has the color olive and another that has the color chocolate. The size of a superpixel or the `larger` relation do not seem to be of relevance for the classifier.

The training accuracy is `0.95`, however we have a high percentage of false positives, even though we let the cost function boost the influence of negative examples. But if you take a look at the single negative coverages of the rules, you can see that this outcome is relativised. Also in all of my experiments, the dataset is highly skewed towards positive examples. This can be attributed to the relatively low setting for ϕ. I could set ϕ to a higher value, but the higher it gets, the more probable it is, that we also do not grasp perfectly fine examples for the class. Imagine a top-1 result of 0.65 for the original image, but ϕ set to 0.75. Even the original image would not be classified as the true class. There is a trade-off being made and setting $\phi = 0.25$ throughout the examples is a design decision that can be challenged in future work.

The theory can be interpreted like this: If an example is covered by at least one of the rules, it is labeled as positive. Otherwise it is labeled as negative. To get a feeling what Aleph has learned, we can produce the image that contains all the superpixels that contain the information stated in the rules (see Figure 10). Note that this is a naive heuristic to try to visualize the learned rules. We will lose expressiveness in doing so.

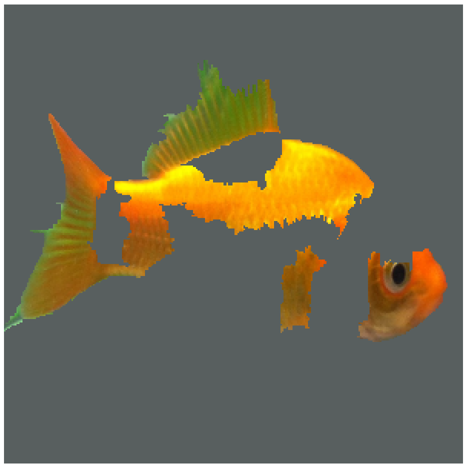
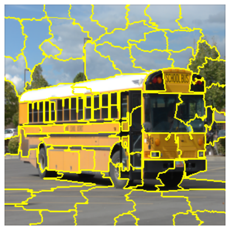


Figure 10: Heuristic visualization of the learned rules for the goldfish. The image results from highlighting all superpixels with the information contained in the rules (`olive, chocolate, saddlebrown, sp_15, sp_37, orange`).

The image shows that the induced rules in fact cover information corresponding to superpixels that a human can identify as being a goldfish.

The next image will be of an american school bus (see Figure 11)

(a) Original input image for the network.


(b) The superpixels found with *Quickshift*.

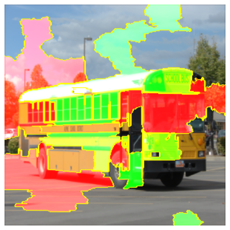

(c) The LIME output.

Figure 11: The original input of a schoolbus[3], the superpixel representation and the LIME output.

The first level of explanation yields the following results indicating that the image is in fact a school bus:

Class index	**Name(s)**	**Predictions**
875	trolleybus	0.000029
706	passenger car	0.000031
94	hornbill	0.000268
97	toucan	0.000849
780	school bus	0.995545

Table 5: Top-5 estimators for the school bus image.

We leave the Aleph modes and settings exactly the same. Also, $s = 200$ and $\phi = 0.25$ stay the same. The induced theory now is a little bit bigger and contains 11 rules:

1. `true_class(A) :- contains(A,B), has_name(B,sp_42), contains(A,C), has_name(C,sp_55).   [Pos cover = 58 Neg cover = 1]`
2. `true_class(A) :- contains(A,B), has_color(B,antiquewhite), contains(A,C), has_color(C,sienna).   [Pos cover = 73 Neg cover = 0]`
3. `true_class(A) :- contains(A,B), has_color(B,peru), contains(A,C), has_color(C,darkolivegreen).   [Pos cover = 86 Neg cover = 1]`
4. `true_class(A) :- contains(A,B), has_name(B,sp_19), contains(A,C), has_color(C,darkgoldenrod).   [Pos cover = 47 Neg cover = 0]`

[3]Taken from `https://en.wikipedia.org/wiki/School_bus`. Shared and modified under the CreativeCommons Attribution-ShareAlike 4.0 International License (Link: `https://creativecommons.org/licenses/by-sa/4.0/legalcode`), Author: An Errant Knight, Title of work: A typical Alpine School District school bus (Utah)

5. `true_class(A) :- contains(A,B), has_name(B,sp_33), contains(A,C), has_name(C,sp_55).  [Pos cover = 53 Neg cover = 1]`

6. `true_class(A) :- contains(A,B), has_color(B,darkgoldenrod), contains(A,C), has_name(C,sp_44).  [Pos cover = 51 Neg cover = 0]`

7. `true_class(A) :- contains(A,B), has_name(B,sp_11), contains(A,C), has_name(C,sp_49).  [Pos cover = 58 Neg cover = 1]`

8. `true_class(A) :- contains(A,B), has_color(B,darkgoldenrod), contains(A,C), has_name(C,sp_52).  [Pos cover = 54 Neg cover = 0]`

9. `true_class(A) :- contains(A,B), has_name(B,sp_15), contains(A,C), has_name(C,sp_23).  [Pos cover = 51 Neg cover = 0]`

10. `true_class(A) :- contains(A,B), has_name(B,sp_33), contains(A,C), has_name(C,sp_46).  [Pos cover = 48 Neg cover = 0]`

11. `true_class(A) :- contains(A,B), has_name(B,sp_14), contains(A,C), has_name(C,sp_34).  [Pos cover = 56 Neg cover = 1]`

The confusion matrix shows good results:

		Actual		
		+	-	
Pred	+	176	5	181
	−	0	19	19
		176	24	200

Table 6: Confusion matrix of the Aleph theory for automatic label annotations only (school bus).

Figure 12 shows again the superpixels that contain the information from the rules as a heuristic.

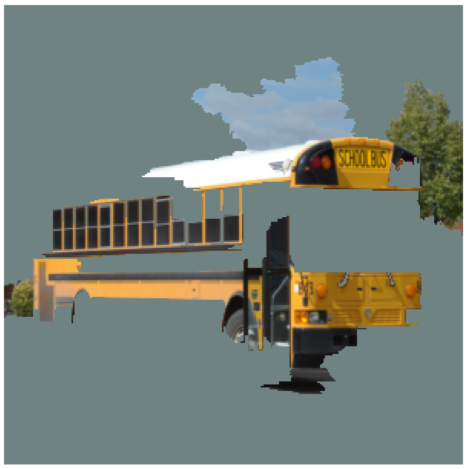


Figure 12: Heuristic visualization of the learned rules for the school bus. The image results from highlighting the superpixels with the information contained in the rules.

The amount of rules is very high in general. This could be an indicator that it is difficult for the system to grasp the details of the very complex decision boundary of the Inception network. You also can see the downside of exposing so many rules to the user. A single rule can be comprehensible, but the theory in full contains many concepts.

For the next experiment, we will also add the automatically inferred spatial relations `left_of(superpixel1, superpixel2)` and `top_of(superpixel1, superpixel2)` into account. We take again the image of the goldfish as input (see Figure 9). We leave the Aleph settings as well as ϕ the same and add the following modes to the previous:

- **Modes**:
  - `:- modeb(*, left_of(+superpixel, +superpixel)).`
  - `:- modeb(*, top_of(+superpixel, +superpixel)).`

  We set $s = 100$ this time.
  Here is the induced theory along with the confusion matrix:

1. `true_class(A) :- contains(A,B), has_name(B,sp_37). [Pos cover = 51 Neg cover = 0]`
2. `true_class(A) :- contains(A,B), has_color(B,olive), contains(A,C), has_name(C,sp_36). [Pos cover = 47 Neg cover = 0]`
3. `true_class(A) :- contains(A,B), has_color(B,chocolate), contains(A,C), left_of(C,B). [Pos cover = 67 Neg cover = 2]`

4. `true_class(A) :- contains(A,B), has_color(B,chocolate), contains(A,C), has_name(C,sp_32).  [Pos cover = 46 Neg cover = 1]`

5. `true_class(A) :- contains(A,B), has_color(B,olive), contains(A,C), has_color(C,chocolate).  [Pos cover = 78 Neg cover = 4]`

		Actual		
		+	-	
Pred	+	91	4	95
	−	0	5	5
		91	9	100

Table 7: Confusion matrix of the Aleph theory for automatic label and relation annotations only (goldfish).

Rule 3 contains in fact a relation predicate. So apparently what is important for the image to be a goldfish is, that there has to be a chocolate colored superpixel and there has to be another superpixel that is to the left of it. This is already more expressive than just visualizing all the superpixels that have this constellation. LIME could only show all the superpixels that meet this criterion. The chocolate colored superpixels in the image are the snoot, the upper tail fin and parts of the tail. So for the image to be a goldfish, you have to take one of these superpixels and one superpixel that is on the left of it. This has been explicitly stated by FOL rules.

4.2 Human annotation

For this experiment, we give Aleph the complete background knowledge of both automatic and manually received labels and relations(The full XML annotation file can be found on the attached CD).

The following list contains the complete mode settings and general settings for this experimental run. s was set to 200 and ϕ remained at 0.25.

- **Modes**:
  - `:- modeh(1, true_class(+example)).`
  - `:- modeb(*, contains(+example, -superpixel)).`
  - `:- modeb(*, has_name(+superpixel, #name)).`
  - `:- modeb(*, has_color(+superpixel, #color)).`
  - `:- modeb(*, has_size(+superpixel, -size)).`
  - `:- modeb(*, larger(+size, +size)).`

  - `:- modeb(*, left_of(+superpixel, +superpixel)).`
  - `:- modeb(*, top_of(+superpixel, +superpixel)).`
  - `:- modeb(*, has_label(+superpixel, #label)).`
  - `:- modeb(*, unlabeled(+superpixel)).`
  - `:- modeb(*, whiter(+superpixel, +superpixel)).`
  - `:- modeb(*, lighter(+superpixel, +superpixel)).`
- **Settings**:
  - `:- set(i, 3).`
  - `:- set(clauselength, 20).`
  - `:- set(minscore, 0).`
  - `:- set(nodes, 20000).`

The last two modes were generated because the annotation included the manual relations `whiter` and `lighter`.

Here is again the induced theory:

1. `true_class(A) :- contains(A,B), has_name(B,eye). [Pos cover = 105 Neg cover = 0]`
2. `true_class(A) :- contains(A,B), has_color(B,olivedrab), contains(A,C), has_label(C,striped). [Pos cover = 120 Neg cover = 4]`
3. `true_class(A) :- contains(A,B), has_name(B,sp_10), contains(A,C), has_label(C,striped). [Pos cover = 93 Neg cover = 3]`

4. `true_class(A) :- contains(A,B), has_label(B,striped), contains(A,C), has_color(C,saddlebrown).  [Pos cover = 126 Neg cover = 4]`
5. `true_class(A) :- contains(A,B), has_color(B,orange), contains(A,C), has_color(C,darkorange).  [Pos cover = 68 Neg cover = 0]`
6. `true_class(A) :- contains(A,B), has_label(B,round), contains(A,C), has_color(C,saddlebrown).  [Pos cover = 122 Neg cover = 6]`
7. `true_class(A) :- contains(A,B), has_label(B,smooth), contains(A,C), has_label(C,striped).  [Pos cover = 136 Neg cover = 6]`
8. `true_class(A) :- contains(A,B), has_color(B,chocolate), contains(A,C), has_color(C,striped).  [Pos cover = 140 Neg cover = 7]`

		Actual		
		+	-	
Pred	+	182	11	193
	−	0	7	7
		182	18	200

Table 8: Confusion matrix of the Aleph theory for human annotations (goldfish).

What catches the eye first, is the usage of the human annotations. The very first rule indicates that the classifier thinks that the fish eye is a strong indicator for the class *goldfish* (it covers 105 out of 182 positive examples; the eye was prominent in more than ca. 58 percent of the positive examples). The other rules contain a mixture of name, color and label information that have to be met.

The downside is, that you can not find any of the relation predicates in the theory. This indicates that for the classifier the introduction of names and labels gave Aleph enough information to work with. Relations do not seem that important to the classifier. You have to mention that the manual relation annotation was rather sparse. For a goldfish, you can not annotate that many meaningful relations.

We conduct the same experiment again for the annotated image of the school bus (The full annotation file can be found on the CD). All the modes and settings as well as s and ϕ stay the same. Here is the induced theory consisting of 9 rules:

1. `true_class(A) :- contains(A,B), has_name(B,sp_31), contains(A,C), has_label(C,rubber).  [Pos cover = 89 Neg cover = 2]`
2. `true_class(A) :- contains(A,B), has_name(B,sp_41), contains(A,C), has_label(C,rubber).  [Pos cover = 80 Neg cover = 3]`

3. `true_class(A) :- contains(A,B), has_color(B,antiquewhite), contains(A,C), has_color(C,darkolivegreen).  [Pos cover = 107 Neg cover = 0]`
4. `true_class(A) :- contains(A,B), has_name(B,sp_16), contains(A,C), has_color(C,peru).  [Pos cover = 56 Neg cover = 0]`
5. `true_class(A) :- contains(A,B), has_name(B,right_indicator), contains(A,C), has_color(C,peru).  [Pos cover = 54 Neg cover = 0]`
6. `true_class(A) :- contains(A,B), has_name(B,sp_18), contains(A,C), has_name(C,windshield).  [Pos cover = 54 Neg cover = 0]`
7. `true_class(A) :- contains(A,B), has_name(B,sp_21), contains(A,C), has_color(C,peru).  [Pos cover = 56 Neg cover = 0]`
8. `true_class(A) :- contains(A,B), has_color(B,sp_19), contains(A,C), has_name(C,right_indicator).  [Pos cover = 52 Neg cover = 0]`
9. `true_class(A) :- contains(A,B), contains(A,C), has_name(C,sp_32), top_of(B,C). [Pos cover = 95 Neg cover = 4]`

		Actual		
		+	-	
Pred	+	176	8	184
	−	0	16	16
		176	24	200

Table 9: Confusion matrix of the Aleph theory for human annotations (school bus).

Again a very big theory like in the last experiment. But this time it actually catches the relation `top_of` in the last rule. It says that there is the superpixel 32 (actually one patch of the back windows) and that there has to be some other superpixel on top of it in order for the class to hold. It seems that relations are more important for the classifier in this image. However, it was only one relation predicate in the whole theory. That could indicate that relations in general perish if there are human annotations that the system uses. I will leave the examination of this to future work.

5 Conclusion and Future Work

The experiments conducted look promising in terms of boosting human interpretability and expressiveness. I have shown that it is possible to incorporate human labelers into the explanation process to make it possible to work with human understandable predicates that are used for a logic theory to be induced. However, as in every system there is room for improvements on a variety of parts which I will elaborate in this section.

When incorporating relations into the annotation, be it manually or automatically, the implicit assumption of the system is, that the relation should be added to the background knowledge if and only if all two superpixels are switched on. This makes sense from a syntactical perspective but it is up to further research how it can be achieved to also incorporate the semantic meaning of the relation.

Another room for improvement is to generate more automatically derived labels and relations. Especially for the relations, the experiments have shown that they still are included very sparsely into the theory.

For the annotation step, one can think of other ways to receive image parts than the standard approach of LIME, quickshift. The parameter settings for the segmentation algorithms were left as they are in standard LIME. However it would give a benefit if the image was analyzed before segmentation and then the parameters are set accordingly. Especially images that contain parts of very different granularity could benefit from algorithms that can adapt to the image. Also methods for obtaining hierarchically structured superpixels could be used here.

The annotation system itself can be made more user friendly. Since the focus of this thesis was not in the field of Human Computer Interaction, at some points certain design decisions were maybe not optimal.

When annotating an image, one idea that could be elaborated further is the incorporation of fuzzy logic. Up to now, when deciding on labels or relations, the labeler has to think of a “normalized” version of the real label. For example the decision if a superpixel shows an object that is more transparent or more opaque is fuzzy. A component in the Annotator that helps the user in the decision (or can deal with fuzzy concepts internally) would bring a benefit.

This thesis is just a small step towards washing black-box classifiers a little bit whiter. The possibility of getting to audit AI systems is nowadays more important than ever. It is my deepest belief that computers and humans should work together. The computer as a powerful tool and the human as the ultimate decision maker.

References

[ACOG18] Marco Ancona, Enea Ceolini, Cengiz Oztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *6th International Conference on Learning Representations (ICLR 2018)*, 2018.

[AM17] Saleh Albelwi and Ausif Mahmood. A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6):242, 2017.

[CS94] Mark W Craven and Jude W Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning Proceedings 1994*, pages 37–45. Elsevier, 1994.

[DSZB17] Yinpeng Dong, Hang Su, Jun Zhu, and Fan Bao. Towards interpretable deep neural networks by leveraging adversarial examples. *arXiv preprint arXiv:1708.05493*, 2017.

[Fag18] Daniel Faggella. Ai for crime prevention and detection – current applications. Website, 2018. Available online `https://www.techemergence.com/ai-crime-prevention-5-current-applications/`; accessed on 7.10.2018.

[FH04] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GBY$^+$18] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An approach to evaluating interpretability of machine learning. *arXiv preprint arXiv:1806.00069*, 2018.

[Her17] Bernease Herman. The promise and peril of human evaluation for model interpretability. *arXiv preprint arXiv:1711.07414*, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Karen] Andrej Karpathy. Convolutional neural networks. Website, no year given. Available online `http://cs231n.github.io/convolutional-networks/`; accessed on 7.10.2018.

[Lab16] Stanford Vision Lab. Imagenet. Website, 2016. Available online `http://image-net.org/`; accessed on 7.10.2018.

[Lai15] Matthew Lai. Giraffe: Using deep reinforcement learning to play chess. *arXiv preprint arXiv:1509.01549*, 2015.

[LBD$^+$89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[MB88] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Machine Learning Proceedings 1988*, pages 339–352. Elsevier, 1988.

[MB00] Stephen H Muggleton and Christopher H Bryant. Theory completion using inverse entailment. In *International conference on inductive logic programming*, pages 130–146. Springer, 2000.

[MDR94] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[Mug91] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[Mug92] Stephen Muggleton. Inverting implication. In *Proceedings of the First European Workshop on Inductive Logic Programming. Vienna, Austria: Unpublished*. Citeseer, 1992.

[Mug95] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[PMJ$^+$16] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[RHvG18] Gabrielle Ras, Pim Haselager, and Marcel van Gerven. Explanation methods in deep learning: Users, values, concerns and challenges. *arXiv preprint arXiv:1803.07517*, 2018.

[RM03] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *IEEE International Conference on Computer Vision*, page 10. IEEE, 2003.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[RSS18] Johannes Rabold, Michael Siebers, and Ute Schmid. Explaining black-box classifiers with ilp–empowering lime with aleph to approximate non-linear decisions with relational rules. In *International Conference on Inductive Logic Programming*, pages 105–117. Springer, 2018.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[sidten] scikit-image development team. Scikit-image documentation for quickshift. Website, no year given. Available online `http://scikit-image.org/docs/dev/api/skimage.segmentation.html#skimage.segmentation.quickshift`; accessed on 07.10.2018.

[SLJ$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[Sri07] Ashwin Srinivasan. The aleph manual. Website, 2007. Available online `https://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html`; accessed on 27.09.2018.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[VS08] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. In *European Conference on Computer Vision*, pages 705–718. Springer, 2008.

[ZMJ16] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. Deepred–rule extraction from deep neural networks. In *International Conference on Discovery Science*, pages 457–473. Springer, 2016.

Note:

Attached to this thesis is a CD that contains the PDF of this work. Also, you can find the code and the auxiliary material for the experiments.