MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN DATA SCIENCE

Unsupervised Concepts Extraction in Neural Networks

CORBUGY, Sacha; Septon, Thibaut

Award date:
2021

Awarding institution:
University of Namur

Link to publication

Université de Namur
Faculté d'informatique
Année académique 2020–2021

Unsupervised Concepts Extraction in Neural Networks

Sacha Corbugy & Thibaut Septon

Maître de stage : Adrien Bibal

Promoteur : ______________________ (Signature pour approbation du dépôt - REE art. 40)
Benoit Frénay

Co-promoteur : Adrien Bibal

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Acknowledgements

Dear reader, through this master thesis, you are about to discover the work that has occupied us for approximately 4 months. Although confusing at times, it showed us a glimpse of what the scientific research field of explainable artificial intelligence is. We hope you will enjoy it as much as we did writing it.

First of all, we would like to thank our promoter Mr Benoit Frénay who gladly helped us during our research. You gave us an expert advice and your help was very valuable.

We would like to thank Adrien Bibal for his endless patience and all his valuable feedbacks, ideas and discussions on our work. The final result would not have been the same without your help and you made the work more enjoyable.

Finally, thanks to our family and friends who were there to listen to what seemed to be nonsense to them.

Abstract

Recently, the domain of Explainable Artificial Intelligence saw the advent of Testing with CAV (TCAV). Although very practical as they allow to check the importance of a concept in the decision making of a neural network, they pose the prerequisite of knowing the concept at stake and having a dedicated dataset. In order to remedy this, a method is proposed to obtain an overview of the different concepts that are important in the decision making process of a neural network. The idea beneath the proposed method is to compare and label the neural network instances based on their activation at a given layer inside the network itself. Using two different kinds of neural network, an image classifier and a game agent, the method is tested to see if an unsupervised extraction of concepts is possible.

Contents

Chapter 1

Introduction

1.1 Context and Problem

Last decade has seen huge improvements over the field of Artificial Intelligence (AI) notably due to the Machine Learning researchers. That period of time saw the emergence of black box applications that proved to be capable of beating the world best player of Go, painting realistic portraits or driving a car without any human help just to cite a few of them. However, with the rise of such applications, emerges some ethical questions too. Should a user place his trust without questioning the reasoning behind such tools? Understanding the reasoning of an uninterpretable model falls within the realm of Explainable Artificial Intelligence (XAI).

This work examines already trained and developed AIs and, if need be, see what can be done to troubleshoot them. To do this, attempting to see if the Neural Networks have learnt high-level concepts can help to better understand their reasoning. If the AI is performing well, it should improve the trust a user place within its decision. If the AI does not operate effectively, learning how it works provides insights on how to improve it.

1.2 Research Question

After exploring the state of the art techniques, it is discussed what constitutes a problem regarding the Testing with CAV (TCAV) method. This technique aims to check the importance of user-defined concepts onto the predictions of a Neural Network model. Therefore, defining these upstream is necessary. However, acquiring the data that represents these concepts in order to apply this method can be hard and need human intervention.

Regarding the problem defined above, this work aims to answer the following question: "**Is an unsupervised extraction of concepts possible given a Neural Network ?**".

1.3 Proposed Solution

In order to answer the research question, this work proposes a new approach to extract the concepts learnt by a Neural Network. Also, to verify the assumptions made, two different kinds of AIs are analysed through that method. One being an image classifier, another one being a video game agent.

1.4 Thesis Structure

To begin with, Chapter 2 introduces the background required to understand the rest of this thesis. Then, Chapter 3 oversees the different techniques that constitutes the state of the art. Thereafter, Chapter 4 presents the contribution of this document to the domain and Chapter 5 and Chapter 6 apply that contribution to two different cases in order to better judge the work done. Finally, Chapter 7 is a conclusion summing up the findings of this work, and discussing at what can be done next in further works.

Chapter 2

Background

This chapter aims to introduce the knowledge necessary for the proposed method and its applications. AI and Machine Learning domains are discussed, as well as the field of Explainable Artificial Intelligence.

2.1 AI for Board Games

For many years, there has been a certain attention for board games in the world of AI. This master thesis considers the specific case of Hearthstone, an online trading card game. It was chosen as it is played at a professional level (e-sports) and its state space is challenging for AI.

2.1.1 Hearthstone's Description

Hearthstone is an online trading card game developed and published by Blizzard Entertainment. At the beginning of a game, the player chooses among heroes available in the game and a 30-card deck, previously created by the player, to fight a battle against an opposing player. The two opponents compete on a virtual game board, similar to a board game. Heroes have 30 health points and one mana point on the first turn of the game, with the mana increasing by one point per turn until reaching a maximum of 10 points. Mana points are used to play the deck cards on the game board, as well as the hero's heroic power. The goal of the game is to reduce the opposing hero's health points to zero, through the use of minions, spells, weapons or through the use of the hero's heroic power controlled by the player. If one of the heroes reaches 0 health points, he loses the game. Each Hearthstone card has a cost in mana to be played. The following is a brief summary.

First, the *minion cards*, who summon a creature on the playing field. It has an attack value, a health value, and possibly a special action (such as an action upon arrival in the game, called a "battle cry," or an action upon death, called a "death rattle". When placed on the game board, a minion must wait one turn before he can attack (except for those with Charge or Rush effect who can attack immediately). When a minion attacks an opposing minion, it receives damage

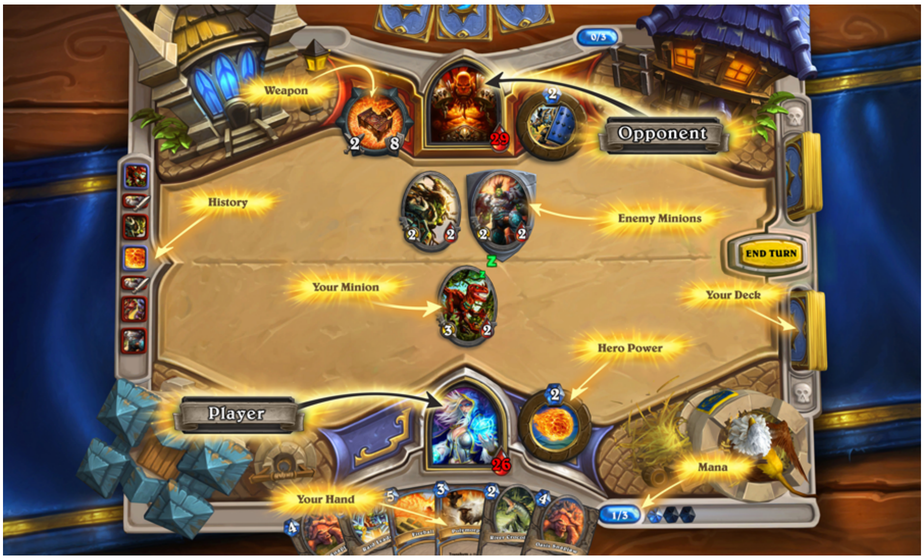


Figure 2.1: Hearthstone UI, taken from hearthstone.gamepedia.com[1]

equivalent to the latter's attack. However, a minion does not take damage if he attacks the opposing hero.

Second, the spell cards, which perform one or more special actions. The spells can be of different categories. First, the *ordinary* spells, which are the ones that do damage, heal, draw cards, steal cards from the opponent, etc. Next are the *secrets*, which are cards that prepare an action unknown to the opponent and which is triggered during his turn, under certain conditions. To conclude, the *quests* spells are cards played at the beginning of the game for one mana point, allowing to receive a reward after completing the quest several turns later.

The next type is weapon cards, which allow the hero to attack in addition to his minions. These cards have an attack value and a number of uses (durability). The weapon loses one durability point when the hero with the weapon attacks. The weapon can also be destroyed in three different ways: when it reaches 0 durability, when a second weapon is equipped (the equipped weapon is then replaced by the new one) or when the opponent plays specific cards to destroy it.

Finally, the legendary heroes, which are "hero" cards that replace standard heroes. When a card of this type is played, it allows you to add several armor points to the hero and replace his heroic power with an improved version.

Each type of minion, spell, weapon or hero, when played or placed on the game board, can generate or benefit from additional special actions, depending on the characteristics of each card.

Figure 2.1 shows what the Hearthstone game board looks like from a player point of view.

[1] https://hearthstone.gamepedia.com/File:Ui-guide-small.png (17/03/21)

Characteristics	Description	
Accessible	If the game state is completely available to its agents.	×
Agents	Number of players in a game.	2
Deterministic	If the next environment state is completely determined by the current state and the action that is going to be performed.	✓
Discrete	If the information and possible actions are clearly defined, then the environment is said to be discrete. Otherwise it is continuous.	✓
Episodic	The agent's experience is divided into episodes. This means it will first perceive and then act.	✓
Static	If the environment can change while an agent is evaluating its choice, it is said to be dynamic, static otherwise.	✓

Table 2.1: Environment characteristics based on [Russell and Norvig, 1995].

The strategic aspects of the game make Hearthstone a competitive game with its own e-sport scene. Indeed, tournaments are regularly organized allowing the best players to compete for cash price. This competitive aspect makes the creation of AI even more interesting. Indeed, a powerful AI can allow players to discover innovative strategies.

Environment

A game environment is the context in which it lets its players evolve in. By relying on some useful environment characteristics described in [Russell and Norvig, 1995] (table 2.1 does a brief summary of these characteristics), the game can be analyzed and characterized.

Hearthstone offers a partially visible game state (it is said to be *inaccessible*) and lets two agents interact with it. This means it is not a perfect information game (like Chess or Go, for instance) and, therefore, offers a challenging problem for Artificial Intelligence (AI). Furthermore it offers a deterministic, static and discrete gameplay. Hearthstone being a turn-based game, it is episodic, which means that the state of a game does not change while an agent evaluates its possible actions before taking one. All these characteristics make Hearthstone a challenging game from an AI point of view. Indeed, the inaccessibility of a game state makes the evaluation of an action complex.

Simulation

Training and using an AI with the official game client is delicate. Directly interacting with the game by evaluating the screen in real time is a challenge on its own. Still this situation would have the advantage to offer the same information as a human player has while playing. Looking at the game logs would be another option. While being feasible, it does not let the AI interact with the game as it is fixed. This is why using a simulator allows to have good performance and to interact with a game through a dedicated API. To this day, there exist a few implementations but the most popular of them is probably

Name	Language	Discontinued
Fireplace	Python	
Heartbreaker	Python	×
Metastone	Java	×
Rosettastone	C++	
Sabberstone	C#	

Table 2.2: Most known Hearthstone simulators.

Fireplace. Written in python and available as a package. See Table 2.2 for a more exhaustive list.

2.1.2 State Space Search

In board games, it is common for an AI to express the problem of finding the best move to be played through a search. This is the process of finding a particular state starting from another one. In that space of states, each state offers a set of actions to be realized, leading to other states having their own properties. Visiting each one of these states is a problem that belongs to the NP problem class, this is why algorithms exist to find the goal state faster.

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an heuristic search algorithm used in decision making [Browne et al., 2012]. It explores the game tree. The root corresponds to the initial configuration of the game. Each node is a configuration and its children are the subsequent configurations. MCTS keeps in memory a tree that corresponds to the already explored nodes of the game tree. A leaf of this tree is either a final configuration (i.e., one of the players has won, or there is a draw), or a configuration from which no simulation has yet been launched (i.e., this part of the tree has not yet been explored). In each node, two numbers are stored; the number of winning simulations and the total number of simulations that passed through that node. Each step of the algorithm is composed of four phases: selection, expansion, simulation and backpropagation (see Figure 2.2).

Selection From the root, children are successively selected until a leaf is reached. In this phase, the choice of children is guided by a trade-off between exploitation (a child that has been shown to be promising) and exploration (another child, who looks less promising but that could lead to better results latter on).

Expansion If the selected node is not a leaf, create all child nodes and choose one of the children.

Simulation Next, it simulates an execution of a random game from this child, until reaching a final configuration.

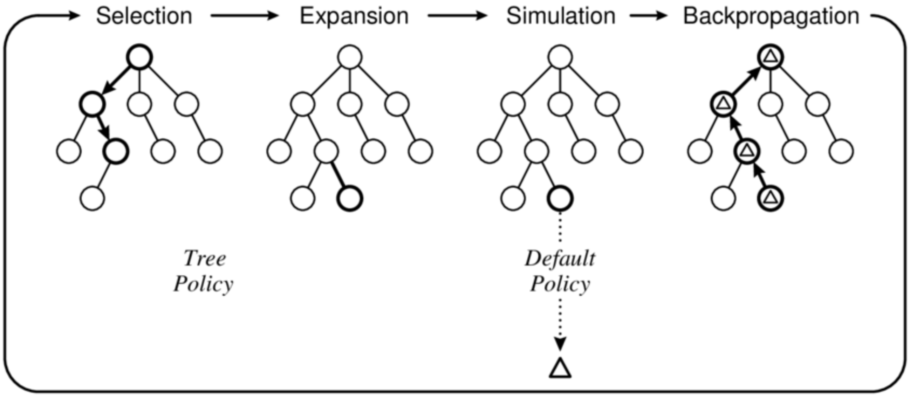


Figure 2.2: MCTS steps illustration [Browne et al., 2012]

Backpropagation Finally, MCTS uses the result of the random game and update the branch information from the child node to the root.

Exploration and exploitation During the selection phase, it is not easy to find a good compromise between exploiting the choices that look promising and exploring the nodes from which few simulations have been performed. To do this, a variant of the Upper Confidence Bound-1 (UCB-1) formula [Auer et al., 2002] is used, which is called Upper Confidence Bound 1 applied to Trees (UCT) [Kocsis and Szepesvári, 2006]. The chosen child is the one that maximizes

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}, \tag{2.1}$$

where w_i is the number of wins for the node considered after the i^{th} move, n_i is the number of times the node i was visited, N_i is the number of times the parent node of i has been visited and c is the exploration parameter, usually chosen empirically.

In Equation 2.1, the first part stands for the exploitation ($\frac{w}{n}$ is high for successors that have a lot of success), while the second stands for the exploration ($c\sqrt{\frac{\ln N}{n}}$ is large for successors that were not involved in a lot of simulations).

2.2 Machine Learning & Deep Learning

The search methods presented in Section 2.1 are often coupled with techniques from machine learning. Machine Learning is an AI field of study that relies on mathematical and statistical approaches to give a computer the ability to *learn*. In order to do so, lots of data is necessary to train a model, which can be thought of as a function mapping inputs (i.e., instances) to outputs (i.e., targets). This set of data where each value is associated with a variable (or attribute) and an observation is called a dataset. Once that process is done, if correctly trained, the model will have the capacity to make predictions on new instances it has never seen before. Machine Learning is constituted of several

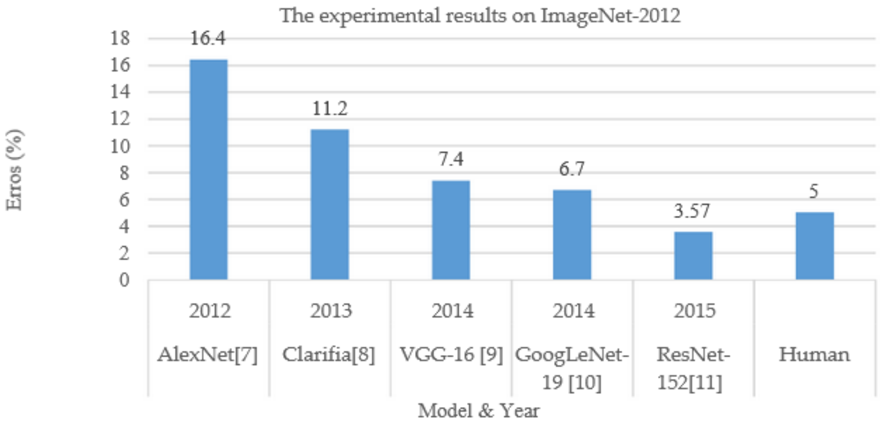


Figure 2.3: Comparison of the error rate of various classification models with respect to humans on the ImageNet dataset, from [Alom et al., 2019].

sub-categories, Deeplearning is one of them. It includes algorithms that are called Neural Network (NN).

2.2.1 Classification Task

Classification is a task in which data must be labelled. Typically, an observation made must be related to a category which it belongs to. To do this, it is common today to use a machine learning algorithm that attempts to find the correct class for any given observation. With the rise of these algorithms in recent years, classification automation has taken a big leap forward.

One interesting problem of recent years that deserves to be considered is image classification. With today's machine learning methods and computing resources, it has been possible to beat humans in many applications. Figure 2.3 shows that some classifiers perform really well on a complex image classification problem. As explained in [Lu and Weng, 2007], there are a lot of algorithms capable of classifying images in an efficient way.

2.2.2 Subdivision of the Machine Learning Field

Machine learning is generally divided into three subcategories.

Supervised Learning

Given a dataset of instances, each one matching its desired target, these algorithms have to find a function that approximates the mapping between these two. In the optimal situation, this function is able to find the correct target for a unseen instance. For example, given a dataset where each instance corresponds to the characteristics of a dog matching its breed, it is possible to find a function that can predict the breed from the characteristics of an unseen dog instance.

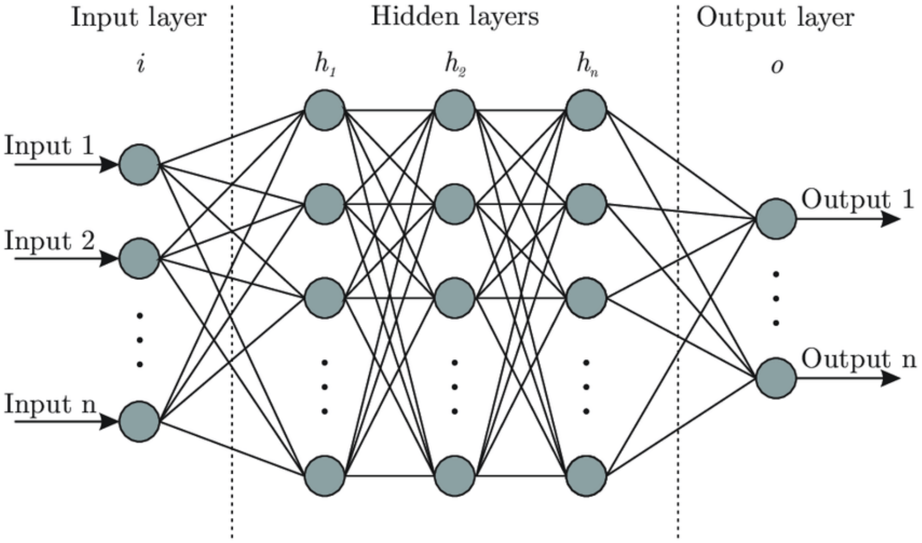


Figure 2.4: Neural Network representation, from [Bre et al., 2018]

Unsupervised Learning

Unlike for supervised learning algorithms, the instances here do not have any target. When training such models, the learning phase is all about finding patterns in the inputs. The best known algorithms are clustering algorithms. These aim to divide a set of data into different homogeneous clusters, so that the data in each subset share common characteristics, which most often correspond to proximity criteria that are defined by introducing distance measures.

Reinforcement Learning

Reinforcement learning consists, for an autonomous agent, in learning the actions to be taken, based on experiences, in order to optimise a quantitative reward over time. Among the first reinforcement learning algorithms, there are Temporal difference learning (TD-learning) [Sutton, 1988] and Q-learning [Watkins and Dayan, 1992].

2.2.3 Neural Networks

Neural networks are a set of machine learning methods. They are generally associated with supervised learning since they tend to learn based on labelled dataset. These methods are called neural networks because they losely mimic the structure and inner working of a human brain. Figure 2.4 gives a visual representation of a Neural Network structure.

Description

An artificial Neural Network is composed of l layers of neurons. The first one is the input, the last one is the output, and in between there are the hidden layers. Each layer constitutes a level of abstraction, the more there are, the more

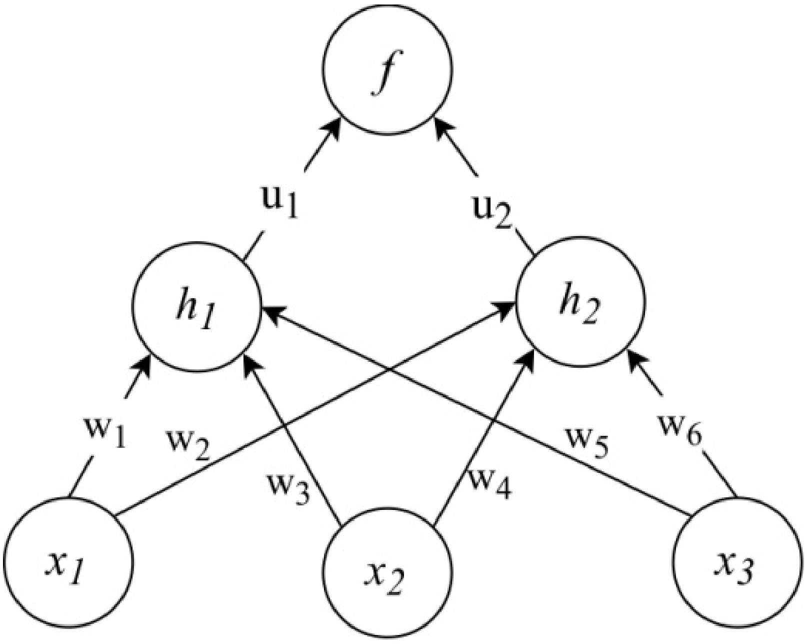


Figure 2.5: Forward propagation illustration, from stackexchange.com[2]

complex the Neural Network becomes. Training a neural network generally takes a lot of time due to the enormous amount of calculation needed. The training is composed of the following steps.

Initialisation The neurons in each layer have a connection to some, or all, neurons in the next layer. For each one of these connections, there exists a specific weight. These weights determine the output of any given input. Each connection has its weight randomly initialized at the start of the learning process.

Forward Propagation This step consists of forwarding an input through the network. An instance from the dataset is taken, passed through the network, and the resulted prediction is then compared with the target.

For instance, a forward propagation for the Neural Network from figure 2.5 would be calculated as follow:

- For each h_i, sum the respective weights time inputs. For instance, $h_1 = x_1 * w_1 + x_2 * w_3 + x_3 * w_5$.
- Then, apply an activation function (see Figure 2.6) to each h_i to get its output.

[2] https://stats.stackexchange.com/questions/295251/forward-propagation-calculation-for-single-layer-neural-network (26/03/21)

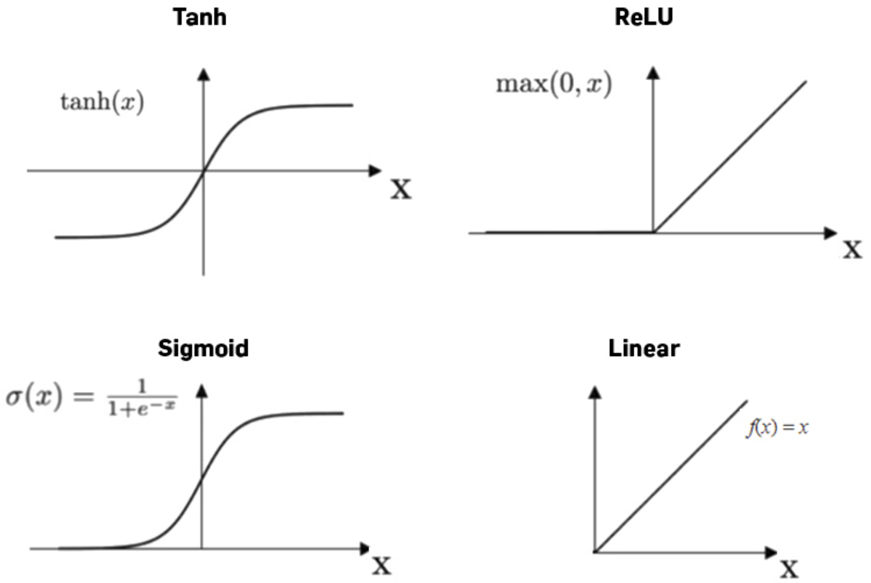


Figure 2.6: Activation functions example, from paperspace.com[3]

- The final output f_{out} is then calculated based on the weighted input $f_{in} = h_1 * u_1 + h_2 * u_2$.

Backpropagation Backpropagation minimises the error E, which is the error between the target and what the forward propagation step outputs for an instance.

In order to do so, a Stochastic Gradient Descent (SGD) is realized to update each weight that contributed to the result. SGD is an iterative gradient descent method used for the minimisation of an objective function. Once done, the training goes back to the forward propagation step.

2.2.4 Convolutional Neural Network

Convolutional Neural Network (CNN) [Indolia et al., 2018] are powerful Neural Networks commonly used for image recognition. They automatically assign to each input image a label corresponding to its class. The Convolutional Neural Network architecture has two parts (Figure 2.7 shows a classic architecture for a CNN).

The first one is the convolutional part whose objective is to extract specific characteristics from each image. This part is composed of three types of layers. The first is the convolution layer, which allows to identify the presence of a set of features in the images received as instances. To do this, convolution filtering are performed. Filters (or convolution kernels) are applied iteratively. They scan the data to extract attributes in order to build a feature map, which tells

[3] https://docs.paperspace.com/machine-learning/wiki/activation-function (12/05/2021)

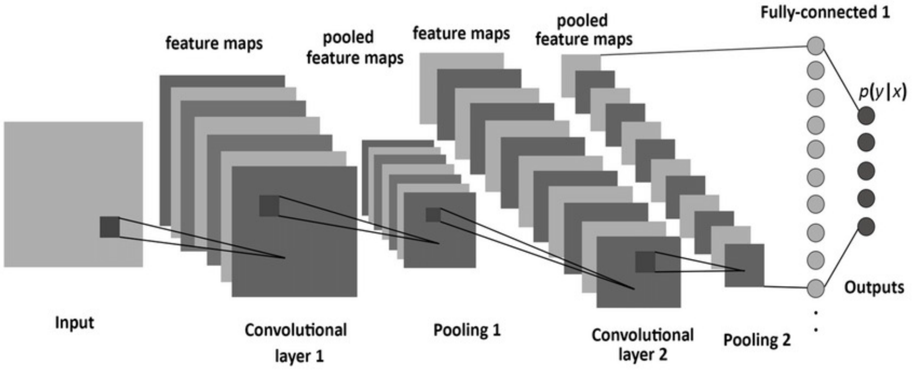


Figure 2.7: Classical architecture of a CNN from [Albelwi and Mahmood, 2017]

where the features are located in the image: the higher the value, the more the corresponding place in the image looks like the feature. The second layer is the pooling layer. It receives as input several feature maps, and applies to each of them the pooling operation. This consists in reducing the size of the images, while preserving their important characteristics. The output is the same number of feature maps as the input, but they are much smaller. The last layer is the ReLu correction layer (see Figure 2.6). Its purpose is to replace all the negative values received as inputs by zeros. It plays the role of an activation function.

The second part of the CNN is the classification part. It consists of fully connected layers and its role is to combine the feature maps in order to classify the image.

2.3 Deep Learning for Board Games

In 1997, a supercomputer called Deep Blue is the first machine to beat the chess world champion Garry Kasparov. Recently, AI has known a lightning advance thanks to AlphaGo, which proved to be capable of beating the world champion of go, something that was thought of to be unachievable before. The inner working of AlphaGo heavily relies on Deep Learning and its successor, AlphaZero, has been adapted to a multitude of other games, Hearthstone being no exception.

2.3.1 AlphaZero

In October 2015, a British company called DeepMind created AlphaGo [Silver et al., 2016], an AI agent that uses reinforcement learning to play the game of go. It became the first AI to beat a professional player. The AI combines machine learning and graph traversal technologies, like explained earlier (see Section 2.1.2). Since then, this algorithm has been further improved over the following versions. AlphaGo Zero [Silver et al., 2017], in October 2017, reached better performance by playing only against itself. This has been followed by a more generic version, AlphaZero [Silver et al., 2018], which is capable of winning not only in go, but also in chess and shōgi (japanese version of chess).

AlphaZero Algorithm

The AlphaZero algorithm takes place in 3 phases [Silver et al., 2018]. The first one consists in playing a certain number of games against itself, the second one retrains the model and the last one evaluates the model.

First Step During this step, AlphaZero creates its training set. To do this, the algorithm uses MCTS of Section 2.1.2. At each move, the following information is stored: the game state, the search probabilities (from the MCTS) and the winner (+1 if the player won, -1 otherwise).

Second Step At this stage, the algorithm retrains the Neural Network to optimise its weights. It then samples a mini-batch of positions with the information (state, search probabilities and winner) stored at the previous step and retrains the Neural Network from these positions. The instances correspond to the game states. The loss function compares the prediction of the Neural Network with the search probabilities and the actual winner.

Third Step This last step consists of evaluating the network. It is done by comparing the latest Neural Network and the current best one. Several games are played between these two networks. Each player uses the MCTS to select their move and their respective Neural Network to evaluate the leaves. To become the new best model, the latest Neural Network must win 55% of the games.

Deep Learning Heuristic Function

As mentioned in Section 2.1.2, standards MCTS uses the Upper Confidence Bound 1 applied to Trees (UCT) formula to find a good compromise between exploration and exploitation. AlphaZero uses a version called Polynomial Upper Confidence Trees (PUCT) [Czech et al., 2020]. PUCT is defined as

$$PUCT(s,a) = Q(s,a) + U(s,a), \tag{2.2}$$

where Q is the mean action value. It corresponds to the average game result of the current simulation that considers action a. In Formula 2.2, $U(s,a) = c_{puct} P(s,a) \frac{\sum_b N(s,b)}{1+N(s,a)}$, where P is the prior probabilities given by the Neural Network and N is number of times this action was taken during current simulations.

All possible actions b are added in the numerator, and the number of visits for the considered action a is in the denominator. Therefore, the less this action was tried, the larger U is. This encourages exploration. When *c_puct* increases, it gives more weight to this exploration term. When it decreases, the exploitation of the expected result Q is given more weight.

Neural Network Architecture

As shown in Figure 2.8, the Neural Network consists of several residual layers [He et al., 2016] that splits into two completely connected heads. One that gives the probability distribution on all actions and one that gives the winning likelihood value for the current state (+1 or -1). An instance corresponds to a game state.

2.4 Explainable Artificial Intelligence

Explainable Artificial Intelligence (XAI) tries to answer different problems that arise from the use of AI. It is necessary (even required sometimes [Došilović et al., 2018]) for certain domains to understand the predictions of a model [Bibal and Frénay, 2016]. An example often mentioned to better understand this need is that of a doctor using a model that gives diagnostics about patients health. If it tells him a certain patient has the flu, the doctor will surely need an overview of how it came to that conclusion before taking any initiative to cure the patient.

Explainable Artificial Intelligence (XAI) is the set of techniques that aim to provide a humanly understandable explanation of a model. When speaking about XAI, two related terms often comes up: explainability and interpretability. Even though they might seem to be synonym and they do not have any fixed definition, it is important to distinct them.

2.4.1 Definitions

Interpretability is about understanding the reasoning behind a model by understanding its inner workings. An interpretable model is also said to be transparent. In opposition to this, explainability is about understanding a black box model, which is said to be uninterpretable by definition. An explanation is therefore an approximation of the internal functioning of a model. Most of the time, these explanations are said to be post-hoc (meaning they are obtained after the training of the model).

Interpretability

Originally, using a transparent method was the approach used to keep a high level of interpretability when dealing with AI. Indeed, there is no better explanations for a model than the model itself. But even transparent models are obviously becoming much more complex than they originally were and it is becoming hard to find relevant explanations that can be understood by humans. Also, transparency reduces the number of algorithms one can choose when elaborating an AI even though some of these nontransparent algorithms would achieve better performance. Indeed, integrated approaches and performance have conflicting objectives [Edwards and Veale, 2017]. Also, this approach can be costly if one already has a working model. This is why XAI methods were created.

Explainability

By contrast, finding an explanation to a model is often achieved after the training, therefore such techniques are said to be post-hoc. A post-hoc approach

[4] https://nikcheerla.github.io/deeplearningschool/2018/01/01/AlphaZero-Explained/ (26/05/2021)

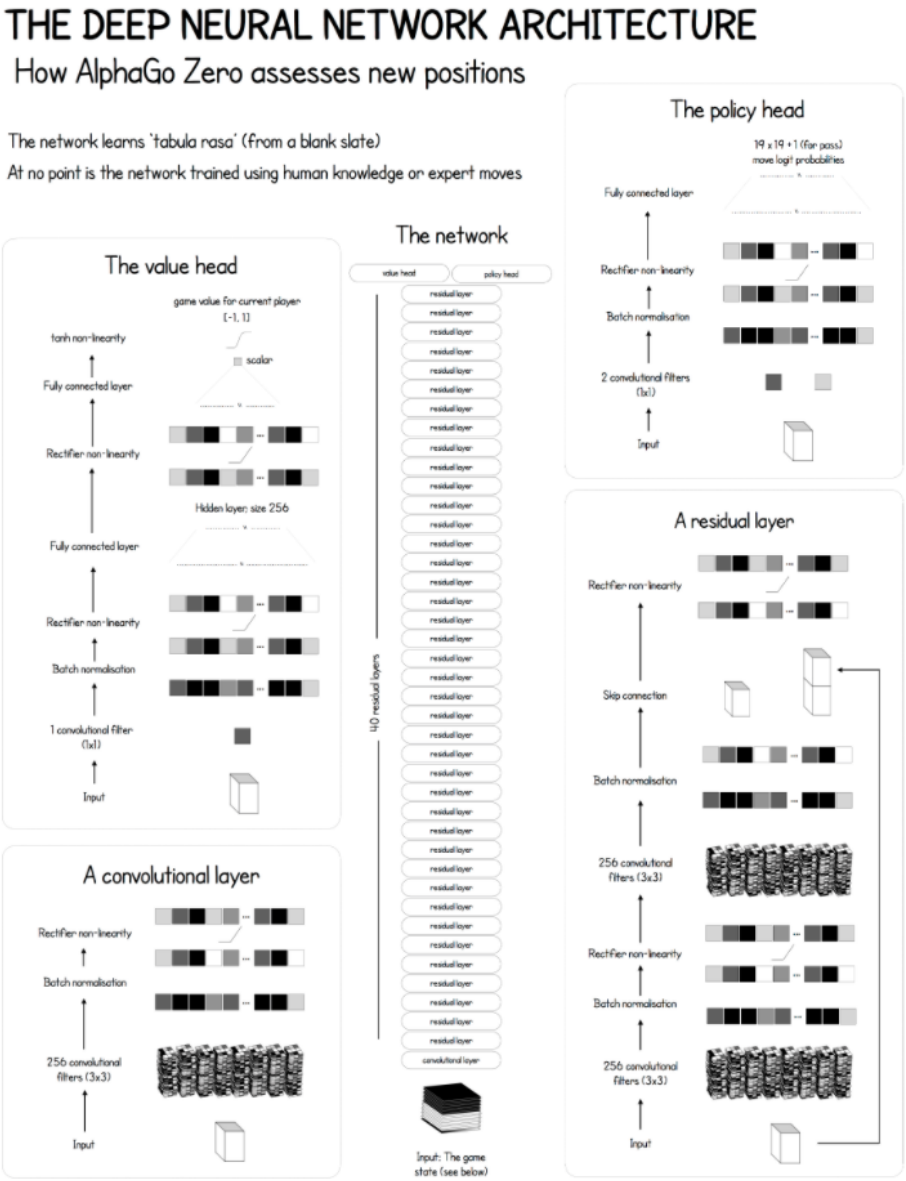


Figure 2.8: Alphazero Neural Network architecture, from https://nikcheerla.github.io[4]

would take an already trained model and try to extract information out of it. This way, one can keep a black-box model as it is, nor does he have to reduce his choice of available algorithms, since most of these techniques are said to be model agnostic, which means they can work regardless of the model nature. This kind of method has the advantage of letting the user keep a black-box model and the performance that comes with it. However, they generally keep the scope of an explanation to a local input. They do not permit the user to learn the inner mechanisms of a model [Montavon et al., 2018].

2.5 Summary

This chapter explains the application of AI in the field of board games and in classification tasks such as image classification. A brief introduction to machine learning and deep learning is also detailed. Finally, Section 2.4 explains the field of Explainable Artificial Intelligence and why it is a necessity nowadays. This allows to understand the next part of this work, which is the different XAI methods that constitute its state of the art.

Chapter 3

Understanding an Artificial Intelligence

In order to answer this master thesis question, it is necessary to explore the different techniques that constitute the state of the art for the Explainable Artificial Intelligence field. Before exploring the post-hoc methods, a brief summary for the integrated methods is done.

3.1 Integrated Methods

This section discusses multiple ways of having a transparent AI, and therefore, making it interpretable.

3.1.1 Purely Interpretable

Using this approach, one is limited to choosing in the set of models that are completely transparent. For instance, decision tree algorithms are considered to be fully transparent since they appear to be exclusively a set of *if-then* condition statements. However, these models often perform poorly compared to more complicated ones and increasing the size of such models does make them more opaque [Došilović et al., 2018].

3.1.2 Hybrid

Hybrid models combine the use of transparent methods with black box ones. This way, it is therefore possible to do the right trade-off between performance and interpretability. For instance, in [Piltaver et al., 2014], they train hybrid tree classifiers where certain leafs are replaced with black box classifiers, therefore trading transparency for better performance.

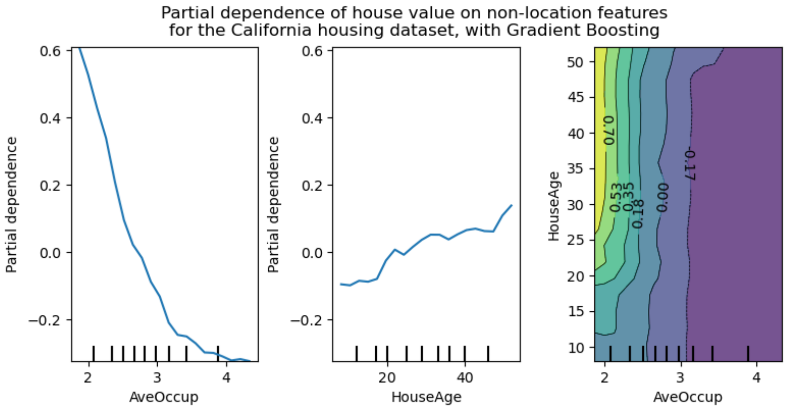


Figure 3.1: Partial dependence Plot, from scikit-learn.org[1]

3.2 Post-hoc Methods

This part focuses on the post-hoc methods that make an uninterpretable model explainable. The structure of the book [Molnar, 2020] has been adapted since it reviews most of the available techniques as for the time of the writing of this work.

3.2.1 Partial Dependence Plot

Partial Dependence Plot (PDP) [Friedman, 2001] is an explanation method that shows the impact of one or two features on a machine learning model prediction. The partial dependence function is defined as

$$\hat{f}_{x_S}(x_S) = E_{x_C}\left[\hat{f}(x_S, x_C)\right] = \int \hat{f}(x_S, x_C) d\mathbb{P}(x_C), \tag{3.1}$$

where x_s are the features relative to the partial dependence function and x_c are the other features. x_s and x_c put together contain all the features. What partial dependence does is marginalising the model's output over the distribution of the features in the set C. By doing so, the function only depends on the features in s.

$\hat{f}_{x_S}$ can be estimated with the Monte Carlo method,

$$\hat{f}_{x_S}(x_S) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)}). \tag{3.2}$$

For given values in the set S, 3.2 gives the average marginal effect on the prediction. In Equation 3.2, $x_C^{(i)}$ are the irrelevant features and n is the number of instances in the dataset. Figure 3.1 shows the partial dependence of house value on non-location features for the California housing dataset.

In the PDP method, features in C must not be correlated with the features in S. Otherwise, the averages calculated for the Partial Dependence Plot will come out with data points that are improbable or impossible.

[1] https://scikit-learn.org/stable/modules/partial_dependence.html (07/05/21)

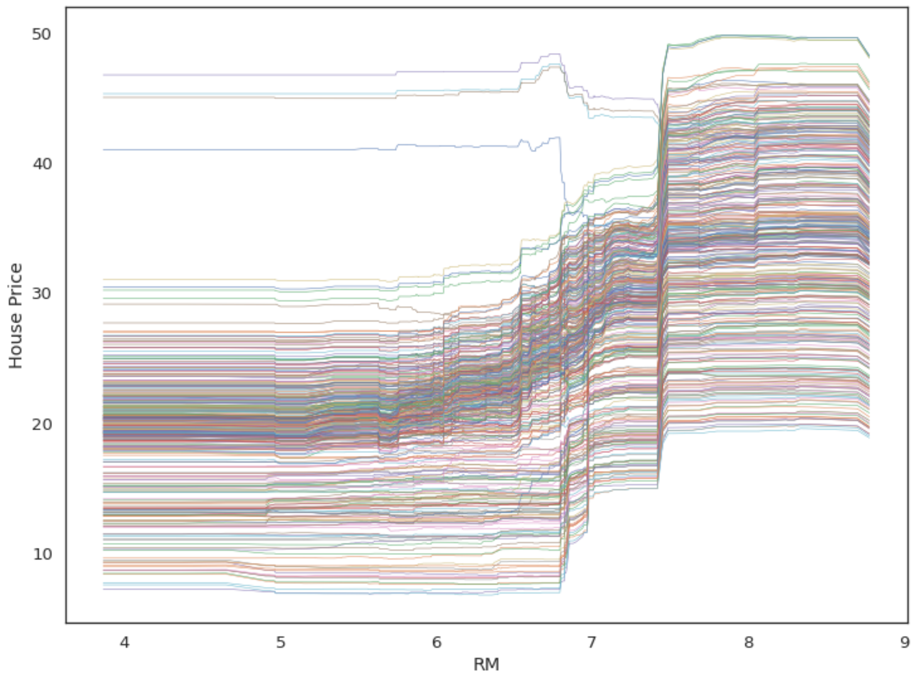


Figure 3.2: ICE plot, from towardsdatascience.com[2]

3.2.2 Individual Conditional Expectation

Individual Conditional Expectation (ICE) plots [Goldstein et al., 2015] is the equivalent of PDP but for an individual instance. A PDP is the average of the lines of an ICE plot.

Concretely, in ICE plots, for each instance in the set $\{(x_S^{(i)}, x_C^{(i)})\}_{i=1}^N$, the curve corresponding to the function $\hat{f}_S^{(i)}$ is plotted against $x_S^{(i)}$, while $x_C^{(i)}$ remains fixed (see Figure 3.2).

3.2.3 Accumulated Local Effects Plot

As explained before in Section 3.2.1, to use the PDP method, the features must not be correlated. If this is not the case, then the result is not reliable. To overcome this, there is a method called M-plots [Apley and Zhu, 2020]. The idea is to average over the conditional distribution of the two correlated features. However, the solution is not optimal since it is an estimation of their combined effect.

The Accumulated Local Effects (ALE) [Apley and Zhu, 2020] method is a solution for correlated features. Instead of calculating the average, it calculates the difference between the features. It avoids to mix the effect of a feature with the effects of correlated features. As explained in Equation 3.1, Partial Dependence Plots average the predictions over the marginal distribution while

[2] https://towardsdatascience.com/how-to-explain-and-affect-individual-decisions-with-ice-curves-1-2-f39fd751546f (07/05/21)

Location	Size	Prediction
good	big	300.000
good	small	200.000
bad	big	250.000
bad	small	150.000

(a) No interaction.

Location	Size	Prediction
good	big	400.000
good	small	200.000
bad	big	250.000
bad	small	150.000

(b) Interaction.

Figure 3.3: Feature interaction example adapted from [Molnar, 2020].

M-plots average the predictions over the conditional distribution. The equation becomes

$$\begin{aligned}\hat{f}_{x_S,M}(x_S) =& E_{X_C|X_S}\left[\hat{f}(X_S, X_C)|X_S = x_s\right] \\ =& \int_{x_C} \hat{f}(x_S, x_C)\mathbb{P}(x_C|x_S)dx_C. \end{aligned} \tag{3.3}$$

ALE plots average the changes in the predictions and accumulate them. The formula is

$$\begin{aligned}\hat{f}_{x_S,ALE}(x_S) =& \int_{z_{0,1}}^{x_S} E_{X_C|X_S}\left[\hat{f}^S(X_s, X_c)|X_S = z_S\right] dz_S - \text{constant} \\ =& \int_{z_{0,1}}^{x_S} \int_{x_C} \hat{f}^S(z_s, x_c)\mathbb{P}(x_C|z_S)dx_C dz_S - \text{constant}, \end{aligned} \tag{3.4}$$

where

$$\hat{f}^S(x_s, x_c) = \frac{\delta \hat{f}(x_S, x_C)}{\delta x_S}. \tag{3.5}$$

3.2.4 Feature Interaction

To estimate the interaction strength between two features, it is possible to measure how much the variation of a prediction depends on the interaction of these two features. This measurement is called H-Stastic [Friedman et al., 2008].

To better understand feature interaction, a great example is given in [Molnar, 2020]. Looking at Figure 3.3, there is the output of two given models that predict the price of a house based on two inputs: its location and its size.

In Table 3.3a, a deconstruction can be done as such: a constant term (150.000) and an effect for each of the features: +100.000 if the house is big but +0 if small, and finally +50.000 if in a good location while +0 if not. In other words, having a big size increase the price by 100.000, while having a good location increases it by 50.000.

However, the above is not enough to explain Table 3.3b. For this, a new term is needed. Introducing one that considers both the size and the location of the house, such that when it meets the two conditions, the price increases by 100.000 can explain the price on the first line of the table. This new term is called an interaction feature.

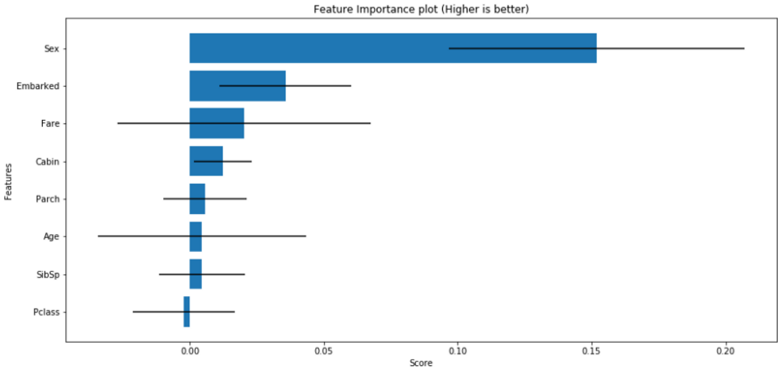


Figure 3.4: Permutation feature importance scores on Titanic dataset with Random forest model, from medium.com[3]

3.2.5 Permutation Feature Importance

Permutation feature importance, introduced by Rudin, Fisher and Dominici [Fisher et al., 2018], is a method that aims to measure the importance of a feature. To do that, it computes the increase in a model prediction's error when permuting the feature. If the permutation of that feature increases the error, then it is an important one. Otherwise it means that the feature is not, or at least less, important. For instance, Figure 3.4 shows the permutation feature importance scores on the Titanic dataset and a *random forest* model.

The algorithm described in [Fisher et al., 2018] proceeds in three steps. It takes four parameters as inputs. The trained model f, the feature matrix X, the target vector y and then the measure $L(y, f)$ that corresponds to the error. The first step of the algorithm is to estimate the original error defined as $e^{orig} = L(y, f(X))$. After, for each feature $i = 1, ..., p$ it computes X^{perm} by permuting feature i in the matrix X, then based on the prediction of the permuted matrix X^{perm} it estimates the error $e^{perm} = L(Y, f(X^{perm}))$, finally it computes permutation feature importance with either $FI^j = e^{perm}/e^{orig}$ or alternatively $FI^j = e^{perm} - e^{orig}$. The last step of the algorithm is sorting features by descending FI.

3.2.6 Saliency Map

It is quite complicated to understand the choice of a model in the context of image recognition. Saliency map is an explanation method used to interpret the predictions of a Convolutional Neural Network (CNN) [Simonyan et al., 2013]. It shows (see 3.5) how much a pixel contributed in the computation of the score (output of the CNN).

A CNN gives the class score $S_c(I)$ for an image I belonging to class c. $S_c(I)$

[3] https://medium.com/@mudassirkhan19/use-this-for-feature-importance-5a8a068a3244 (07/05/21)

[4] https://usmanr149.github.io/urmlblog/cnn/2020/05/01/Salincy-Maps.html (22/03/21)

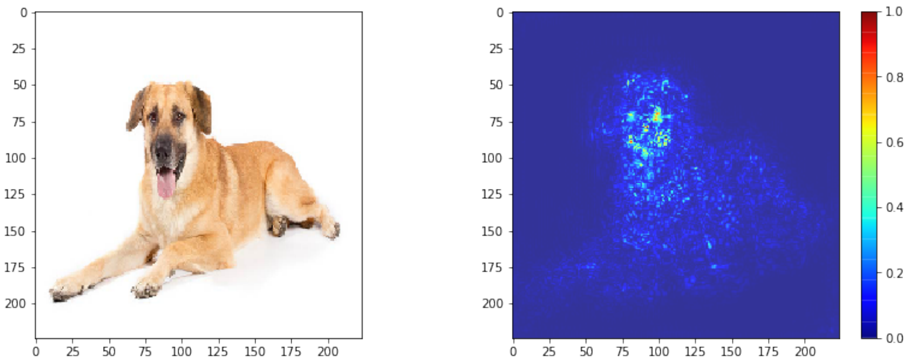


Figure 3.5: Saliency map, from usmanr149.github.io[4]

can be approximated with the first-order Taylor expansion such as,

$$S_x(I) \approx w^T I + b \tag{3.6}$$

where b is the bias and w is the derivative of S_c with respect to the image I,

$$w = \frac{\delta S_c}{\delta I}. \tag{3.7}$$

As mentioned in [Simonyan et al., 2013], the magnitude of w indicates what pixels need to be changed the least to affect the class score the most.

Recently, in [Adebayo et al., 2018], it has been proposed a new technique to evaluate the efficiency of saliency maps in giving human explanations they can understand. This article highlights the fact "that visual inspection is a poor guide in judging whether an explanation is sensitive to the underlying model and data". Furthermore, this method, while being interesting when working with images, is not applicable otherwise.

3.2.7 LIME

Local Interpretable Model-Agnostic Explanations (LIME) is a technique that aims to give local explanation for a given prediction. Its main goal is to give a local approximation to a black box model using an interpretable model so that he or she can trust the output prediction of a model.

What LIME does is solving

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \; \mathcal{L}(f, g, \pi_x) + \Omega(g), \tag{3.8}$$

where G is the set of potentially interpretable models, $\Omega(g)$ is the complexity of g, $\pi_x(z)$ is the proximity measure of z regarding to x. $f : \mathbb{R}^d \to \mathbb{R}$ is the model that needs explanations, $\mathcal{L}(f, g, \pi_x)$ is the measure of how unfaithful g is for approximating f in the locality defined by π_x.

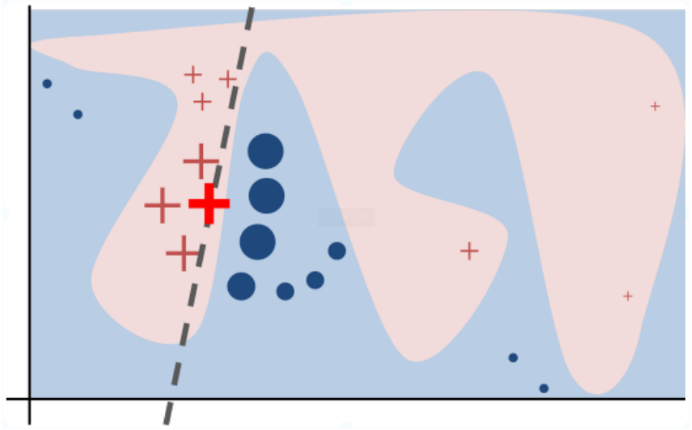


Figure 3.6: Faithful local explanation using LIME [Ribeiro et al., 2016].

It is therefore simply a matter of minimizing $\mathcal{L}$, while keeping the complexity of g low enough for the explanation to be understandable by a human [Ribeiro et al., 2016].

Figure 3.6 gives a better understanding of what LIME does. An explanation for the Figure is given in [Ribeiro et al., 2016]: "The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful.".

3.2.8 Anchors

Anchors, from the same authors behind LIME, are model-agnostic explanations based on if-then rules. This method has the benefit of being intuitive and easy to understand. A defined anchor "anchors" (hence the name) its prediction locally such that for any instance on which the anchor stands, the prediction will (almost) be the same [Ribeiro et al., 2018]. Figure 3.7 is an Anchor example while a comparison between LIME and a corresponding anchor explanation is

Feature	Value
Age	20
Sex	Female
Class	First
Ticket Price	300$
...	...
Survived	**True**

(a) Instance example.

IF SEX = female,
AND Class = first,
THEN PREDICT Survived = true,
WITH PRECISION 97%,
AND COVERAGE 15%

(b) Corresponding *if-then* explanation.

Figure 3.7: Anchors example from [Molnar, 2020].

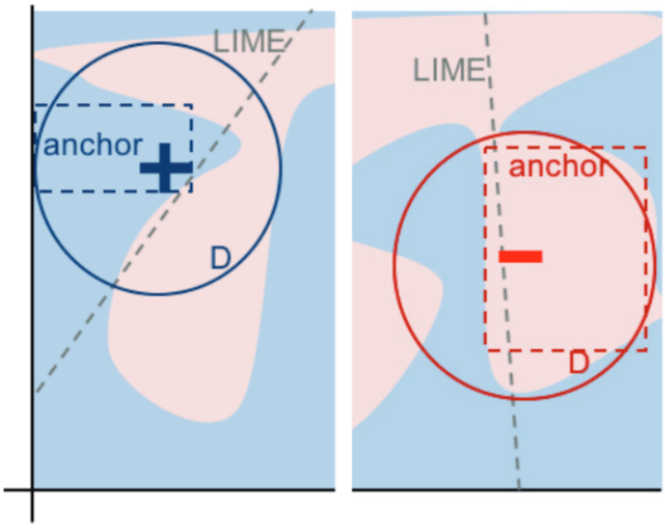


Figure 3.8: LIME approximation for an instance compared to its Anchor explanation, from [Ribeiro et al., 2018]

given in Figure 3.8.

Given the example in Table 3.7, [Molnar, 2020] explains that anchors provide essential insights into a model's prediction and its reasoning. "The result shows which attributes were taken into account by the model, which in this case, is the female sex and first class. Humans, being paramount for correctness, can use this rule to validate the model's behavior. The anchor additionally tells that it applies to 15% of perturbation space's instances. In those cases the explanation is 97% accurate, meaning the displayed predicates are almost exclusively responsible for the predicted outcome." [Molnar, 2020].

In [Ribeiro et al., 2018], they argue the usefulness of anchors explanations on a multitude of domains and models: Text classification, structured prediction, tabular classification, image classification and visual question answering.

More formally, the definition of an anchor is

$$\mathbb{E}_{\mathcal{D}_x(z|A)}[1_{f(x)=f(z)}] \geq \tau, A(x) = 1, \tag{3.9}$$

wherein $x \in X$ represents the instance being explained, f is the model to be explained, A, is a set of predicates such that $A(x) = 1$ if all its feature predicates are true for instance x. $D_x(\cdot|A)$ is the conditional distribution around x, when rule A applies, and finally, $0 \leq \tau \leq 1$ is a specified threshold.

3.2.9 SHAP

SHAP (SHapley Additive exPlanations) by Lundberg and Lee is a game theoretic approach aiming to explain the output of any machine learning model [Lundberg and Lee, 2017].

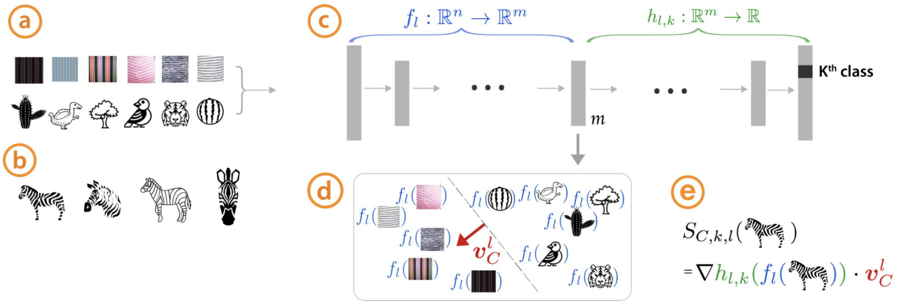

a
b
c
$f_l : \mathbb{R}^n \to \mathbb{R}^m$
$h_{l,k} : \mathbb{R}^m \to \mathbb{R}$
K^th class
m
d
$f_l(\ )$
v_C^l
e
$S_{C,k,l}(\ )$
$= \nabla h_{l,k}(f_l(\ )) \cdot v_C^l$

Figure 3.9: TCAV illustration from [Kim et al., 2018].

This method is based on Shapley Values introduced by L.S. Shapley in 1953 [Shapley, 1953]. In game theory (for cooperative games), the Shapley value gives a fair distribution of payoffs to the players. The players collaborate to obtain a certain gain. The problem is then the distribution of this gain among the different players. Shapley proposed a "fair" distribution of the payoffs of the coalition of n players.

The idea behind SHAP, is to compute the contribution of each feature in the prediction of an instance x.

SHAP defines the explanation as

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j, \tag{3.10}$$

where g is the explanation model, $z' \in \{0,1\}^M$ is the coalition vector, M is the maximum coalition size and $\phi_j \in \mathbb{R}$ is the feature attribution for a feature j, the Shapley values. In the coalition vector, 1 means that the corresponding feature value is "present" and 0 that it is "absent". To compute Shapley values, it simulates that only some features values are playing and some are not. The explanation can be simplified with a "trick" to a linear model and it is done to simplify the computation of ϕ's. An instance x has a coalition vector x' with only 1's (all features are present):

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j. \tag{3.11}$$

3.2.10 Testing with CAV

As introduced in [Kim et al., 2018], Testing with CAV (TCAV) provides an interpretation of a Neural Network internal state in terms of human-friendly concepts, which are called High Level Concept (HLC). CAV stands for Concept Activation Vectors. Concretely, TCAV uses the directional derivative to quantify the sensitivity of the model to a certain HLC represented by a CAV. Figure 3.9 shows a schematic explanation of the method. The steps are detailed in the following section.

User-defined Concepts as Sets of Examples

The first step in this method is to define some HLCs. To do this, one must choose sets of examples, each representing one of the chosen concept or find independent datasets with the labelled concepts.

CAVs

The next step is to create the Concept Activation Vectors (CAV) in the space of activations of layer l. To find this vector, it is explained in [Kim et al., 2018] that "we consider the activations in layer l produced by input examples that in the concept set versus random examples [*sic*]. We then define a *concept activation vector* (or CAV) as the normal to a hyperplane separating examples without a concept and examples with a concept in the model's activations".

In a more mathematical way, with C representing the concept, one actually has to define a dataset containing the concept, let it be P_C, and another one made of random examples, N. A binary linear classifier has then to be trained to distinguish between the layer activations of the two sets: $\{f_l(x) : x \in P_C\}$ and $\{f_l(x) : x \in N\}$. The resulting $v_C^l \in \mathbb{R}^m$ is a linear CAV that is defined for the concept C [Kim et al., 2018].

Directional derivatives and Conceptual Sensitivity

Let $v_C^l \in \mathbb{R}^m$ be a CAV vector for concept C at layer l, and $f_1(x)$ the activation for input x for the very same layer, the sensitivity for class k, to concept C, can be computed as the directional derivative

$$\begin{aligned} S_{C,k,l}(x) &= \lim_{\epsilon \longrightarrow 0} \frac{h_{l,k}(f_1(x) + \epsilon v_C^l) - h_{l,k}(f_1(x))}{\epsilon} \\ &= \Delta h_{l,k}(f_1(x)) \cdot v_C^l \end{aligned} \tag{3.12}$$

where $h_{l,k} : \mathbb{R}^m \longrightarrow \mathbb{R}$ [Kim et al., 2018].

As explained in [Kim et al., 2018], this is because "CAV and directional derivatives permit the calculation of the sensitivity of machine learning predictions to changes towards the direction of a concept, at neural activation layer l".

Testing with CAV (TCAV)

The following formula allows to compute the TCAV score of a given concept,

$$\mathrm{TCAV}_{Q_{C,k,l}} = \frac{|\{x \in X_k : S_{C,k,l(x)} > 0\}|}{|X_k|}. \tag{3.13}$$

It can be defined as the ratio of inputs x from class k that have a positive directional derivative $S_{C,k,l}(x)$ towards the concept C at layer l given all inputs from that very same class k [Kim et al., 2018].

Statistical significance testing

The drawback of using only one TCAV is that it could be meaningless. Indeed, any randomly generated set, instead of P_C, would still produce a CAV, and therefore, would not be relevant at all. To prevent such pitfall, [Kim et al., 2018] proposes a statistical test approach. Instead of training a CAV once on a random set N, one could do it about 500 times. Therefore, if the TCAV scores behave consistently across the runs, it can be considered meaningful.

TCAV Extensions: Relative TCAV

Using CAVs to distinguish between similar concepts will not mean that these CAVs will be orthogonal between them. Indeed, related HLCs will likely overlaps. In [Kim et al., 2018], a solution for such comparison is proposed: relative TCAV. Taking two distinctive but alike concepts, C and D, training a classifier on $f_l(P_C)$ and $f_l(P_D)$ yields a vector $v^l_{C,D} \in \mathbb{R}^m$. That vector is defining a one dimensional space where the projection of any $f_l(x)$ tells to which concept x is more relevant.

3.3 Discussion

All the methods detailed in this chapter are what constitute the state of the art in the Explainable Artificial Intelligence domain. Although they are diverse and serve different purposes, not all of them are usable to answer this work question.

While methods relative to features perform well with few features, they become complicated to use when working with images and large feature space in general.

Developing further the idea behind Saliency maps would have been an interesting approach to the research question. However, they have received a lot of criticism lately [Adebayo et al., 2018] and most importantly, they are not usable with other types of networks than image recognition Neural Network.

Lime, SHAP and the Anchors methods are local explanations to a model prediction. This master thesis aims to extract global concepts learnt by a Neural Networks. Therefore, they may not be the most appropriate for the scope of this thesis.

The idea behind TCAV is appealing. Having a way of determining the direction of a concept given an activations space is interesting. However, it assumes that the concepts, to be identified, are known to the user and therefore, that he has a dataset to train the CAV on. Finding or creating datasets containing the concepts to extract may be a very tedious task. Also, it can be very hard, if not impossible, to be exhaustive. Therefore, the next chapter focuses on the problem that comes from the use of TCAV, and gives a method to extract the concepts from a Neural Network.

Chapter 4

Unsupervised Extraction of Concepts

Given the problem that TCAVs need concepts to be defined in advance, and the creation of their respective datasets, this chapter aims to automatically determine the concepts learnt by a Neural Network and to build a dataset of these concepts. Thus, this chapter describes an unsupervised method to extract concepts learnt by a Neural Network, Figure 4.1 illustrates the method.

4.1 Retrieving Activation Vectors

The first step of the proposed method is to obtain instances for the given Neural Network. These instances are passed through the network, and, for each one of them, their activation vector (i.e., their output) at a chosen layer are retrieved and saved. Choosing a layer cannot be automated and largely depends on the network architecture. Testing different layers is necessary since the method results differ depending on the chosen layer.

4.2 Extracting the Concepts

This step extracts the learnt concepts of the Neural Network. For this, the previously retrieved activation vectors from Section 4.1 are regrouped based on their similarities. A clustering is therefore done based on their position in the space that they form. It allows comparing their respective instances and to understand what links them together. A dataset is created for each cluster formed, which contains the instances corresponding to the activation vectors present in the cluster.

The datasets formed therefore contains instances with common concept, they are referred to as *concept datasets* in the following sections.

To perform the clustering, an unsupervised classification method, called K-Means [Schubert and Rousseeuw, 2019], is used. As it can be seen in Figure 4.2, the K-Means algorithm separates the data in k clusters. Given a set of

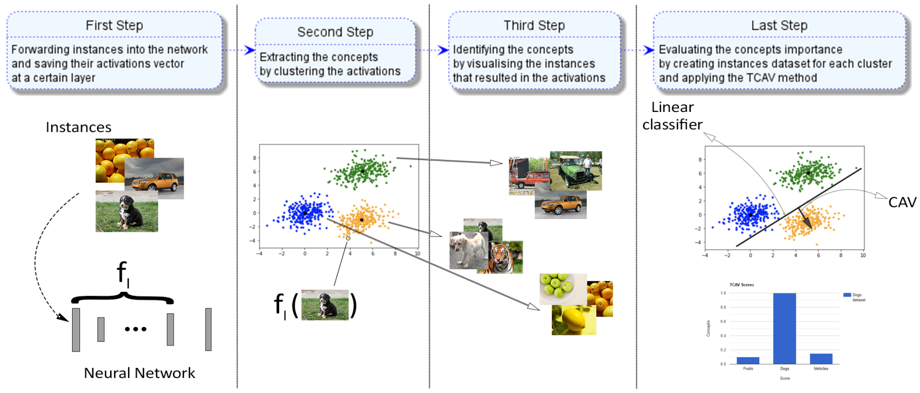

First Step
Forwarding instances into the network and saving their activations vector at a certain layer
Second Step
Extracting the concepts by clustering the activations
Third Step
Identifying the concepts by visualising the instances that resulted in the activations
Last Step
Evaluating the concepts importance by creating instances dataset for each cluster and applying the TCAV method
Instances
f_l
...
Neural Network
f_l ( )
Linear classifier
CAV

Figure 4.1: Illustration of the 4 different steps used to extract learnt concepts out of a Neural Network.

points $\{x_1, x_2, ..., x_n\}$, it partitions the n points into a chosen number k of sets $S = \{S_1, S_2, ..., S_k\}$ $(k \leq n)$ by minimising the distance between points within each partition,

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - \mu_i \|^2, \tag{4.1}$$

where μ_i is the mean of the samples in the cluster S_i.

4.3 Understanding the Concepts

When the concepts are extracted, the method identifies them. How it is done, widely depends on the nature of the instances. Methods to identify the concepts for two different types of instances are presented below.

4.3.1 Visualising Images

If instances are images, it is possible to visualise the images that resulted in a cluster of activation vectors in order to better understand what is the common concept between them. Visualising only one image does not help to identify what links multiple activation vectors together. However, processing all of them is tedious for a human. The proposed solution is to take a reasonable amount of instances per cluster (i.e., 3 or 4) and show them to the user. That way, it is possible to oversee the concept linking the image activation vectors.

A solution to find representative instances for a cluster is to perform a second clustering on its activation vectors. Visualising the instances corresponding to each center of the calculated sub-clusters allows having an overview of the overall cluster concept.

To perform the second clustering, another unsupervised classification method, called K-Medoids [Schubert and Rousseeuw, 2019], is used. It is a clustering algorithm more robust to outliers than the traditional K-Means [MacQueen

[0] https://stanford.edu/∼cpiech/cs221/handouts/kmeans.html (10/05/2021).

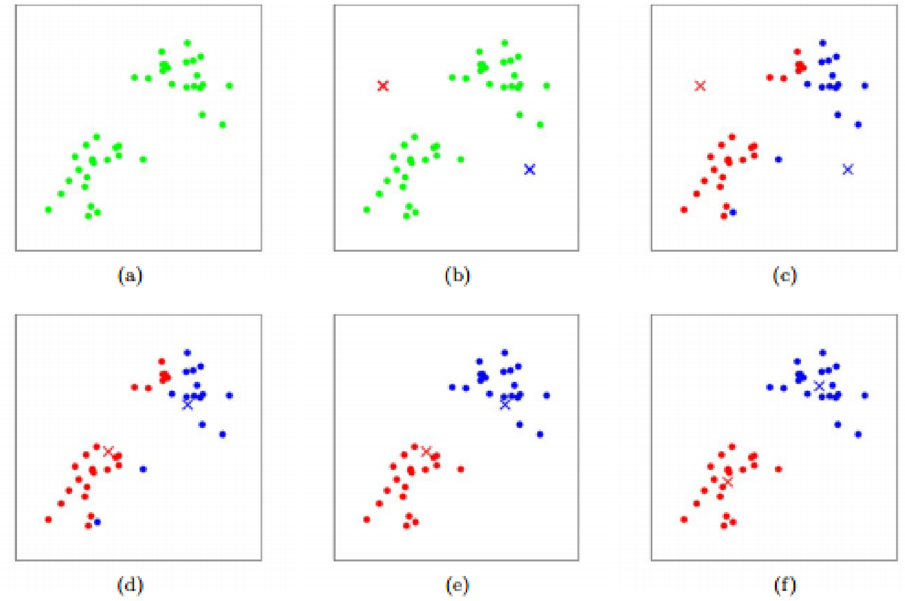


Figure 4.2: K-Means illustration taken from stanford.edu.

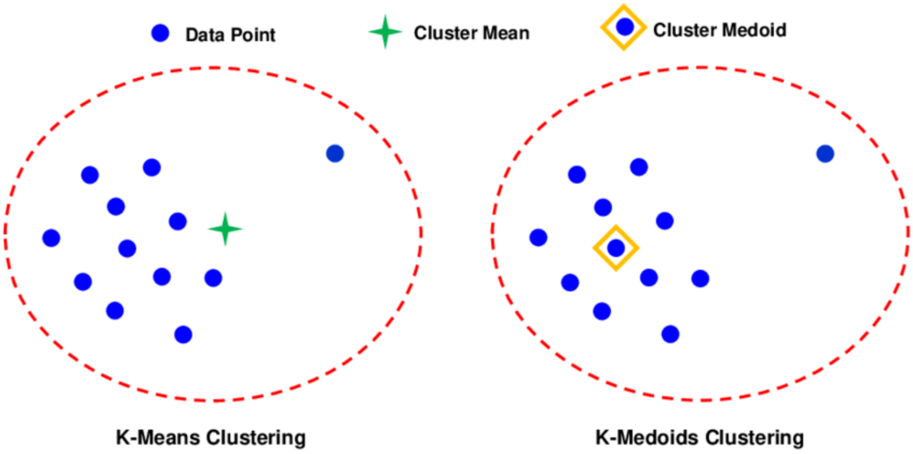


Figure 4.3: K-Means vs K-Medoids illustration taken from [Entezami et al., 2020].

et al., 1967] (see Figure 4.3 for a visual comparison of the two algorithms). Furthermore, K-Medoids gives the possibility to retrieve a cluster center that is an existing activation vector (i.e., the medoid), instead of the mean activation vector that is provided by K-Means.

The K-Medoids algorithm minimises the mean squared error, which is the distance between the points of the class and its medoid. The medoid is the most representative object of a cluster, for which the average dissimilarity with respect to all objects in the cluster is the smallest. The medoid is defined as

$$x_{medoid} = \arg\min_{y \in X} \sum_{i=1}^{n} d(y, x_i), \tag{4.2}$$

where X is a set of n instances $\{x_1, x_2, ..., x_n\}$ in a space and d is a distance function [Struyf et al., 1997].

4.3.2 Visualising Tabular Data

Tabular data instances cannot be viewed and understood as easily as images, they require another method to better understand the concept behind the datasets formed by the clustering in Section 4.2. Each instance in the datasets is a vector of features. To understand what links the instances together in one concept dataset, it is helpful to determine which features are common between them. These features are the ones that do not change too much between the different instances.

To sort out the features that do not change too much from the ones that do, the Shannon entropy [Shannon, 1948], which is a measure of disorder, is useful. Given a source X with n symbols, a symbol x_i having a probability $p_i = P(X = x_i)$ to appear, the entropy H is defined as

$$H(X) = -\sum_{i}^{n} p_i \log_2(p_i). \tag{4.3}$$

To omit the irrelevant features (i.e, those that change too much) and have a clearer representation of each cluster, a *mask* is created. The mask is a way to visualise the features that characterise a concept dataset (those who do not change, or change only a little). To be representative of the concept, the mask must match a significant amount of the instances in the concept dataset.

The mask is created in several steps. First, the mask is set to equal the instance that resulted in the activation vector that is the closest to the cluster center. The entropy of each feature is computed on all instances of the concept dataset. The closer the entropy is to zero the more similar the feature values are. At the start, the features that do not change (the entropy is equal to 0) and the feature with the smallest non zero entropy value are kept. Then the instances where the selected features have the same value as the mask are counted, ignoring the values of the unselected features. If not enough instances are counted, the next feature with the smallest non zero entropy is added. Every time a feature is added to the mask, it selects more and more instances matching with the mask until reaching the desired percentage. At this moment, the mask is a subset of features that represents the concept dataset.

4.4 Evaluating the Concepts

Once the datasets of instances are created and their concepts identified, the TCAV method can therefore be used to determine the influence of the concepts over the model's predictions. Classifiers needs to be trained on each clusters from Section 4.2 to retrieve their corresponding Concept Activation Vectors, each one pointing at the center of the cluster. To do so, the following is repeated for each cluster.

All the activation vectors are taken in the cluster of interest and a corresponding label for each is created. These labels are set to "1", meaning that these activation vectors correspond the identified concept. As negative samples, the same amount of activation vectors is randomly taken from the other clusters and their labels are set to "0", meaning that they do not contain the concept.

A linear classifier is then trained over the previously labelled activation vectors in order to differentiate the activation vectors that represent the concept from those that do not. The corresponding CAV is the vector that is orthogonal to the decision boundary of the classifier. At the end of this process, a CAV for the evaluated cluster is obtained.

When the above is done for each determined cluster, it is possible to check what concept prevails over a particular class of instances using the calculated CAVs and the TCAV score method described in [Kim et al., 2018]. Taking a class dataset and then applying the Formula 3.13 explained in Section 3.2.10 gives a score for each identified concept, making it possible to compare them in order to know how much each influence the Neural Network's outputs.

4.5 Summary

This chapter gives a solution to the unsupervised extraction of concepts regarding a Neural Network. By retrieving multiple activation vectors for the concerned Neural Network, an algorithm (i.e., K-Means) can extract the concepts by clustering the activation vectors. Once the clusters are formed, depending on the nature of the instances, it is possible to identify the underlying concepts by finding the most representative instances behind the activation vectors forming the cluster.

Finally, to test the importance of an identified concept, the TCAV method can be used based on the concepts dataset formed with this method. In the incoming chapters, the method is applied to two different problems (images classification and a Hearthstone agent) in order to evaluate its performance. The two chosen problems are solved by models that have two different structures and are working on different types of data, allowing this work to better evaluate its method and to get more diversified results.

Chapter 5

Method Evaluation on Image Classifier

This chapter aims to assess the performance of the method introduced in Chapter 4. The GoogLeNet image classifier used in [Kim et al., 2018] is introduced as well as the ImageNet dataset. Then, an unsupervised extraction of concepts is done over the classifier using 22 different ImageNet's classes and the technique explained in Chapter 4.

5.1 Model Introduction

[Kim et al., 2018] use two models to produce their results. Only one of them, GoogLeNet, is used in the following part of this chapter, since the two models are ImageNet classifiers. The model's architecture is given in [Szegedy et al., 2015] and can be observed at Figure 5.1.

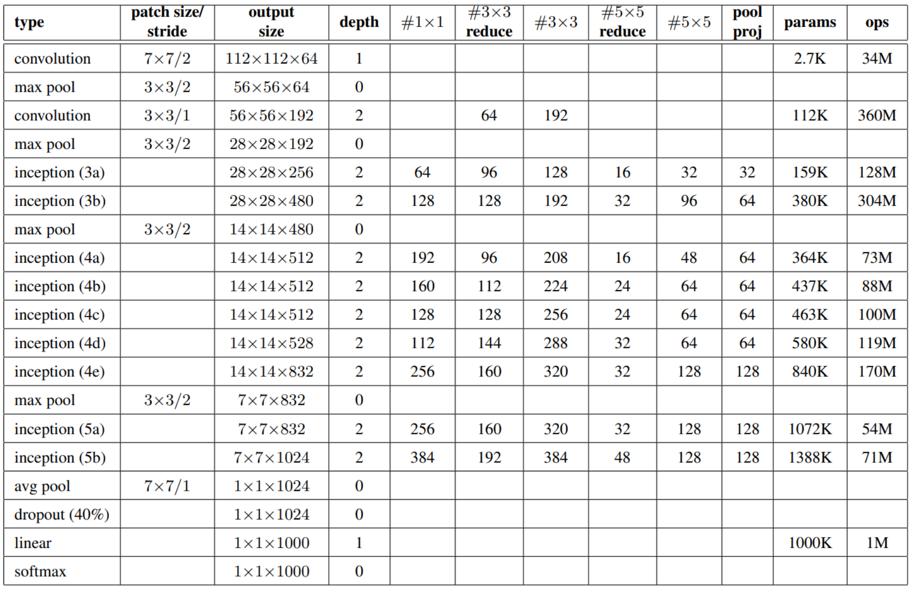
GoogLeNet is a classifier trained on the ImageNet dataset[1], which is a large dataset of labelled images used for computer vision research, with more than 1000 classes, each represented by hundreds and thousands of images. The predictions given by GoogLeNet are the probabilities for an input image to be part of a class of the dataset.

When introduced, GoogLeNet outperformed its opponents with an error rate of 6.67%. It is therefore not a matter of debugging the model but of understanding the reasoning behind its predictions.

5.2 Unsupervised Extraction of Concepts

In this section, the theory presented in Chapter 4 is applied to the GoogLeNet model. Also, it shows that the clusters formed by the method are linked to high level concepts.

[1] https://www.image-net.org/



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 5.1: GoogLeNet's architecture, from [Szegedy et al., 2015].

5.2.1 Retrieving Activation Vectors

[Kim et al., 2018] use a subset of the ImageNet dataset containing only zebra-labelled images. The method introduced in Chapter 4 is unlikely to be as precise as the concept of stripes extracted in [Kim et al., 2018] (due to its own dedicated dataset). A larger subset of ImageNet is therefore taken to better isolate different concepts. Indeed, the method is unsupervised and a single class out a 1000 is not representative of the potential concepts learnt by the Neural Network.

To constitute the concept datasets, different classes that are likely to contain concepts in common are chosen. However, in order to diversify these concepts, some distant classes are taken too. For instance, different classes of fruits (lemon, orange and banana) are taken, as well as various classes containing motorised vehicles (cabs, steam trains, *etc.*). There are also a few breeds of dogs, different types of fish (white sharks, lionfish, *etc.*) and finally the zebra and tiger classes. In total, there are 22 different classes. For computing reasons, the chosen subset of classes only include 20 to 35 images of the original dataset classes. The images are then forwarded through the network and their activation vectors are saved. This is done at the same layer [Kim et al., 2018] selected, which is the layer named *inception4c* on Figure 5.2.

5.2.2 Extracting the Concepts

Once the activation vectors are retrieved, K-Means is used to create the clusters that aim to extract each concept. It is therefore necessary to select the right amount of clusters. For this, it is common to use the elbow method [Kingrani et al., 2018] coupled with the silhouette score [Kingrani et al., 2018] of the clustering. For the latter, the score is a measure of the similarity of an instance

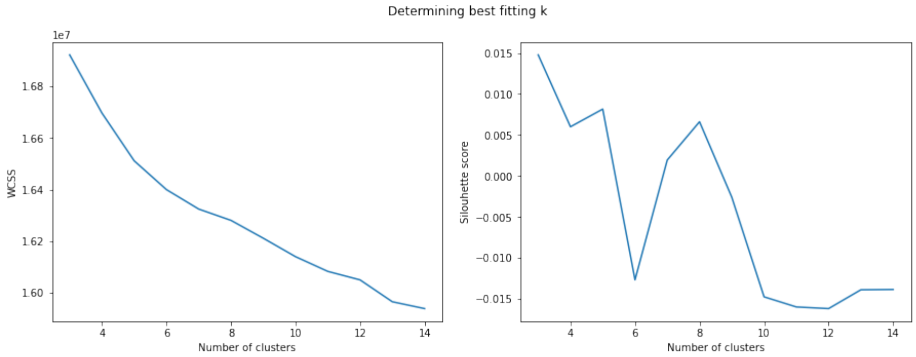


Figure 5.2: Using the elbow method and the silhouette score to determine the best number of clusters for the GoogLeNet network.

with its own cluster compared to others. For the former, it is entirely subjective to choose the value that is supposed to be optimal (generally, the one in the elbow of the curve, hence the name “Elbow method”). It is the value that maximises the score that is supposed to be the best number of clusters for the silhouette technique. Looking at Figure 5.2, two is the value that maximises the silhouette score. However, 2 clusters for 22 classes is unlikely because it means that there is only two concepts learnt by the Neural Network over all these classes.

Since the value that maximises the silhouette score is unlikely to be the best number of cluster, the other scores that peak on the graph at Figure 5.2 are reviewed below. These values are $k = 5$ and $k = 8$, and are more plausible.

It is important to mention that Figure 5.2 indicates very poor silhouette scores. Indeed, the silhouette score lies between 1 and -1, with the former representing well defined clusters and the latter poorly distinguished clusters.

5.2.3 Understanding the Concepts

Like explained in Section 4.3.1, looking at only one image per cluster is not helpful to identify its extracted concept, it is therefore necessary to refine the results.

A K-Medoids clustering is applied on each one of the clusters formed with $k = 5$ and $k = 8$. The images corresponding to the medoids for the sub-clusters are retrieved. Visualising these images helps identifying the concept behind each clusters. Although subjective, it exists a pattern between these images.

Figure 5.3 and 5.4 show the results of the K-Medoids clustering over the different clusters for $k = 5$ and $k = 8$ respectively. The two figures show 4 images per cluster that correspond to the images of the sub-clusters medoids.

First, the observations made over $k = 5$ are developed. When looking at Figure 5.3, even though each class is seen to be completely different from the Neural Network perspective, the clustering made over the activation vectors isolate similar classes together. Looking at the first cluster, it regroups the dogs breeds

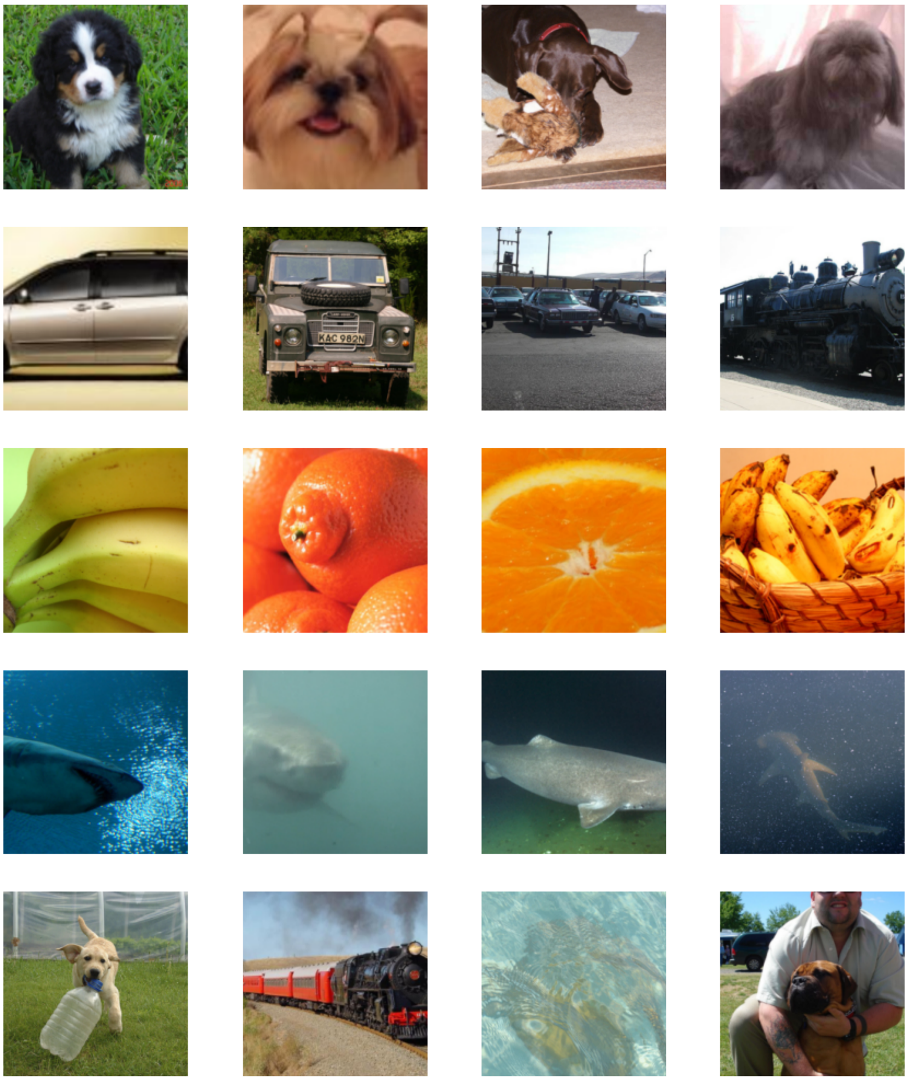


Figure 5.3: GoogLeNet sub-medoids corresponding images for $k = 5$ (each row corresponding to a different cluster).

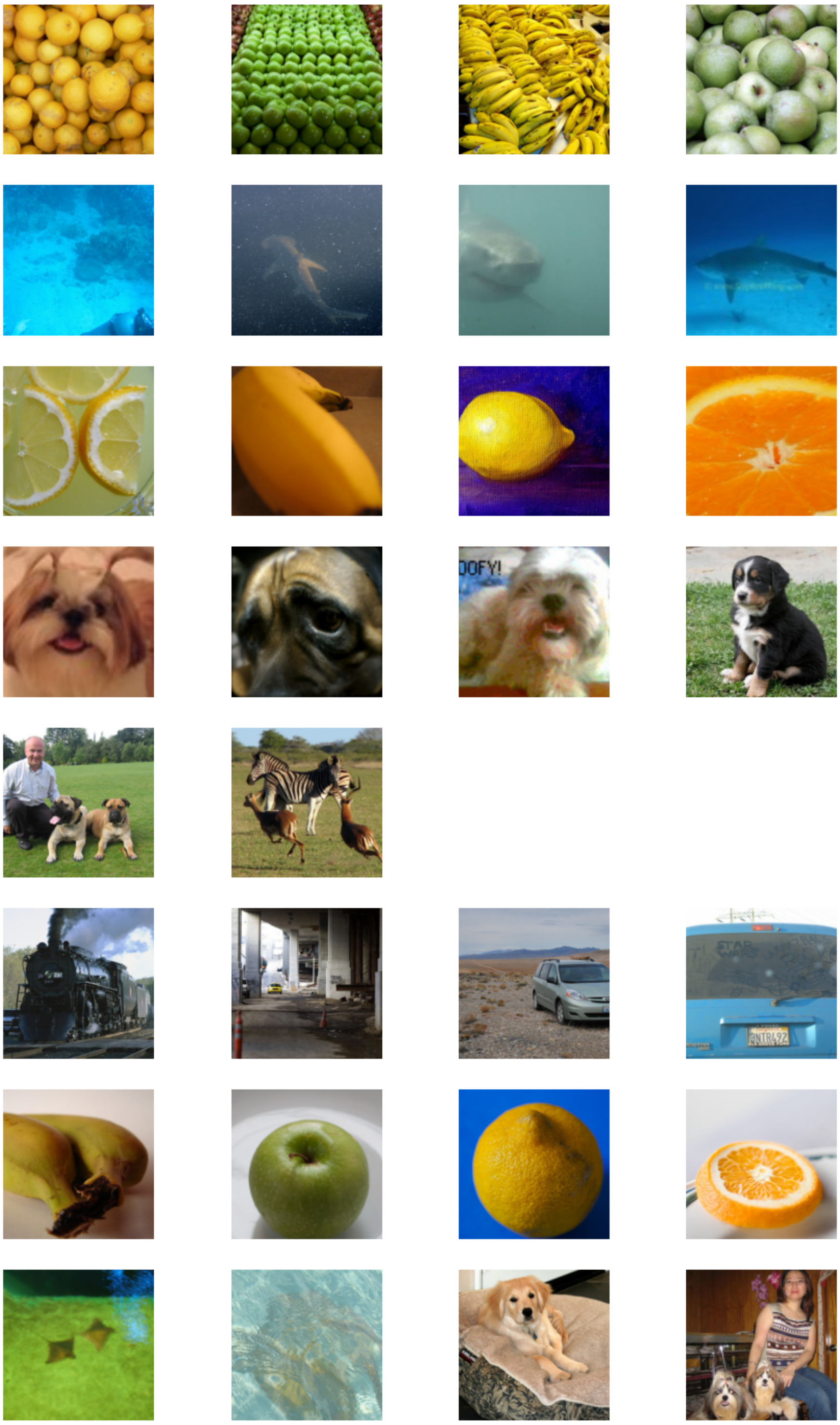


Figure 5.4: GoogLeNet sub-medoids corresponding images for $k = 8$ (each row corresponding to a different cluster).

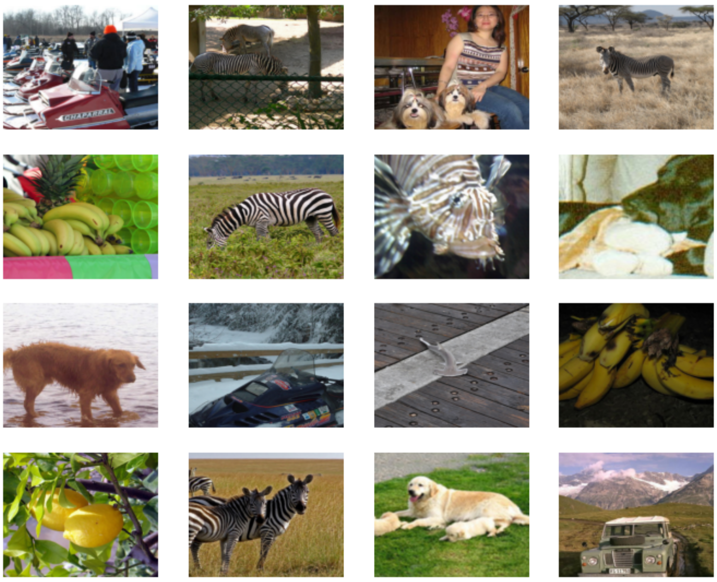


Figure 5.5: 16 images taken from the 5^{th} cluster for $k = 5$.

together (but also images of tigers), it does the same for the motorised vehicles and for the third cluster, the different sort of fruits can be seen too. The concept behind the images in cluster 4 is related to water or to the colour blue at least. The concept behind the last group formed is unclear from the 3 medoids displayed. Figure 5.5 shows more images forming the latter, and it can be seen that there is no apparent pattern between the images. Indeed it contains a lot of images from all the different classes and they have no distinct pattern.

Concerning the identified concepts mentioned above, it is important to keep in mind that it is an unsupervised method and so, that descriptions for the concepts were chosen based on what is seen of the images.

Figure 5.6 shows 5 images selected by hand from the fourth cluster, which are the most representative of the cluster diversity. It is interesting because each image belongs to a different class but they all have a very blue colorimetry and are related to water most of the time.

The clusters made with $k = 8$ is now analysed. The medoids extracted from the different clusters (see Figure 5.4) do not help to identify any new concept over ones that are identified when $k = 5$. Instead, some seem to repeat themselves. However, looking more precisely at the first and third ones, even though they both contain images of fruits, this does not seem to be the prime concept of the first cluster. Looking at Figure 5.7 shows the difference between the two. It can be observed that the first cluster shows heaps of fruits sometimes grouped in a

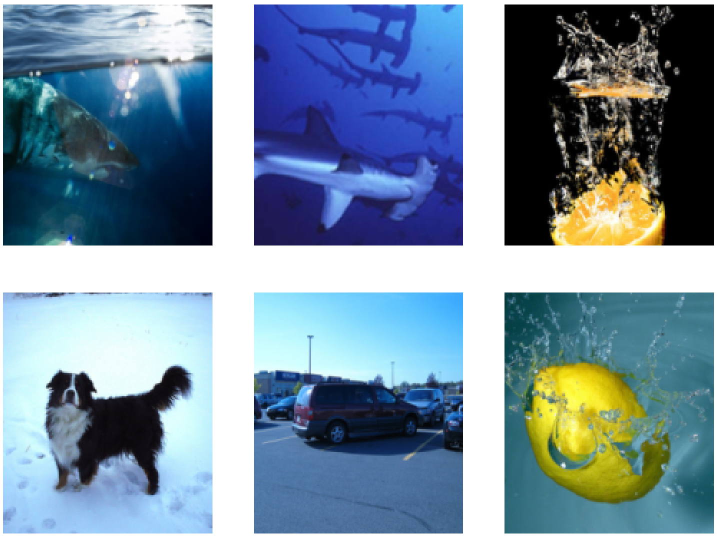
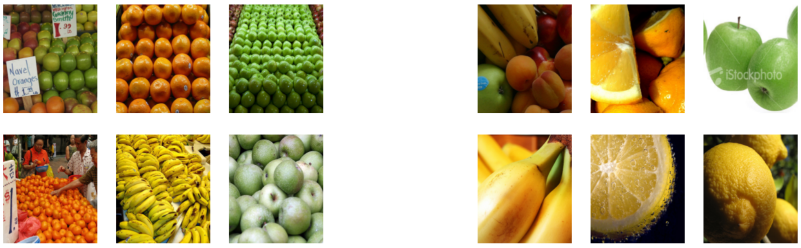


Figure 5.6: 5 images taken from the 4^{th} cluster for $k = 5$.



(a) 1^{st} Cluster.

(b) 3^{rd} Cluster.

Figure 5.7: Comparison of the first and the second cluster for $k = 8$.

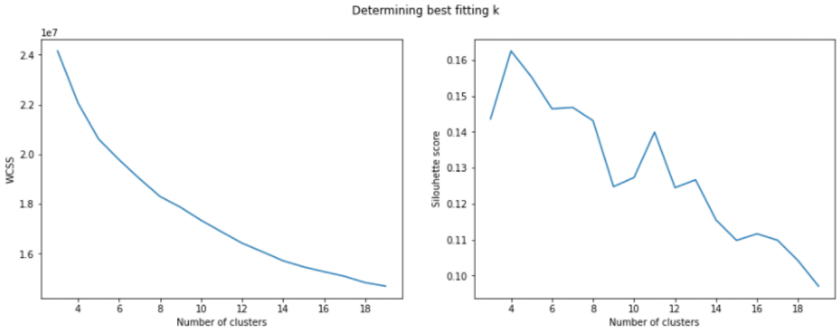


Figure 5.8: Elbow method and silhouette scores for GoogLeNet after applying the PCA reduction.

very geometrical manner while the third cluster has its images containing a lot less fruits. The clustering made with $k = 5$ does not make this distinction and so the clustering with $k = 8$ has extracted a higher level concept.

It is important to note that no matter how the clustering is trained, it seems that there is always a least one cluster containing images from the 22 different classes with no apparent pattern. Indeed, they seem to correspond to all the activation vectors that could not be labelled in a distinct cluster. Moreover, some concept datasets contain too few images to be representative of a concept, and therefore cannot produce a classifier robust enough to build a CAV.

5.2.4 Fixing the Results

The drawbacks of the results (i.e., the cluster with no apparent concept and the very poor silhouette scores) obtained in the above experiments could be due to the high dimensionality of activation vectors. To test this hypothesis, this section presents additional experiments where a dimensionality reduction is performed on the activation vectors before the clustering. Principal component analysis (PCA) [Ringnér, 2008] is used to reduce the dimensionality of activation vectors from 100.352 ($= 14 \times 14 \times 512$) to 20. Then, the clustering is performed using K-Means and concepts are extracted from the clusters *in the new space with lower dimensionality*. Since the corresponding CAVs live in the lower-dimensional space, TCAV cannot be obtained directly. This is left for further work.

Using the 22 same Imagenet's classes from Section 5.2.1, results show that the clustering is more stable: The silhouette score in Figure 5.8 is better (with larger values) and a peak at $k = 4$ is visible.

Figure 5.9 shows the sub-medoids for the 4 clusters. The concepts are likely the same (fur, fruits, vehicles and water) than the ones obtained without the application of the PCA algorithm but there are some important difference; There is no cluster regrouping images with no apparent similarities, no cluster containing too few images and the clusters are containing a lot less images that seems to be outliers. The clusters are clearly better formed. Therefore, it confirms that there was a dimensionality problem due to the high dimension of the activation

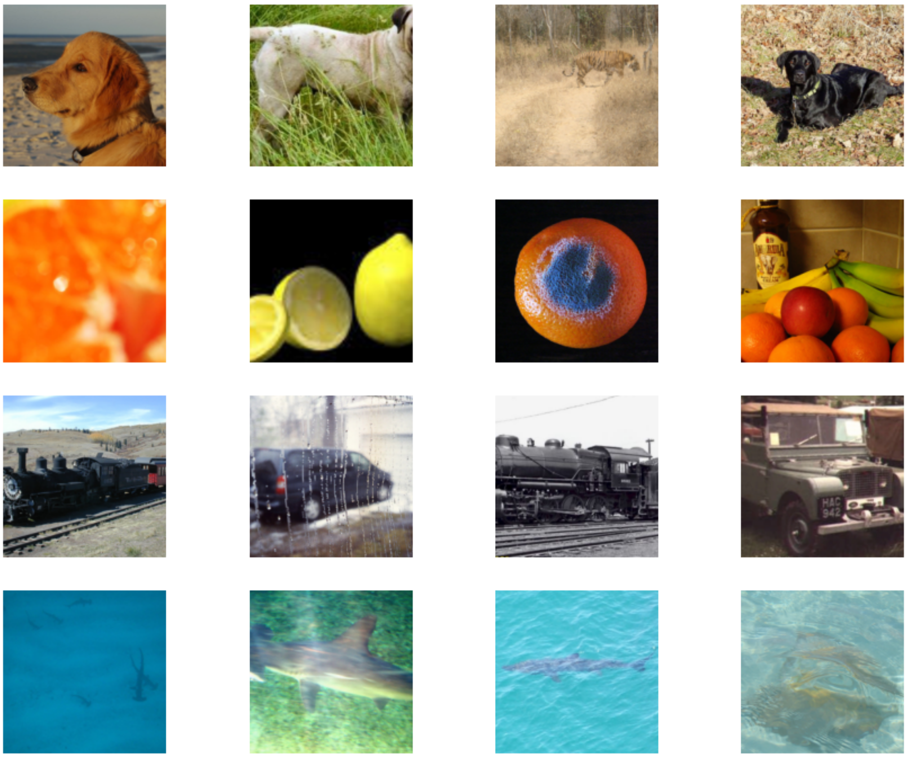


Figure 5.9: GoogLeNet instances corresponding to the sub-medoids of the 4 different clusters (each row corresponding to a different cluster).

vectors.

5.3 Conclusion

The results obtained with the GoogLeNet model are encouraging. Indeed, the method is able to extract some concepts. However, these concepts do not compete with the stripped concept presented in [Kim et al., 2018], which is of higher level. The extracted concepts still seem to be linked to classes. It remains to be seen whether a larger dataset of activation vectors would allow more precision in the concepts extraction.

Since the high dimensionality problem is solved by reducing the dimensions, the CAVs cannot be created directly because they are in a lower dimension than the initial activation vectors. Regarding this, TCAV scores still remain to be done in further work.

The results presented by the clustering performed without PCA show a distinction between fruits. One cluster contains the images with a lot of fruits and the other contains the images with few fruits. This result comforts the idea that a much more precise concepts can be extracted.

This section tends to prove that concepts extraction works on images with the proposed method. Therefore, next chapter will use the method in order to troubleshoot a Hearthstone game agent and to show that concept identification for tabular-data is more complex.

Chapter 6

Method Evaluation on Board Game Agent

In this chapter, the technique from Chapter 4 is applied on a Hearthstone game agent. The nature of the instances is different. Unlike images, a representation of the Hearthstone game state is not designed to be visually understandable and therefore, the mask solution from Section 4.3.2 needs to be applied to extract concepts from the constructed clusters.

6.1 Model Introduction

In this section, different models implementations for Hearthstone are presented followed by the reflection that led to the selection of one of them.

6.1.1 Hearthstone Existing Implementations

Finding an efficient Hearthstone AI is not something easy. Indeed, there exist different implementations, but a lot of them are not viable options since they are not documented or even finished. The following are the most important ones.

There exist inside some of the simulators themselves (such as Sabberstone or Metastone) some already implemented AIs. A well known AI is Silverfish. It was used during a time by the Hearthstone players to let them improve their rank in the game, but the AI was not very effective against pro players. Also, it is hard finding information about this particular AI since it seems its source code is not available anymore (only a few copies are available on Github). Also, there exists HearthAgent, which is an implementation using only MCTS, but its results are not convincing. All these AIs are not suitable for this work since they are not Neural Networks.

Since the advent of AlphaZero (Section 2.3.1) in 2017, there has been multiple repositories on Github that try to adapt it for Hearthstone. Few of them are documented or even usable since they are mainly the side project of enthusiasts.

The problem with such implementations is that AlphaZero was designed and has shown results for games with a completely known game state such as Chess, Shogi and Go. Its performance seems to fall off when it comes to imperfect-information games such as Hearthstone (see Table 2.1 for an exhaustive list of the game's characteristics). More recently, Facebook introduced ReBeL to overcome this problem, but no implementation for Hearthstone was found. For the following section, an AI implementing AlphaZero is presented, its name is Alphastone. It is the game agent chosen for the concept extraction part of this chapter.

6.1.2 Alphastone

Alphastone[1] is a repository on Github that consists in a reinforcement learning bot that uses MCTS and a Neural Network to play Hearthstone. The implementation uses Firestone, a simulator written in Python.

The author of the project briefly explains its choice of architecture on his website[2]. It is a work forked from an implementation[3] based on the original AlphaZero paper [Silver et al., 2018]. One important point to consider is that the game state representation is incomplete. Cards are not represented, only their characteristics, which means the model ignores their special effects.

A trained model is available on the repository, which is said to have a winning rate of 82% against a random AI, but the training methodology adopted is not optimal. The author of the repository trained his model using randomly generated decks over each new iteration of the AlphaZero algorithm. While this approach makes the AI encounters any card the game has to offer, it can also lead to unusable decks. Furthermore, a player would never learn to play with every cards available, he rather specialises himself on a particular type of deck. This is something that can partially explain the performance of the model, which are therefore not astonishing. However, this is not a problem for this evaluation since it aims at extracting the concepts learned by the Neural Network whether they are good or not.

6.2 Unsupervised Extraction of Concepts

In this section, the theory presented in Chapter 4 is applied to the Neural Network that Alphastone uses. The application of the method shows how it can help to understand the strategies learnt by an agent and then how to troubleshoot - partially - the Alphastone model.

6.2.1 Retrieving Activation Vectors

First, it is necessary to generate a number of valid game states. In order to do so, the agent plays a high number of games thanks to the Fireplace simulator, and each time it enters in a new state, it is saved in a file. Once this is done, these states are loaded and only the unique ones are kept. Each one of them are

[1] https://github.com/sirmammingtonham/alphastone (26/02/2021).

[2] https://sirmammingtonham.github.io/projects/alphastone.html (22/04/2021)

[3] https://github.com/suragnair/alpha-zero-general

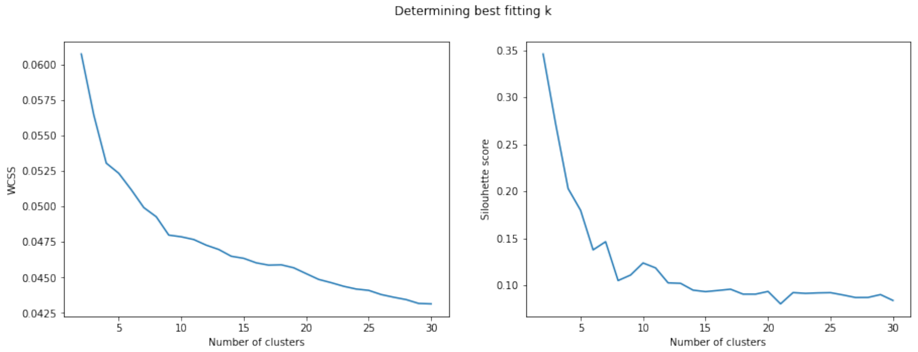


Figure 6.1: Using the elbow method and the silhouette score to determine the best number of clusters for the Alphastone network.

then passed through the network and the corresponding activation vector at a certain layer is saved.

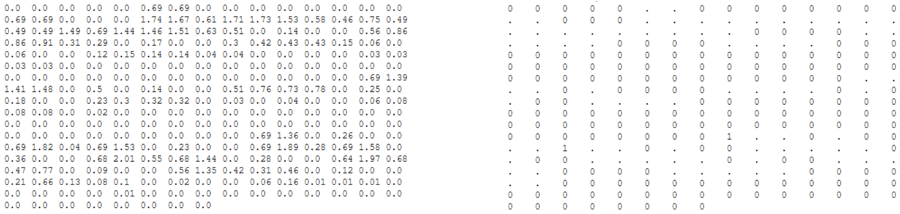
6.2.2 Extracting the Concepts

To determine the best number of clusters, the elbow method is used combined with the silhouette score as for GoogLeNet (see Section 5.2.2). Looking at Figure 6.1, it seems counter intuitive that there would be only two distinctive clusters for all the activation vectors. Also, the elbow technique does not confirm that assumption. Comparing the two methods, it looks like the best fitting value for k, the number of clusters, is somewhere between 4 and 10. The choice of k being not clear, for the rest of this chapter, $k = 5$ is considered in order to have a first look at what comes out.

6.2.3 Understanding the Concepts

Once the clustering is done, it is necessary to find a way to understand what each cluster represents. Since the model instances are tabular data, applying the dedicated method explained in Section 4.3.2 is relevant. This means that for each cluster, a mask is created based on the state corresponding to its closest activation vector to the cluster center, which is a representative game state for the cluster. Concretely, each state corresponding to the cluster instances is taken and the entropy of each feature is computed as explained in Section 4.3.2. Once this is done, the mask is created based on the resulting entropy values.

Figure 6.2a shows the entropy values of the first cluster for each feature. When the value is equal to 0.0 it means that the value is the same for every state in the cluster. Figure 6.2b shows the corresponding mask. To generate this mask, the theory from Section 4.3.2 is applied. The percentage of states in the cluster that must fit to the mask is fixed to 80% and the mask is generated until it reaches this desired percentage. In Figure 6.2b, the values replaced by a dot are the ones that change too much. It corresponds to the features with an entropy value that is considered too high. These values are not meaningful to understand the concept in the cluster.



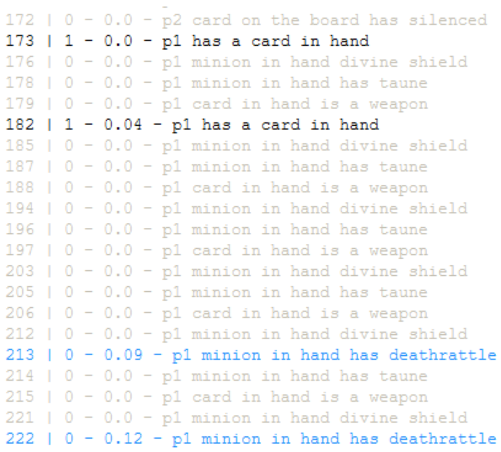
```
0.0  0.0  0.0  0.0  0.0  0.69 0.69 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.69 0.69 0.0  0.0  0.0  1.74 1.67 0.61 1.71 1.73 1.53 0.58 0.46 0.75 0.49
0.49 0.49 1.49 0.69 1.44 1.46 1.51 0.63 0.51 0.0  0.14 0.0  0.0  0.56 0.86
0.86 0.91 0.31 0.29 0.0  0.17 0.0  0.0  0.3  0.42 0.43 0.43 0.15 0.06 0.0
0.06 0.0  0.0  0.12 0.15 0.14 0.14 0.04 0.04 0.0  0.0  0.0  0.0  0.03 0.03
0.03 0.03 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.69 1.39
1.41 1.48 0.0  0.5  0.0  0.14 0.0  0.0  0.51 0.76 0.73 0.78 0.0  0.25 0.0
0.18 0.0  0.0  0.23 0.3  0.32 0.32 0.0  0.03 0.0  0.04 0.0  0.0  0.06 0.08
0.08 0.08 0.0  0.02 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.69 1.36 0.0  0.26 0.0  0.0
0.69 1.82 0.04 0.69 1.53 0.0  0.23 0.0  0.0  0.69 1.89 0.28 0.69 1.58 0.0
0.36 0.0  0.0  0.68 2.01 0.55 0.68 1.44 0.0  0.28 0.0  0.0  0.64 1.97 0.68
0.47 0.77 0.0  0.09 0.0  0.0  0.56 1.35 0.42 0.31 0.46 0.0  0.12 0.0  0.0
0.21 0.66 0.13 0.08 0.1  0.0  0.02 0.0  0.0  0.06 0.16 0.01 0.01 0.01 0.0
0.0  0.0  0.0  0.0  0.01 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

(a) Entropy values

```
0  0  0  0  0  .  .  0  0  0  0  0  0  0  0
.  .  0  0  0  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  0  0  0  0  .  .
.  .  .  .  0  0  0  0  .  .  .  .  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  .  .
.  .  0  .  0  0  0  0  .  .  .  .  0  .  0
.  0  0  .  .  .  .  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  1  .  .  0  .  0  0
.  .  1  .  .  0  .  0  0  .  .  .  .  .  0
.  0  0  .  .  .  .  .  0  .  0  0  .  .  .
.  .  0  0  0  0  .  .  .  .  .  0  0  0  0
.  .  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
```

(b) Mask

Figure 6.2: Entropy values and their corresponding mask for the first cluster.



```
172 | 0 - 0.0 - p2 card on the board has silenced
173 | 1 - 0.0 - p1 has a card in hand
176 | 0 - 0.0 - p1 minion in hand divine shield
178 | 0 - 0.0 - p1 minion in hand has taune
179 | 0 - 0.0 - p1 card in hand is a weapon
182 | 1 - 0.04 - p1 has a card in hand
185 | 0 - 0.0 - p1 minion in hand divine shield
187 | 0 - 0.0 - p1 minion in hand has taune
188 | 0 - 0.0 - p1 card in hand is a weapon
194 | 0 - 0.0 - p1 minion in hand divine shield
196 | 0 - 0.0 - p1 minion in hand has taune
197 | 0 - 0.0 - p1 card in hand is a weapon
203 | 0 - 0.0 - p1 minion in hand divine shield
205 | 0 - 0.0 - p1 minion in hand has taune
206 | 0 - 0.0 - p1 card in hand is a weapon
212 | 0 - 0.0 - p1 minion in hand divine shield
213 | 0 - 0.09 - p1 minion in hand has deathrattle
214 | 0 - 0.0 - p1 minion in hand has taune
215 | 0 - 0.0 - p1 card in hand is a weapon
221 | 0 - 0.0 - p1 minion in hand divine shield
222 | 0 - 0.12 - p1 minion in hand has deathrattle
```

Figure 6.3: Part of the mask description for the first cluster. Blue if the value is 0 and the entropy is different than 0. Black if the value is 1. Grey if the value and the entropy are 0.

As it stands, it is quite difficult to interpret the clusters based on their respective masks. That is why more detailed descriptions are generated to understand what each value of the mask represents. Since it is long and tedious, Figure 6.3 shows an example taken from the first cluster. A description can be observed for some of the unmasked features with their corresponding value and entropy score. Since each instance has 263 features, describing them all for the 5 different clusters would overload the document. For the same reason, detailed matrices for cluster 2, 3, 4 and 5 were placed in the Appendix A.

Analyzing the masks and the descriptions remains difficult. Apart from defining the concept as what is literally described by the mask and its corresponding description, it is difficult to summarise what exactly it consists of. For instance, when looking at Figure A.5b, only features set to 0 are present in the mask. Since it is not possible to differentiate the useless features from the useful ones, the amount of features left to process is too high.

Finally, it is clear that the model has learnt very low level concepts. For instance, for the first cluster, it is about having a card or two in hand, which corresponds to a loosing state in the game. This makes sense because the representation of the game state is quite poor and does not allow the model to learn higher level strategies like playing a specific card in a specific situation.

6.3 Conclusion

This section is an evaluation of the method based on the results obtained in Section 6.2.

The clustering identified groups of concepts but the visualising method dedicated to *tabular-data* does not allow to highlight them easily. Since it is difficult for a human to process too much text, the problem would rather lie in the given solution of the mask which remains to be improved as well as in the difficulty imposed by the representation of the game state.

Indeed, describing the observations made with this method is very time consuming, especially for larger instances. The Alphastone game state representation, with its 263 values, is a real barrier to the concepts identification. Indeed, even when cleaned up using the mask technique, it remains abstract and difficult for a human to grasp the concepts. Still, the Neural Network has learnt concepts and the clustering isolated them, the difficulty remains in their representation.

Also, as mentioned in Section 6.1.2, the representation of a game state is incomplete. The cards are not distinguished by an ID but by their characteristics (life, attack,...), which does not really allow to differentiate them. For example, 2 spell cards having a mana cost of 2 do not have the same effects but cannot be distinguished in the state representation used in this work. Fixing the misrepresentation can certainly improve the ability of the Neural Network to develop higher level concepts. Applying the method from Chapter 4 highlights that misrepresentation.

This chapter shows that the task consisting of extracting concepts from a Neural Network that has tabular data as instances is much harder than for image-based instances. Still, the extraction of concepts is feasible, but getting a clear representation is the real difficulty.

Chapter 7

Conclusion

This master thesis proposes a new method in the field of Explainable Artificial Intelligence that aims to extract concepts learnt by a Neural Network. The method performs a clustering on the activation vectors retrieved from a Neural Network, to afterwards identify concepts that link these activation vectors inside each cluster. Instances corresponding to the clusters are then used to build concept datasets, making possible to automate the creation of Concept Activation Vectors [Kim et al., 2018].

The applications of the method made over the GoogLeNet Neural Network in Chapter 5 proves that an unsupervised extraction of concepts is feasible. Also, the method application over Alphastone in Chapter 6 shows that the extraction of concepts highlights what is problematic with a model.

Although the proposed method is able to extract concepts, it remains to be seen if the extraction of much more precise concepts is feasible. Using a larger number of instances to better diversify each type of activation vectors would be an approach to the problem and is left for further work.

Also, the feasibility of applying the TCAV method using a concept dataset realised with the proposed method still needs to be demonstrated since the high dimensionality issue from Chapter 5 prevents the creation of CAVs directly. The CAV evaluation is left for further work.

It is important to keep in mind that the initialisation of the K-Means algorithm affects the construction of the clusters and thus on the overall method results. The choice of k is also essential and using the silhouette method does not assure to have the best well isolated concepts as seen in Chapter 5 and Chapter 6. An important improvement to mention is to optimise the selection of k, while decreasing the user importance in that choice. Chapter 5 shows the problem of high dimensional activation vectors. Indeed, K-Means does not perform well when dealing with too much features. Performing a dimension reduction with an algorithm like PCA improves the clustering, but loses information. Finding the right trade-off between information loss and performance is left for further work.

Finally, improving the concept identification step for tabular-data instances is also necessary. In its current state, it is not user-friendly to grasp the idea behind the *masks* formed using this work method as seen in Chapter 6. Improving the method for concept identification over tabular-data is left for further work.

Bibliography

[Adebayo et al., 2018] Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., and Kim, B. (2018). Sanity checks for saliency maps. *arXiv preprint arXiv:1810.03292*.

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

[Alom et al., 2019] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Hasan, M., Van Essen, B. C., Awwal, A. A., and Asari, V. K. (2019). A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3):292.

[Apley and Zhu, 2020] Apley, D. W. and Zhu, J. (2020). Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086.

[Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256.

[Bibal and Frénay, 2016] Bibal, A. and Frénay, B. (2016). Interpretability of machine learning models and representations: an introduction. In *ESANN*.

[Bre et al., 2018] Bre, F., Gimenez, J. M., and Fachinotti, V. D. (2018). Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158:1429–1441.

[Browne et al., 2012] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

[Czech et al., 2020] Czech, J., Korus, P., and Kersting, K. (2020). Monte-carlo graph search for alphazero. *arXiv preprint arXiv:2012.11045*.

[Došilović et al., 2018] Došilović, F. K., Brčić, M., and Hlupić, N. (2018). Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 0210–0215. IEEE.

[Edwards and Veale, 2017] Edwards, L. and Veale, M. (2017). Slave to the algorithm: Why a right to an explanation is probably not the remedy you are looking for. *Duke L. & Tech. Rev.*, 16:18.

[Entezami et al., 2020] Entezami, A., Sarmadi, H., and Razavi, B. S. (2020). An innovative hybrid strategy for structural health monitoring by modal flexibility and clustering methods. *Journal of Civil Structural Health Monitoring*, 10(5):845–859.

[Fisher et al., 2018] Fisher, A., Rudin, C., and Dominici, F. (2018). Model class reliance: Variable importance measures for any machine learning model class, from the" rashomon" perspective. *arXiv preprint arXiv:1801.01489*, 68.

[Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

[Friedman et al., 2008] Friedman, J. H., Popescu, B. E., et al. (2008). Predictive learning via rule ensembles. *Annals of Applied Statistics*, 2(3):916–954.

[Goldstein et al., 2015] Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Indolia et al., 2018] Indolia, S., Goswami, A. K., Mishra, S., and Asopa, P. (2018). Conceptual understanding of convolutional neural network-a deep learning approach. *Procedia computer science*, 132:679–688.

[Kim et al., 2018] Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., et al. (2018). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pages 2668–2677. PMLR.

[Kingrani et al., 2018] Kingrani, S. K., Levene, M., and Zhang, D. (2018). Estimating the number of clusters using diversity. *Artificial Intelligence Research*, 7(1):15–22.

[Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

[Lu and Weng, 2007] Lu, D. and Weng, Q. (2007). A survey of image classification methods and techniques for improving classification performance. *International journal of Remote sensing*, 28(5):823–870.

[Lundberg and Lee, 2017] Lundberg, S. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*.

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, number 14 in 1, pages 281–297. Oakland, CA, USA.

[Molnar, 2020] Molnar, C. (2020). *Interpretable machine learning*. Lulu.com.

[Montavon et al., 2018] Montavon, G., Samek, W., and Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15.

[Piltaver et al., 2014] Piltaver, R., Luštrek, M., and Gams, M. (2014). Multi-objective learning of accurate and comprehensible classifiers–a case study. In *STAIRS 2014*, pages 220–229. IOS Press.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "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.

[Ribeiro et al., 2018] Ribeiro, M. T., Singh, S., and Guestrin, C. (2018). Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[Ringnér, 2008] Ringnér, M. (2008). What is principal component analysis? *Nature biotechnology*, 26(3):303–304.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Prentice Hall Series in Artificial Intelligence*. Prentice Hall Englewood Cliffs, NJ:.

[Schubert and Rousseeuw, 2019] Schubert, E. and Rousseeuw, P. J. (2019). Faster k-medoids clustering: improving the pam, clara, and clarans algorithms. In *International conference on similarity search and applications*, pages 171–187. Springer.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

[Shapley, 1953] Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

[Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[Simonyan et al., 2013] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

[Struyf et al., 1997] Struyf, A., Hubert, M., Rousseeuw, P., et al. (1997). Clustering in an object-oriented environment. *Journal of Statistical Software*, 1(4):1–30.

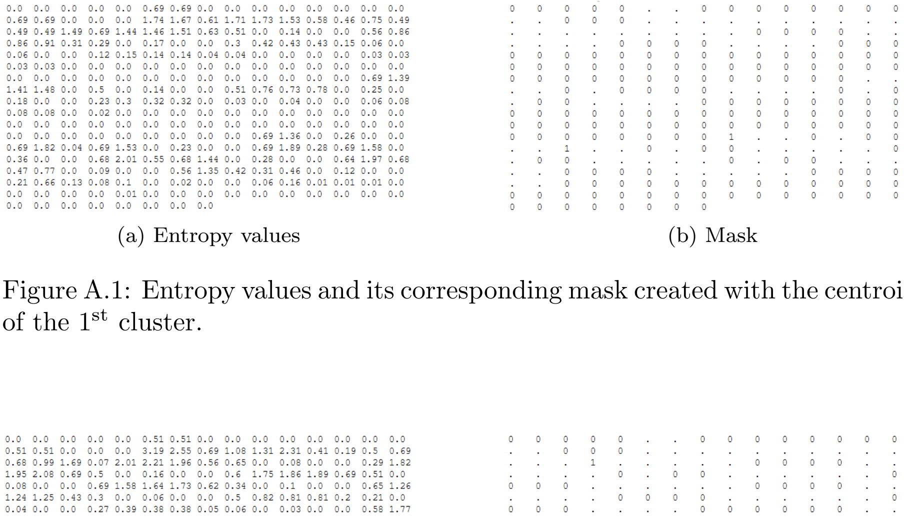
[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Appendix A

Alphastone Matrices



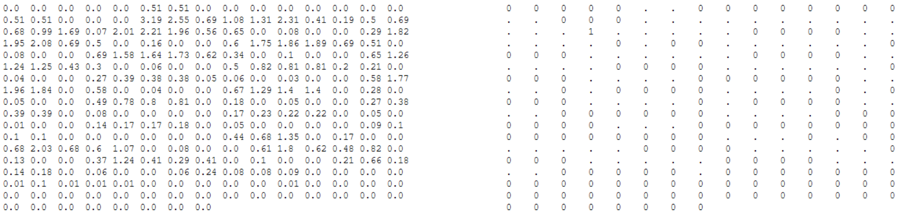
```
0.0  0.0  0.0  0.0  0.0  0.69 0.69 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.69 0.69 0.0  0.0  0.0  1.74 1.67 0.61 1.71 1.73 1.53 0.58 0.46 0.75 0.49
0.49 0.49 1.49 0.69 1.44 1.46 1.51 0.63 0.51 0.0  0.14 0.0  0.0  0.56 0.86
0.86 0.91 0.31 0.29 0.0  0.17 0.0  0.0  0.3  0.42 0.43 0.43 0.15 0.06 0.0
0.06 0.0  0.0  0.12 0.15 0.14 0.14 0.04 0.04 0.0  0.0  0.0  0.0  0.03 0.03
0.03 0.03 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.69 1.39
1.41 1.48 0.0  0.5  0.0  0.14 0.0  0.0  0.51 0.76 0.73 0.78 0.0  0.25 0.0
0.18 0.0  0.0  0.23 0.3  0.32 0.32 0.0  0.03 0.0  0.04 0.0  0.0  0.06 0.08
0.08 0.08 0.0  0.02 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.69 1.36 0.0  0.26 0.0  0.0
0.69 1.82 0.04 0.69 1.53 0.0  0.23 0.0  0.0  0.69 1.89 0.28 0.69 1.58 0.0
0.36 0.0  0.0  0.68 2.01 0.55 0.68 1.44 0.0  0.28 0.0  0.0  0.64 1.97 0.68
0.47 0.77 0.0  0.09 0.0  0.0  0.56 1.35 0.42 0.31 0.46 0.0  0.12 0.0  0.0
0.21 0.66 0.13 0.08 0.1  0.0  0.02 0.0  0.0  0.06 0.16 0.01 0.01 0.01 0.0
0.0  0.0  0.0  0.0  0.01 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

(a) Entropy values

```
0    0    0    0    0    .    .    0    0    0    0    0    0    0    0
.    .    0    0    0    .    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    0    0    0    0    .    .
.    .    .    .    0    0    0    0    .    .    .    .    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    .    .
.    .    0    .    0    0    0    0    .    .    .    .    0    .    0
.    0    0    .    .    .    .    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    1    .    .    0    .    0    0
.    .    1    .    .    0    .    0    0    .    .    .    .    .    0
.    0    0    .    .    .    .    .    0    .    0    0    .    .    .
.    .    0    0    0    0    .    .    .    .    .    0    0    0    0
.    .    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0
```

(b) Mask

Figure A.1: Entropy values and its corresponding mask created with the centroid of the 1st cluster.



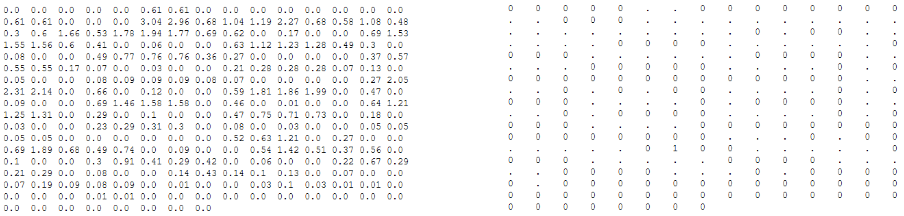
```
0.0  0.0  0.0  0.0  0.0  0.51 0.51 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.51 0.51 0.0  0.0  0.0  3.19 2.55 0.69 1.08 1.31 2.31 0.41 0.19 0.5  0.69
0.68 0.99 1.69 0.07 2.01 2.21 1.96 0.56 0.65 0.0  0.08 0.0  0.0  0.29 1.82
1.95 2.08 0.69 0.5  0.0  0.16 0.0  0.0  0.6  1.75 1.86 1.89 0.69 0.51 0.0
0.08 0.0  0.0  0.69 1.58 1.64 1.73 0.62 0.34 0.0  0.1  0.0  0.0  0.65 1.26
1.24 1.25 0.43 0.3  0.0  0.06 0.0  0.0  0.5  0.82 0.81 0.81 0.2  0.21 0.0
0.04 0.0  0.0  0.27 0.39 0.38 0.38 0.05 0.06 0.0  0.03 0.0  0.0  0.58 1.77
1.96 1.84 0.0  0.58 0.0  0.04 0.0  0.0  0.67 1.29 1.4  1.4  0.0  0.28 0.0
0.05 0.0  0.0  0.49 0.78 0.8  0.81 0.0  0.18 0.0  0.05 0.0  0.0  0.27 0.38
0.39 0.39 0.0  0.08 0.0  0.0  0.0  0.0  0.17 0.23 0.22 0.22 0.0  0.05 0.0
0.01 0.0  0.0  0.14 0.17 0.17 0.18 0.0  0.05 0.0  0.0  0.0  0.0  0.09 0.1
0.1  0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.44 0.68 1.35 0.0  0.17 0.0  0.0
0.68 2.03 0.68 0.6  1.07 0.0  0.08 0.0  0.0  0.61 1.8  0.62 0.48 0.82 0.0
0.13 0.0  0.0  0.37 1.24 0.41 0.29 0.41 0.0  0.1  0.0  0.0  0.21 0.66 0.18
0.14 0.18 0.0  0.06 0.0  0.0  0.06 0.24 0.08 0.08 0.09 0.0  0.0  0.0  0.0
0.01 0.1  0.01 0.01 0.01 0.0  0.0  0.0  0.0  0.0  0.01 0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

(a) Entropy values

```
0    0    0    0    0    .    .    0    0    0    0    0    0    0    0
.    .    0    0    0    .    .    .    .    .    .    .    .    .    .
.    .    .    1    .    .    .    .    .    0    0    0    0    .    .
.    .    .    .    0    .    0    0    .    .    .    .    .    .    0
0    0    0    .    .    .    .    .    .    0    0    0    0    .    .
.    .    .    .    0    0    0    0    .    .    .    .    .    .    0
0    0    0    .    .    .    .    0    0    0    0    0    0    .    .
.    .    0    .    0    0    0    0    .    .    .    .    0    .    0
0    0    0    .    .    .    .    0    .    0    0    0    0    .    .
.    .    0    0    0    0    0    0    .    .    .    .    0    0    0
0    0    0    0    .    .    .    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    .    .    .    0    .    0    0
.    .    .    .    .    0    0    0    0    .    .    .    .    .    0
0    0    0    .    .    .    .    .    0    0    0    0    .    .    .
.    .    0    0    0    0    0    .    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0
```

(b) Mask

Figure A.2: Entropy values and its corresponding mask created with the centroid of the 2nd cluster.



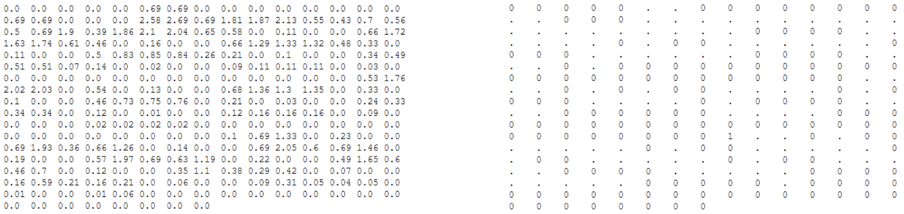
```
0.0  0.0  0.0  0.0  0.0  0.61 0.61 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.61 0.61 0.0  0.0  0.0  3.04 2.96 0.68 1.04 1.19 2.27 0.68 0.58 1.08 0.48
0.3  0.6  1.66 0.53 1.78 1.94 1.77 0.69 0.62 0.0  0.17 0.0  0.0  0.69 1.53
1.55 1.56 0.6  0.41 0.0  0.06 0.0  0.0  0.63 1.12 1.23 1.28 0.49 0.3  0.0
0.08 0.0  0.0  0.49 0.77 0.76 0.76 0.36 0.27 0.0  0.0  0.0  0.0  0.37 0.57
0.55 0.55 0.17 0.07 0.0  0.03 0.0  0.0  0.21 0.28 0.28 0.28 0.07 0.13 0.0
0.05 0.0  0.0  0.08 0.09 0.09 0.09 0.08 0.07 0.0  0.0  0.0  0.0  0.27 2.05
2.31 2.14 0.0  0.66 0.0  0.12 0.0  0.0  0.59 1.81 1.86 1.99 0.0  0.47 0.0
0.09 0.0  0.0  0.69 1.46 1.58 1.58 0.0  0.46 0.0  0.01 0.0  0.0  0.64 1.21
1.25 1.31 0.0  0.29 0.0  0.1  0.0  0.0  0.47 0.75 0.71 0.73 0.0  0.18 0.0
0.03 0.0  0.0  0.23 0.29 0.31 0.3  0.0  0.08 0.0  0.03 0.0  0.0  0.05 0.05
0.05 0.05 0.0  0.0  0.0  0.0  0.0  0.0  0.52 0.63 1.21 0.0  0.27 0.0  0.0
0.69 1.89 0.68 0.49 0.74 0.0  0.09 0.0  0.0  0.54 1.42 0.51 0.37 0.56 0.0
0.1  0.0  0.0  0.3  0.91 0.41 0.29 0.42 0.0  0.06 0.0  0.0  0.22 0.67 0.29
0.21 0.29 0.0  0.08 0.0  0.0  0.14 0.43 0.14 0.1  0.13 0.0  0.07 0.0  0.0
0.07 0.19 0.09 0.08 0.09 0.0  0.01 0.0  0.0  0.03 0.1  0.03 0.01 0.01 0.0
0.0  0.0  0.0  0.01 0.01 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

(a) Entropy values

```
0    0    0    0    0    .    .    0    0    0    0    0    0    0    0
.    .    0    0    0    .    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    0    .    0    0    .    .
.    .    .    .    0    0    0    0    .    .    .    .    .    .    0
0    0    0    .    .    .    .    .    .    0    0    0    0    .    .
.    .    .    0    0    0    0    0    .    .    .    .    0    .    0
0    0    0    0    0    0    0    0    0    0    0    0    0    .    .
.    .    0    .    0    .    0    0    .    .    .    .    0    .    0
0    0    0    .    .    .    .    0    .    0    0    0    0    .    .
.    .    0    .    0    0    0    0    .    .    .    .    0    .    0
0    0    0    .    .    .    .    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    .    .    .    0    .    0    0
.    .    .    .    .    0    1    0    0    .    .    .    .    .    0
0    0    0    .    .    .    .    .    0    0    0    0    .    .    .
.    .    0    0    0    0    .    .    .    0    .    0    0    0    0
0    .    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0
```

(b) Mask

Figure A.3: Entropy values and its corresponding mask created with the centroid of the 3rd cluster.



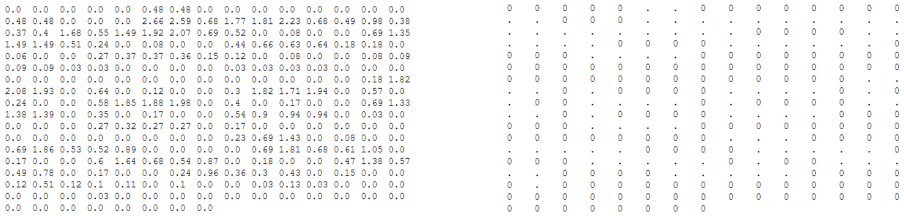
```
0.0  0.0  0.0  0.0  0.0  0.69 0.69 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.69 0.69 0.0  0.0  0.0  2.58 2.69 0.69 1.81 1.87 2.13 0.55 0.43 0.7  0.56
0.5  0.69 1.9  0.39 1.86 2.1  2.04 0.65 0.58 0.0  0.11 0.0  0.0  0.66 1.72
1.63 1.74 0.61 0.46 0.0  0.16 0.0  0.0  0.66 1.29 1.33 1.32 0.48 0.33 0.0
0.11 0.0  0.0  0.5  0.83 0.85 0.84 0.26 0.21 0.0  0.1  0.0  0.0  0.34 0.49
0.51 0.51 0.07 0.14 0.0  0.02 0.0  0.0  0.09 0.11 0.11 0.11 0.0  0.03 0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.53 1.76
2.02 2.03 0.0  0.54 0.0  0.13 0.0  0.0  0.68 1.36 1.3  1.35 0.0  0.33 0.0
0.1  0.0  0.0  0.46 0.73 0.75 0.76 0.0  0.21 0.0  0.03 0.0  0.0  0.24 0.33
0.34 0.34 0.0  0.12 0.0  0.01 0.0  0.0  0.12 0.16 0.16 0.16 0.0  0.09 0.0
0.0  0.0  0.0  0.02 0.02 0.02 0.02 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.1  0.69 1.33 0.0  0.23 0.0  0.0
0.69 1.93 0.36 0.66 1.26 0.0  0.14 0.0  0.0  0.69 2.05 0.6  0.69 1.46 0.0
0.19 0.0  0.0  0.57 1.97 0.69 0.63 1.19 0.0  0.22 0.0  0.0  0.49 1.65 0.6
0.46 0.7  0.0  0.12 0.0  0.0  0.35 1.1  0.38 0.29 0.42 0.0  0.07 0.0  0.0
0.16 0.59 0.21 0.16 0.21 0.0  0.06 0.0  0.0  0.09 0.31 0.05 0.04 0.05 0.0
0.01 0.0  0.0  0.01 0.06 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

(a) Entropy values

```
0    0    0    0    0    .    .    0    0    0    0    0    0    0    0
.    .    0    0    0    .    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    0    0    0    0    .    .
.    .    .    .    0    .    0    0    .    .    .    .    .    .    0
0    0    0    .    .    .    .    .    .    0    0    0    0    .    .
.    .    0    .    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    .    .
.    .    0    .    0    .    0    0    .    .    .    .    0    .    0
0    0    0    .    .    .    .    0    .    0    0    0    0    .    .
.    .    0    0    0    0    0    0    .    .    .    .    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    1    .    .    0    .    0    0
.    .    .    .    .    0    .    0    0    .    .    .    .    .    0
.    0    0    .    .    .    .    .    0    .    0    0    .    .    .
.    .    0    0    0    0    .    .    .    .    .    0    0    0    0
.    .    .    .    .    0    0    0    0    0    .    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0
```

(b) Mask

Figure A.4: Entropy values and its corresponding mask created with the centroid of the 4th cluster.



```
0.0  0.0  0.0  0.0  0.0  0.48 0.48 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.48 0.48 0.0  0.0  0.0  2.66 2.59 0.68 1.77 1.81 2.23 0.68 0.49 0.98 0.38
0.37 0.4  1.68 0.55 1.49 1.92 2.07 0.69 0.52 0.0  0.08 0.0  0.0  0.69 1.35
1.49 1.49 0.51 0.24 0.0  0.08 0.0  0.0  0.44 0.66 0.63 0.64 0.18 0.18 0.0
0.06 0.0  0.0  0.27 0.37 0.37 0.36 0.15 0.12 0.0  0.08 0.0  0.0  0.08 0.09
0.09 0.09 0.03 0.03 0.0  0.0  0.0  0.0  0.03 0.03 0.03 0.03 0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.18 1.82
2.08 1.93 0.0  0.64 0.0  0.12 0.0  0.0  0.3  1.82 1.71 1.94 0.0  0.57 0.0
0.24 0.0  0.0  0.58 1.85 1.88 1.98 0.0  0.4  0.0  0.17 0.0  0.0  0.69 1.33
1.38 1.39 0.0  0.35 0.0  0.17 0.0  0.0  0.54 0.9  0.94 0.94 0.0  0.03 0.0
0.0  0.0  0.0  0.27 0.32 0.27 0.27 0.0  0.17 0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.23 0.69 1.43 0.0  0.08 0.0  0.0
0.69 1.86 0.53 0.52 0.89 0.0  0.0  0.0  0.0  0.69 1.81 0.68 0.61 1.05 0.0
0.17 0.0  0.0  0.6  1.64 0.68 0.54 0.87 0.0  0.18 0.0  0.0  0.47 1.38 0.57
0.49 0.78 0.0  0.17 0.0  0.0  0.24 0.96 0.36 0.3  0.43 0.0  0.15 0.0  0.0
0.12 0.51 0.12 0.1  0.11 0.0  0.1  0.0  0.0  0.03 0.13 0.03 0.0  0.0  0.0
0.0  0.0  0.0  0.03 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

(a) Entropy values

```
0    0    0    0    0    .    .    0    0    0    0    0    0    0    0
.    .    0    0    0    .    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    0    0    0    0    .    .
.    .    .    .    0    0    0    0    .    .    .    .    .    .    0
0    0    0    .    .    .    .    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    .    .
.    .    0    .    0    0    0    0    .    .    .    .    0    .    0
.    0    0    .    .    .    .    0    .    0    0    0    0    .    .
.    .    0    .    0    0    0    0    .    .    .    .    0    0    0
0    0    0    .    .    .    .    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    .    .    .    0    0    0    0
.    .    .    .    .    0    0    0    0    .    .    .    .    .    0
0    0    0    .    .    .    .    .    0    .    0    0    .    .    .
.    .    0    0    0    0    .    .    .    .    .    0    0    0    0
0    .    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0
```

(b) Mask

Figure A.5: Entropy values and its corresponding mask created with the centroid of the 5th cluster.

Acronyms

AI Artificial Intelligence. 5–7, 9–11, 16, 18, 20, 21, 47, 48

ALE Accumulated Local Effects. 23, 24

CAV Concept Activation Vectors. 29–31, 36, 44, 46, 52

CNN Convolutional Neural Network. 15, 16, 25

HLC High Level Concept. 29–31

ICE Individual Conditional Expectation. 23

LIME Local Interpretable Model-Agnostic Explanations. 26, 27

M-plots Marginal Plots. 23, 24

MCTS Monte Carlo Tree Search. 10, 11, 17, 47, 48

NN Neural Network. 5, 6, 12–15, 17, 19, 29, 31–33, 36, 38, 39, 47, 48, 51, 52

PCA Principal component analysis. 44, 46, 52

PDP Partial Dependence Plot. 22, 23

PUCT Polynomial Upper Confidence Trees. 17

SGD Stochastic Gradient Descent. 15

TCAV Testing with CAV. 2, 5, 29–32, 36, 44, 46, 52

TD-learning Temporal difference learning. 13

UCB-1 Upper Confidence Bound-1. 11

UCT Upper Confidence Bound 1 applied to Trees. 11, 17

XAI Explainable Artificial Intelligence. 2, 5, 7, 18, 20, 21, 31, 52