Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Neuro-Argumentative Machine Learning with Images

Supervisor:
Prof. Francesca Toni

Co Supervisors:
Maurizio Proietti
Emanuele De Angelis
Avinash Kori

Second Marker:
Dr Wenjia Bai

Author:
Abdul Rahman Jacob

June 17, 2024

Abstract

We introduce a novel Neural Argumentative Learning (NAL) pipeline which integrates Assumption-Based Argumentation (ABA) with deep learning methods to learn from image data. The NAL pipeline's ability to derive comprehensible rules from images facilitates transparency and interpretability, addressing the challenges inherent in the black-box nature of deep learning models. This implementation is inspired by the strategies proposed in previous NeuroSymbolic AI research.

The NAL pipeline comprises a neuro-module, which segments and encodes images into facts using object-centric learning, and a symbolic module that applies ABA learning to learn new ABA frameworks with additional rules. These ABA frameworks can be mapped onto logic programs with negation as failure, enabling logical reasoning about image content. The main advantage of the NAL pipeline is its capacity to not only give insights into image classification tasks but also learn underlying rules with minimal examples.

Our experimental results, evaluated on synthetic datasets, demonstrate that the NAL pipeline is competitive with state-of-the-art alternatives in terms of accuracy and explainability. Overall, this work contributes to the advancement of NeuroSymbolic AI by providing a novel combination of neural and symbolic processes, showcasing the potential of neural symbolic learners in real-world applications.

Acknowledgements

I am deeply grateful to my supervisors, Prof. Francesca Toni, Maurizio Proietti, Emanuele De Angelis and Avinash Kori, for their invaluable guidance and support throughout this project.

Contents

Chapter 1

Introduction

1.1 Motivations

Over the last decade, Artificial intelligence (AI), namely deep learning, has become more prevalent in our everyday lives. They are widely used in a range of fields including healthcare [1, 2] and finance [3, 4]. However, as we rely more on these technologies to make critical decisions, concerns regarding their safety, reliability and explainability naturally emerge. Deep learning models are considered black boxes as their internal systems are not easily interpretable by domain experts, resulting in a lack of trust in their predictions by users. This has prompted the need for explainable AI (XAI).

One proposal for achieving XAI is through Neuro-Symbolic AI [5] which attempts to combine the powerful generalisation and pattern-recognition capabilities of deep learning models such as Neural Networks with the interpretability of symbols present in Symbolic AI. There are two main approaches to achieving this integration: extending neural networks with logical aspects or extending logical frameworks with neural constructs [6]. The former approach uses methods such as using logic as regularizers to ensure neural network predictions adhere to logic rules and/or encoding logical constructs into the network through differential operations. The latter approach, however, aims to create an interface between the neural network and logical framework to allow both systems to be optimised together. The various ways of implementing Neuro-Symbolic AI have led to the common question: What is the best way of representing Neuro-Symbolic AI? [5]

Computational argumentation (CA) has emerged in XAI as a powerful tool for reasoning and knowledge representation [7]. In argumentation theory, arguments are logical statements which can be attacked by rebuttals and undercutting. Most of the recent work in CA has been in argumentation frameworks which are structured ways of representing arguments. A popular argumentation framework is Assumption-Based Argumentation (ABA) [8] where arguments are deductions from inference rules containing assumptions which can be attacked by their contraries. ABA framework can be mapped into logic programs with negation as failure which has allowed

them to be leveraged in logic-based learning settings [9]. Through this method, ABA frameworks can be extended with learned rules based on positive and negative examples.

1.2 Objectives

Despite numerous works employing computational argumentation in Neuro-Symbolic AI, none has utilised assumption-based argumentation as the symbolic reasoner in their systems. This project aims to explore the unification of ABA and neural networks to be able to create a novel architecture in the Neuro-Symbolic space as envisioned in [10]. The specific goals of this project are:

- To investigate the use of logic programs in image classification. This involves finding methods to discretise image data into suitable predicates for the logic-based algorithm to learn from.
- To explore the use of computational argumentation, particularly ABA, within a logic-based learning setting to extract rules from images.
- To evaluate the output program from the pipeline, assessing their accuracy in image classification and their explainability.

1.3 Contribution

To accomplish the objectives of this project, we propose a neural argumentative learning pipeline (Figure 5.1) that combines the perceptual abilities of neural networks through slot attention with a novel logic-based learning algorithm in ABA learning. This pipeline can be applied to image classification tasks by reasoning about images through their constituent objects in an argumentative manner.

We investigate the effectiveness of the pipeline methodology through synthetic datasets including our own SHAPES dataset, which uses ASP rules to generate images that conform to the specifications of these rules.

Finally, we evaluate the pipeline by comparing it with neural and neural-symbolic architectures. Additionally, we leverage gradient visualisation techniques to generate explanations, enabling a more detailed comparison of explainability with the output programs of our pipeline.

1.4 Report Structure

In Chapter 2, we discuss some of the ethical considerations of machine learning and explainable AI

In Chapter 3, we introduce the theoretical concepts essential for the project. This includes object-centric learning via slot attention and clustering using K-means. We also delve into the syntax and semantics of assumption-based argumentation, exploring methods for learning such frameworks and interpreting logic programs. Finally, we discuss standard evaluation metrics and related architectures in the Neuro-Symbolic field.

In Chapter 4 we present the datasets used to investigate the pipeline, detailing the rules that govern the images within each dataset.

In Chapter 5 we outline the neural argumentative learning pipeline. We examine the inputs, the neural and symbolic sections, their interactions, and the format in which the model produces its outputs.

In Chapter 6 we introduce the experiments we conducted on the neural argumentative pipeline outlining the training process used, the tasks defined on each dataset and the methodology of evaluation.

In Chapter 7 we analyse the results of the classification tasks defined on each dataset, comparing the pipeline's performance with other neural and neuro-symbolic architectures.

In Chapter 8 we proposed some possible avenues of future research in the space of neural argumentative learning addressing the improvement of each section of the pipeline.

Chapter 2

Ethic Discussion

Ethics has been a significant topic in Artificial Intelligence and this project also has tried to keep these in mind. The neuro-argumentative architecture aims to make AI more explainable however, adding additional transparencies could lead to more points of misuse for AI. In this chapter, we discussed some ethical matters neuro-argumentative learning introduces.

Model Usage

The NAL architecture developed in this project processes images. Due to this, its application can be used in a wide range of scenarios including medical diagnosis, and vision systems such as self-driving cars and classifiers. Many reports show that humans are more likely to trust computers than other humans (including themselves) and hence shouldn't take predicted results from the model as the undisputed truth (especially for critical system decisions e.g. medical diagnosis). Due to the added explainability, users can view the reasons for an explanation and use their judgment to verify correctness.

Security of XAI

AI systems being unexplainable adds an extra layer of security to the system as a person cannot maliciously engineer the internal model for misuse. The transparency created by neuro-argumentative architectures' symbolic modules removes these layers as rules could be injected into the background knowledge which could lead to engineered prediction by the model. When training and using the model, the user must make sure to keep their system secure so such an event won't occur.

Data Usage & Data Protection

The data used in this project were publically available under the Creative Commons (CC) License which allows users to copy, redistribute, remix, transform and build upon the datasets as long as we give appropriate credit. When training the model, we should use an unbiased set of training data to prevent bias in the knowledge base and also ensure that any predefined rules don't introduce bias. The data must also

be obtained by a trusted source (with permission if necessary). Additionally, some data are categorised as personal or sensitive which means that data protection laws must be adhered to. Keeping this consideration in mind while working with models is critical to reducing the risk of biased models.

Overall, these ethical considerations are crucial to making sure that innovative solutions to problems don't cause unintentional harm to society. XAI including neuro-argumentative learners aims to help make AI more trustworthy and improve our relationship with the technology which is shaping our world today.

Chapter 3

Background

The integration of symbolic reasoning and deep learning models requires the incorporation of two distinct modules. Within this chapter, we discuss the background knowledge of the methodologies employed in this paper to implement each module. Specifically

- We overview the goal of object-centric learning for detecting objects in an unsupervised setting
- We introduce K-means clustering: An unsupervised learning algorithm used for grouping data points
- We discuss attention, its variants and introduce the slot attention architecture used to perform object-centric learning.
- We introduce the topic of Logic Programming and Computational Argumentation including our choice of argumentation framework: Assumption-Based Argumentation
- We discuss how ABA frameworks can be learned using logic-based learning approaches.
- We overview some evaluation metrics used in the space of machine learning
- We provide a literature review of some of the work done in the neuro-symbolic space

3.1 Object-Centric Learning

Object-centric learning is a method used to extract symbolic elements from images. It is a learning paradigm which learns meaningful features from images by spitting them into composite images. More formally, given an input image, it assumes that we can split it into K images where each image contains one object present in the input including the background (Figure 3.1). Object-centric learners are trained unsupervised and optimised using the reconstruction loss: The mean distance between

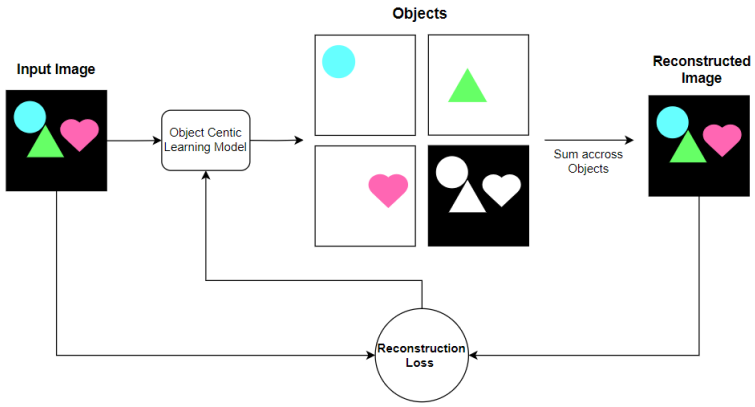

Figure 3.1: Object-Centric Learning Pipeline

the original image and the reconstructed image from the K object images.

The output of such models includes the reconstructed objects, a mask denoting which object a pixel belongs to and a latent representation of the object which can be used for downstream tasks (e.g. classification). Many model architectures exist to perform object-centric learning. MONet uses the UNet model to segment the image into objects and a variational autoencoder to produce latent representations of each of them while Slot Attention uses an attention mechanism with a recurrent process to estimate the latent space for each attended object. Despite the difference architectures present, most architectures perform well in object-centric learning tasks.

3.2 K-Means Clustering

K-means clustering is an unsupervised learning algorithm used to group instances within a feature space into distinct clusters. The algorithm consists of one main hyperparameter, K, which specifies the number of clusters. K-means begins by randomly selecting K points as initial centroids. Each data point is then assigned to the nearest centroid, forming K clusters. Subsequently, the centroids are updated by computing the mean position of all points in each cluster. Data points are then reassigned to the nearest updated centroid, and this process repeats until the centroids no longer change significantly between iterations. The goal of k-means clustering is to minimize within-cluster variance, which is achieved by reducing the sum of squared distances between each point and its assigned centroid (Equation 3.1). The algorithm terminates when centroid updates are minimal, indicating that the clusters are well-formed.

$$J = \frac{1}{N} \sum_{i=1}^{N} ||x^{(i)} - \mu_{c^{(i)}}||^2 \tag{3.1}$$

3.3 Attention Mechanisms

Attention mechanisms have been widely used in machine learning to enhance the performance of models in both computer vision and natural language processing. They work by allowing models to focus on specific parts of the input data which are relevant to the current execution step. This allows them to better capture long-term dependencies and contextual information within the data. There are several variants of attention mechanisms including dot product attention, self-attention and cross-attention which we discuss more in this section.

3.3.1 Dot-Product Attention

Dot-product attention is a specific type of attention mechanism commonly employed in transformer-based architectures. In dot-product attention, the input embeddings are associated with three vectors: Query, Key, and Value which are learnt during training.

$$\texttt{Attention}(Q, K, V) = \texttt{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{3.2}$$

The attention scores are computed by taking the dot product between a query vector and the set of key vectors. This calculation denotes the similarity between the query and key vectors, indicating how much attention each input element should receive. These scores are then normalised to produce the final attention weights. The weights are then applied to the value vectors producing scalars that represent the weighted combination of the corresponding value vectors. These weighted values represent the attended information, highlighting the most relevant features or elements from the input sequence.

Equation 3.2 shows scaled dot product attention which divides the dot product result with the square root of the vector dimension. This is done to provide stability during training by reducing the chances of encountering the vanishing gradients problem.

3.3.2 Self and Cross Attention

Self-attention and Cross-attention are both mechanisms which use the same procedure as dot-product attention to compute attention scores. However, the difference between these two mechanisms is where they derive their query, key and value vectors from and what they attend to:

Self-attention attends to different positions within the same input sequence. Each element in the sequence is associated with query, key, and value vectors. The attention scores are computed based on the similarity between the query vector for a given position and the key vectors for all positions in the sequence. This allows the model to weigh the importance of each element in the sequence relative to others, enabling it to capture dependencies and relationships within the sequence.

Cross-attention attends to information from a source sequence while processing a target sequence. The query vectors are derived from the target sequence, while the key and value vectors are derived from the source sequence. The attention scores are computed based on the similarity between the query vectors in the target sequence and the key vectors in the source sequence. This allows the model to focus on relevant information from the source sequence while processing the target sequence

3.4 Slot Attention Models

Slot Attention is a set-based attention model designed for image-processing tasks. Introduced in [11], Slot Attention integrates an attention mechanism to capture the structural representations of an image. It is an architecture which is used in object-centric learners to create an interface between perceptual representation (images) and structured representations (symbols).

3.4.1 Perceptual Backbone

Obtaining object-centric representation from unstructured data, such as images requires the use of a perceptual backbone to extract the relevant information for potential objects. This is done via a Convolutional Neural Network (CNN) which extracts lower-level features through a series of convolutional layers and pooling layers. The Convolutional layers apply filters (also known as kernels) that convolve over the image, creating feature maps to capture patterns across various regions. The pooling layer reduces the feature map's spatial dimensions, which helps lower the computational complexity for subsequent steps in the architecture.

The perceptual backbone is also augmented with a positional embedding since Slot Attention is invariant to position. The input image is converted to a grid-like representation where each point on the grid is associated with a 4-dimensional feature vector that encodes its distance (normalized to [0, 1]) to the borders of the feature map along each of the four cardinal directions. This extra information is used to capture the spatial positions, improving Slot Attention's ability to recognise and segment objects.

3.4.2 Slot Attention Module

The Slot Attention module aims to map a set of N input feature vectors to K output vectors referred to as slots. These slots are latent representations of the different objects in the image. Algorithm 1 describes how the modules work in pseudo-code. The initial inputs to the Slot Attention modules are the feature maps augmented with their positional embedding and slots which have dimensions $\mathbb{R}^{K \times D_{\text{slots}}}$. The slots are randomly initialised by taking IID samples from a Gaussian distribution.

Algorithm 1: Slot Attention Module [11]

Input : inputs $\in \mathbb{R}^{N \times D_{\text{inputs}}}$, slots $\sim \mathcal{N}(\mu, \text{diag}(\sigma)) \in \mathbb{R}^{K \times D_{\text{slots}}}$
Layer params: k, q, v: linear projections for attention; GRU; MLP; LayerNorm (x3)
inputs = LayerNorm(inputs)
for $t = 0 \dots T$ **do**
  slots_prev = slots
  slots = LayerNorm(slots)
  attn = Softmax($\frac{1}{\sqrt{D}} k(\text{inputs}) \cdot q(\text{slots})^T$, axis = 'slots')
  updates = WeightedMean(weights = attn + ϵ, values = v(inputs))
  slots = GRU(state = slots_prev, inputs = updates)
  slots += MLP(LayerNorm(slots))
return slots

An iterative attention mechanism is used to map from its inputs to the slots. For each iteration, an attention score is calculated via cross-attention 3.3. The query vectors are associated with the slots while the key vectors are associated with the inputs. The attention weights are then derived by applying a temperature softmax (dividing the attention score by the size of the dimensions). The softmax is normalised over the slot which induces competition between slots, resulting in each feature being assigned to a slot. Finally, the weights and slots are updated. A weighted mean is used to update the attention weights due to the normalisation over the slots rather than the inputs. A GRU is used to update the slots which learns how to update the slot representation after each iteration. Common practices such are using LayerNorms are also used to help stabilise training and allow for faster convergence.

At each iteration, the Slot Attention mechanism assigns more features to slots which have been previously assigned similar features. This process can be thought of as a meta-clustering algorithm of latent features. The slots return will be associated with features which correspond to the objects present in the input data. These latent features can then be used for downstream tasks such as object discovery and set prediction.

3.5 Logic Programming

3.5.1 Answer Set Programming

Answer Set Programming (ASP) is a form of declarative programming oriented towards combinatorial problems (e.g. search)[12]. It is rooted in the area of Knowledge Representation and Reasoning (including non-monotonic reasoning). ASP is based on stable model semantics which deals with computing stable models (or answer sets) from logic programs with negation as failure.

Definition 1 (ASP Syntax) ASP programs consist of rules which are written in the form

$$\texttt{p :- q}_1, \ldots, \texttt{q}_k, \texttt{ not q}_{k+1}, \ldots, \texttt{not q}_n$$
$$\text{or}$$
$$\texttt{:- q}_1, \ldots, \texttt{q}_k, \texttt{ not q}_{k+1}, \ldots, \texttt{not q}_m$$

where $\texttt{p}, \texttt{q}_1, \ldots \texttt{q}_n, \texttt{q}_1, \ldots \texttt{q}_m$ are atoms, $\texttt{k} \geq 0, \texttt{n} \geq 0, \texttt{m} \geq 1$ and `not` denotes negation as failure. `p` is referred to as the head of a rule and $\texttt{q}_k, \texttt{ not q}_{k+1}, \ldots, \texttt{not q}_n$ as the body. rules without a head are known as constraints and rules with an empty body are referred to as fact.

3.5.2 Answer Sets

Answer sets semantics aims to determine the meaning of an answer set program. Given a set of rules and facts, An interpretation is a function that specifies the meaning of each symbol in the domain. The answer set of a program is a consistent and minimal interpretation of the program rules. It represents a set of facts (or atoms) that satisfy all the rules without leading to any contradictions.

Example 1 Below is a simple ASP program P:

```
road(london, bath).  road(london, oxford).  road(oxford, manchester).
road(manchester, London).  accident(london, oxford).

route(X,Y) :- road(X,Y), not accident(X,Y).
route(X,Y) :- route(Y,X).
```

The answer set of ASP program P denoted as $ans(P)$ is calculated by firstly grounding the program $ground(P)$ using substitution to produce P' which contains no variables and only includes constants available from the domain

Example 2 ASP program $ground(P) = P'$:

```
road(london, bath).  road(london, oxford).  road(oxford, manchester).
road(manchester, London).  accident(london, oxford).

route(london, bath) :- road(london, bath), not accident(london, bath).
route(london, bath) :- route(bath, london).
route(bath, london) :- route(london, bath).
...
```

The answer set is then computed by iteratively including the facts that appear in P and facts that can be deduced through these facts and the grounded rules.

$ans(p) =$
{
road(london, bath), road(london, oxford),
road(oxford, manchester), road(manchester, london),
accident(london, oxford), route(london, bath),
route(oxford, manchester), route(manchester, london),
route(london, manchester), route(manchester, oxford),
route(bath, london)
}

Notice how the fact `route(london,oxford)` doesn't appear in $ans(P)$. This is due to the fact that we can prove `accident(london,oxford)` using the facts from the program. Hence having `route(london,oxford)` in our answer set contradicts the rule `route(london,oxoford):- road(london,oxford), not accident(london,oxford)`

3.5.3 Reasoning over Answer Sets

Example 1 falls into the class of stratified programs which have the property that one can find an ordering for the evaluation of the rules in the program, such that the value of negative literals can be predetermined [13]. For example, when evaluating the negative literal `not accident(X,Y)` we firstly check if `accident(X,Y)` is true. If so then the negative literal is false and the rule is not applied rule. If it is false then the rule is applied. This forms a chain of evaluation which is used to find the answer sets of a program with negation as failure.

When a program chain of execution contains a cycle i.e. there are rules which depend on each other, it results in the program being unstratified. One case of this is when two or more predicates are mutually defined over "`not`". This is seen in the following example:

Example 3 Unstratified ASP Program Q

```
man(john).
working(X) :- man(X), not relaxing(X).
relaxing(X) :- man(X), not working(X).
```

Example 3 has two answer sets which denote the different possibilities which are consistent with the information presented in program Q:

$$ans_1(Q) = \{\texttt{man(john)}, \texttt{working(john)}\}$$
$$ans_2(Q) = \{\texttt{man(john)}, \texttt{relaxing(john)}\}$$

To reason over the truth of an atom q in programs which emit more than one answer set, one can choose to reason bravely or cautiously. **Brave Reasoning** would conclude that q is true in an ASP program if it appears in some answer set. However **Cautious Reasoning** would conclude that q is true in an ASP program if it appears

in all answer sets.

Cautious reasoning can be seen as trying to find one explanation for the scenario described in the program. This is represented by the intersection of all answer sets. This reasoning can sometimes be too strong leading to no explanation for an atom that only appears in some of the solutions set. Brave reasoning is represented by the union of all answer sets. It contains all the possible solution sets that the program encodes. This is a weaker constraint and allows atoms `working(john` and `relaxing(john)` in example 3 to be considered true. Choosing the best way to reason is dependent on the problem as each reasoning method has drawbacks in the consequences they can derive.

3.6 Computational Argumentation

Argumentation is a powerful reasoning method that many of us use in everyday conversations to support or attack an idea/theory given by another person(s). An argument can be seen as a set of logical statements whereby each statement's validity can be determined by whether or not it can 'defend' itself against other arguments which attack it (Dung 1995 [14]). Through this definition, arguments are modelled as directed graphs where the nodes are the logical statements and the edges show the direction of attack between two nodes.

AF allow us to investigate different arguments regardless of the intricate properties of the arguments themselves. Because arguments naturally follow a non-monotonic logical framework, we can use argumentation frameworks to perform non-monotonic reasoning (Figure 3.2). The abstract natures of these steps have allowed different formalisms to be built upon argumentation to perform non-monotonic entailment including ASPIC+[15] and Assumption-Based Argumentation (ABA) [8] which we will discuss more in the next section.

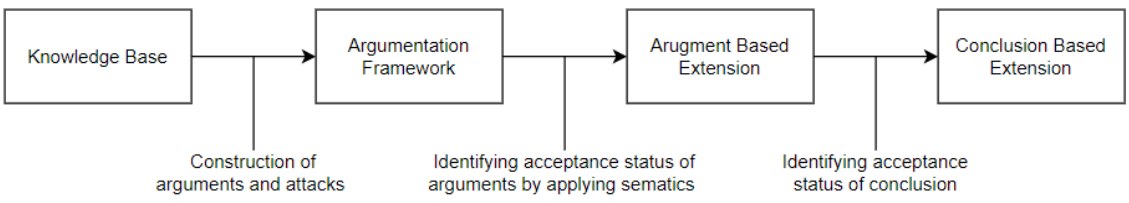

Figure 3.2: Argumentation for inference [16]

3.6.1 Assumption-Based Argumentation

Assumption-based argumentation (ABA) is a popular framework used for reasoning argumentative in non-monotonic settings. It is an instance of Abstract Argumentation whereby instead of arguments and attacks between arguments being abstract

and primitive, they are deductions (using inference rules in an underlying logic) supported by assumptions. An attack by one argument against another is achieved when a deduction of the contrary of an assumption supporting the argument can be made.

Defined by [8], an ABA framework is a tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ where:

- $\langle \mathcal{L}, \mathcal{R} \rangle$ is a deductive system where $\mathcal{L}$ is the language and $\mathcal{R}$ is the set of (inference) rules.
- $\mathcal{A} \subseteq \mathcal{L}$ is the set of assumptions
- $\overline{\ }$ is a total mapping from $\mathcal{A}$ to $\mathcal{L}$ where $\bar{a}$ is referred to a contrary of a ($a \in \mathcal{A}$)

Rules $\mathcal{R}$ are assumed to have the form $\rho : h \leftarrow b_0, \ldots, b_m$ ($m \geq 0, b_i \in \mathcal{L}$ for $0 \leq i \leq m$). h is referred to as the head, $b_0, \ldots, b_m$ as the body of the rule $h \leftarrow b_0, \ldots, b_m$ and ρ as the identifier of the rule. A rule ρ with an empty body is referred to as a fact and we also assume $\mathcal{L}$ to be finite.

Example 4 An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ may consist of:

$$\mathcal{L} = \{p(X), q(X), r(X), s(X), t(X) \mid X \in \{a, b\}\}$$

$$\begin{aligned}\mathcal{R} = \{&\rho_1 : p(X) \leftarrow q(X),\\ &\rho_2 : q(X) \leftarrow r(X),\\ &\rho_3 : s(X) \leftarrow p(X), t(X),\\ &\rho_4 : r(a) \leftarrow,\ \rho_5 : p(b) \leftarrow\}\end{aligned}$$

$$\mathcal{A} = \{t(X)\}$$

$$\overline{t(X)} = r(X)$$

The rules here are written in schematic i.e. $t(X)$. X is considered a variable which can be instantiated across the language $\{a, b\}$ hence $t(X)$ is shorthand for $\{t(a), t(b)\}$ To reason in this argumentation framework we need to construct arguments given a knowledge base. In ABA arguments are deductions of claims using the rules in the deductive system $(\mathcal{L}, \mathcal{R})$. More formally:

Definition 2 (Argument): An argument for the claim $c \in \mathcal{L}$ supported by $A \subseteq \mathcal{A}$ and $R \subseteq \mathcal{R}$ (denoted as $A \vdash_R c$) is a finite tree with nodes labelled by sentences in $\mathcal{L}$ or by τ denoting *true* (Figure 3.3), the roots labelled by c, the leaves either *true* or assumptions in A, and non-leaves c' with, as children, the elements of the body of some rule in R with the head c'.

Example 5 Derived Deductions from the ABA Framework in Example 4:

$$\{\} \vdash_{\{\rho_2,\rho_4\}} q(a) \qquad \{\} \vdash_{\{\rho_5\}} p(b) \qquad \{t(a)\} \vdash_{\{\}} t(a)$$

$$\{t(a)\} \vdash_{\{\rho_1,\rho_2,\rho_3,\rho_4\}} s(a)$$

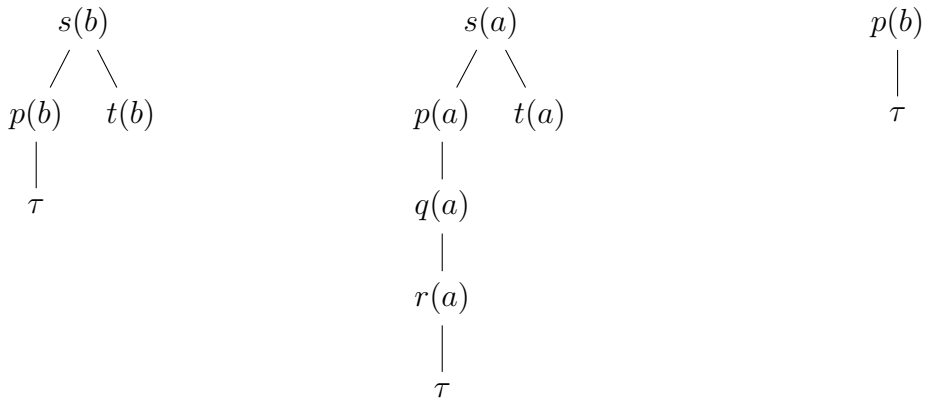

Figure 3.3: Arguments $\{t(b)\} \vdash_{\{\rho_3,\rho_4\}} s(b)$, $\{t(a)\} \vdash_{\{\rho_1,\rho_2,\rho_3,\rho_4\}} s(a)$ and $\{\} \vdash_{\{\rho_5\}} p(b)$ represented as trees

An interesting case is arguments of the form $\{\alpha\} \vdash_{\{R\}} \beta$ where $\alpha, \beta \in \mathcal{A}$. This intuitively means that an assumption has validated another assumption in the framework. To allow ABA frameworks to be mapped to logic programs we keep an assumption acceptance independent from other assumptions in $\mathcal{A}$. This restriction is referred to Flat ABA framework:

Definition 3 (Flat ABA Framework): An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ is flat if and only if for every $A \subseteq \mathcal{A}$, A is closed.

Note that if an assumption is not the head of a rule (a conclusion) then we guarantee the ABA framework to be flat. In the rest of this report, we only deal with Flat ABA frameworks.

Attacks in ABA Frameworks can also be formalised using the deductions which describe an argument. These arguments can only be directed at assumptions which are the implicit beliefs or premises that are taken for granted and often unstated in an argument.

Definition 4 (Attack): An Argument $A \vdash_{R_1} c_1$ *attacks* argument $A_2 \vdash_{R_2} c_2$ if and only if $s_1 = \overline{a}$ for some $a \in A_2$

Example 6 Attacks on some arguments derived from Example 5

$\{\} \vdash_{\{\rho_4\}} r(a)$ attacks $\{t(a)\} \vdash_{\{\rho_1,\rho_2,\rho_3,\rho_4\}} s(a)$ because $\overline{t(a)} = r(a)$

$\{\} \vdash_{\{\rho_4\}} r(a)$ attacks $\{t(a)\} \vdash_{\{\}} t(a)$, again because $\overline{t(a)} = r(a)$

3.6.2 ABA Semantics

We use argumentation semantics to find sets of arguments which can be accepted given the relationships described in the argumentation frameworks.

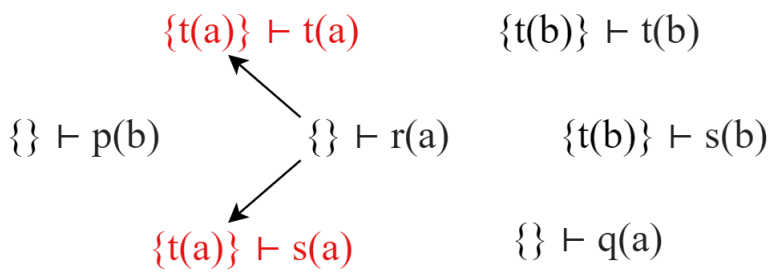

Figure 3.4: Attacks as directed graph

Definition 5 (Abstract Argumentation): Given an ABA framework let $Args$ be the set of arguments and $Att = \{(\alpha, \beta) \in Args \times Args | \alpha$ attacks $\beta\}$ then $(Args, Att)$ is an Abstract Argumentation [14].

Definition 6 (Stable Semantics): $S \subseteq Args$ is a stable extension iff

(i) $\nexists \alpha, \beta \in S$ such that $(\alpha, \beta) \in Att$ (i.e S is conflict-free)

(ii) $\forall \beta \in Args \backslash S, \exists \alpha \in S$ such that $(\alpha, \beta) \in Att$ (i.e S attacks all it doesn't contain)

From Example 4 we can deduce the set of arguments (we omit the set of rules which builds the argument)

$$Args = \{\{\} \vdash r(a),\ \{\} \vdash p(b),\ \{\} \vdash q(a),\ \{t(a)\} \vdash t(a),\\ \{t(a)\} \vdash s(a),\ \{t(b)\} \vdash t(b),\ \{t(b)\} \vdash s(b)\}$$

By definition 6 we find that this framework has only one stable extension $\Delta = \{p(b), r(a), q(a), s(b), t(b)\}$. Note how $s(a)$ and $t(a)$ is not included as the argument can be attack by $r(a)$ as depicted in the directed graph shown in figure 3.4

Definition 7 (Type of Stable Extension): Sometimes we can have frameworks that emit multiple stable extensions. ABA frameworks can be reasoned using different semantics which allow us to decide whether a sentence is accepted. These include:

(i) **Sceptical (Cautious) Semantics**: An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ is said to cautiously cover a sentence $c \in \mathcal{L}$ under stable semantics if for every stable extension $\mathcal{S}$ of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$, c is the clam of an argument $\alpha \in \mathcal{S}$

(ii) **Credulous (brave) Semantics**: An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ is said to bravely cover a sentence $c \in \mathcal{L}$ under stable semantics if there exists a stable $\mathcal{S}$ of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$, c is the clam of an argument $\alpha \in \mathcal{S}$

3.7 Learning ABA Frameworks

ABA frameworks produce a structured knowledge representation that can be used to reason argumentatively. They are also an instance of logic programs whereby the

assumptions are negation as failure literals ($not\ p$)and the contrary of this being p [9]. These properties allow us to learn the structure of an argument through logic-based learning approaches [17, 18]. The learnt framework represents explainable concepts through argumentation. This process is known as ABA Learning.

ABA Learning takes elements from inductive learning programs to learn concepts. It is driven via training data which is a non-empty set of positive labels and a (possibly empty) set of negative examples. With the help of background knowledge (which itself is an ABA framework), the learner aims to introduce concepts to cover all the positive examples and none of the negative.

3.7.1 ABA Learning Task

Definition 8 (Coverage): Given an ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$, an example e is covered by $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle \models e$ iff $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle \models e$ and is not covered by $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle \models e$ iff $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle \not\models e$

Definition 9 (Background Knowledge): A background is defined as a ABA Framework $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$. (We omit the $\mathcal{L}$ for defining ABA frameworks thus forth as this can be reconstructed by the other components of the framework.)

Example 7 Background Knowlege $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$

$$\mathcal{R} = \{\rho_1 : s_1(a) \leftarrow,\ \rho_2 : s_1(b) \leftarrow,\ \rho_3 : s_2(b) \leftarrow\}$$

Positive Examples: $\mathcal{E}^+ = \{\ c_1(a)\ \}$
Negative Examples: $\mathcal{E}^- = \{\ c_1(b)\ \}$

We can ascribe concrete readings to the abstract predicates in the background knowledge from example 7 inspired by the type of tasks performed in later sections. $s_i(X)$ represents the slot (properties) active in image $X \in \{a, b\}$ and c_i represents the concept we are trying to extract from the set of images.

Example 8 Learnt ABA framework $\langle \mathcal{R}', \mathcal{A}', \overline{\ }' \rangle$

$$\begin{aligned}\mathcal{R} = \{&\rho_1 : s_1(a) \leftarrow,\\ &\rho_2 : s_1(b) \leftarrow,\\ &\rho_3 : s_2(b) \leftarrow,\\ &\rho_4 : c_1(X) \leftarrow s_1(X), \alpha(X),\\ &\rho_5 : c\_\alpha(X) \leftarrow s_2(X)\}\end{aligned}$$

$$\mathcal{A} = \{\alpha(X)\}$$

$$\overline{\alpha(X)} = c\_\alpha(X)$$

Example 8 Illustrates the final ABA Framework learnt when completing the ABA goal successfully learning the concept c_1. This goal is defined formally by [9]:

Definition 10 (ABA Learning Goal): Given a background knowledge $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$, positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$, with $\mathcal{E}^+ \cap \mathcal{E}^- = \emptyset$, the goal of ABA learning is to construct $\langle \mathcal{R}', \mathcal{A}', \overline{\ }' \rangle$ such that $\mathcal{R} \subseteq \mathcal{R}'$, $\mathcal{A} \subseteq \mathcal{A}'$ and for all $\alpha \in \mathcal{A}$, $\overline{\alpha}' = \overline{\alpha}$ such that

(i) (Existence): $\langle \mathcal{R}', \mathcal{A}', \overline{\ }' \rangle$ admits at least one extension under the chosen ABA semantics

(ii) (Completeness): for all $e \in \mathcal{E}^+$, $\langle \mathcal{R}', \mathcal{A}', \overline{\ }' \rangle \models e$

(iii) (Consistency): for all $e \in \mathcal{E}^-$, $\langle \mathcal{R}', \mathcal{A}', \overline{\ }' \rangle \not\models e$

From definition 10, we see that the ABA framework $\langle \mathcal{R}', \mathcal{A}', \overline{\ }' \rangle$ in example 8 admits one stable extension which covers all the positive examples and none of the negative examples for both semantics. To achieve the goal, we make use of transformation rules to algorithmically augment the framework.

3.7.2 Transformation Rules

Notation: Similarly to [9] we assume that all rules are normalised meaning they are written in the form:

$$\rho : p_0(X_0) \leftarrow eq_1, \ldots, eq_k, p1(X_1), \ldots p_n(X_n)$$

where $p_i(X_i)$ for $0 \geq i \geq n$ is an atom are in $\mathcal{L}$ and eq_i for $0 \geq i \geq k$ is an equality between variables in $X_0, \ldots, X_i$ e.g. $X_1 = a$

Transformation rules allow us to construct new ABA frameworks for ABA learning. These are:

- ***Rote Learning***: Introduces an argument from a ground atom $p(t)$ by adding $\rho : p(X) \leftarrow X = t$ to $\mathcal{R}$ Hence, $\mathcal{R}' = \mathcal{R} \cup \{p\}$
  In ABA learning setting, rote learning is usually applied to positive examples
  Example: Considering the example from 7 we apply rote learning on the positive example $c_1(a)$ to obtain $\mathcal{R}' = R \cup \{\rho_4 : c_1(X) \leftarrow X = a\}$

- ***Equality Removal***: Generalises a rule by removing an equality from its body. More formally we replace a rule $\rho_1 H \leftarrow eq_1, Eqs, B$ in $\mathcal{R}$ with $\rho_2 : H \leftarrow Eqs, B$ obtaining $\mathcal{R} = (\mathcal{R} \backslash \{C_1\}) \cup \{C_2\}$
  Here $eq1$ is an equality, Eqs is a (possibly empty) set of equalities and B is a (possibly empty) of atoms

- ***Folding*** Generalises a rule by replacing some atoms in its bodies with their consequences using a rule in $\mathcal{R}$. I.e. Given Rule $\rho_1 : H \leftarrow Eqs_1, B_1, B_2$ and $\rho_2 : K \leftarrow Eqs_1, Eqs_2, B_1$ replace ρ_1 by $\rho_3 : H \leftarrow Eqs_2, K, B_2$. Thus $\mathcal{R}' = (\mathcal{R} \backslash \{\rho_1\}) \cup \{\rho_3\}$
  Example: We can apply folding to $\rho_4 : c_1(X) \leftarrow X = a$ to ρ_1 to obtain $\rho_5 : c_1(X) \leftarrow X = a, s_1(a)$.

- ***Subsumption*** Allows the removal of rules which are $\theta-$subsumed by rules in the background knowledge [19]. For example if $\mathcal{R}$ contains rules $\rho_1 : H \leftarrow Eqs_1, B_1$ and $\rho_2 : H \leftarrow Eqs_2, B_2$
  Example: A rule $flies(X) \leftarrow X = b$ would be subsumed by $flies(X) \leftarrow bird(X)$ [9] if the only instances of b is within the rule $bird(b)$

- ***Assumption Introduction*** Introduces exceptions to rules by replacing $\rho_1 : H \leftarrow Eqs, B$ by $\rho_2 : H \leftarrow Eqs_1, B_1, \alpha(X)$ where X is a tuple of variables taken from $vars(H) \cup vars(B)$ and $\alpha(X)$ is a assumption with contrary $\chi(X)$ Hence $\mathcal{R}' = (\mathcal{R} \backslash \{\rho_1\}) \cup \{\rho_2\}, \mathcal{A}' = \mathcal{A} \cup \{\alpha(X)\}$, $\overline{\alpha(X)}' = \chi(X)$ and $\overline{\beta}' = \overline{\beta}$ for all $\beta \in \mathcal{A}$ (I.e the remaining contraries are unmodified)

In summary, ***Assumption Introduction*** and ***Rote Learning*** allow us to expand or constrain the space of possible framework so we cover positive examples and no negative example ***Equaity Removal***, ***Folding*** and ***Subsumption*** generalises rules to cover more of the search space.

3.7.3 ABA Learning Algorithm

To allow ABA frameworks to be learnt, the rules in section 3.7.2 have to be performed in a certain order to ensure that the search space is adequately searched. Proposed in [9] a general strategy for learning ABA framework is illustrated in Figure 3.5.

Input: Background Knowledge: ABA Framework $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$
Training Data: $\mathcal{E}^+$ and $\mathcal{E}^-$ with $\mathcal{E}^+ \cap \mathcal{E}^- = \emptyset$

Algorithm:
Step 1 [Target Selection]: Select a predicate p such that $\exists p(c) \in \mathcal{E}^+$
Step 2 [Rule Introduction]: For each positive example $p(c)$ in $\mathcal{E}^+$ add the rule $p(X) \leftarrow X = c$ to $\mathcal{R}$ via Rote Learning
Step 3 [Rule Generation]: For each rule in $\mathcal{R}$, perform one of the following transformation:
1. Subsumption - To possibly remove the rule from $\mathcal{R}$
2. Folding and Equality Removal Repeatedly - Until all constants are removed

Step 4 [Exception Learning]: Repeat until for all $p(d) \in \mathcal{E}^-$, $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle \nvdash p(d)$
1. Select $p(d) \in \mathcal{E}^-$
2. Select from $\mathcal{R}$ a rule $\rho : p(X) \leftarrow Eqs, B$ such that we can construct an argument for $p(d)$ with ρ as a top rule
3. Construct a set $A = \{a_1(Y_1), \ldots, a_k(Y_k)\} \subseteq B$ that can generate two ground instances A^+ and A^- such that
   (a) For every example $p(e) \in \mathcal{E}^+$ we can build an argument for $p(e)$ with ρ as the top rule and the ground atoms in A^+ are children of $p(e)$
   (b) For every example $p(e) \in \mathcal{E}^-$ we can build an argument for $p(e)$ with ρ as the top rule and the ground atoms in A^- are children of $p(e)$
   (c) Appy Assumption Introduction by adding the new assumption $\alpha(Y_1), \ldots, (Y_k)$ with contrary $c\_\alpha(Y_1), \ldots, (Y_k)$ to the body of the rule ρ
4. Add $c\_\alpha(consts(A^+))$ to $\mathcal{E}^-$ and add Add $c\_\alpha(consts(A^-))$ to $\mathcal{E}^+$

Step 5 [Example Removals]: Remove examples from predicate p form the training data
Step 6 [Termination Check]: If $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ is complete and consistent then terminate. Otherwise GOTO *Step 1*

Figure 3.5: ABA Learning Algorithm [18]

Example 9 We demonstrate how the ABA learning algorithm can generate the learnt ABA framework (example 8) from example 7

(ITERATION 1) **Input:**
$\mathcal{R} = \{\ \rho_1 : s_1(a) \leftarrow,\ \rho_2 : s_1(a) \leftarrow,\ \rho_3 : s_2(b) \leftarrow,\ \}$
$\mathcal{E}^+ = \{c_1(a)\} \qquad \mathcal{E}^- = \{c_1(b)\}$

Algorithm:
Step 1 [Target Selection]: Select predicate: c_1
Step 2 [Rule Introduction]:
Apply Rote Learning to obtain $\mathcal{R}_1 = \mathcal{R} \cup \{\rho_4 : c_1(X) \leftarrow X = a\}$
Step 3 [Rule Generation]:
Apply Folding using p_1 and Equality Removal to generalise rule p_4
Folding: $\rho_5 : c_1(X) \leftarrow X = a, s_1(X)$
Equality Removal: $\rho_6 : c_1(X) \leftarrow s_1(X)$
Hence we obtain $\mathcal{R}_2 = (\mathcal{R}_1 \backslash \{\rho_4\}) \cup \{\rho_6\}$
Step 4 [Exception Learning]: Repeat until for all $p(d) \in \mathcal{E}^-$, $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle \nvdash p(d)$
1. We firstly select $c_1(b) \in \mathcal{E}^-$
2. We select $\rho_6 : c_1(X) \leftarrow s_1(X)$ as the rule to construct the argument for
3. We Construct a set $A = \{c_1(Y)\}$ with $A^+ \in \{\{c_1(a)\}\}$ and $A^- \in \{\{c_1(b)\}\}$
4. Apply Assumption Introduction replacing $\rho_6 : c_1(X) \leftarrow s_1(X)$ with $\rho_7 : c_1(X) \leftarrow s_1(X), \alpha(X)$
Obtain $\mathcal{R}_3 = (\mathcal{R}_2 \backslash \{\rho_6\}) \cup \{\rho_7\}$ and $\overline{\alpha(X)} = c\_\alpha(X)$
5. Add contraries to training examples
$\mathcal{E}^+ = \{c_1(a), c\_\alpha(b)\}$ and $\mathcal{E}^- = \{c_1(b), c\_\alpha(a)\}$
Since none of the negative examples are covered we can move to the next step
Step 5 [Example Removals]: Remove examples from predicate p from the training data
$\mathcal{E}^+ = \{c\_\alpha(b)\}$ and $\mathcal{E}^- = \{c\_\alpha(a)\}$
Step 6 [Termination Check]: Since $\langle \mathcal{R}_3, \mathcal{A}', \overline{\ }' \rangle$ not complete and consistent we return to *Step 1*

(ITERATION 2) **Input:**
$\mathcal{R} = \{\ \rho_1 : s_1(a) \leftarrow,\ \rho_2 : s_1(a) \leftarrow,\ \rho_3 : s_2(b) \leftarrow,\ \rho_7 : c_1(X) \leftarrow s_1(X), \alpha(X)\ \}$
$\mathcal{E}^+ = \{c\_\alpha(b)\} \qquad \mathcal{E}^- = \{c\_\alpha(a)\}$

Algorithm:
Step 1 [Target Selection]: Select predicate: $c\_\alpha(b)$
Step 2 [Rule Introduction]:
Apply Rote Learning to obtain $\mathcal{R}_4 = \mathcal{R}_3 \cup \{\rho_8 : c\_\alpha(X) \leftarrow X = b\}$
Step 3 [Rule Generation]:
Apply Folding using p_3 and Equality Removal to generalise rule p_8
Folding: $\rho_9 : c\_\alpha(X) \leftarrow X = b, s_2(X)$
Equality Removal: $\rho_{10} : c\_\alpha(X) \leftarrow s_2(X)$
Hence we obtain $\mathcal{R}_5 = (\mathcal{R}_4 \backslash \{\rho_8\}) \cup \{\rho_{10}\}$
Step 4 [Exception Learning]: Since none of the negative examples are covered we can move to the next step
Step 5 [Example Removals]: Remove examples from predicate p from training data
$\mathcal{E}^+ = \emptyset$ and $\mathcal{E}^- = \emptyset$
Step 6 [Termination Check]: $\langle \mathcal{R}_5, \mathcal{A}', \overline{\ }' \rangle$ is complete and consistent so we terminate

The final ABA Framework is $\langle \mathcal{R}_5, \mathcal{A}', \overline{\ }' \rangle$ which corresponds to example 8

3.8 ABA and Logic Programming

ABA frameworks can be converted into logic programs by mapping the contraries as negation as failure. For all contraries $\overline{\alpha(X)} = p(X)$ we replace $\alpha(X)$ with $not\ p(X)$, in the set of rules $\mathcal{R}$ and remove it from the language $\mathcal{L}$. The resulting frameworks can then be written in logic programming languages such as Prolog and Answer Set Programming (ASP) to automatically compute stable extensions. This transformation allows us to leverage logic-based approaches to learning ABA frameworks and has led to the following tools being developed:

3.8.1 ABALearn

ABAlearn [18] is an Automated Logic-Based Learning System for learning ABA Frameworks. It takes an ABA Framework written as a Prolog program, positive and negative examples and runs the algorithm defined in Figure 3.5 producing a new program with the learnt rules. The program can then be executed with new facts to be able to test for satisfiability

3.8.2 ASP-ABALearn

ASP-ABALearn [17] is a tool which leverages ASP to learn new ABA frameworks. To achieve this, it first encodes a framework to an ASP program by converting rules to ASP syntax (Example 10). The learning goal is then encoded by writing positive examples as `:- not e` and negative examples as `:- e`. This states that any solution learnt must cover all positive examples and none of the negative examples.

Example 10 ASP Encoding of Example 7

```
slot_1(A) :- A= img_1.
slot_1(A) :- A= img_2.
slot_2(A) :- A= img_2.
```

$\mathcal{E}^+ = \{\texttt{class\_1(img\_1)}\}$ $\mathcal{E}^- = \{\texttt{class\_1(img\_2)}\}$

A new framework is learnt using the ASP-ABALearn strategy, which employs two procedures: Rote and Gen, as depicted in Figure 3.6. These procedures utilise a subset of transformation rules to learn new rules. The Rote procedure aims to add facts to the background knowledge of the input framework so that the new framework is a (non-intensional) solution. A non-intensional solution is one that contains facts. The Gen procedure then transforms this non-intensional solution into an intensional solution. The ASP-ABALearn strategy alternates between these two procedures until a intensional solution is produced which satisfies the ABA Learning goal.

Example 11 We demonstrate how the ABA-ASP learning strategy can generate the learnt ABA framework (Example 8) from Example 10

Input: ABA Learning Problem $(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle , \langle \mathcal{E}^+, \mathcal{E}^- \rangle)$
RoLe$(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle , \langle \mathcal{E}^+, \mathcal{E}^- \rangle)$; *GEN*$(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle, \langle \mathcal{E}^+, \mathcal{E}^- \rangle)$.

Procedure *RoLe*$(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle , \langle \mathcal{E}^+, \mathcal{E}^- \rangle)$:
$P := ASP^*(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle , \langle \mathcal{E}^+, \mathcal{E}^- \rangle, \mathcal{A})$;
if P is unsatisfiable
then fail;
for all $\mathtt{c}\_\alpha(\mathtt{t}) \in \mathcal{C}(P) \setminus \mathcal{C}(ASP(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle)$, where $\mathtt{c}\_\alpha(t) = \overline{\alpha(t)}$ for some $\alpha(t) \in \mathcal{A}$ **do**
apply *Rote Learning* and get $\mathcal{R} := \mathcal{R} \cup \{c\_\alpha(X) \leftarrow X = t\}$;
for all $p(u) \in \mathcal{E}^+$ such that $\mathtt{p(u)} \notin \mathcal{C}(ASP(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle)$ **do**
apply *Rote Learning* and get $\mathcal{R} := \mathcal{R} \cup \{p(X) \leftarrow X = u\}$;
if $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ does not entail $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$
then fail;

Procedure *GEN*$(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle , \langle \mathcal{E}^+, \mathcal{E}^- \rangle)$:
while there exists a non-intensional rule $\rho_1 \in \mathcal{R}_{learnt}$ **do**
// **Folding:**
$\rho_2 :=$ *fold-all*(ρ_1); $\mathcal{R} := (\mathcal{R} \setminus \{\rho_1\}) \cup \{\rho_2\}$;
// **Assumption Introduction:**
if $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ does not entail $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$ **then**
apply *Assumption Introduction* and get ρ_3: $H \leftarrow B, \alpha(X)$ where α is a new predicate symbol and $X = vars(H \leftarrow B)$;
$\mathcal{R} := (\mathcal{R} \setminus \{\rho_2\}) \cup \{\rho_3\}$;
$\mathcal{A} := \mathcal{A} \cup \{\alpha(X)\}$, with: $\overline{\alpha(X)} = c\_\alpha(X)$;
// **Rote Learning:**
for all $\mathtt{c}\_\alpha(\mathtt{t}) \in \mathcal{C}(ASP^+(\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle , \langle \mathcal{E}^+, \mathcal{E}^- \rangle, \{\alpha(X)\}))$ **do**
apply *Rote Learning* and get $\mathcal{R} := \mathcal{R} \cup \{c\_\alpha(X) \leftarrow X = t\}$;
if $\langle \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ does not entail $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$ **then** fail;
// **Subsumption:**
for all ρ: $p(X) \leftarrow X = t \in \mathcal{R}_{learnt}$ **do**
if $\mathtt{p(t)} \in \mathcal{C}(ASP(\langle \mathcal{R} \setminus \{\rho\}, \mathcal{A}, \overline{\ } \rangle)$ **then**
apply *Subsumption* and delete ρ: $\mathcal{R} := \mathcal{R} \setminus \{\rho\}$.

Figure 3.6: ABA-ASP Algorithm [17]

Input:
$\mathcal{R} = \{\rho_1 : \texttt{slot\_1(A) :- A= img\_1}, \quad \rho_2 : \texttt{slot\_1(A) :- A= img\_2.},$
. $\quad \rho_3 : \texttt{slot\_2(A) :- A= img\_2.}\}$
$\mathcal{E}^+ = \{\texttt{class\_1(img\_1)}\} \qquad \mathcal{E}^- = \{\texttt{class\_1(img\_2)}\}$

Algorithm:
RoLE: Select predicate `class_1` from $\mathcal{E}^+$:
obtain $\mathcal{R} = \mathcal{R} \cup \{\rho_4 : \texttt{class\_1(X) :- X = img\_1}\}$ by rote learning
Since $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$ does not entail $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$ Execute GEN
GEN: Since there exists a non-intensional rule ρ_4
obtain $\mathcal{R} = (\mathcal{R} \cup \{\rho_5 : \texttt{class\_1(X) :- slot\_1(X)}\}) \backslash \{\rho_4\}$ by folding
Since $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$ does not entail $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$
Assumption Introduction
Obtain ρ_6 : `class_1(X) :- slot_1(X), alpha(X).`
Replace Rule $\mathcal{R} = (\mathcal{R} \backslash \{\rho_4\}) \cup \{\rho_6\}$
Add assumption $\mathcal{A} = \mathcal{A} \cup \{\texttt{alpha(X)}\}$ with contrary `c_alpha(X)`
Rote Learning
Obtain $\mathcal{R} = \mathcal{R} \cup \{\rho_7 : \texttt{c\_alpha(X) :- X=img\_2.}\}$
$\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$ does entail $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$
No Subsumption
GEN: Since there exists a non-intensional rule ρ_7
obtain $\mathcal{R} = (\mathcal{R} \cup \{\rho_8 : \texttt{c\_alpha(X) :- slot\_2(X)}\}) \backslash \{\rho_7\}$ by folding
$\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$ does entail $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$
GEN: No non-intenstional Rule END

Hence the final program we obtain after converting to ASP :

```
slot_1(A) :- A= img_1.
slot_1(A) :- A= img_2.
slot_2(A) :- A= img_2.
class_1(X) :- slot_1(X), alpha(X).
c_alpha(X) :- not alpha(X), slot_2(X).
```

The ASP-ABALearn exhibits non-determinism in folding non-intentional rules. For instance, when folding `X:-img_2`, the algorithm faces a choice between `slot_1` and `slot_2`. The strategy uses Clingo to initially verify if folding would yield a program which is not unsatisfiable. If the program is satisfiable, the algorithm proceeds with that fold; however, it remains uncertain whether this represents the optimal fold at that stage. Consequently, this approach results in the generation of multiple possible rules for the same input, with some processes experiencing increased termination times due to extensive backtracking.

3.9 Evaluation

3.9.1 Evaluating Object Centric Learners

Evaluating object-centric learners can be relatively hard compared to other models due to the fact that object-centric learners produce permutationally invariant results. For this reason, we need evaluation metrics which can assess the quality of the object recognised without depending on the order of the output. Some metrics we will use include:

Adjusted Rand Index
Adjusted Rand Index (ARI) is a metric which measures the similarity between two clusters. It is a correction of the Rand Index (RI) which is a basic measure of similarities. RI assesses clustering similarity by comparing pairs of data points in both algorithm-generated clusters and the ground truth. It measures the proportion of these pairs that are correctly clustered producing a score between 0 and 1 where 1 is a perfect match. A disadvantage of RI is that it doesn't account for chance i.e. some agreement between two clusterings can occur by chance. ARI take this into account and is calculated by Equation 3.3.

$$\text{ARI} = \frac{\text{RI} - \text{Expected\_RI}}{\text{max\_RI} - \text{Expected\_RI}} \tag{3.3}$$

ARI can be applied to evaluating object-centric learners by considering each pixel in an image as a data point and an object as a cluster. Each pixel is thus assigned to an object by considering the attention mask of each object with the highest value. This metric hence gives us a good indication of how well the object-centric learner can segment the image into objects.‘

Average Precision
Average Precision (AP) metric used for evaluating models in the fields of object detection and segmentation. AP measures the ability of a model to correctly identify and localize objects within images by evaluating the trade-off between precision and recall. Precision is defined as the ratio of true positive detections to the total number of detections made by the model, while recall is the ratio of true positive detections to the total number of actual objects present in the ground truth.

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.4}$$

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.5}$$

To calculate AP, the precision and recall values are first computed at various threshold levels of the detection confidence score. These values are used to plot a precision-recall curve, where precision is plotted on the y-axis and recall on the x-axis. The AP is then determined by calculating the area under this precision-recall curve (AUC). Higher AP values indicate better performance, as they reflect a higher proportion of

correct detections across all levels of recall. This metric provides a comprehensive assessment of a model's detection capabilities, balancing the need for both accurate localisation and correct classification of objects.

3.9.2 Evaluating Logic-Based Learners

Evaluating a logic-based learning system requires assessing its performance in rule generation and the correctness of the rule produced to classify concepts as positive and negative. A metric we can use to evaluate correctness is the F1 score (3.6) which is defined as the harmonic mean between the precision and recall. F1 provides a more reliable measure of performance compared to accuracy which can be susceptible to data imbalances.

$$F_1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall} \tag{3.6}$$

Additionally, assessing the system's efficiency in terms of computational resources, such as time and memory usage, is important for practical applications as well as scalability: how well the learner performs as the size and complexity of the input data increase. These can all be measured by running different experiments altering the number of examples used in ABA learning and increasing the object domain. Finally, we can evaluate the quality of rules which computational argumentation through human feedback to verify how explainable and interpretable the rules are.

3.10 Related Work

The field of explainable AI is very diverse which has resulted in many systems which aim to make the results of models more interpretable. Most of these systems integrate a type of symbolic reasoner into traditional neural networks. These symbolic algorithms allow users to be able to follow how a system reaches its prediction. Other systems however add tools to aid users in understanding the key features which result in a prediction.

3.10.1 GradCAM

Gradient-weighted Class Activation Mapping (Grad-CAM) [20] is a visualization technique that enhances the interpretability of CNNs. It does this by identifying which regions of an input image are most influential in the network's decision-making process. Grad-CAM computes the gradients of the target class score concerning the feature maps of a specific convolutional layer and then applies global average pooling to obtain importance weights for each feature map. These weighted feature maps are combined and passed through a ReLU activation to highlight the positive contributions, followed by upsampling to the original image size. The resulting heatmap allows for a clear visualization of the areas that the model considers critical for its predictions. Grad-CAM not only aids in understanding and debugging model behaviour but also enhances trust in the model by revealing its focus areas,

making it particularly valuable in fields like medical imaging and autonomous driving.

3.10.2 NeSyFold

NeSyFOLD [21] is a neuro-symbolic learner used for image classification tasks. It aims to learn global features using a CNN however, it replaces the final layer with a binarisation of the CNN kernels and extracts an ASP program from this. This has resulted in a novel approach in the neuro-symbolic space allowing for the classification task to be fully interpretable via ASP.

NeSyFOLD uses a rule-based machine learning algorithm called FOLD-SE-M to derive the ASP program from the CNN. This algorithm learns a rule set from data as a default theory (a non-monotonic logic for commonsense reasoning). To classify the image, the framework uses quantisation to binarise the output kernel of the trained CNN. It then applies the FOLD-SE-M algorithm to generate an ASP program from the binarised kernel however these rules are all abstract. The NeSyFOLD framework then uses semantic labelling to assign meaningful words to the abstract rules (see paper for the algorithm). The output is an ASP program with human-like concepts whereby when run, will return us the stable model corresponding to the final classification.

3.10.3 Neuro-Symbolic Concept Learner

The Neuro-Symbolic Concept Learner [22] is an advanced neuro-symbolic learning system that employs explanatory interactive learning (XIL) to derive meaningful concepts from images. It uses slot attention as a feature extractor to identify objects within images, while its reasoning modules incorporate set transformers to extract and generate explanations for each object's concepts. These explanations are presented as grid-like representations, enabling users to understand the concepts applied in decision-making and derive additional rules.

One challenge the Neuro-Symbolic Concept Learner faced was its tendency to focus on incorrect reasons. To address this, the paper proposed using XIL, which leverages human feedback to guide the model towards the correct concepts. This intervention allows the Concept Learner to adjust its internal representations, resulting in improved outcomes.

Chapter 4

Datasets

In this chapter, we analyse some of the datasets that were used to train the Neuro-Argumentative Learning pipeline.

4.1 Requirements

To evaluate how well neuro-argumentative learning can learn rules on images, we required datasets which have labelled segmentation maps of the objects in an image. This allows us to verify that the object-centric learner can identify and localise objects in an image. We also require logical rules/constraints to which objects or sets of images adhere to. These rules can be compared with outputted rules where we can assess the readability and correctness of the logic-based learner.

4.2 SHAPES

SHAPES [23] is a synthetic dataset for understanding spatial and logical relationships among multiple objects. It comprises of 300x300 pixel images which contain objects organised in a grid-like fashion occupying nine set regions. Objects can be one of three shapes `(square, circle, triangle)` each of which can be of three colours `(red, green, blue)` and two absolute sizes `(small, large)`. This amounts to a maximum of 72 attributes per image. Each image is labelled with the object occupying each region and its attributes.

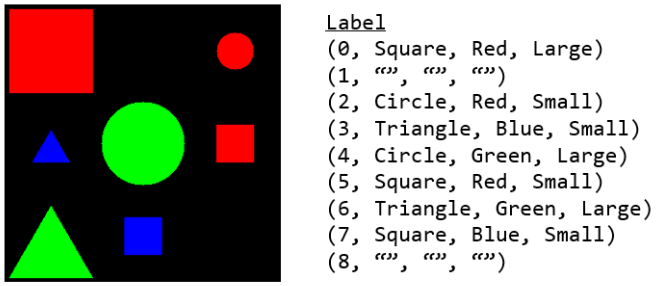

Figure 4.1: Image from SHAPES dataset

4.2.1 Generating SHAPES

The SHAPES dataset originally associated questions relating to the spatial and logical relationship of the shapes in the images. Each image is assigned a yes or no label depending on the objects meeting the rules of the question. The task for this dataset is the opposite: given positive and negative examples of a rule, can we recover the rule? To allow the dataset to suit our needs better, we implemented a generator to produce SHAPES-like images based on input rules and used this to generate a new SHAPES dataset.

The SHAPES generator takes input in ASP syntax, consisting of rules that specify conditions objects in the image must meet. These rules fall into three categories: Basic Rules, which condition on the presence of objects in the image; Positional Rules, which condition on the arrangement of objects; and Exception Rules, which specify conditions where the negation must be satisfied. To implement this, we randomly generated images and developed functions to assess whether each image conformed to the specified rule. The generator also allows for blank sections in the images to increase the variability of images. Figure 4.2 displays images generated under the rule: `class1(A) :- image(A), in(A,B), square(A), blue(A).` which translates to "An image is an instance of class 1 if it includes a blue square".

To produce our new SHAPES dataset, we generated 3K images for 6 rules leading to a dataset of 18K images. Within the 3K images for each rule, half of which was the negative instance of the rule resulting in a 1.5k split for positive and negative examples. We then took 500 of each split as testing data. In total, we had 12K images for training and 6K for testing. Table 4.1 outlines the 6 rules classes we used to generate the dataset

We also conducted a brief analysis of the images generated for the shape dataset to ensure each attribute was adequately represented. Table **??** indicates that squares and blue shapes are slightly underrepresented, while triangles and green shapes are slightly over-represented. Despite these minor disparities, no attributes were severely underrepresented or over-represented. Therefore, the generator produced a balanced dataset in terms of the shape attributes.

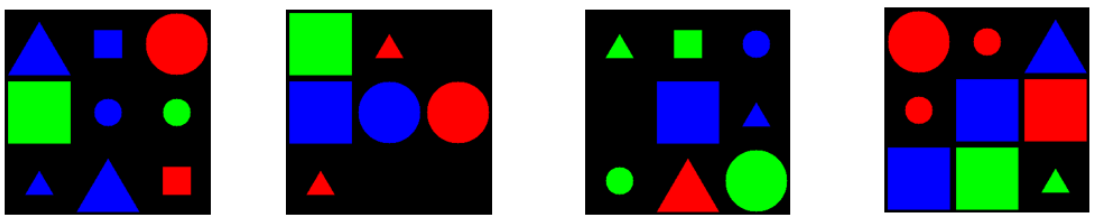

Figure 4.2: SHAPES Generated Images from Class 1

SHAPES Dataset ASP Rules
class1(A) :- image(A), in(A,B), square(B), blue(B).
class2(A) :- image(A), in(A,B), triangle(B), small(B), green(B)
class3(A) :- image(A), in(A,B), in(A,C), triangle(B), blue(B), circle(C), red(C), large(C).
class4(A) :- image(A), in(A,B), in(A,C), circle(B), red(B), square(C), blue(C), above(B,C).
class5(A) :- image(A), in(A,B), in(A,C), triangle(B), red(B), circle(C), green(C), left(B,C).
class6(A) :- not exception(A), image(A). exception(A) :- image(A), in(A,B), circle(B), blue(B) .
SHAPES Datasets Rules Interpretation
class1(A): - Class 1 images must contain a blue square
class2(A) :- Class 2 images must contain a small green triangle
class3(A) :- Class 3 images must contain both a blue triangle and a large red circle
class4(A) :- Class 4 images must contain a blue square above a green triangle
class5(A) :- Class 5 images must contain a red triangle left of a small green circle
class6(A) :- Class 6 images are without blue circles

Table 4.1: ASP rules used to generate the 6 classes for the SHAPES dataset

4.3 CLEVR-Hans

The CLEVR dataset is a diagnostic dataset designed to evaluate and analyze the reasoning capabilities of machine learning models, specifically focusing on visual question-answering (VQA) systems. The images in the CLEVR dataset are synthetic and computer-generated, featuring simple 3D geometric shapes: spheres, cubes, and cylinders. These shapes vary in colour, size, and material. An image may contain a maximum of 10 objects leading to a total of 150 attributes. Objects are arranged in various spatial configurations within plain backgrounds. With these elements, CLEVR creates a complex visual scene that challenges models to demonstrate genuine understanding and reasoning, free from natural image biases.

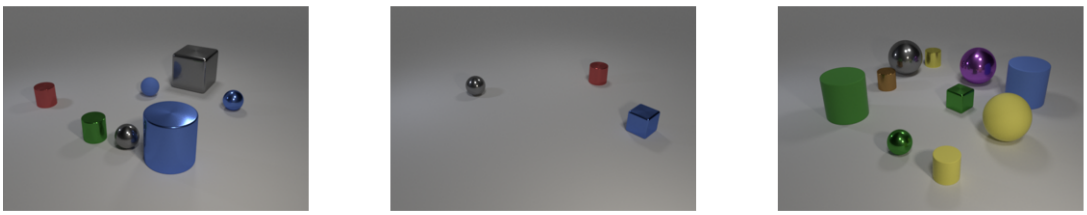

Figure 4.3: Example Images from CLEVR Dataset

CLEVR-Hans [24] adapts the dataset by assigning CLEVR images to non-overlapping

CLEVR-Hans 3 Rules	
class 1:	`Large (Gray) Cube and Large Cylinder`
class 2:	`Small metal Cube and Small (metal) Sphere`
class 3:	`Large blue Sphere and Small yellow Sphere`

Table 4.2: Class rules for CLEVR-Hans

classes. The membership of a class is based on combinations of objects' attributes and relations (Table 4.2). Each class is represented by 3000 training images, 750 validation images, and 750 test images. The training, validation, and test set splits contain 9000, 2250, and 2250 samples. The class distribution is balanced for all data splits. CLEVR-Hans allows us to stretch both parts of the neural-argumentative architecture while simultaneously testing the system on non-binary classification.

The CLEVR-Hans dataset enhances the difficulty of learning tasks by including specific instances of class objects in the training set and generalising them in the test dataset. For instance, in the training set, all instances of large cubes in class 1 images are grey, but in the test set, they could appear in any colour. This allows us to test the robustness of the rules learned from the training data understanding if they overfit to the examples provided or produce generalised rules.

Chapter 5

Neural Argumentative Learning

This chapter presents the Neuro-Argumentative Learning (NAL) pipeline (Figure 5.1) used to learn argumentation frameworks from images.

The NAL pipeline is split into two sections, the neuro-module and the symbolic module. The neuro-modules goal is to convert the images into symbols and feed this information to the symbolic modules which apply ABA learning to learn new concepts. These concepts can be leveraged for downstream tasks such as classification and common-sense reasoning.

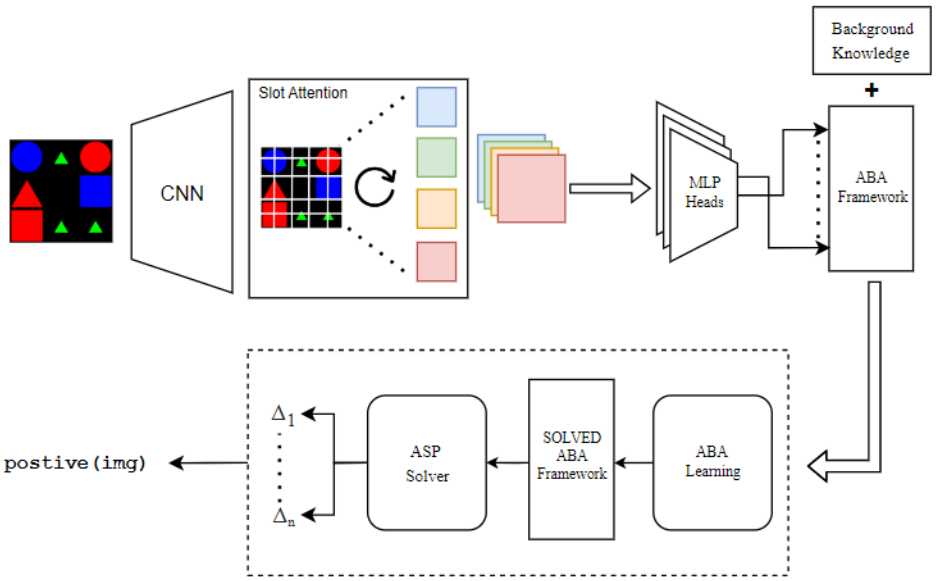

Figure 5.1: Neural Argumentative Learning Pipeline

5.1 Inputs

The NAL pipeline accepts a dataset X of images $x_i \in X$ whereby $X := [x_1, \ldots, x_N]$. Images in the dataset can be divided into a subset of N_c classes $\{X_1, \ldots, X_{N_c}\}$. The NAL pipeline also takes in an object dictionary 5.2 which contains all the attributes the model needs to identify in the images. A special entry in the dictionary `object` details the type of objects the attribute refers to. This dictionary is also used to label the predicate in the symbolic module. Without it, the architecture will define abstract properties to extract from each object.

```
object_info = {
    "object" : ["Triangle", "Circle", "Square"],
    "colour" : ["Red", "Green", "Blue"],
    "size:"  : ["Large", "Small"]
}
```

Figure 5.2: Object dictionary for SHAPES dataset

Other input configurations include the number of slots for the slot attention unit (which is set to the length of the object list plus one for the background) and the size of the input image for the neuro-module. The symbolic module also takes a list of tuples $[(\mathcal{E}^+, \mathcal{E}^-) \ldots]$ where each tuples contains the classes which are to be considered positive and negative examples. We also input the number of examples which the ABA Learning algorithm should use. Before training and inference, the images are automatically preprocessed consisting of normalisation to the range of [-1, 1] and resizing the image to the desired scale.

5.2 Neuro-Module

The neuro-module consists of a slot attention autoencoder and a set of multi-layer perceptrons (MLP) to convert the images into predicates. The slot attention autoencoder is trained using the process described in section 3.4.2 segmenting the image into K slots. Each of these slots are latent representations of objects containing information on their attributes.

These attributes are then extracted via the MLP. The architecture has two types: A classification MLP which uses a softmax activation in the final layer to predict the most likely attribute for a given feature. The second type is a regression MLP head which is used to predict the location of objects and determine whether the slot has attended to a real object or not.

The results of these predictions are then concatenated to form our final perception of the input image. We also append the confidence of each prediction due to the sensitivity of the symbolic module. Finally, we output the mask of each slot which can be used to add positional background knowledge in the ABA framework

```
% Image Facts
in(A,B) :- A=img_1, B=object_1.
circle(A) :- A=object_1.
blue(A) :- A=object_1.
in(A,B) :- A=img_1, B=object_2.
circle(A) :- A=object_2.
blue(A) :- A=object_2.
image(A) :- A=img_1.

in(A,B) :- A=img_20, B=object_68.
triangle(A) :- A=object_68.
blue(A) :- A=object_68.
in(A,B) :- A=img_20, B=object_69.
triangle(A) :- A=object_69.
red(A) :- A=object_69.
in(A,B) :- A=img_20, B=object_70.
triangle(A) :- A=object_70.
blue(A) :- A=object_70.
image(A) :- A=img_20.

% Command to run ABA ASP
aba_asp('shapes_bk.aba',[c(img_1)],[c(img_20)]).
```

Figure 5.3: Example of ABA Learning input file

5.3 Symbolic-Module

The symbolic module receives the output predictions of the neuro-module and processes them to an ABA Framework (Figure 5.3) for the ABA Learning algorithm.

The module's initial task involves selecting appropriate positive and negative examples from the sets X_{N_c} listed in $(\mathcal{E}^+, \mathcal{E}^-)$. This is accomplished by aggregating the slots and performing clustering. The number of clusters corresponds to the desired number of examples, and an image is chosen from each cluster. We also check the confidence of each prediction and prune off any image below a certain threshold.

The predictions are then passed to concept embedding functions which use the object dictionary to convert the raw predictions to predicate. This is achieved using the ABA Framework class which which allows users to add background knowledge, run the ABA-ASP algorithm and extract learnt rules for inference. For each image and object, an identifier is given in the form of the predicate `image(img_i)`, and the constant `object_i` respectively. Then for each prediction, we take the argmax to identify the properties in the object dictionary which are attributed to the object. We encode this as a predicate in the framework e.g. `blue(object_i)`. This is done for all objects and images.

Once all images and objects are encoded into the ABA Framework, we generate the ABA-ASP command `aba_asp('filename.aba',[e_pos],[e_neg])`. The command specifies which images are positive and negative using $(\mathcal{E}^+, \mathcal{E}^-)$. The ABA-ASP learning strategy is then run to produce a new ABA Framework which includes the learnt rule. The ABA Framework class extracts these learnt rules into a `.SOLVED.aba` file.

5.4 Inference

At inference time, we run a slightly different pipeline to obtain final results. For a new image we wish to see if it's a positive instance of the concept, we passed the image through the neuro-module slot attention unit and MLP heads. This produces predictions of the objects and their attributes which are converted into predicates. From this, we create an ASP program which contains these predicates, the learnt rules from training and any extra background knowledge. This file is then solved using an ASP solver (e.g. Clingo) to compute the stable models that the program emits. Depending on the ABA semantics, the program may emit more than one stable model.

For classification tasks, these stable models are analysed to see the presence of the concept predicate. If so the model predicts positive and negative if absent. For other downstream tasks, the analysis of the model may vary hence one can define the conditions needed for a positive instance.

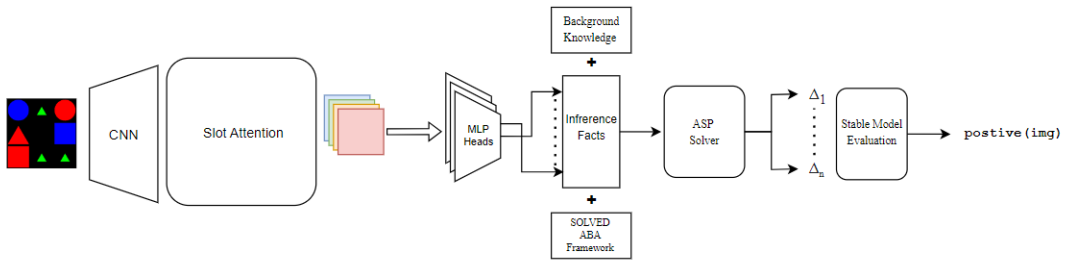

Figure 5.4: NAL Inference Pipeline

Chapter 6

Experiment Setup

In this chapter, we describe the training settings used to prepare the NAL architecture for our experiments. Additionally, we outline the various tasks used in these experiments and detail the baseline models used for comparison against our architecture.

6.1 Models and Training

6.1.1 Training Neural Modules

Training the neural modules involved setting up a training pipeline which combined the goals of the slot attention unit and the MLP head. Images were paired with one-hot encoded label $y_i \in [0,1]^{K \times P+1}$ where $P = \sum A_i$ and A_i represents the length of the list corresponding to the i-th key in the object dictionary (5.2). These labels represent all the objects in an image and their corresponding attributes. The slot attention then produce slots $\hat{Z}_i \in \mathbb{R}^{d_O \times d_S}$. Each slot is then passed through MLP heads to produce a prediction $y_i \in [0,1]^{K \times P+1}$.

The loss function to train this model was a combination of reconstruction loss (for slot attention) and Binary Cross Entropy (BCE). However, one problem we ran into was the positional invariance property of the slot attention output. This meant that our label order didn't match the output order from the slot attention model. To resolve this, we used Hungarian matching: an optimisation technique which aims to find the optimal way to match elements of two distinct sets. To apply this, we created a cost matrix which stored the BCE of all combinations of labels and predictions. We then run the algorithm to find matches which result in the minimum cost. With this, we define equation 6.1 as the loss function which the model tries to minimise.

$$\text{Loss} = \text{MSE}(x_i, \hat{x}_i) + \alpha \min_{\sigma \in S_N} \sum_{i=1}^{N} \text{BCE}(y_i, \hat{y}_{\sigma(i)}) \tag{6.1}$$

where S_N is the set of all possible permutations of true labels and predicted probabilities

For our experiments, we trained each model for 1500 epochs on all the images in the dataset and set alpha in equation 6.1 to 0.7 to balance the reconstruction loss and the Hungarian loss. For every 4 epochs, we evaluated the validation set and used the model which performed best on the validation set as our final model.

6.1.2 Training Symbolic Module

The symbolic section of the architecture used ABA-ASP to learn a new ABA framework. In our experiments, the algorithm was a black box hence, we couldn't tailor the learning strategy for image data tasks. To address this limitation, we focused on optimising the learning process via the inputs and the setting that ABA-ASP provided.

ABA-ASP Examples

The nature of the ABA-ASP learning strategy means that we cannot provide the entire dataset as input. Instead, we provide a small subset of positive and negative examples. These examples define the search space that the learning strategy uses to generate rules. Therefore, the input examples must be representative, non-redundant, and contrastive to enable the algorithm to identify rules that distinguish between the classes.

To achieve this, we performed K-Means clustering on the aggregated slot representations of all images in each class. We experimented with various aggregation methods, such as averaging and concatenation. The number of centroids was set to match the number of examples needed for input. After fitting the model, we selected the slots closest to the centroids, based on Euclidean distance, as the examples.

ABA-ASP Leaning Settings

ABA-ASP offered numerous settings that allowed us to adjust the learning strategy for rule generation despite the core algorithm staying the same. We experimented with various configurations to identify the optimal combination that facilitated efficient rule learning and produced high-quality rules. Two of the most impactful settings were the learning mode and the folding mode. The learning mode determined whether the ABA semantics were brave or cautious, while the folding mode defined whether the algorithm conducted a top-down or bottom-up search.

Our experiments showed that using brave semantics led to faster termination. However, this learning mode produced multiple stable model outputs, which were difficult to interpret for our downstream tasks. Additionally, we found the bottom-up search strategy infeasible for learning rules due to the number of predicates each example had. Consequently, we opted for cautious learning and a top-down search. Although this approach was more challenging and resulted in longer execution times, the ABA-ASP tool was still able to produce adequate rules to meet the learning goals.

ABA-ASP Folding Order

Despite optimising the search strategy, ABA-ASP occasionally took much longer to generate rules. This delay was caused by the algorithm folding on auxiliary predicates first, leading it to search redundant sections of the search space. To address this, we defined a folding order for the algorithm, prioritising domain-specific predicates. Additionally, we ordered the predicates by their arity, enabling the algorithm to construct rules using simpler predicates before progressing to more complex ones. These changes significantly improved the termination speeds of ABA-ASP in our experiments.

6.1.3 Baseline Models

In the field of neuro-symbolic learning, few systems have successfully integrated neural networks and symbolic learning as we have in the NAL pipeline. One notable exception is NeSyFold, which employs a different neural network system and symbolic reasoner to learn rules from images. Unfortunately, the code for NeSyFold was unavailable, preventing us from using it as a baseline model. As a result, we opted to compare our NAL architecture with sub-symbolic learners and other tools which helped to make deep learning models more explainable.

Baseline 1: ResNet-34 with GradCAM

Our first baseline purpose is to compare how well our neuro-symbolic model compares to fully neural models. We use a Resent-34 model which extends CNNs with residual connections to prevent vanishing gradients. We trained the model for 100 epochs as it converged faster than our slot attention model and passed the final layer through a GradCAM function to visualise the regions leading to a classification.

Baseline 2: Neuro-Symbolic Concept Leaner

Our second baseline uses the Neuro-Symbolic Concept Learner (NS-CL) as a sub-symbolic system for comparison. Similarly to our architecture, NS-CL uses slot attention to extract objects and their attributes. However, instead of pure symbolic reasoning, it employs a set transformer to obtain a final classification. NS-CL also outputs visual explanations, highlighting the concepts utilised in the classification process.

6.2 NAL Experiments

6.2.1 Investigating NAL with SHAPES

The SHAPES dataset was considered our baseline dataset for testing if argumentation was a viable strategy for learning rules on image data. To conduct this experiment using the SHAPES dataset, we defined six binary classification tasks for the

neural argumentative architecture. The goal of each was to learn rules which could segment the positive and negative examples of each class (4.1). After training the neuro module on the dataset, we train the symbolic module for each of the classification tasks varying the number of examples and ordering of predicate. Our ResNet baseline model with GradCAM achieved perfect classification scores on all tasks, suggesting that these tasks are well-suited for evaluating the viability of our proposed architecture.

6.2.2 CLEVR-HANS

Once establishing an understanding of the capabilities of the neuro-argumentative architecture, we defined a multi-class classification task on the CLEVR dataset.

Similarly to the SHAPES dataset, we trained the neuro-module using all the training images. One challenge we encountered when training the symbolic module was the binary inputs (positive and negative examples) it needed to learn. For the CLEVR-HAN dataset, which contained 3 classes, we addressed this by training the symbolic modules twice. Initially, we trained with one class as positive and the rest as negative. Then, we repeated the process with the remaining class as positive and the rest as negative. Consequently, during inference, we executed the first ABA program. If the classification was negative, we proceeded with the second program to obtain a final classification for the remaining two classes.

To evaluate this method we used the ResNet model with GradCAM and the Neuro-Symbolic Concept learner to compare the performance of the multi-class classification and the expalinton produced by each model.

Chapter 7

Experimental Results

In this chapter, we evaluate how well the NAL pipeline performs against each dataset providing qualitative and quantitative analysis on both sections of the pipeline

7.1 Neural Module Evaluation

The perceptual ability of the neuro module was an important task for the architecture to ensure that the facts written to the symbolic module were accurate. To evaluate the performance of this task we analysed how well the model segmented objects within images and the identification of attributes from each segmented object. We make use of the evaluation metric discussed in section 3.9

7.1.1 Image Segmentation

The slot attention unit performed well in segmenting the image into its constitute objects. From Figure 7.1 each slot correctly attends to a specific object, resulting in a nearly identical reconstruction. To further evaluate performance, we used the Adjusted Rand Index (ARI) metric to assess the unit's ability to localize and segment images. The model achieved high ARI scores, with 0.8 for SHAPES and 0.91 for CLEVR, demonstrating consistent performance regardless of the number of objects, positions, or sizes.

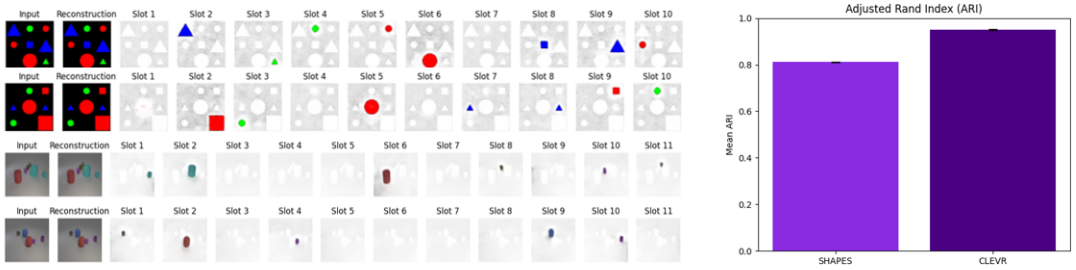

Figure 7.1: (LEFT) Example images segmented into slots; (RIGHT) ARI Plot for each dataset

7.1.2 Object Classification

Given the latent representation from the slot attention unit, the MLP heads performed well in extracting the attributes of each object. The chart in Figure 7.2 shows that for both CLEVR and SHAPES images, the model could extract attributes with an accuracy greater than 80%. The F1 score for both is above 70% suggesting that the model can minimise false positives and false negatives. This is a good indication that the symbolic module can reason accurately about the images. However, the results from the CLEVR dataset were slightly worse than the SHAPES. This could be due to the number of attributes being larger in CLEVR than in SHAPES suggesting that the attribute size can affect prediction performance.

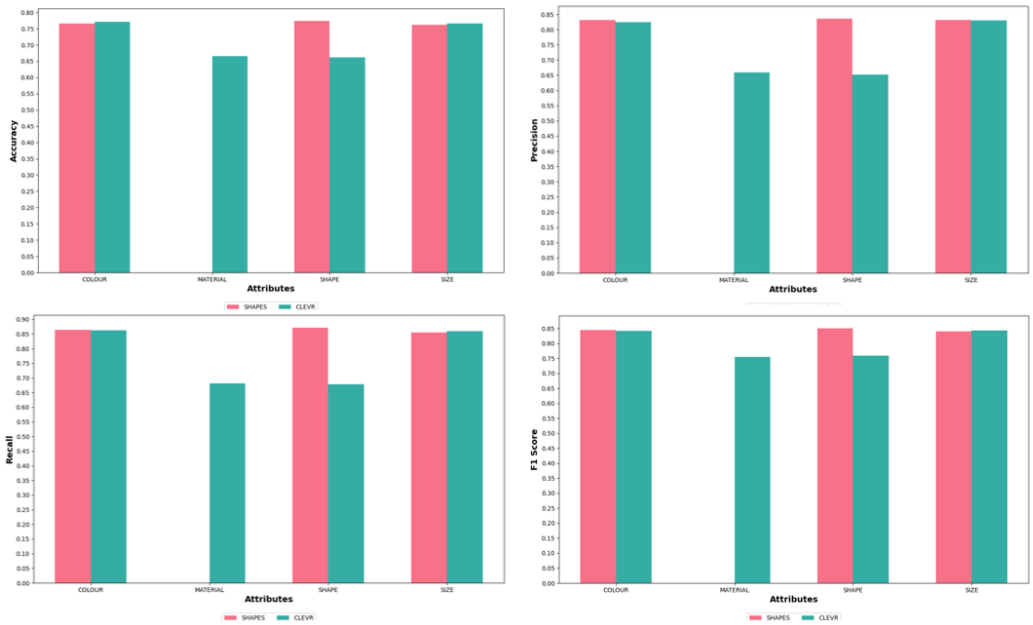

Figure 7.2: Standard evaluation metrics of predicate classification on the SHAPES and CLEVR dataset

We also calculated the average precision (AP) of the neuro-modules for both datasets. At infinite range, the CLEVR dataset scored 0.88 while SHAPES scored 0.91. These high AP values indicate that the modules can effectively classify and localise objects. This reinforces the idea that the predicates used for the symbolic modules accurately represent the images.

7.2 Symbolic Module Evaluation

The symbolic section of the pipeline used the ASP-ABALearn learning strategy to produce programs for the final classification task. To evaluate this section, we looked at the common pitfalls in logic-based learning algorithms e.g. scalability and quality of results.

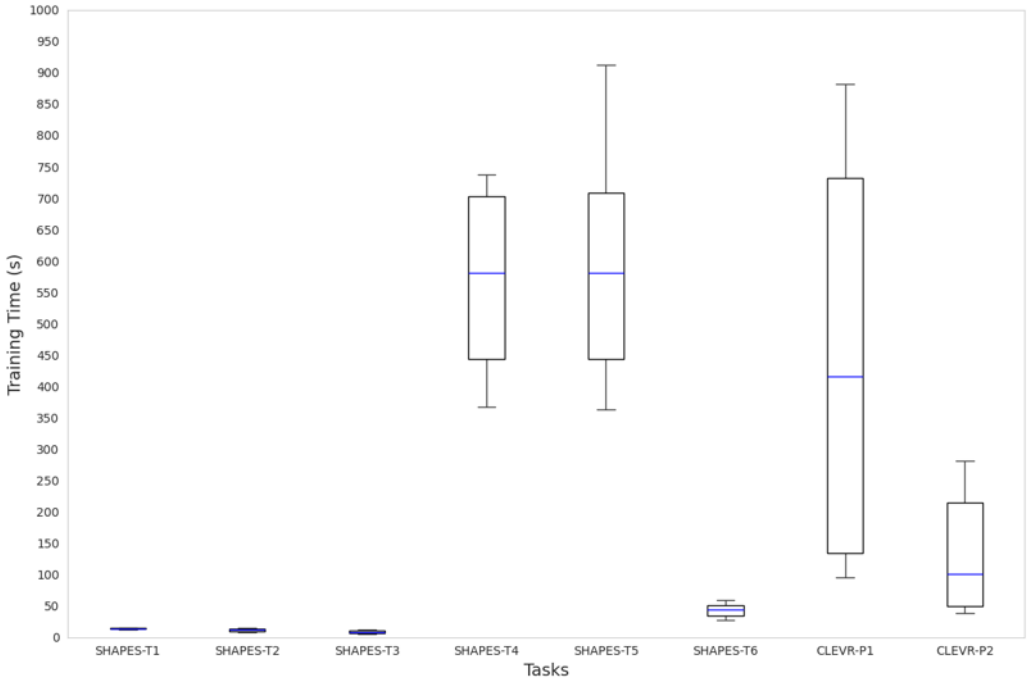

Figure 7.3: Box plots of training time of ASP-ABALearn

Our experiments revealed that the non-determinism of ABA-ASP folding led to variability in both the quality of results and the system's running time. This variability was exacerbated by the complexity of the task, such as increasing the number of facts through background rules or tasks with large object domains. As illustrated in the box plot in Figure 7.3, simpler tasks like SHAPES-T1 exhibited low variance, whereas more complex tasks like SHAPES-T5 and CLEVR-P1 showed high variance in running time. Furthermore, we found that with longer running times, the quality of the results decreased due to heavily nested rules. Because of this, we chose to evaluate the best program outputted by ASP-ABALearn.

The ASP-ABALearn tool also faced scalability issues, which limited the number of examples we could use. As we increased the number of examples, training times grew significantly, with some runs having to be manually terminated after several days. This could be due to the tool having a larger example set to process, which increases the likelihood of exploring irrelevant sections of the search space. Additionally, the higher risk of incorrect predicates produced by the neuro-modules could lead to examples no longer representing the positive class. If this occurs, the learning strategy needed to search the entire space before realising the problem is unsolvable. With these findings, we used between 10-20 examples to keep the search space feasible.

7.3 NAL with SHAPES

The SHAPES Dataset's primary purpose was to evaluate the viability of using argumentation to reason about objects in images. From Table 7.4, we see that the NAL

	Accuracy	Precision	Recall	F1-Score
Task 1	99.0 ± 0.0	100.0 ± 0.0	97.5 ± 0.0	99.0 ± 0.2
Task 2	96.0 ± 0.0	99.0 ± 0.0	93.5 ± 0.0	96.0 ± 0.2
Task 3	98.0 ± 0.0	100.0 ± 0.0	97.5 ± 0.2	98.0 ± 0.0
Task 4	75.0 ± 3.0	61.0 ± 5.2	98.5 ± 0.4	75.0 ± 0.2
Task 5	86.0 ± 4.1	77.0 ± 3.8	96.5 ± 0.2	84.0 ± 0.2
Task 6	99.0 ± 0.0	98.0 ± 0.0	100.0 ± 0.0	99.0 ± 0.0

Figure 7.4: Standard evaluation metrics denoting how well NAL can distinguish between positive and negative instances of rules present in SHAPES images

pipeline performed well in generating rules which could differentiate the positive and negative examples. For most rules, the recall was the lowest of the standard ml metric meaning that positive examples were classified as negative. This could be a result of the NAL architecture producing rules that don't fully capture the ground truth rule.

Two results which are of interest in the result class 4 and class 5 rules which produced a lower F1 score due to the lack of precision. These class rules described more complex relationships between objects. A low precision suggests that the NAL architecture learnt rules which were too general resulting in many negative examples being classified as positive. This is further evident by the high recall as a result of fewer false negatives.

7.3.1 Rule Analysis

From table 7.4, we found that the pipeline performed well on classifying images conditioned on the presence of an object/attribute. These types of rules were represented by tasks 1 to 3. Figure 7.5 shows the learned ABA program which ABA Learning produced from task 1. The program can be interpreted as an image belonging to the positive set if the image contains a square except if the square is red or green. This interpretation has the same meaning as the rule used to generate the image albeit expressed in a convoluted way. This could be a result of the argumentative nature of the learning process which learns by finding exceptions to the original rule devised.

```
% Background Knowledge
% Learnt Rules
class_1(A) :- alpha_2(B,A), square(B), in(A,B).
c_alpha_2(A,B) :- image(B), red(A).
c_alpha_2(A,B) :- image(B), green(A).
alpha_2(A,B) :- not c_alpha_2(A,B), square(A), in(B,A).
```

Figure 7.5: NAL output for SHAPES images generated by rule 1

A limitation of the rule generated by NAL for these tasks was its inability to capture all elements of a rule in conjunction. Highlighted by the rule learnt for task 3, the resulting program focuses solely on the second object in the conjunct rather than both objects. This led to some misclassification when only one of the two objects appeared in the image. A solution to force ABA-ASP to focus on both conjuncts would be to include more of these cases as negative examples.

```
% Background Knowledge
% Learnt Rules
class_3(A) :- alpha_2(B,A), circle(B), in(A,B).
c_alpha_2(A,B) :- image(B), green(A).
c_alpha_2(A,B) :- image(B), blue(A).
c_alpha_2(A,B) :- image(B), small(A).
alpha_2(A,B) :- not c_alpha_2(A,B), circle(A), in(B,A).
```

Figure 7.6: NAL output for SHAPES images in Task 3

We also saw some performance issues with the NAL pipeline on Task 4 and Task 5. For these tasks, the positive set consisted of images with objects in certain configurations. For these rules, we injected background knowledge which defined what it meant to be above and left and grounded the program to produce a program of grounded facts for ABA-ASP. Looking at the output of Task 5 (Figure 7.7), we find that the program captures some elements of the rule used to generate the images: e.g. the circle being green and that a triangle not being to the right of the circle. However, the program doesn't cover all the exceptions leading to it being too general. This could be a result of the increased search space as a consequence of the grounding with the background knowledge.

```
% Background Knowledge
above(S1, S2, I) :- position(I, S1, X1, Y1), position(I, S2, X2, Y2),
Y1 - Y2 < 0.
left(S1, S2, I) :- position(I, S1, X1, Y1), position(I, S2, X2, Y2),
X2 - X1 > 0.
% Learnt Rules
class_5(A) :- alpha_2(B, A), small(B), circle(B), in(A, B).
c_alpha_2(A, B) :- image(B), blue(A).
c_alpha_2(A, B) :- image(B), red(A).
c_alpha_2(A, B) :- green(C), triangle(A), left(A, C, B).
alpha_2(A, B) :- not c_alpha_2(A, B), small(A), circle(A), in(B, A).
```

Figure 7.7: NAL output for SHAPES images generated by rule 5

Task 6 featured images which were classified as positive via the absence of an object/attribute. From the previous results, we found that the pipeline generated rules by conditioning on objects found in the positive set and then finding counterexamples. However, since the rule was a counter-example itself, the pipeline couldn't use this

method and hence failed to learn a rule for this task. This case led to the realisation that the pipeline also needs to reason about the image as a whole first before looking at the object contained in the image. To allow for this we introduced ordering predicate by arity so the algorithm would search for rules conditioned on the image before objects. This improvement led to the pipeline being able to learn a program which accurately reasons on exception rules.

```
% Background Knowledge
% Learnt Rules
class_6(A) :- alpha_2(A), image(A).
c_alpha_2(A) :- alpha_3(B,A), circle(B), in(A,B).
c_alpha_3(A,B) :- image(B), red(A).
c_alpha_3(A,B) :- image(B), green(A).
alpha_2(A) :- not c_alpha_2(A), image(A).
alpha_3(A,B) :- not c_alpha_3(A,B), circle(A), in(B,A).
```

Figure 7.8: NAL output for SHAPES images in Task 6

7.4 NAL with CLEVR-Hans

The CLEVR-hans task was defined as a multiclass classification task where each class corresponded to a type of object being present (4.2). Looking at some standard evaluation metrics in Figure 7.9 we see that the NAL architecture had some success in uncovering rules to distinguish the classes. It achieved a score of around 70% in each category performing better than the ResNet baseline by 10%. However, it performed slightly worse than the NeuroSymbolic Concept Learner which achieved scores in the 80% region. This could be because the NS-CL uses a set transformer to classify the images rather than pure symbolic reasoning.

	Accuracy	Precision	Recall	F1-Score
NAL	69.1 ± 0.3	70.0 ± 0.1	69.1 ± 0.2	68.0 ± 0.2
ResNet with GradCAM	65.2 ± 0.3	66.2 ± 0.4	65.2 ± 0.2	61.0 ± 0.2
Neuro-Symbolic CL	84.7 ± 0.1	86.1 ± 0.2	84.7 ± 0.2	84.0 ± 0.2

Figure 7.9: Standard evaluation metrics denoting how well NAL can distinguish between positive and negative instances of rules present in SHAPES images

The confusion matrices in Figure 7.13 reveal that the NAL architecture struggled the most with distinguishing between Class 1 and Class 2. This difficulty likely arose because the classification required the presence of two objects with different shapes. In such cases, the ABA learning algorithm tends to be lazy, conditioning on only one object, since both conjuncts must be true for the conjunction to be valid.

Despite this, the NAL architecture outperformed the baseline. The training sets for

Class 1 and Class 2 included only a single type of colour/material, whereas the test set featured a variety of these attributes. Consequently, the baseline model and the concept learner may have bound to incorrect concepts, leading to reduced accuracy in the test set.

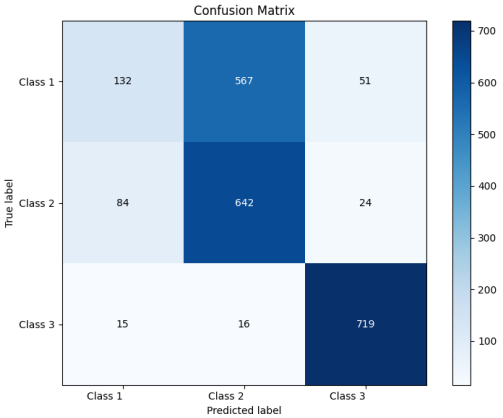

Figure 7.10: ResNet with GradCAM

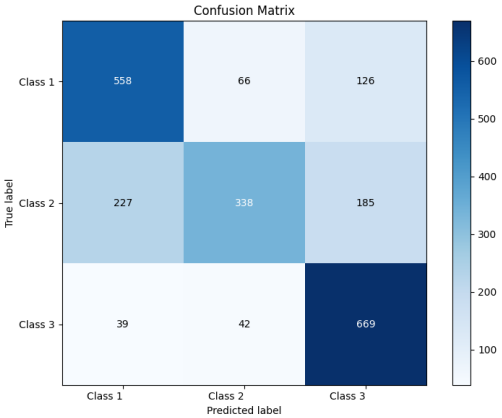

Figure 7.11: NAL Architecture

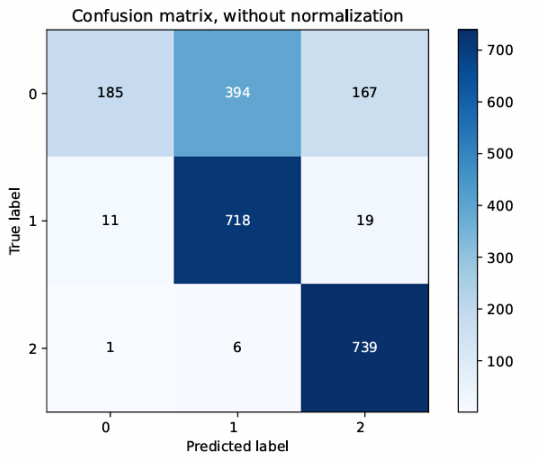

Figure 7.12: NS-CL

Figure 7.13: Confusion matrices of the different models for the test set of CLEVR-Hans3.

7.4.1 Rule Analysis

As described above we trained the symbolic-modules twice producing two frameworks. The first framework (shown in Figure 7.14) suggests that class 3 images are images which contain a sphere except if they possess any of the attributes listed. Analysing this further the exceptions indicate that the sphere can only be small and yellow (and any material). This only captures half of the rule neglecting that large blue spheres are also classified as class 3. Furthermore, the exception lists large and blue spheres as arguments against class 3 images. Despite half the rule missing, the framework was able to distinguish class 3 objects as in most cases either both types

of spheres appear or none. However many misclassifications happened due to the rule being too general.

```
% Background Knowledge
% Learnt Rules
class_3(A) :- alpha_2(B,A), sphere(B), in(A,B).
c_alpha_2(A,B) :- image(B), brown(A).
c_alpha_2(A,B) :- image(B), green(A).
c_alpha_2(A,B) :- image(B), cyan(A).
c_alpha_2(A,B) :- image(B), red(A).
c_alpha_2(A,B) :- image(B), large(A).
c_alpha_2(A,B) :- image(B), blue(A).
c_alpha_2(A,B) :- image(B), gray(A).
alpha_2(A,B) :- not c_alpha_2(A,B), sphere(A), in(B,A).
```

Figure 7.14: NAL output for classifying CLEVR images as class 3

The next framework is shown in Figure 7.15 is then used to classify the remaining images as class 1 and class 2. The framework suggests that cubes that are not small are class 1 images. This rule was devised as class 1 images comprised of larger cubes while class 2 small cubes. The ABA learning found that this was the differentiating factor. Despite it's many more images were classified as class 1 when they were supposed to be class 2. This could be as the rule is too general and a class 1 image may contain a small cube as well leading to a wrong classification.

```
% Background Knowledge
% Learnt Rules
class_1(A) :- alpha_2(B,A), cube(B), in(A,B).
c_alpha_2(A,B) :- image(B), small(A).
alpha_2(A,B) :- not c_alpha_2(A,B), cube(A), in(B,A).
```

Figure 7.15: NAL output for classifying CLEVR images as class 1

7.5 Explainablity and Interpretablity

One of the main reasons for developing a NAL pipeline is its enhanced explainability and interpretability. The programs generated by NAL are easily interpretable by humans because the rules are written in natural English. Users can follow the model's reasoning by constructing an argumentation of the predicates and ensuring that no parts of the reasoning tree are attacked. Additionally, users can inject their own rules to address cases the outputted program missed or to serve as regularisers, preventing the rules from being triggered in unwanted situations.

To evaluate these properties we conducted user studies asking 5 end users to interpret and explain the output of each model. All users had some knowledge of machine learning while a subset had previous experience in logic programming.

7.5.1 NAL Explanations

The explanation provided by the NAL pipeline was easy to interpret by the end users. Given the output programs from SHAPES and CLEVR learning tasks, all users could construct detailed reasoning as to why an image would be classified. However, for more complex programs, users could only understand what the program stated but struggled to articulate the program's meaning in natural English. The complexity was related to the nesting of negation. The more levels of negations which were present the more difficult the program was to reason about. This suggested that the NAL programs lack some interpretability. Another issue highlighted was the error posed by the neuro section of the pipeline. In some cases, the prediction was contrary to what the users believed due to some perceptual errors. Despite not only being able to fix these errors, users could pinpoint the incorrect fact demonstrating the tranparacy of the output.

7.5.2 GradCAM Explanations

An alternative explanation we provided for users was GradCAM images. These highlighted the regions of the images which contributed to the final classification (Figure 7.16). All users could formulate reasons as to why the reference images by stating the objects presented in the highlighted region. This provided them with some reassurance that the model had picked up on the right attribute. However, these regions didn't capture the deeper meaning of a classification e.g. the relation of a shape with another or the material of a shape. Because of this, users needed many images to recover this information.

Figure 7.16: Classification Heatmaps produced by ResNet-GradCAM model on SHAPES Images. (Left) Class 4 classification (Right) Class 6 classification

7.5.3 NS-CL Explanations

The explanations provided by NS-CL featured a visual grid that encompassed all the concepts involved in the final classification. This added information allowed users to deliver more detailed explanations compared to GradCAM images. The grid's high interpretability also facilitated the integration of Explainable Interactive Learning (XIL). XIL enables users to interactively guide the model by indicating which concepts to focus on or ignore, thereby enhancing model transparency. However, a limitation of this approach is that the grid does not encode more complex information, such as positional data. Consequently, users needed multiple images to retrieve this information, similar to GradCAM images.

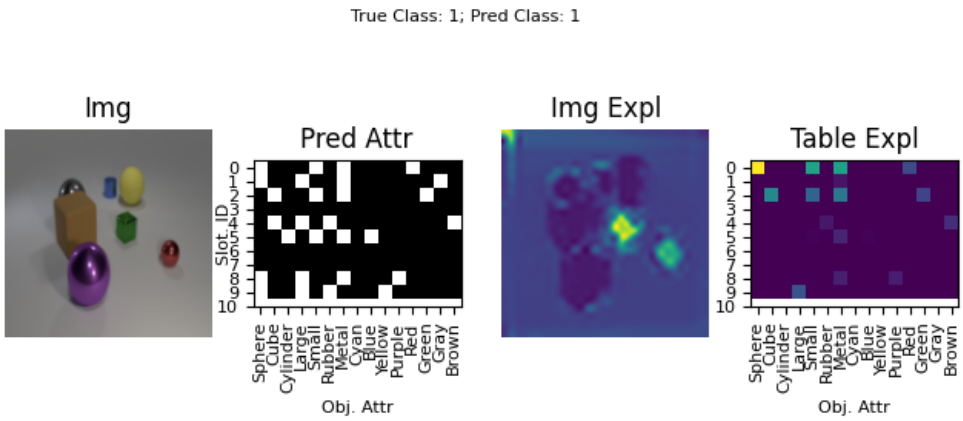

Figure 7.17: NC-CL Visual Explanation

Chapter 8

Future Work

The outcomes of the neuro-argumentative pipeline demonstrate the promise of integrating slot attention for object-centric learning with computational argumentation for reasoning. Nevertheless, the experiments have identified intriguing avenues for future research.

8.1 Quantifying Uncertainties

In the current implementation of our neuro-argumentative pipeline, the outputs neural section are treated as perfect facts which don't take into account the inherent uncertainties in their predictions. A line of research could focus on incorporating epistemic uncertainty within the slot attention to better capture and quantify the confidence in the extracted facts. This enhancement could then be incorporated into the ABA Learning algorithm to reason more effectively by considering the reliability of the information, leading to more robust rule-learning and decision-making processes. Techniques such as probabilistic object-centric learning [25] and integrating probabilities with logical facts like that of DeepProbLog [26] could be explored to quantify and integrate this uncertainty within the neuro-argumentative pipeline thereby improving the overall performance and accuracy of the system.

8.2 Computation Argumentation

Another avenue for future work involves addressing the limitations of using a pure assumption-based argumentation learning algorithm in our pipeline for image classification. ABA Learning primarily focuses on finding exceptions to differentiate between positive and negative examples, which has proven to be less effective for rule learning in this context. To enhance the system's ability to learn robust classification rules, a tailored learning framework that initially identifies the most general rules and subsequently refines them through exception learning could be developed. By prioritising the discovery of broad, generalisable rules before honing in on specific exceptions, this refined approach could yield more accurate and comprehensive classification programs, ultimately improving the overall performance of the pipeline.

8.3 Interpretabilty of Results

A final avenue for future work involves improving the interpretability of the classification programs learned by the neuro-argumentative pipeline. Currently, using brave semantics results in the production of multiple stable models at inference, making it challenging to interpret a final classification and/or understand the reasoning of classification from convoluted and nested rules. To address this, we propose adopting the concept of visual debates [27]. This approach involves creating visual representations of the argumentative process, allowing users to see and understand how different arguments and counterarguments lead to specific classifications. By leveraging visual debates, we can provide more intuitive and transparent explanations for the classification decisions made by the pipeline, ultimately improving user trust.

Chapter 9

Conclusion

Revisiting the objectives in the first section, the primary goal of the project was to explore neural argumentative machine learning within the context of images. To achieve this, we proposed a novel neural argumentative learning pipeline capable of learning ABA frameworks to classify images. The pipeline comprised two sections: the first utilised slot attention for object-centric learning, and the second used ASP-ABALearn to reason about the objects in the image.

Additionally, we aimed to evaluate the viability of computational argumentation as a method for reasoning about images. By using synthetic datasets such as CLEVR and our generated SHAPES, we assessed the extent to which neural argumentation learning could recover the logical rules underlying the images.

9.1 Contributions and Challenges

9.1.1 Neural Argumentative Learning Pipleine

In Chapter 4 we introduced the neural argumentative learning pipeline which combines the perceptual capabilities of neural networks with the explainability of a symbolic reasoner. The pipeline uses slot attention to extract objects from the image and their attributes. It then converts this to predicates using an object dictionary and runs the ABA-ASP learning strategy to learn new rules. The pipeline can then be run in inference mode to reason on new images using these rules. The pipeline can be seen as the main contribution of this project as it is a novel combination in the neuro-symbolic space.

Challeges

A key challenge during the pipeline's development was using logic-based learning (via ASP-ABALearn) to learn rules about the predicates identified by the neural section. Non-representative examples and incorrect perception of facts from these examples by the neural section could distort the learning problem, leading to prolonged execution times. In section 6.1.2 we discuss some measures to mitigate these

issues through clustering and thresholding. Although these approaches led to some improvements, they also increased the processing time between the neural and symbolic sections.

Another challenge was applying the slot attention model to real-world images. The model struggled to perceive objects accurately due to the requirement for linearly separable inputs and the presence of noise. Additionally, defining an object dictionary to describe the potentially infinite features and objects in the real world was infeasible. To address this, we briefly explored using the DINOSAUR architecture [28] to enhance object perception and generated abstract facts from the latent representation.

9.1.2 Exploring CA for Images

Another contribution of this project was the investigation of computational argumentation (CA) within the context of images. We introduced our own SHAPES dataset generation in Chapter 3 which creates SHAPES-like images which conform to the rules inputted in ASP. This allowed us to perform an in-depth analysis of the type of rules ASP-ABALearn could recover.

As discussed in Chapter 7, we found that the ABA Learning strategy could uncover most types of rules, but it tended to reproduce them in a more convoluted manner as nested exceptions. These nested exceptions made it challenging for users to interpret the results. However, with tools to simplify and visualise the reasoning, these explanations could be made more interpretable.

9.2 Concluding Remarks

Overall, this project has provided significant insights into the potential of computation argumentation in the space of image classification. The development of the Neural Argumentative Learning (NAL) pipeline demonstrates that an object-centric approach combined with computational argumentation such as ABA, is a viable neural symbolic approach for enhancing the transparency and explainability of AI systems.

Bibliography

[1] Yim J, Chopra R, Spitz T, Winkens J, Obika A, Kelly C, et al. Predicting conversion to wet age-related macular degeneration using deep learning. Nature Medicine. 2020 June 1,;26(6):892-9. Available from: https://www.nature.com/articles/s41591-020-0867-7. pages 1

[2] Chen H, Engkvist O, Wang Y, Olivecrona M, Blaschke T. The rise of deep learning in drug discovery. Drug discovery today. 2018;23(6):1241-50. ID: 271275. Available from: https://www.sciencedirect.com/science/article/pii/S1359644617303598. pages 1

[3] Nguyen TT, Tahir H, Abdelrazek M, Babar A. Deep Learning Methods for Credit Card Fraud Detection. arXiv preprint arXiv:201203754. 2020 December 7. Available from: http://arxiv.org/abs/2012.03754. pages 1

[4] Lui H, Long Z. An improved deep learning model for predicting stock market price time series. Digital Signal Processing. 2020 -07-01;102:102741. Available from: https://www.sciencedirect.com/science/article/pii/S1051200420300865. pages 1

[5] d'Avila Garcez A, Lamb LC. Neurosymbolic AI: the 3rd wave — Artificial Intelligence Review. Artificial Intelligence Review. 2023 March 15,;56(11):12387-406. Available from: https://doi.org/10.1007/s10462-023-10448-w. pages 1

[6] Hitzler P, Sarker MK. Neuro-symbolic artificial intelligence: The state of the art. IOS Press; 2022. pages 1

[7] Cyras K, Rago A, Albini E, Baroni P, Toni F. Argumentative XAI: A Survey. vol. 5; 2021/08/09. p. 4392-9. Available from: https://www.ijcai.org/proceedings/2021/600. pages 1

[8] Minh DP, A KR, Francesca T. In: Guillermo S, Iyad R, editors. Assumption-Based Argumentation. Argumentation in Artificial Intelligence. Boston, MA: Springer US; 2009. p. 199-218. Available from: https://doi.org/10.1007/978-0-387-98197-0_10. pages 1, 13, 14

[9] Proietti M, Toni F. Learning Assumption-based Argumentation Frameworks; 2023. Available from: http://arxiv.org/abs/2305.15921. pages 2, 17, 18, 19, 20

[10] Proietti M, Toni F. A Roadmap for Neuro-argumentative Learning. 2023. pages 2

[11] Francesco L, Dirk W, Thomas U, Aravindh M, Georg H, Jakob U, et al. Object-centric learning with slot attention. AdvNeural InformProcessSyst. 2020;33:11525-38. pages 9, 10

[12] Lifschitz V. Answer set programming. vol. 3. Springer Heidelberg; 2019. pages 10

[13] Eiter T, Ianni G, Krennwallner T. Answer Set Programming: A Primer. vol. 5689; 2009. Available from: https://www.researchgate.net/profile/Thomas-Krennwallner/publication/221510765_Answer_Set_Programming_A_Primer/links/0912f50b5f87b1bca6000000/Answer-Set-Programming-A-Primer.pdf. pages 12

[14] Dung PM. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence. 1995;77(2):321-57. ID: 271585. Available from: https://www.sciencedirect.com/science/article/pii/000437029400041X. pages 13, 16

[15] Modgil S, Prakken H. The ASPIC + framework for structured argumentation: a tutorial. Argument Computation. 2014 /01/01;5(1):31-62. Available from: https://content.iospress.com/articles/argument-and-computation/869766. pages 13

[16] Baroni P, Caminada M, Giacomin M. In: Abstract argumentation frameworks and their semantics. Handbook of formal argumentation. College Publications; 2018. p. 159-236. pages 13

[17] Angelis ED, Proietti M, Toni F. ABA Learning via ASP. ElectronProcTheorComputSci. 2023;385:1-8. Comment: In Proceedings ICLP 2023, arXiv:2308.14898. Available from: http://arxiv.org/abs/2308.15877. pages 17, 22, 23

[18] Tirsi C, Proietti M, Toni F. ABALearn: An Automated Logic-Based Learning System for ABA Frameworks. Lecture Notes in Computer Science. Springer Nature Switzerland; 2023. p. 3-16. pages 17, 20, 22

[19] Pettorossi A, Proietti M. Transformation of logic programs: Foundations and techniques. The Journal of Logic Programming. 1994;19-20:261-320. Available from: https://www.sciencedirect.com/science/article/pii/0743106694900280. pages 19

[20] Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Int J Comput Vis. 2020;128(2):336-59. Comment: This version was published in International Journal of Computer Vision (IJCV) in 2019; A previous version

of the paper was published at International Conference on Computer Vision (ICCV'17). Available from: http://arxiv.org/abs/1610.02391. pages 26

[21] Padalkar P, Wang H, Gupta G. NeSyFOLD: Neurosymbolic Framework for Interpretable Image Classification; 2023. Available from: http://arxiv.org/abs/2301.12667. pages 27

[22] Mao J, Gan C, Kohli P, Tenenbaum JB, Wu J. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision; 2019. Comment: ICLR 2019 (Oral). Project page: http://nscl.csail.mit.edu/. Available from: http://arxiv.org/abs/1904.12584. pages 27

[23] Andreas J. SHAPES Dataset; 2017. https://paperswithcode.com/dataset/shapes-1. pages 28

[24] Stammer W, Schramowski P, Kersting K. Right for the Right Concept: Revising Neuro-Symbolic Concepts by Interacting with their Explanations. arXiv preprint arXiv:201112854. 2020. pages 30

[25] Kori A, Locatello F, Santhirasekaram A, Toni F, Glocker B, Ribeiro FDS. Identifiable Object-Centric Representation Learning via Probabilistic Slot Attention; 2024. Available from: http://arxiv.org/abs/2406.07141. pages 50

[26] Manhaeve R, Dumančić S, Kimmig A, Demeester T, Raedt LD. DeepProbLog: Neural Probabilistic Logic Programming; 2018. Comment: Accepted for spotlight at NeurIPS 2018. Available from: http://arxiv.org/abs/1805.10872. pages 50

[27] Kori A, Glocker B, Toni F. Explaining Image Classification with Visual Debates; 2023. Available from: http://arxiv.org/abs/2210.09015. pages 51

[28] Seitzer M, Horn M, Zadaianchuk A, Zietlow D, Xiao T, Simon-Gabriel CJ, et al.. Bridging the Gap to Real-World Object-Centric Learning; 2023. Comment: ICLR 2023 camera-ready version. Available from: http://arxiv.org/abs/2209.14860. pages 53

[29] Kabra R, Burgess C, Matthey L, Kaufman RL, Greff K, Reynolds M, et al.. Multi-Object Datasets; 2019. https://github.com/deepmind/multi-object-datasets/. pages