The advantages of non-monotonic logic in modular architectures: High performance and interpretable outputs with limited training data

Heather Riley

A thesis submitted in fulfilment of the requirements for the degree of Master of Engineering in computer systems engineering, the University of Auckland, 2019. This thesis is for examination purposes only and is confidential to the examination process.

Department of Computer Systems Engineering
University of Auckland
New Zealand

February 20, 2019

Abstract

Deep learning has become the state of the art algorithm for many areas of machine learning research. Deep learning can achieve high performance but it requires a lot of training data and is not readily interpretable. This makes it difficult to develop algorithms for new applications, as errors cannot easily be resolved when the algorithm's reasoning is obscure, and large datasets cannot always be obtained. To address these limitations logic knowledge bases can be used for interpretable reasoning while deep learning is used used for any processing that is not suitable for representation by a knowledge base. A knowledge base requires no training data and provides interpretable outputs, and in addition, the intermediate outputs from knowledge base components can be used to guide learning for the deep learning components, making training more efficient. These advantages were central to the design of a novel architecture, which used deep learning for image processing and a combination of non-monotonic logic and decision tree induction for interpretable reasoning. Performance was evaluated with visual question answering (VQA), and a state of the art planning architecture was used to show the advantages of knowledge bases in dynamic domains. The VQA architecture's performance was evaluated in two domains: estimating the stability of simulated block towers, and determining the messages conveyed by traffic signs. Experimental results showed that the proposed architecture achieved interpretability and high performance with small training datasets. The planning architecture was evaluated in a dynamic tower-building domain. Another domain in which a simulated Turtlebot acted as an office assistant was used to show that the VQA and planning architectures could be combined. The performance of knowledge base reasoning was improved through the use of automated axiom learning, which was shown in all four domains. In the stability and traffic domains the use of axiom learning to add to the knowledge base improved VQA accuracy, while in the tower building and office assistant domains the action plans developed after axiom learning were more sensible and efficient than those developed before.

Acknowledgements

The author would like to thank Dr. Mohan Sridharan for supervising the work that went into this thesis and for providing useful guidance and advice throughout the year. The author would also like to thank Lucien Koefoed and Steve DeSouza for suggesting improvements after reading early drafts of this thesis.

Contents

1. Introduction

Artificial intelligence (AI) algorithms are increasingly being used in critical application domains such as security, healthcare and autonomous navigation. In general, it is likely that humans interacting with an autonomous system designed for critical domains will want to know why and how the system arrived at particular conclusions. For example, consider automated diagnosis systems that are designed to take symptoms and patient history as input and generate diagnoses and treatment options as output. Patients and doctors are both going to want to know why a particular diagnosis or treatment option was suggested so that they can determine the best course of action. Designing more interpretable algorithms - that is, algorithms whose internal reasoning can be seen and understood by humans - is important if humans are to place trust in AI in critical domains. Interpretability is also important as a diagnostic tool for complex AI applications.

Deep learning is a machine learning algorithm that is increasing in popularity and has become the state of the art for many pattern recognition tasks. Deep neural networks can achieve high performance when sufficient training data is provided. Unfortunately, their training data requirements are prohibitive and there are many specialised domains (e.g. automated diagnosis of rare diseases) where the amount of training data required to implement a purely deep learning algorithm cannot be obtained. Additionally, deep learning algorithms have poor interpretability, due to the black box nature of a neural network. Due to its capacity for high performance and ease of implementation, deep learning has become a go-to method of increasing prevalence, particularly in the area of computer vision where they are used for both feature extraction and decision making. This in itself is not a problem, and the use of neural networks for image processing is a sensible choice. However, many computer vision applications require additional processing for which neural networks may not be the most suitable algorithm, such as reasoning about the features of an image after they have been extracted. The use of end-to-end deep learning models puts unnecessary restrictions on what constitutes an acceptable domain for these applications. Sufficient training data must be provided to train the large network, and the uninterpretable nature of neural networks leads to a lack of trust in these algorithms, as well as making it difficult to diagnose and correct any problems in their performance.

The architecture described in this thesis was designed for training efficiency and interpretability, making it suitable for domains with limited training data and for applications that humans need to understand and trust. This architecture combines deep learning with non-monotonic logic reasoning and decision tree induction to get the best of both worlds. Deep learning was used to provide high performance for complex tasks such as image processing, while knowledge bases and decision trees were used to provide interpretable outputs for tasks that could be represented by axioms or dendrograms. Only applying the deep learning components to specific tasks reduced the overall training data requirement.

Knowledge bases are useful, particularly in domains with limited training data, because they do not require training and they provide interpretable outputs. One downside to knowledge bases, however, is that they need to be created by a human designer, as opposed to neural networks which automatically learn how to correctly match input to output. Knowledge bases can present particular difficulty in complex domains, where many axioms are necessary to fully represent the domain and it is difficult for a human designer to capture all of them. One way to overcome this drawback is to use an automated 'axiom learning' methodology to add to an incomplete knowledge base. This thesis presents an axiom learning system that can be used to add new state constraints and statics to a knowledge base. The methodology presented is novel but was inspired by a similar methodology, presented in [38], that was designed to add executability conditions to knowledge bases.

One of the biggest computer vision challenges is visual question answering (VQA), as in addition to the common computer vision tasks of image processing, feature extraction and object detection, a VQA algorithm must reason about a question and determine the answer. The questions asked might be things like 'what colour is the man's shirt' or 'is there a cat in this image'. Traditional VQA datasets contain tens or hundreds of thousands of images [6, 7], so applying VQA to a domain that has limited training data is a challenging goal. A standard VQA architecture that forms the base of most state of the art algorithms is composed of a convolutional neural network (CNN) for image processing and a recurrent neural network (RNN) for question answering. The proposed architecture used, in addition, a knowledge base and a decision tree for classification and a knowledge base for question answering. This reduced the load on the deep learning components of the architecture, making their tasks easier to learn with limited training data. The answer sets generated by the knowledge base, or the branches used by the decision tree, were parsed and the information used to construct explanations for the classification decisions made. In addition, to show that non-monotonic logic is useful in a variety of domains, a planning architecture that relies on a planning knowledge base was applied to a robotic task, and this architecture was then combined with the proposed VQA architecture.

It was hypothesised that the proposed architecture would outperform a baseline architecture that was purely based on deep networks when the size of the training dataset was small, and that it would be able to provide explanations for its classification decisions. It was expected that the proposed architecture could be combined with a planning architecture for dynamic domains, and that an automated axiom learning process would improve the performance of knowledge base reasoning, in both static and dynamic domains. Experimental results confirmed these expectations.

The rest of this thesis is laid out as follows: Chapter 2 describes previous research related to the goals of the thesis, Chapter 3 describes the proposed architecture and a methodology that can be used to apply the proposed architecture to new domains, and Chapter 4 discuses experimental results.

2. Related work

Deep learning is a popular machine learning technique, and it is increasingly being used for applications with complex inputs, such as computer vision [1, 2, 3, 4, 5]. There are three issues currently affecting many deep learning models: they require a lot of training data and thus are difficult to train, they lack interpretability, and they are often affected by dataset bias. In this thesis it is hypothesised that the use of commonsense knowledge reasoning can help to alleviate these issues. The following sections explore four topics: training data requirements, interpretability, dataset bias, and reasoning with knowledge.

2.1 Training data volume

Most deep learning algorithms need to be trained on large datasets in order to achieve high performance. For visual question answering (VQA) this training data is usually obtained from benchmark databases such as Microsoft Common Objects in Context (MSCOCO) [6] or the Dataset for Question Answering on Realworld Images (DAQUAR) [7]. Constructing such large databases can be an expensive and time-consuming process, meaning that most research is only being done on a few popular generic datasets, and finding enough training data for specific applications can be difficult. A discussion of the methods currently being used to mitigate the training data requirements of computer vision algorithms follows.

Transfer learning was used in [46, 47] to reduce the amount of training data required. With transfer learning, a neural network that has already been trained on a large, generic dataset (such as ImageNet [48]) is fine-tuned with a domain-specific dataset. This improves performance when the domain-specific dataset is too small to train a neural network from scratch. The model for ship classification presented in [46] was a pre-trained Resnet-50 network [59] which was fine-tuned on a database of approximately 2000 images of ships. A similar model for landscape classification was based on a pre-trained CNN which was fine-tuned on a database of approximately 5000 military vehicle images [47]. In this paper pre-training was done with 50,000 unlabelled landscape images (similar to the landscapes in the target images, but without the military vehicles). Though this reduced the labelling requirement, a large image dataset was still required. Transfer learning cannot be applied to all domains as many do not have a large dataset available that is similar enough to their target data.

Active learning was used for VQA in [10] and for image classification in [45] to reduce the amount of labelled data required. For the VQA application a large question-image database was used, with only some of the question-image pairs labelled with answers. The model was first trained with the labelled training data. It then iteratively expanded its training set by selecting image-question pairs involving concepts it was uncertain about and asking an ‘oracle’ (human annotator) for the answer. Experimental results showed that using active learning instead of annotating all questions in the database with answers saved on time spent annotating answers by about 20%. However, a large database of question-image pairs still had to be constructed.

Another way to reduce the amount of training data required is to use an attentional model to determine which region of the image the model should get the answer from. A model was proposed that used stacked attentional networks to improve VQA performance [8]. The words of a question were embedded to a vector, and at each time step a word vector was passed to a long short term memory (LSTM) [60] network that updated its hidden state. Once every word in the question had been processed, the final hidden layer in the LSTM was used as the question vector. A visual geometry group neural network (VGGNet) [61] was used to extract the image feature map from a raw image. The question vector and image feature map were given as inputs to a single layer neural network, the output of which was passed through a softmax function to generate an attentional distribution over the image. This was then used to calculate the weights of the image vectors, each from a different region in the image. These image vectors were combined with the question vector to form a query vector that encoded both question information and the visual information that was relevant to the potential answer. The query-attention process was done iteratively using multiple attention layers, each extracting more fine-grained visual attention information for answer prediction. With this method objects were first identified and located, and then concepts in the question used to rule out irrelevant objects (e.g. if the question asked ‘what is sitting in the basket’, any objects that did not fit the concept ‘sitting in’ could be ignored). Once as many objects as possible had been ruled out the remaining object was the answer to the question.

In a similar paper a co-attentional model was proposed to improve robustness to variations in question wording [9]. Most attentional models create a spatial map highlighting which regions of the image to pay attention to. This model went a step further and highlighted which words of the question to pay attention to. For example, the questions ‘how many horses are in this image’ and ‘how many horses can you see’ have the same meaning but different wording, which could affect the performance of simpler VQA models. However, with a co-attentional model, the first three words ‘how many horses’ are identified as the most important words in the question, so the wording will not affect the answer. Two different co-attention mechanisms were presented, parallel co-attention, and alternating co-attention. Parallel co-attention attended to the image and question simultaneously. The similarity between image and question features at all pairs of image-locations and question-locations was calculated as an affinity matrix. This affinity matrix was used as a feature and the model trained to predict image and question attention maps from it. In alternating co-attention the model sequentially alternated between generating image and question attention. First, the question was summarized into a single vector. The question vector was then used to attend to the image. The attended image feature was then used to attend to the question. At each step an attention operation took the features and attention guidance (derived from either the image or question) as inputs, and output the attended question or image vector. The answer was predicted with the coattended image and question features as inputs. Experimental results from both of these papers [8, 9] showed that the attentional models outperformed baselines where neural networks were used without any attentional model. Improved performance with the same amount of training data suggests that the attentional models would be able to achieve the same performance as plain neural network models with less training data, however, this was not a focus of these papers.

A Deep Attention Neural Tensor Network (DA-NTN) for VQA used tensor-based representations to discover joint correlations between images, questions and answers [72]. An attention module selected a discriminative reasoning process, and optimized the DA-NTN through regression with KL-divergence losses to improve scalability of training and convergence. Schwartz and colleagues describe a general-purpose attention mechanism in the context of VQA [73]; the approach learned high-order correlations between various data modalities, directing attention to elements in the data modalities that were relevant to the task at hand.

2.2 Interpretability

A model designed to explain neural network decisions was proposed to address the issue of how difficult to understand their reasoning can be [19]. GradCAM used the gradient information flowing into the last convolutional layer of a CNN to understand the importance of each neuron for a decision of interest. The feature map of a convolutional layer of a neural network was used to compute the gradient score for a target class. These gradients were global-average-pooled to obtain neuron importance weights, with the weight representing the importance of the target class. Although this was a step in the right direction, the explanations it provided (they were represented as heat maps) were not very intuitive.

A similar model was used to improve the interpretability of deep neural network EEG classifications [51]. The technique was called ‘layer-wise relevance propagation’ and the output was a heatmap showing the relevance of each input for the final outcome.

A model called ‘interpretable CNN’ was designed in such a way that each filter in the higher convolutional layers represented a specific object part [49]. The model automatically assigned object parts to filters during training. As these object parts were selected by the algorithm, not by a human, some of them were not intuitive. Images could be generated with detected object parts highlighted to indicate what parts of the image the CNN was focusing on.

Fong and Vedaldi used image blurring to detect the most relevant part of an input image [50]. Sections of an image were blurred and the blurred images run through a CNN. The results were compared to the result from the unblurred image, and the blurred shape that suppressed the CNN’s softmax prediction was used to create an image mask highlighting which section of the image was most relevant to the CNN’s classification. This showed what part of the image the algorithm is paying attention to.

Researchers have also developed general approaches for understanding the predictions of any given machine learning algorithm. Second-order approximations of influence diagrams can be used to trace any model's prediction through a learning algorithm back to the training data in order to identify training samples most responsible for any given prediction [74]. Ribeiro and colleagues developed a framework that analyzed any learned classifier model by constructing a interpretable simpler model that captured the essence of the learned model, and formulated the task of explaining models based on selected representative instances and explanations as a submodular optimization problem [75].

The methods listed above are useful for highlighting which parts of a neural network are the most active or relevant to the output, or which parts of an image it is paying the most attention to. However there is still a gap in the interpretability literature with regards to explanations. An explanation is an additional output generated by an algorithm in which it explains, with words (e.g. a paragraph of text printed to the screen), its reasons for selecting a particular class or value for the main output.

2.3 Dataset bias

Dataset bias occurs when a neural network is trained to simply match inputs to the outputs that they were most often paired with in the training dataset, rather than learning to reason about the information it is given. In visual question answering an issue that can arise from training a model with some of the popular large datasets is that the algorithm learns to answer questions based on similar question-answer pairs that it has seen, without reasoning about the image at all [1, 13, 14]. These models often achieve high accuracy if testing and training data are similar, and this can give a false idea of how well the algorithm would work in a real world application where it needs to reason about its environment. The following papers present datasets and models designed to mitigate the effect of dataset bias.

A balanced version of the VQA dataset was presented to address the issue of models relying on language priors to give them the answer without actually understanding the image [15]. In this database a complementary image was added for every question so that each question could be paired with two separate images that give two different answers. A similar balanced database that includes only yes/no questions was presented in [16]. With every question paired with multiple different answers, algorithms cannot rely on language priors to achieve high performance.

A similar paper presented both a dataset designed to reduce bias and a model designed to not rely on bias were presented [17]. The databases were new splits of the VQA v1 and VQA v2 datasets, called Visual Question Answering under Changing Priors (VQA-CP v1 and VQA-CP v2 respectively), which were arranged to have different answer distributions in the training set and testing set for each type of question (e.g. for yes/no questions the training set might have mostly yes answers but the testing set might have mostly no answers). This meant that in order to perform well a model had to rely on information from the image rather than relying on common answers from the training set. Along with the dataset, a model called Grounded Visual Question Answering (GVQA) was designed specifically to ignore language priors and rely on information from the image. GVQA separated the tasks of locating relevant objects and their features in the image, and choosing from a range of possible answers, so that the question was used only to analyse the image and could not be directly used to guess the answer. An example given in the paper was asking 'what colour is the dog?' The question was parsed to find out what visual concepts need to be located in the image (in this case 'dog') and what type the answer should be (in this case 'colour'). The model then located the patch in the image corresponding to 'dog', detected features such as black and furry, and finally output black because it matched the required answer type. The algorithm makes two assumptions about the contents of an image: (1) the attended image patch required to answer a question has at most one dominant object in it; and (2) every object has at most one dominant attribute from each attribute category.

Another way to reduce dataset bias would be to include image-question pairs that do not make sense, so that the model cannot learn how to predict answers from the question alone. However it would be disadvantageous if the model responded to nonsensical questions with nonsensical answers. In one VQA architecture a pre-trained captioning model was used to determine whether a question was relevant to an image, and only if it was relevant was an answer attempted [18].

2.4 Reasoning with knowledge

It has been suggested in a review of previously attempted knowledge-directed vision techniques (mostly from the 1980s) that a more modern attempt is called for [58]. The proposed architecture presented in this thesis uses non-monotonic logic knowledge bases for reasoning about computer vision tasks such as VQA and planning. This section presents some other ways in which knowledge has been used in computer vision or question answering applications. The techniques used include production-system languages, a declarative programming language called Loom, physics engines, predicate knowledge bases, description logic, temporal logic, fuzzy logic, and first-order logic. This section also introduces non-monotonic logic and describes some ways in which it has previously been used.

A declarative programming language called Loom has been used for the detection of objects such as runways and taxis in aerial Airport images [54]. Production-system languages have been used for selecting which algorithm to use for segmentation [56]. A recent approach for VQA used physics engines and prior knowledge (of domain objects) to realistically simulate and explore different situations. These simulations guided the training of deep network models that anticipated action outcomes and answered questions about hypothetical situations [40].

One question answering architecture used domain knowledge to reason about general-knowledge questions regarding objects in an image, such as 'what country is this animal from', thus enabling a wide range of questions to be answered without making the amount of training data needed impractical

[11]. The model detected objects in the image and extracted information regarding those objects from a knowledge base. It then used information from both the image and the knowledge base to answer a natural language question. The knowledge base was composed of predicates that encoded relations such as *(giraffe, from country, Africa).* Though this enabled the algorithm to answer a wider range of questions with a standard amount of training data, it did not reduce the amount of training data required to answer questions about the image.

Another question answering architecture used description logic to encode information about the structures of buildings [41]. Information such as lists of external features (e.g. wall, roof), internal features (e.g. stairwell, room), and the materials that the building was constructed from were encoded in a hierarchical list that could be parsed for the answers to questions. A summary of recent progress combining descriptive logic with computer vision was given in [53], with examples of description logic being used for pattern classification and image interpretation.

Temporal logic can be used for scene recognition [52]. An algorithm was presented that was able to, though video processing, recognise different users and interpret their gestures. The gestures tracked over time were compiled in a knowledge base that reasoned about the actions being taken by different people to determine what activity was being performed in the video.

Another form of logic that has been combined with computer vision is fuzzy logic [42, 43, 44]. Fuzzy logic is different from most other types of logic, which tend to only assign the values true, false, or sometimes unknown. Fuzzy logic, on the other hand, can represent things qualitatively and is useful for probabilistic reasoning. Fuzzy logic and computer vision are often combined in robotic applications [42], with computer vision predicting object locations and fuzzy vision tracking object trajectories over time, allowing the robot to easily interact with the object.

A question answering architecture was developed that used a first-order logic knowledge base to answer common knowledge questions such as ‘what is the capital city of France’ [39]. Another architecture was presented that reasoned with first-order logic representations and incrementally refined action operators [76]. Cognitive architectures have also been developed to extract information from perceptual inputs to revise domain knowledge represented in first-order logic [80], and to combine logic and probabilistic representations to support reasoning and learning in robotics [81, 65]. Out of the approaches used in the current literature, the one most similar to non-monotonic logic is first-order logic. Both forms of logic can be encoded in knowledge bases of axioms. First-order logic is more efficient and less computationally expensive that non-monotonic logic, but it cannot represent uncertainty or revise beliefs.

A complete review of knowledge-based computer vision is given in [55]. It was written in 2010, so does not discuss the most recent advances, but the only type of logic mentioned is first-order logic. From the related work presented in this section of the thesis the conclusion that there are currently no algorithms using non-monotonic logic for interpreting the content of images or videos can be drawn.

Non-monotonic logic formalisms such as Answer Set Prolog (ASP) have been developed and used in cognitive robotics [64] and other applications [66]. ASP has been combined with inductive learning to monotonically learn causal laws [77], and the theory of actions has been expanded to learn and revise domain knowledge represented as ASP programs [78, 79].

A disadvantage of logic programming methods is that they do not support incremental learning of new axioms or generalisation. These challenges can be addressed using interactive task learning, a general framework for acquiring knowledge using labeled examples or reinforcement signals obtained from observations, demonstrations, or human instructions [82, 83]. Sridharan and Meadows developed such a framework that combined non-monotonic logical reasoning and relational reinforcement learning to learn action models or search control rules to be used for reasoning or learning in dynamic domains [68].

It is hypothesised that reasoning with knowledge will allow artificial intelligence architectures to reduce training data requirements and improve interpretability. Though the related works mentioned here have interesting and useful capabilities, none have made significant progress on the goal this thesis addresses: creating a knowledge-based architecture that is designed in such a way that it minimises training data requirements, provides interpretable outputs, can be used for a variety of applications and is unaffected by dataset bias. This thesis proposes an architecture that is designed to achieve that goal, and it is described in detail in the next chapter.

3. Proposed architecture

This thesis proposes a novel architecture for the task of visual question answering (VQA), in which an algorithm must generate an answer to a question about an image. This is a good example of a task that can make use of incremental learning and explainable reasoning, but the proposed architecture could easily be adapted for other applications. The methodology behind the architecture's design combines deep learning with knowledge bases (and decision trees when additional flexibility is required) to get the best of both worlds - high performance when processing raw inputs such as images, interpretable outputs, and manageable training data requirements. This methodology is generalisable and can be used to design architectures for a variety of tasks. As well as the proposed VQA architecture, this thesis describes a state of the art planning architecture that follows the same principle of combining knowledge bases with deep learning.

The VQA architecture was designed to improve training efficiency and provide explanations for the image classification desicions made. This architecture is made up of four main components, as can be seen in the block diagram in Figure 3.1. The planning architecture is discussed in Section 3.6. An advantage of the modular nature of these architectures is that components can be exchanged, allowing architectures designed for different tasks to be combined for applications that require several functionalities.

1. The yellow blocks in the block diagram represent the feature extraction component. This is a group of convolutional neural networks (CNNs) that process the input image and determine what features are present.

2. The blue blocks represent the classification component, which uses two systems - a knowledge base and a decision tree. The knowledge base does not need to be trained, but it is not very flexible and cannot deal with unexpected feature combinations (this could happen if a CNN misclassifies a feature). The decision tree is used to classify inputs that the knowledge base cannot. Using the decision tree on only a small subset of the feature space (i.e. combinations that the knowledge base has no classification rule for) reduces the amount of training data required.

3. The orange blocks represent the question answering component. Like the classification component, this uses two systems - a knowledge base and an RNN. Once again this allows the best of both words, with the knowledge base dealing with most inputs to reduce the amount of training data required, and an RNN providing the flexibility needed for unexpected inputs.

4. The purple block represents the explanation generation component. This takes as input either the

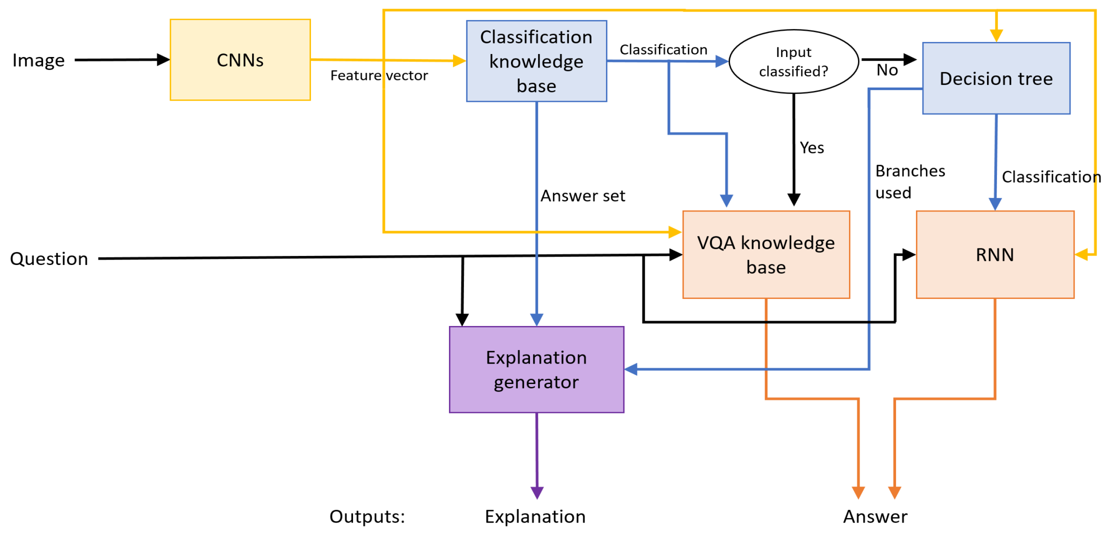

Figure 3.1: Block diagram of the components of the proposed architecture.

answer set from the knowledge base or a list of the branches used by the decision tree, depending on which system was used for classification. This input is parsed and the information gained used to construct an intuitive explanation for the classification decision made.

The proposed architecture used deep learning for processing the image and answering unexpected questions, and used a knowledge base and decision tree for classifying the image and answering expected questions. Expected questions are defined as those that are paired with a combination of features (extracted from the image) that the knowledge base has axioms for. An unexpected combination of features can be produced if any of the CNNs misclassify a feature, and in these cases the flexibility of an RNN which can learn to map even incorrect inputs to the correct outputs, so long as the same mistakes are commonly made, is useful. This design gained the advantages of deep learning while avoiding the disadvantages; deep learning was used to provide high performance for tasks that would have been difficult to encode in a knowledge base, and because these tasks were made as specific as possible the training data requirement was kept to a minimum. The use of the non-monotonic logic reasoning and decision tree induction also provided interpretable intermediate outputs. Another advantage of separating the VQA task into classification and question answering was that it avoided a problem that can occur with biased datasets, where the algorithm simply chooses the most common answer without reasoning about the image at all. With a separate classification step being trained specifically for classification accuracy the algorithm was forced to learn how to reason about the image. This architecture was designed to be versatile and can be applied to a wide variety of domains.

The steps involved in applying the proposed VQA architecture to a new domain are:

1. Choose relevant features

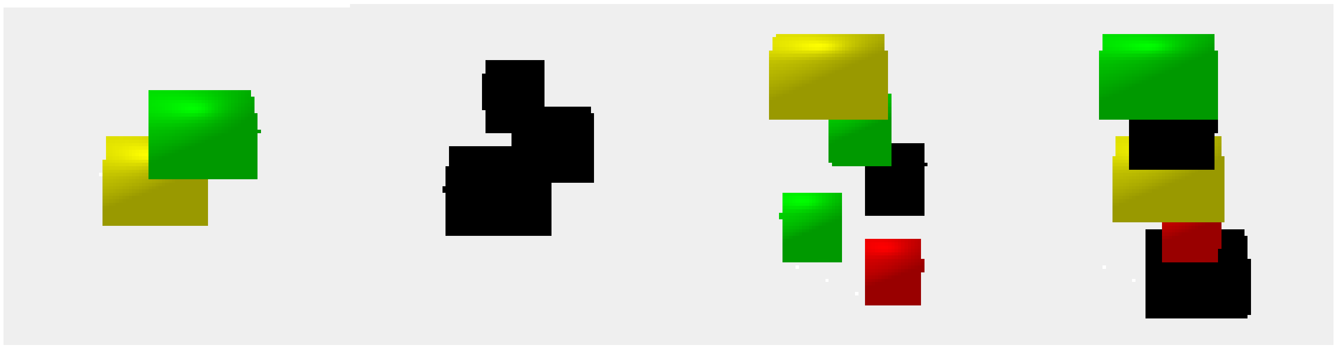

Figure 3.2: Illustrative images of structures of blocks from a physics-based simulator.

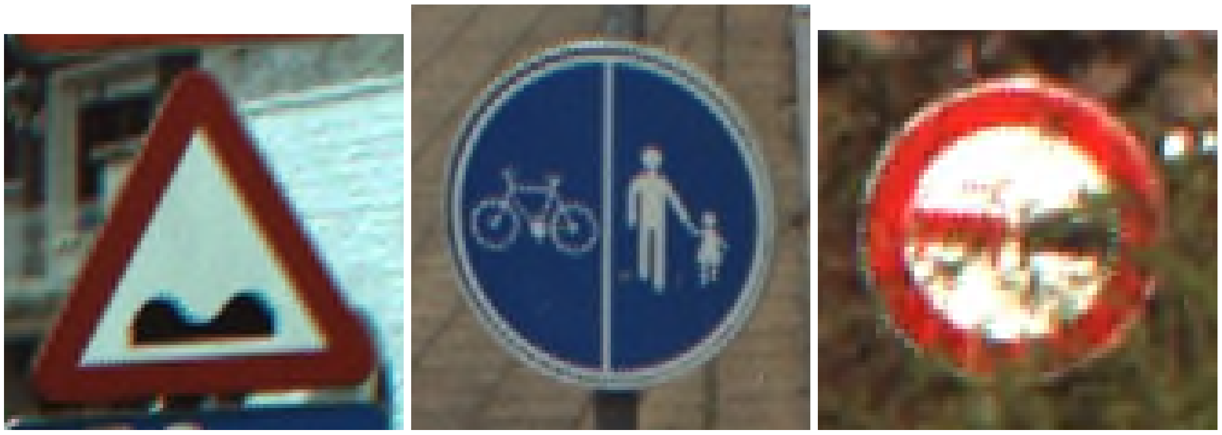

Figure 3.3: Illustrative images of traffic signs from the BelgiumTS dataset [33].

2. Design neural network structures
3. Create classification knowledge base
4. Create decision tree algorithm
5. Create question answering knowledge base
6. If the domain is dynamic, create planning knowledge base
7. If performance needs to be improved, implement axiom learning

Example code from four example domains is available in a Github repository [84]. Examples include scripts for training and using CNNs, RNNs, and decision trees, as well as ASP files showing classification, question answering and planning knowledge bases. A controller script can be written in any sequential language (e.g. Python [57]) to manage the inputs and outputs between the components of the architecture. Figure 3.1 can be referred to as a guide for this. Methodology sections throughout this chapter describe each of the above steps in more detail. First, four domains that are used throughout this thesis to provide explanatory examples are introduced.

3.1 Domains

The following list describes the domains that were used to obtain experimental results for the proposed architecture. The first two were used for the VQA task, the third was used for the planning task, and the fourth was applied to both tasks simultaneously.

1. **Structure Stability (SS):** This domain used 2500 images of structures created in the PyBullet physics simulator [35]. These structures were composed of varying numbers of blocks with randomly chosen sizes, colours and positions (see Figure 3.2 for examples). The relevant features of this domain were the number of blocks, whether the structure was on a lean, whether the structure had a narrow base, and whether any block was displaced (placed so far to the side) such that it was not well balanced on top of the block below. The objective was to classify structures as being stable or unstable, and to answer explanatory questions such as 'why is this structure unstable?' and 'what needs to happen to make the structure stable?'.

2. **Traffic Sign (TS):** For this domain the BelgiumTS benchmark dataset [33] was used with ≈ 5000 real-world images (total) of 62 different traffic signs (see Figure 3.3 for examples). The features of this domain were the primary symbol/sign in the middle, secondary symbol, shape (e.g., circle, hexagon), color (main, border), and the background image. The objective in this domain was to classify the traffic signs in images and to answer explanatory questions such as 'what is the sign's message?' and 'how should the driver respond to this sign?'.

3. **Tower Building (TB):** This dynamic domain involved a simulated robot building a tower of boxes. The robot's objective was to make the tower as high as possible without risking collapse. There were five boxes available - a large blue one, a medium orange one, a medium yellow one, a medium red one, and a small green one. Putting a box on top of a smaller box was unstable and usually resulted in the tower collapsing. If the tower collapsed or the robot's attempt to place a box did not result in the box being balanced on top of the tower the robot would use the state and action that led to that to learn axioms about which box combinations would be unstable. More details on this domain are given in Section 3.6.

4. **Robot Assistant (RA):** This dynamic domain was simulated in Gazebo [84], using models from the 3DGEMS database [85]. A simulated Turtlebot (model 2.0), a small wheeled mobile robot, was used to deliver messages to the workers in an office world (see Figure 3.4). After delivering a message it would observe the room and objects around it. It would then return to the sender of the message, and answer questions about its observations. More details on this domain are given in Section 3.6.2.

The proposed architecture was designed for use in domains with two main features: firstly, they should be specialised enough that all the relevant features can be feasibly captured within a knowledge

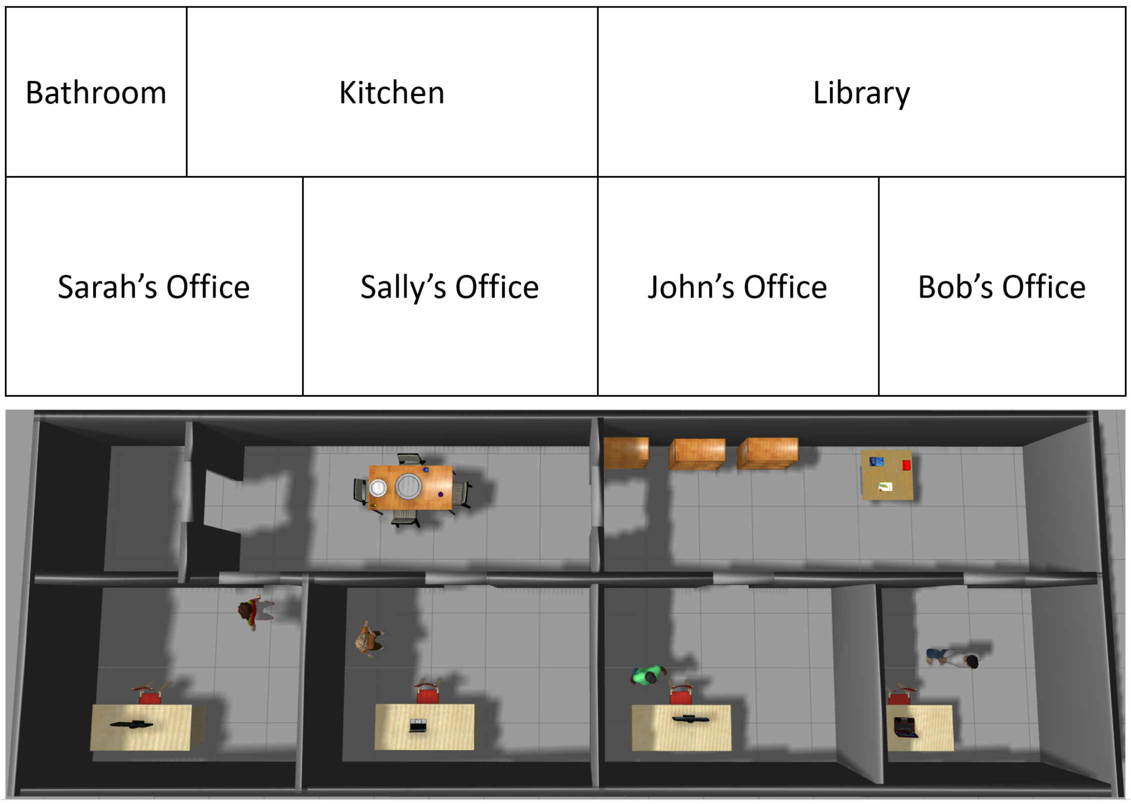

Figure 3.4: The layout of the office building in the Gazebo simulation of the RA domain. The top half of the figure shows a map of the layout with room names, while the bottom half shows the composition of the rooms with models of objects and people included.

base, and secondly, they should only have a limited amount of training data. While the proposed architecture can also be used for domains with large training datasets, end-to-end deep learning algorithms are likely to perform just as well with these datasets so the proposed architecture may not be the best choice (unless interpretability is of high importance e.g. in critical domains where humans want to understand the automated system). There is currently a gap in the state of the art for specialised domains with small training datasets. Most state of the art VQA algorithms are designed for benchmark datasets with huge amounts of training data. These benchmark datasets would be unsuitable for the proposed architecture because they contain a wide variety of image and question types and it would not be feasible to create a knowledge base capable of classifying all of the image types and answering all of the questions. For these reasons the SS and TS domains were selected as suitable candidates for testing the proposed architecture's VQA performance.

3.1.1 Methodology: Choosing relevant features

Each domain will have a separate list of features that are used for classification. The user must select these based on their knowledge of the domain. The features should be visual, and simple enough that

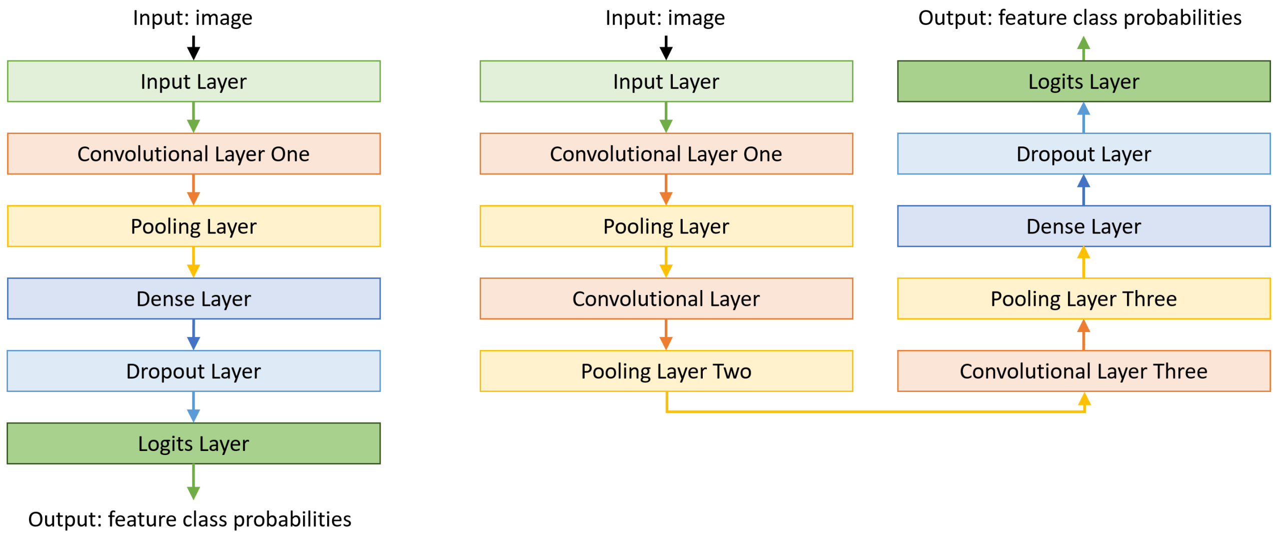

Figure 3.5: Illustrative example of the CNN architecture used for feature extraction in the SS and TS domains. Part (a), on the left, shows the simplest possible CNN that would be used in this architecture while part (b), on the right, shows a more standard CNN structure.

CNNs can detect them in images without much training. They should also be class-specific enough for a knowledge base and decision tree to use the features for classifying the images. For example, in the TS domain the features are the shape of the sign, the colour, the colour around the border, the symbol, the secondary symbol, the background image, and whether there is a cross. If, during the creation of the classification knowledge base (discussed in Section 3.3.2) accurate classification axioms cannot be found, it is likely that more refined features need to be chosen.

3.2 Feature extraction with CNNs

The first component of the architecture takes images as input and extracts user-defined features that can then be used to classify the image. For the VQA example domains (SS and TS), semi-automated annotation was used to obtain the relevant features from images of different scenes. The feature representation for each domain was selected based on domain expertise about the characteristic features of the corresponding domain.

In the SS domain the features were:

- The number of blocks in the structure (between two and five);
- Whether the structure was on a lean (true or false);
- Whether the structure had a narrow base (true or false);

- Whether any block was displaced, i.e., placed far to one side such that it was not balanced on top of the block below it (true or false).

In the TS domain the features were:

- The primary symbol in the middle of the traffic sign (there were 39 different primary symbols such as *bumpy road, slippery road, stop, left turn,* and *speed limit*);
- The secondary symbol in the traffic sign (there were 10 different secondary symbols such as *disabled, car* and *fence*);
- The shape of the sign (*circle, triangle, square, hexagon, rectangle, wide rectangle, inverted triangle,* or *diamond*);
- The main color in the middle of the sign (*red, white,* or *blue*);
- The border color at the edge of the sign (*red, white,* or *blue*);
- The sign's background image (some signs had the symbol placed over a background image such as a square or a triangle);
- The presence of a cross, in red or black, across a sign to indicate the end or invalidity of a zone (with the sign without the cross indicating the beginning or validity of the zone), e.g. a parking sign with a cross implied no parking.

As stated earlier, CNNs were trained and used for this task. A separate CNN was trained for each feature, and each CNN was designed using the same training methodology, which is described in the following subsection.

3.2.1 Methodology: Neural network design

There are a variety of languages and libraries available for the implementation of machine learning algorithms. For the implementation of the decision trees and neural networks presented as examples in this thesis Python [57] and TensorFlow [62] were used.

The CNNs and RNN should be designed to be as simple as possible while still achieving high accuracy. Because some features are more complex than others, a separate CNN should be designed for each feature. The methodology for neural network structure design is given below, and it should be applied separately to each CNN and the RNN.

1. Start with the basic version of the network you are creating. For a CNN refer to Figure 3.5(a) for a guide to the layers which should be included. For an RNN the basic structure is an LSTM with one hidden layer.

2. Test the network's accuracy.
3. Add a layer / layers (for the CNN one pooling layer and one convolution layer should be added; for the RNN one hidden layer).
4. Test the network's accuracy again.
5. If the accuracy has increased: repeat steps 1-5.
6. If not: remove the last layer(s) added and stop.

The number of CNNs should be equal to the number of features to be extracted for the application domain, and the structure of CNNs trained for different features may be different. One of the common CNN architectures learned for feature extraction in the example domains had three convolutional layers and three pooling layers, as shown in Figure 3.5(b). For more complex features, the architecture also allows the option of fine-tuning previously trained CNN models (transfer learning) instead of starting from scratch. This was not necessary for any of the example domains.

3.3 Classification with logic or decision tree induction

Once the feature vector had been extracted from an image, it needed to be assigned a class label. In the proposed architecture the class label was assigned using non-monotonic logical reasoning or a decision tree-based classifier. In the examples presented in this thesis the logic classifier was implemented using Answer Set Prolog (ASP) [36], while the decision tree classifier was implemented using Python [57]. Both classification methods are described below.

3.3.1 Logic-based inference with commonsense knowledge

Answer Set Prolog (ASP), a non-monotonic logical reasoning paradigm, was used to represent incomplete commonsense knowledge about the domain. ASP is based on stable model semantics, and supports *default negation* and *epistemic disjunction*, e.g., unlike "$\neg a$" that states *a is believed to be false*, "$not\ a$" only implies *a is not believed to be true*, and unlike "$p \lor \neg p$" in propositional logic, "$p\ or\ \neg p$" is not tautological. Each literal can be true, false or unknown, i.e., the agent does not have to believe anything that it is not forced to believe. ASP can represent recursive definitions, defaults, causal relations, special forms of self-reference, and language constructs that occur frequently in non-mathematical domains, and are difficult to express in classical logic formalisms [34]. Also, unlike classical first-order logic, ASP supports non-monotonic logical reasoning, i.e., it can revise previously held conclusions (equivalently reduce the set of inferred consequences) based on new evidence, aiding in recovery from errors due to incomplete knowledge. ASP and other paradigms that support reasoning with domain knowledge are often criticized for requiring considerable (if not complete) prior knowledge and manual supervision, and for being unwieldy in large, complex domains. However, modern ASP solvers support efficient reasoning in large knowledge bases with incomplete knowledge, and are used by an international research community for cognitive robotics [64, 65] and other applications [66]. For instance, existing work has demonstrated that ASP-based commonsense reasoning can be combined with probabilistic reasoning for reliable and efficient planning and diagnostics [67], and interactive learning for incrementally learning or revising such commonsense domain knowledge based on input from sensors and humans [68].

An ASP *program* (Π) has a *sorted signature* Σ and axioms. Σ includes *sorts*, *statics*, i.e., domain attributes that do not change over time, and *fluents*, i.e., domain attributes whose values can be changed. The specific sorts and attributes encoded depend on the domain under consideration. For instance, in the TS domain, Σ includes sorts such as $shape$, $main\_color$, $border\_color$, $symbol$, and $secondary_symbol$c. In the SS and TS domains, some statics and fluents of interest include:

$$\begin{aligned}
&num\_blocks(blocks),\ structure(feature), \\
&symbol(symbol),\ main\_color(main\_color) \\
&secondary\_symbol(secondary\_symbol)
\end{aligned} \tag{3.1}$$

These relations are described in terms of their arguments' sorts.

The axioms of Π encode some rules that are used for inference with commonsense knowledge. Some examples from the SS and TS domains are:

$$\begin{aligned}
structure(unstable) &\leftarrow structure(block\_displaced). \\
structure(stable) &\leftarrow num\_blocks(2), \\
&\quad \neg structure(lean). \\
class(no\_parking) &\leftarrow main\_color(blue), \\
&\quad symbol(blank).
\end{aligned} \tag{3.2}$$

where the first axiom says that any structure that has a block that is significantly displaced is unstable, and the second axiom says that any pair of blocks that does not have a significant lean is stable. The third axiom says that a traffic sign that is blue and blank is a no parking sign.

Additionally, in dynamic domains, Π includes fluents which can have a different value in each timestep, a predicate $holds(fluent, step)$ is used to indicate that a particular fluent holds true at a particular timestep, actions with their preconditions and effects, and a history *H* that allows the ASP program to reason about a robot's previous interactions with the environment. The domain knowledge in practical domains often includes default statements that are true in all but certain exceptional circumstances. For example, we may know in the SS domain that structures with two blocks of the same size are usually stable. ASP supports the elegant encoding of, and reasoning with, such defaults and exceptions (if any). Recent work expanded the notion of history to represent defaults describing the values of fluents in the initial state, and demonstrated how this knowledge could be used for planning and diagnostics in a robotics domains [67]. This capability is supported in the proposed architecture.

Key tasks of an agent equipped with a system description D and history H include reasoning with this knowledge for inference, planning and diagnostics. In the proposed architecture, these tasks were accomplished by translating the domain representation to a program $\Pi(D, H)$ in CR-Prolog, a variant of ASP that incorporates consistency restoring (CR) rules [69]. The program Π included the signature and axioms of D, inertia axioms, reality checks, closed world assumptions for defined fluents and actions, and observations, actions, and defaults from H. In addition, the features extracted from an input image were encoded as the initial state of the domain in Π. Each answer set of $\Pi(D, H)$ represented the set of beliefs of an agent associated with this program. Algorithms for computing entailment, and for planning and diagnostics, reduced these tasks to computing answer sets of CR-Prolog programs. Answer sets of CR-Prolog programs were computed using a system called Sorted ASP with Consistency Restoring Rules (SPARC) [37]. The CR-Prolog programs for the example domains are available in an open-source software repository [84]. For the classification task in the example domains, the relevant literal in the answer set provided the class label. Note that the accuracy of the inferences drawn from the encoded knowledge depends on the accuracy and extent of the knowledge encoded, but encoding comprehensive domain knowledge is difficult. The decision of what (and how much) knowledge to encode is made by the designer (i.e., domain expert).

In some cases (if the knowledge base was incomplete or one of the features extracted by the CNNs was inaccurate), the knowledge base might not have an axiom for classifying the set of features it received as input. In these cases the decision tree would be used for classification instead.

3.3.2 Methodology: Create classification knowledge base

Algorithm 1 shows a methodology for creating accurate classification axioms that use the features selected in Section 3.1.1. The user must first choose a target accuracy, which for the rest of this section will be referred to as X. They must then create a skeleton ASP file, including sorts, predicates, and generic axioms such as the closed world assumption [36]. Once this has been done they will be ready to add classification axioms to the knowledge base.

Following the methodology in Algorithm 1, the user must iterate through their list of relevant features and test axioms of increasing complexity until they find one that has their desired accuracy. For each feature, and each value that that feature can have, the training dataset should be used to determine whether that feature value is always (or mostly always) associated with a particular class. If not, more specificity is required and therefore another feature should be included in the axiom. If it is impossible to find axioms that have the desired accuracy then it is likely that more detailed features need to be selected in

```
Data: Features and training samples
Result: Classification axioms
for each feature in features do
    for each possible value of feature do
        for each possible image class do
            candidate axiom format: image(class) if feature(value);
            test classification accuracy of this axiom with the training dataset (this can be done
            with a script);
            if accuracy is greater than or equal to X then
                add axiom to knowledge base;
            end
        end
        if accuracy of X not achieved for any class then
            for each feature_two in features do
                for each possible value_two of feature_two do
                    for each possible image class do
                        candidate axiom format: image(class) if feature(value),
                        feature_two(value_two);
                        test classification accuracy of this axiom;
                        if accuracy is greater than or equal to X then
                            add axiom to knowledge base;
                        end
                    end
                    if accuracy of X not achieved for any class then
                        Continue including new features (i.e. adding nested for loops to this
                        methodology) until you have attempted axioms that combine every feature
                        that can be extracted by the CNNs;
                    end
                end
            end
        end
    end
end
```

Algorithm 1: Methodology for manually discovering all relevant classification axioms.

the feature selection step, which is outlined in Section 3.1.1.

Once the algorithm has been completed the user should go through the classification axioms added to the knowledge base and remove any that elaborate on another, simpler one.

3.3.3 Decision tree classifier

If ASP-based inference could not classify the feature vector from an image, it was classified by a decision tree learned from labeled training samples. In a decision tree classifier, each node is associated with a question about the value of a particular feature, with the child nodes representing the different answers to the question, i.e., the possible values of the feature. Each node is also associated with samples that satisfy the corresponding values of the features along the path from the root node to this node. In the proposed architecture a standard implementation of a decision tree classifier was used [71]. In this implementation, the Gini measure was used to compute information gain (equivalently, the reduction in entropy) that would be achieved by splitting an existing node based on each particular feature that has not already been used to create a split in the tree. Among the features that provided a significant information gain, the feature that provided the maximum information gain was selected to split the node. If none of the features would result in any information gain this node was set to be a leaf node associated with a particular class label, e.g., one of the types of traffic sign in the TS domain. Note that the decision tree's search space was quite specific since it only considered samples that could not be classified by ASP-based reasoning. The decision tree did not need to generalize as much as it would have if it had to process every training (or test) sample in the dataset. Also, although overfitting is much less likely, it is still possible; pruning can be used to minimize the effects of overfitting, although this was not found necessary in any of the example domains. Figure 3.6 shows part of a learned decision tree classifier; specific nodes used to classify a particular example are highlighted to indicate that 94% of the observed examples of structures that have fewer than three blocks, do not have a significant lean, and do not have a narrow base, correspond to stable structures. These 'active' nodes along any path in the decision tree that is used to classify an example can be used to explain the classification outcome in terms of the values of particular features that were used to arrive at the class label assigned to a specific image under consideration. The classifier used in this stage of the architecture determined which question answering system would be used in the next stage. If the knowledge base was used for classification, it was also used to answer a question about the classified image. If the decision tree was used, its output was passed to an RNN for question answering.

3.3.4 Methodology: Decision tree design

There are decision tree libraries available, but for the SS and TS domains it was sufficient to create simple decision tree algorithms in Python. As mentioned in the above subsection, the decision trees designed for the SS and TS domains used the Gini measure to compute information gain, selected the feature that

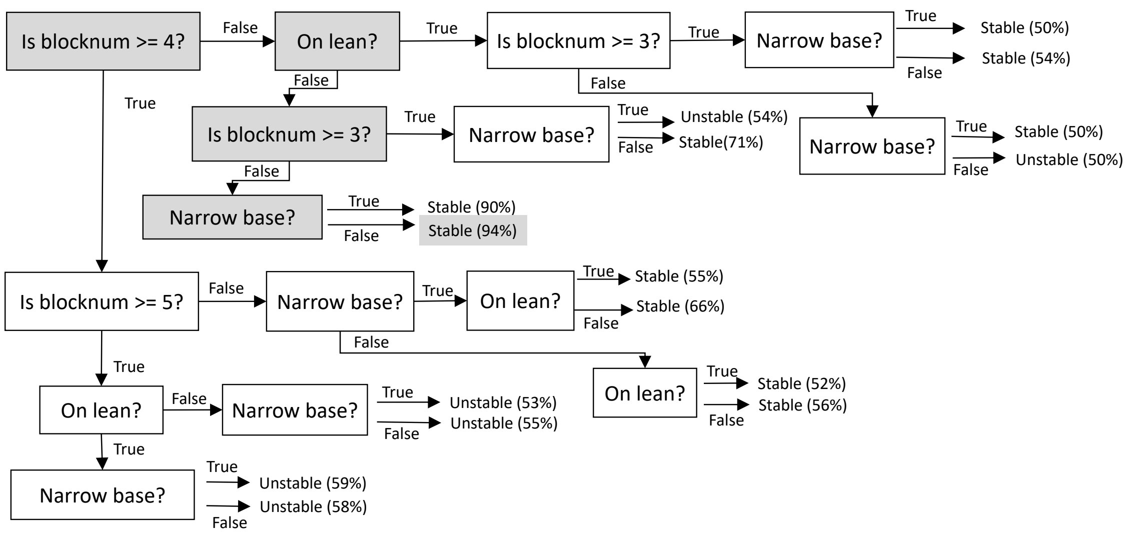

Figure 3.6: Illustrative example of part of a decision tree constructed for classification. Nodes used to classify a particular example are highlighted.

would provide the maximum information gain to split a node, and if no features provided information gain the node was designated a leaf node. Pruning was not found to be necessary. This design is not a strict requirement of the proposed architecture, and if a different decision tree algorithm is a better match for the target domain then the use of more sophisticated algorithms, libraries, alternative information gain measures or pruning are all permissible.

3.4 Question answering with logic or neural networks

As mentioned earlier, two systems were used for question answering. The non-monotonic logic component, implemented as an ASP knowledge base, answered questions for inputs that the knowledge base was able to classify. The neural network component, an RNN, answered questions for inputs that were classified by the decision tree. The inputs to the question answering systems were the transcribed question, the vector of features extracted from the image under consideration, and the classification output.

The human designer must provide pre-determined templates for questions and their answers. In the SS and TS domains a list of the questions used for training was provided, as well as a controlled vocabulary containing all the words deemed relevant to the domain (as well as basic words such as ‘the’ or ‘why’ that were thought necessary for phrasing questions). For every input the classifier used would determine which question answering system would be used, and the user would them be prompted to select or write a question. For the knowledge base the user would have to select from a provided list of questions.

The chosen question would then be mapped to a number and the information provided to the knowledge base as a fact (an axiom with no conditions, which is always true) with the format *question(X)*. For the RNN the user could type out any question so long as it only contained words from the preset vocabulary. The vocabulary defined mappings of each word to a number, and this would be used to convert the question string into a word vector. Answers were chosen from a list of possible answers (50 for the SS domain, 157 for the TS domain). Examples of questions in the SS domain include: ‘is this structure stable/unstable?’, ‘what is making this structure stable/unstable?’, and ‘what would need to be changed to make this structure stable/unstable?’. Examples of questions in the TS domain include: ‘what sign is this?’, ‘what is the sign’s message?’, and ‘how should the driver respond to this sign?’.

3.4.1 Logic question answering

The first method for answering explanatory questions was based on the understanding that if the feature vector extracted from the image was classified using ASP-based reasoning, it was also possible to reason with the existing knowledge to answer explanatory questions about the scene.

To support question answering in the knowledge base, the signature Σ and system description $\mathcal{D}$ had to be augmented. For instance, sorts such as $question$ and $answer$, and suitable relations to represent questions and answers, were included. A constraint axiom was used to ensure that a question did not result in multiple contradictory answers. Once the program Π had been augmented, a solver was used to compute the answer set of the program. For any given question, the answer set was then parsed to extract literals that were used to construct the answer (based on pre-determined templates).

3.4.2 Methodology: Create question answering knowledge base

To avoid the need to create another skeleton ASP file, the question answering axioms can be added to the same file as the classification axioms.

This step is simple. For every possible combination of classification and question (in some domains the features will need to be considered also), add an axiom that selects the answer.

An example of a question answering axiom that relies on the feature-classification-question combination in the SS domain is:

$$\begin{aligned} answer(20) \leftarrow\ & structure(stable), num\_blocks(2), structure(narrow\_base), \\ & structure(lean), structure(block\_displaced), question(2). \end{aligned} \tag{3.3}$$

Question two means ‘what is making this structure stable?’ and answer twenty means ‘it has a small number of blocks’.

An example of a question answering axiom that relies on the classification-question combination in the TS domain is:

$$answer(101) \leftarrow class(50), question(1). \tag{3.4}$$

Class fifty is a parking sign, question one means ‘what is the sign’s message?’ and answer 101 means ‘parking available here’.

This mapping of strings to numbers in order to encode question information in the knowledge base is only one way of using templates for question answering. The RNN maps individual words to numbers and compiles them into a word vector representing the question, rather than mapping the entire question to a single number. The user can use any template style that suits their application or implementation.

3.4.3 Neural network question answering

The second method for providing answers to explanatory questions was invoked if the decision tree was used to classify the vector of image features. In this case, the inability to classify the feature vector through ASP-based reasoning was taken to imply that the encoded domain knowledge was insufficient to answer explanatory questions about the scene. An LSTM was trained and used to answer questions. This RNN took as input the feature vector, classification result, and a vector representing the transcribed and parsed query. The output (provided during training) was in the form of answers in the predetermined templates. As with the CNNs for feature extraction, an incremental strategy was used to design the network structure with the goal of making it as simple as possible while achieving high performance. The strategy begins with a single hidden layer and increases complexity until the accuracy stops increasing. Further details are given in Section 3.2.1. Adding a stack of LSTMs can be effective if adding individual layers does not improve network accuracy significantly and the performance is still below the desired target. Figure 3.7 shows an example RNN with two hidden layers. In the SS domain this RNN design method resulted in a network with 12 hidden layers, while in the TS domain the RNN had 24 hidden layers.

3.5 Axiom learning

The components of the architecture described above support reasoning with commonsense knowledge, decision trees, and deep networks, to answer explanatory questions about the scene and an underlying classification problem. In some complex domains, the available knowledge is incomplete, and this can result in the knowledge base not having enough axioms to process a large percentage of the inputs. In order to minimise the amount of training data required, the variety of inputs that have to be classified by the decision tree should be restricted as much as possible. Additionally, in planning domains, an incomplete planning knowledge base can result in suboptimal plans being developed that do not achieve the goal or

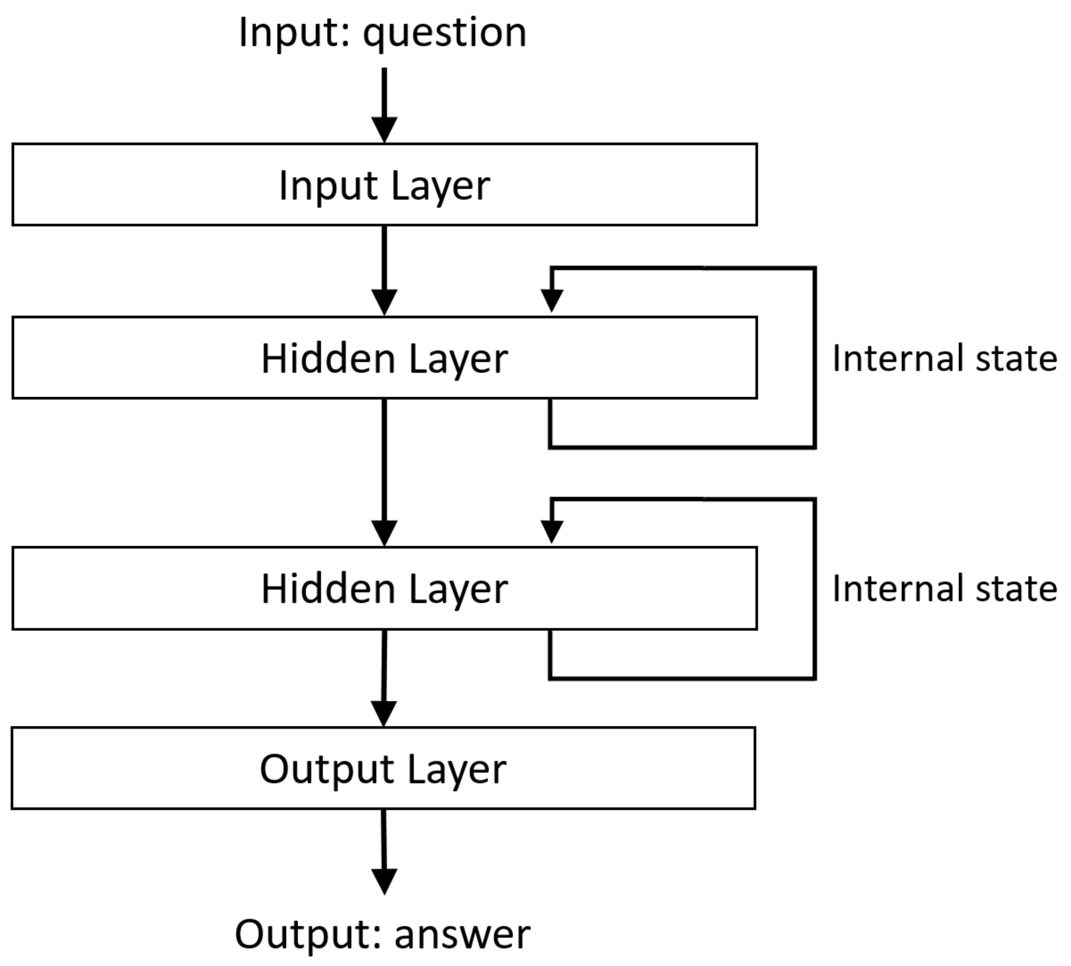

Figure 3.7: Block diagram of an RNN with two hidden layers.

take an unnecessarily long time to do so. Figure 3.8 shows an updated block diagram of the proposed VQA architecture, with axiom learning included as part of the classification section. This indicates the inputs and outputs the axiom learning component requires in order to update the classification knowledge base with new axioms. Note that axiom learning is not restricted to just the classification knowledge base, but can also be used to update question answering or planning knowledge bases.

In domains where it is not feasible to create entire knowledge bases by hand a human expert can create a simple version of the knowledge base, and then axiom learning can be used to automatically add new rules to the knowledge base and make it more complete. This thesis presents a new axiom learning methodology for learning state constraints, a format that is particularly useful for VQA domains in which all the classification and question answering rules are state constraints, and statics, which are useful for encoding knowledge about the physical world in a planning domain. The inclusion of axiom learning with the proposed architecture was inspired by Sridharan and Meadows [38], who presented an axiom learning methodology for executability conditions.

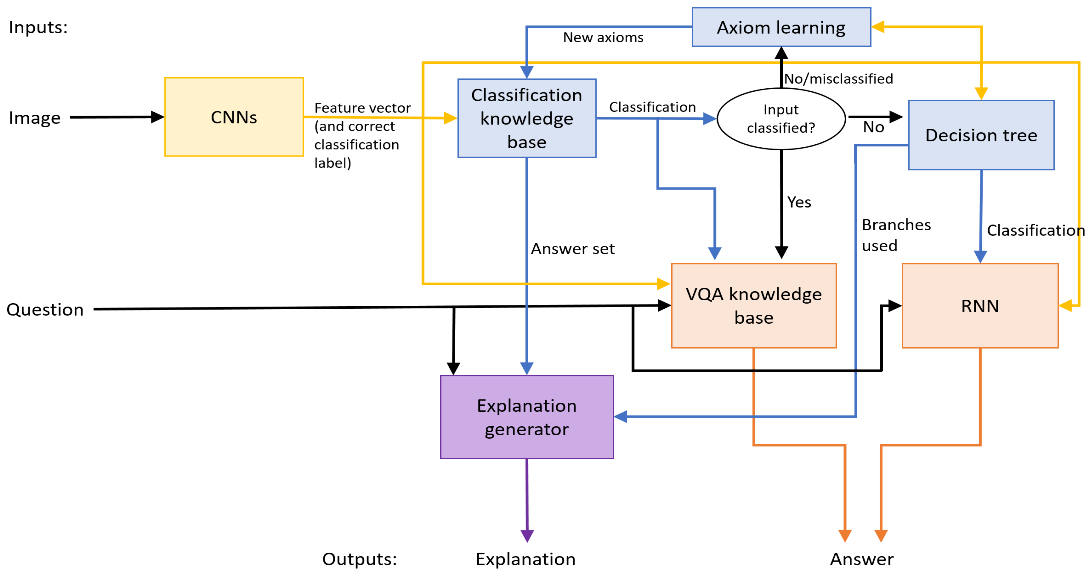

Figure 3.8: Block diagram of the components of the proposed architecture with axiom learning included.

To perform axiom learning the training dataset was first run through the original knowledge base, and any inputs that it failed to produce the correct output (classification or answer) for were added to a list. Decision tree induction was used to generate candidate axioms from these lists, and after a rigorous testing procedure any candidate axioms deemed worthy were added to the knowledge base. In planning domains there is usually not a premade training dataset of the results of actions available, so the agent should be given numerous goals to achieve and any unexpected action outcomes collected into a list. Descriptions of two such domains are included in Section 3.6.

3.5.1 Methodology: Implement axiom learning

Axiom learning can be used to make a classification, question answering or planning knowledge base more complete. A methodology for automatically learning new state constraints or statics is given below.

- All the training samples are run through the knowledge base, and any that it does not classify correctly are added to a list.
- A decision tree is trained on the list of misclassified/unclassified samples.
- Paths that end in certain predictions (i.e. all the samples that fit that path had the same label) and are supported by at least ten percent of the training samples are translated into axioms.
- Candidate axioms are generalised. For example, the axioms

$$structure(stable) \leftarrow \neg structure(lean), \neg structure(narrow\_base), num\_blocks(4). \quad (3.5)$$

and

$$structure(stable) \leftarrow \neg structure(lean), \neg structure(narrow\_base), num\_blocks(3). \quad (3.6)$$

have almost all the same conditions, and have the same classification, suggesting that the condition they do not have in common (the number of blocks) is irrelevant. The generalised axiom would be:

$$structure(stable) \leftarrow \neg structure(lean), \neg structure(narrow\_base). \quad (3.7)$$

- The candidate axioms are tested one at a time. The candidate being tested is added to the knowledge base. A number of samples equal to ten percent of the dataset are randomly selected from the training dataset and classified with the updated knowledge base. Only samples that are relevant to the new axioms are used; if a sample's features do not match the conditions of any of the new axioms it is rejected and a different sample selected. If there are not enough relevant samples to test it properly the candidate axiom is rejected. Any axiom that results in a misclassification is rejected.
- The candidate axioms are compared to one another and if any axiom has the same conditions as another, plus extra (i.e. if an axiom elaborates on another axiom) it is removed.
- The remaining axioms are added to the knowledge base and inputs are now run through this new, updated knowledge base.

Although this methodology applies only to learning state constraints and statics, other axiom learning methodologies can be implemented to learn any kind of axiom; for example, a methodology that can be used to learn executability conditions is presented in [38].

3.6 Planning

The proposed VQA architecture is modular and can be combined with planning components. Figure 3.9 shows a block diagram of a standard ASP planning architecture. The controller sends an updated history to the knowledge base whenever a new plan needs to be developed, receives an action plan in return, instructs the executer to execute each action one by one, and instructs the computer vision component to make observations about relevant fluents. The planning and VQA architectures can be combined by replacing the computer vision component with the VQA architecture, and using answers to certain questions (e.g. 'what is the robot's location?') as observations.

In this thesis the domains used to show planning capabilities are the tower building (TB) and robotic assistant (RA) domains, introduced in Section 3.1. To support planning and diagnostics, the planning architectures each include a planning knowledge base with a system description D augmented with fluents and actions, as discussed in Section 3.3. A history H must also be provided. The history H of a

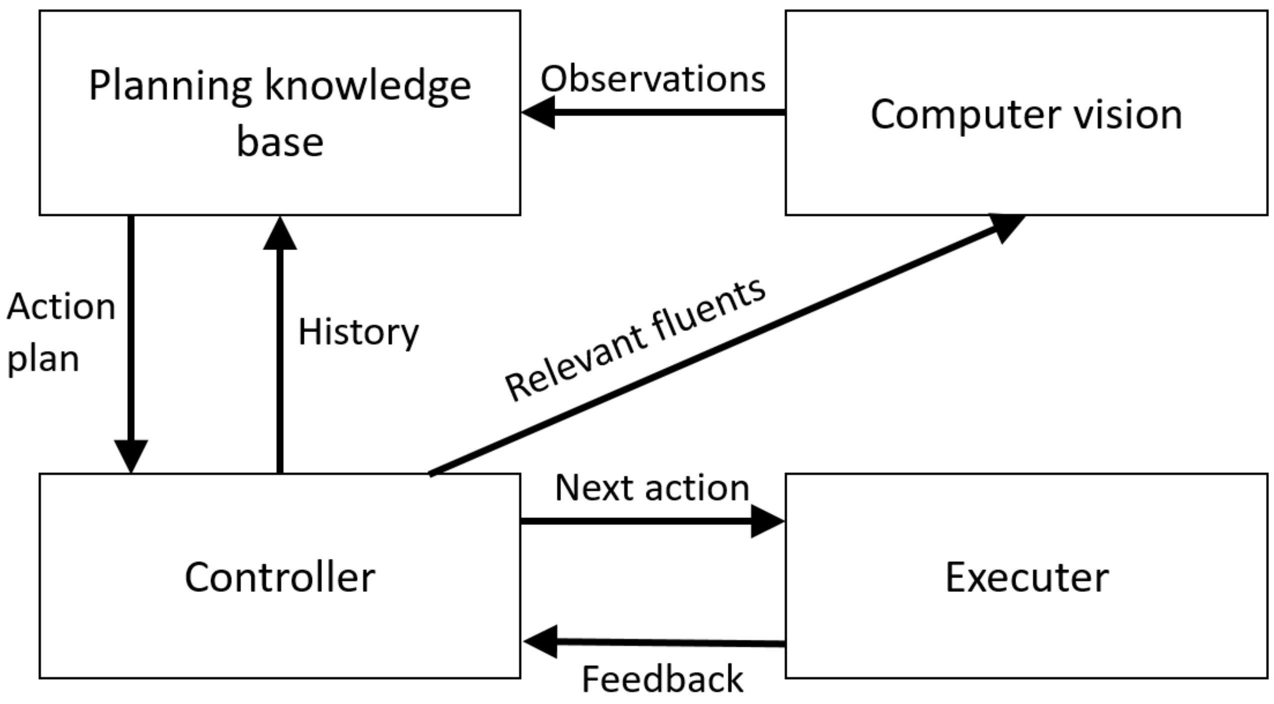

Figure 3.9: A block diagram of a logic-based planning architecture.

dynamic domain is a record of the fluents observed to be true at particular time steps, encoded in the form *obs(fluent, step)*, and the execution of actions at particular time steps, encoded in the form *hpd(action, step)*. The TB and RA domains will be described as examples of dynamic domains. Planning knowledge bases from both domains are presented in a Github repository [84].

In the TB domain the signature Σ of the system description D had basic sorts such as *box, box_place, tower_level, location* and *step*. These could be arranged hierarchically, e.g. the sorts *place* and *tower_level* were subsorts of *location*. Σ also included ground instances of these sorts, e.g. *red_box*, *green_box* and *yellow_box* were instances of the sort *box*. Domain attributes and actions were described in terms of the sorts of their arguments. In the TB domain there were no statics, and fluents included *loc(box, location)*, to describe the location of each box, *in_hand(box)*, which denoted whether a particular box was in the agent's hand, *unstable_placement(box)*, which denoted whether placing a particular box would make the tower unstable, and *empty(tower_level)*, which denoted whether a particular level of the tower had a box in it. The actions in this domain were *pickup(box)* and *place(box, tower_level)*. The signature Σ also included the relation *holds(fluent, step)* to imply that a particular fluent was believed to be true at a particular time step.

3.6.1 Methodology: Create planning knowledge base

A planning knowledge base for a dynamic domain needs to include actions, fluents, and a goal. An agent is the entity (a robot or a simulation) who's task is to achieve the goal. The actions should encompass everything the agent might need to do to accomplish the goal. The fluents should encompass everything in the environment that can change and that is relevant to the goal or actions. Before creating the ASP file, the user should first come up with a list of possible goals, as well as all of the actions and fluents that are relevant to the domain.

For example, the TB domain includes the following:

- Actions: pick up boxes, and place boxes.
- Fluents: the location of each box, whether each box is in the robot's hand, the stability of any potential box placement, and the status of each tower level (empty or filled).
- Goals: in this domain the goal is always to increase the tower height by one.

Once these lists have been created they should be converted into axioms in an action language (AL) file. This is a form of pseudocode which helps to define the domain before writing axioms in the less readable ASP file. The format of an AL file is described in detail in [36]. It should include:

- State constraints, which are used to define what is and is not allowed in the domain. For example, when the condition of the robot being in a certain location is met, all other robot locations are set to false. This constrains the robot's location so that it cannot be in two places at once.

  not loc(Box, Location2) **if** *loc(Box, Location1), Location1 != Location2.*

- Causal laws, which are used to define the effect that performing an action will have on the agent and environment. For example, picking an object up causes it to be in the robot's hand.

  pickup(Box) **causes** *in_hand(Box).*

- Executability conditions, which are used to define circumstances in which it is impossible to perform a certain action. For example, it is impossible to place a box on top of an empty tower level.

  impossible *place(Box, Location)* **if** *empty(Location-1).*

  In the TB domain tower locations are represented by numbers, so 'Location-1' refers to the tower level below 'Location'.

These examples include full words to improve readability, but to make axioms more concise it is generally advisable to use only a single letter for each variable name. The remaining examples in this section follow the single letter variable name format.

As with the classification knowledge base, a skeleton ASP file should be created before the domain-relevant axioms are added. As well as the axioms included in a static knowledge base, a dynamic ASP file needs an inertial axiom, a constraint forcing the answer set to include the goal being achieved, and constraints to avoid more than one action being performed in one time step [36].

To translate AL axioms into ASP, time steps need to be included, and the keyword 'holds' used to indicate that a fluent's value at a particular time step is a belief held by the agent at that time, not an absolute truth. The following list gives translation guidelines; for more detailed instructions see [36]. For each example, the first equation is the AL axiom, and the second is the ASP axiom.

- Translating state constraints: fluents such as location ('loc') become part of the 'holds' predicate, along with the timestep at which the fluent is believed to have that particular value (i.e. *fluent(value)* becomes *holds(fluent(value),timestep)*.

$$\textit{not loc}(B, L2) \textbf{ if } loc(B, L1),\ L1! = L2. \tag{3.8}$$

$$\neg holds(loc(B, L2), I) \leftarrow holds(loc(B, L1), I),\ L1! = L2. \tag{3.9}$$

- Translating causal laws: actions must now become part of the 'occurs' predicate, along with the time step at which they were performed (i.e. *action(action_variables)* becomes *occurs(action (action_variables), timestep)*. The effects of an action must become part of the 'holds' predicate, with the time step at which that fluent is believed to take on the value caused by the action being the time step *after* the one in which the action occurred.

$$pickup(B) \textbf{ causes } in\_hand(B). \tag{3.10}$$

$$holds(in\_hand(B), I + 1) \leftarrow occurs(pickup(B), I). \tag{3.11}$$

- Translating executability conditions: once again, actions must become part of the 'occurs' predicate, and fluents must become part of the 'holds' predicate. The format of executability conditions in ASP is as follows: an action does not occur in a particular time step if certain conditions are met in that time step.

$$\textbf{impossible } place(B, L) \textbf{ if } empty(L - 1). \tag{3.12}$$

$$\neg occurs(place(B, L), I) \leftarrow holds(empty(L - 1), I). \tag{3.13}$$

3.6.2 Combining two architectures

Another dynamic domain, the robot assistant (RA) domain, was used to show that the VQA and planning architectures could be combined. The planning knowledge base had the same layout as the TB domain's planning knowledge base, and the elements included are listed below.

- Sorts: *location, person, robot, status,* and *agent*. *person* and *robot* are subsorts of *agent*.
- Ground instances: Example instances of the *location* sort are *bobs_office, sarahs_office, kitchen, library.* Example instances of the *person* sort are *bob, sarah, sally.* The Turtlebot is the only instance of the *robot* sort and it is referred to as *rob1*. The instances of the *status* sort are *delivered* and *undelivered*.
- Statics: *workplace(person,location)*, which indicated which room a person was most likely to be found in, and *next_to(location,location)*, which encoded the layout of the office world by describing which rooms were next to one another.
- Fluents: *loc(agent,location)* means that the *agent* (a *person* or a *robot*) is currently at *location*. *message_status(message_id, person, status)* means that the message for *person* is currently either *delivered* or *undelivered*, depending on which *status* is used.
- Actions: *move(robot,location)* and *deliver(robot,message_id,person)*.

The above list does not describe the entire domain, as the domain description was split between the classification knowledge base and the planning knowledge base. The planning knowledge base dealt with navigation and message delivery, while the classification knowledge base encoded information about objects and the states of rooms. The classification domain was static, i.e., objects were not reasoned about dynamically and were assumed to stay in place during the time that the Turtlebot was performing its task. This was considered a reasonable assumption as the robot only took a few minutes to deliver a message and return to the sender, and its beliefs about object locations would be reset when it was given a new goal, so the movement of objects between delivery tasks would not confuse the robot. Objects were represented by sorts that divided them into separate categories, such as *object, ground_object, tabletop_object, kitchen_object, kitchen_tabletop_object*, etc. These sorts were hierarchical. For example: *kitchen_tabletop_object* and *kitchen_ground_object* were subsorts of *kitchen_object*; *kitchen_tabletop_object* and *library_tabletop_object* were subsorts of *tabletop_object*; and *tabletop_object* and *ground_object* were subsorts of *object*. The objects included in the domain are listed below.

- Kitchen tabletop objects: plate_one, plate_two, cup_one, cup_two.
- Kitchen ground objects: table, chair_one, chair_two, chair_three, chair_four.

- Library tabletop objects: book_one, book_two, book_three.
- Library ground objects: bookshelf_one, bookshelf_two, bookshelf_three, coffee_table.
- Sarah's office tabletop objects: sarahs_computer.
- Sarah's office ground objects: sarahs_desk, sarahs_chair.
- The offices of Sally, John and Bob had the same objects as Sarah's office.

The combination of two architectures allowed a wide range of functionality to be achieved - the simulated Turtlebot could move around an office world, deliver messages, make observations, and answer questions about observed scenes, supplying the humans with any information their coworkers wanted them to receive or that they wanted to know about the state of the office rooms. The order in which the robot accomplished its tasks was as follows:

- Receive instructions from sender (intended message and recipient must be provided) - this is encoded as the goal in the planing knowledge base
- Search office world until message recipient has been found
- Deliver message
- Observe surroundings
- Navigate back to sender
- Answer questions about the room the recipient was in

The robot had to reason about its environment, which can be seen in Figure 3.4, in three separate ways: it had to navigate through the rooms and maintain an estimate of its position; it had to recognise individuals and deliver messages to the correct recipients; and it had to observe the objects in its vicinity and reason about the state of its environment (e.g. is a room tidy or messy). Domain knowledge was encoded to help it achieve all three tasks.

The planning knowledge base included the static *next_to(location,location)* to describe which rooms were next to one another, allowing the robot to come up with efficient plans for navigation. An executability condition, shown below, ensured that the robot did not try to move directly from one room to another unless they were next door.

$$\neg occurs(move(R, L1), I) \ \leftarrow \ holds(loc(R, L2), I), \ \neg next\_to(L1, L2). \qquad (3.14)$$

The planning knowledge base also included axioms that ensured that any plan the robot came up with would involve achieving its goal (which was to deliver a message). When it first set out on this task the robot would not know what room the recipient of the message was in, so it would make an assumption and then go to the room that it thought the recipient was in. If it did not observe the recipient of the message there it would include this fact in its history H and replan. Eventually it would find the correct room. An executability condition, shown below, ensured that the robot would not deliver a message unless it had observed the recipient of the message to be in the same room as it.

$$\neg occurs(deliver(R, M, P), I) \leftarrow holds(loc(R, L), I),\ not\ holds(loc(P, L), I). \tag{3.15}$$

Section 4.4 describes how axiom learning was used to update the Turtlebot's domain knowledge with information on which rooms certain people were most likely to be in, allowing it to find message recipients more efficiently.

Finally, the Turtlebot had to observe objects and reason about its surroundings. A separate CNN was trained for the observation of each object with a training dataset of 500 images taken from the simulated Turtlebot's camera, showing various viewpoints through the office world. Labels were manually provided indicating which objects were visible in each image. During the observation of a room the robot would take four images, rotating 90 degrees in between each image. If an object was observed in any of the four images it was considered to be in the room. In order to include as much relevant information as possible, in addition to the four images taken after the recipient had been found, the robot would take one image each time it entered a new room. If any object was observed (in these images or in the four images taken after delivering the message) this information was provided as an input to the classification knowledge base in the form *loc(object,room)*. Some objects, such as cups and plates in the kitchen, were usually on top of a high table and therefore out of the robot's field of view (which was close to the ground). For these objects defaults were encoded in the classification knowledge base so that if a robot had not observed a tabletop object to be in another room, it was assumed to be in its usual location (e.g. cups and plates were usually in the kitchen, books were usually in the library). Ground objects (such as chairs and desks), on the other hand, had to be observed or the robot would believe them to not be in the room.

As well as reasoning about object locations, the classification knowledge base used the information it had about which objects were present (or assumed to be present) in a room to determine the state of that room. The room was classified into multiple different categories. It was considered to be untidy if any object not belonging to the room was detected, e.g. if plate was spotted in the library. It was considered to have objects missing if any of the objects usually in the room weren't detected (ground objects) or had been observed somewhere else (ground or tabletop objects). It was considered cluttered if the number of objects in the room was above the usual amount (e.g. if the library contained all of the bookshelves, books and the coffee table, as well as a laptop, the library was cluttered). The room was also categorised as a particular type based on the prevalent object category within the room. This information might be important to managers; for example, if a manager often noticed the robot classifying the library as a 'kitchen' because it frequently observed cups and plates there, it might indicate to the manager that there were not enough chairs in the kitchen and employees had resorted to sitting in the library during lunch breaks. All of these classification decisions were made with state constraint axioms. Some examples are given below.

$$tidy(L) \leftarrow \; not \; mess(L). \tag{3.16}$$

$$\neg tidy(L) \leftarrow \; not \; tidy(L). \tag{3.17}$$

$$mess(L) \leftarrow \; loc(O, L), \; \neg designated\_area(O, L). \tag{3.18}$$

$$object\_missing(L) \leftarrow \; \neg loc(O, L), \; designated\_area(O, L). \tag{3.19}$$

$$\neg object\_missing(L) \leftarrow \; not \; object\_missing(L). \tag{3.20}$$

The first three axioms state that a room is tidy if it is not known to be messy, that it is not tidy if it is not known to be tidy, and that it is messy if it contains an object which who's designated area is not that room. The last two axioms state that a room has an object is missing if any object that has that room as its designated area is not there, and that no objects are missing if the room is not known to have any objects missing.

The robot would use these classifications to answer questions about the recipient's location such as 'what type of room was it', 'was the room tidy', 'was the room cluttered', etc. The robot would also report on the name of the room, which was expected to match the type but wouldn't always, for example if Sarah left a lot of books in her office the robot might tell the manager that 'Sarah's office' was classified as a library. The objects detected by the robot were used to provide explanations for the classification decision made. See Section 4.3 for examples.

3.7 Methodology: Training

The following components of the architecture need to be trained: the CNNs, the RNN, the decision tree, and any knowledge bases for which axiom learning is being applied. These components should be trained separately, so that sensible outputs from one fully-trained component can then be used to train the next. This requires less training data than end-to-end training [63]. The training methodology assumes that the target architecture has a planning knowledge base and that axiom learning is being applied to all knowledge bases. If this is not the case simply ignore the methodology sections that are irrelevant to the target architecture. Training should proceed in the following order: train the CNNs with images and feature labels; train the classification knowledge base and decision tree with the features output by the trained CNNs and classification labels; train the question answering knowledge base and RNN with classification outputs and answer labels (the answer labels should bes correct with respect to the ground truth classification, not the output from the classification components). Finally, once the VQA architecture has been fully trained its outputs should be incorporated into the planning architecture. Test the planning architecture for a pre-determined number of iterations and train the planning knowledge base with any unexpected action outcomes.

3.7.1 Training the CNNs

The VQA architecture should have a separate CNN for each feature. They do not need to be trained in any particular order, and can be trained in parallel or sequentially. Each CNN should be trained on all of the images in the training dataset, with the ground truth values for that CNN's feature used as labels. If, at the end of training, any CNN's are still performing badly, the user may need to rethink their choice of features.

3.7.2 Training the classification components

Once the CNNs have been completely trained, the training data should be run through them once again to get outputs that are similar to what the CNNs will predict during testing. The feature combinations output by the CNNs are then used as inputs for the classification components. These feature combinations should first be passed through the classification knowledge base. Following the axiom learning methodology in Section 3.5.1, any inputs that the knowledge base cannot classify correctly should be used to learn new axioms and update the knowledge base. Once this has been done the feature combinations should be passed through the updated knowledge base. Any that still cannot be classified should be used as training data for the decision tree. If the knowledge base is complete (i.e. all relevant axioms were found during manual creation and/or axiom learning), it is possible that it will be able to classify *all* the inputs. Even if this is the case it is still not a good idea to leave the decision tree untrained, as there may be some feature combinations in the testing dataset that were not seen during training and cannot be classified by the knowledge base. If the classification knowledge base was able to classify all inputs, then the entire training set of feature combinations should be used to train the decision tree, although only the predictions generated by the knowledge base will be used in the next step. If testing shows the decision tree performing much better with training data than with testing data, then pruning may be used to prevent overfitting, but this was not necessary for the example domains presented in this thesis.

3.7.3 Training the question answering components

The question answering (QA) training dataset consists of classifications made by the classification knowledge base or decision tree, paired with questions. Every image is paired with every question to get all possible combinations, so the size of the QA training dataset will be equal to the size of the classification training dataset multiplied by the number of questions. The labels used for learning should answer the questions correctly with respect to the ground truth classification, not the classification predictions made by the architecture. Training should proceed in a similar way to the classification components: axiom learning is first performed to update the knowledge base; any questions that still cannot be answered by the knowledge base are used to train the RNN; and if the knowledge base can answer all questions then the whole dataset should be used to train the RNN, although the knowledge base's answers will be the preferred output. Training the RNN for multiple epochs is recommended in order to achieve the best possible performance. A good target to prevent overfitting is 5% accuracy increase after each epoch; if this is no longer being achieved, training should stop.

3.7.4 Axiom learning for the planning knowledge base

The planning architecture needs to be used so that unexpected action outcomes can be observed before axiom learning commences. The number of testing iterations (with one iteration being an attempt to achieve a goal) is defined by the user, as are the goals and initial conditions of each trial. In general, more complex domains in which the agent can perform numerous actions will require more iterations in order to uncover all of the gaps in the agent's domain knowledge. Thirty iterations is reasonable for simple domains with only a few possible actions. During testing, the agent should observe the outcomes of every action it performs. If an unexpected outcome is observed the transition (a state, action, state tuple) should be saved as training data. Axiom learning can then proceed with this list of transitions. The axiom learning methodology presented in Section 3.5.1 can only learn state constraints and statics. In the TB domain state constraint axioms were added to the planning knowledge base to indicate which conditions made particular box placements unstable. The example given below is a learned axiom that states that the red box (which is medium sized) would be an unstable placement on top of any tower which has the green box (which is small) as its base (tower level one is the base of the tower).

$$holds(unstable\_placement(red\_box), I) \ \leftarrow \ holds(loc(green\_box, 1), I). \tag{3.21}$$

In order to make use of this information during planning, the causal law for box placement was designed such that the box's location only becomes the next tower level if the placement is not unstable. This axiom prevents the robot from designing action plans that include unstable block combinations.

$$\begin{aligned} holds(loc(Box, Location), I+1) \ \leftarrow \ & \neg holds(unstable\_placement(Box), I+1), \\ & occurs(place(Box, Location), I). \end{aligned} \tag{3.22}$$

If learning executability conditions would be more useful for the target domain then the methodology presented in [38] should be used.

Once every component has been trained the architecture will be fully functional and can be used for the target domain. It is hypothesised that the modular designs of these architectures mean that they can be successfully combined, and that the use of deep learning for processing numerical inputs such as images (arrays of pixel numbers), and knowledge bases and decision trees to provide interpretable outputs, will make it easy to tune parameters and achieve high performance even with limited training data. Experimental results supporting these hypotheses are presented in the next chapter.

4. Experimental setup and results

This section describes the results of experimental evaluation of the proposed architecture. The following hypotheses are considered:

1. **H1**: the proposed architecture provides intuitive explanations for classification decisions;

2. **H2**: the proposed architecture outperforms architectures based on just deep networks when the size of the training dataset is small;

3. **H3**: the proposed architecture can be successfully combined with a planning architecture.

4. **H4**: axiom learning can be used to automatically add relevant axioms to incomplete knowledge bases, and this improves performance;

These hypotheses were evaluated in the context of the SS, TS, TB and RA domains introduced in Section 3.1. Execution traces are provided in support of hypothesis $H1$, and quantitative results comparing the proposed VQA architecture with a deep learning baseline are provided in support of $H2$. Demonstration of the functionality of the RA domain, which combines the VQA and planning architectures, is shown in support of $H3$. Comparison of both VQA and planning performance with incomplete knowledge bases before and after axiom learning has been applied are provided in support of $H4$.

For the VQA domains (SS and TS) accuracy (the percentage of questions that were answered correctly) was used as the measure of performance. The amount of data used for training varied from 100 images to 4000, and results are reported separately for each training dataset size. In each domain 500 unseen images were used for testing and each image was combined with each testing question. This process was repeated thirty times and the average of these trials reported in Section 4.2.

The measures for the RA domain's planning performance were how long the Turtlebot took to deliver a message and return to the sender, how much computation time was needed, how many times replanning needed to occur, and how many actions were completed before the task was complete. The average VQA performance was also calculated, and execution traces showing questions and answers were recorded.

In the planning domain (TB) the agent's goal was to build towers as high as possible, and tower height (measured in the number of blocks) and planning time (measured in seconds) were used as performance measures, with any collapsed tower being considered to have a height of zero.

The following sections describe the experimental results. Section 4.1 describes execution traces that demonstrate the proposed VQA architecture's explanation capabilities. Section 4.2 provides quantitative results comparing the proposed architecture with a deep learning baseline. Section 4.3 presents execution traces to demonstrate VQA and planning abilities in the RA architecture, to show that both functionalities can be achieved simultaneously. Finally, Section 4.4 analyses performance of both VQA and planning architectures before and after axiom learning.

4.1 Execution traces

The proposed architecture generates explanations with a three-paragraph format. The first paragraph states what features were extracted from the image, the second states which classifier was used (knowledge base or decision tree), and the third states what classification was made and the reasons why (the reasons are parsed from either the axiom or the branches used to make the classification). The following execution traces illustrate the reasoning, explanation generation, and question answering capabilities of the proposed architecture.

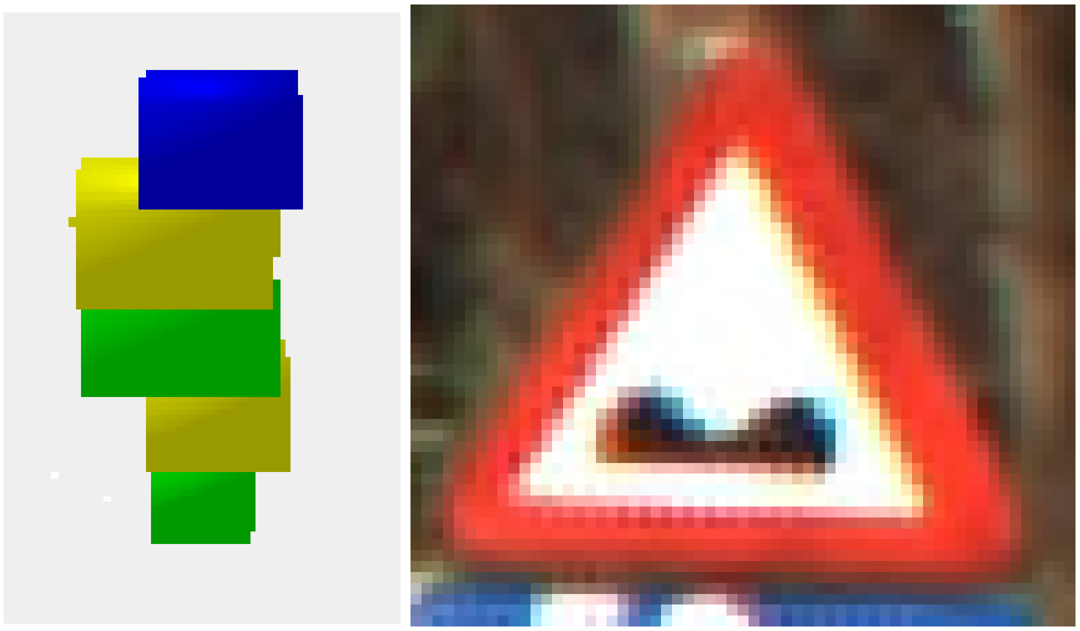

Figure 4.1: Images used as inputs for the execution traces in Section 4.1.

[Question Answering in SS domain]
The input image for this example is that shown on the left in Figure 4.1.

- The question posed: “what is making this structure unstable?”
- The architecture’s answer: “nothing; it is stable.”
- The explanation:

  “The CNNs extracted the following features from this image: the structure has five blocks, the structure has a narrow base, the structure is standing straight, and all of the blocks are in place.

This example was classified by the knowledge base.

This example was classified as stable because the following conditions were met: the structure is standing straight and all of the blocks are in place."

[Question Answering in TS domain]
The input image for this example is that shown on the right in Figure 4.1.

- The question posed: "what is the sign's message?"
- The architecture's answer: "uneven surfaces ahead."
- The explanation:

  "The CNNs extracted the following features from the image: the sign is triangle shaped, the sign is white, the sign has a red border, the sign has no background image, the sign has a bumpy_road symbol, the sign has no secondary symbol, the sign has no cross.

  This example was classified by the decision tree.

  Branches used to classify this example as a bumpy road sign:
  Is main colour white? - true"

Note that although the sign being white is not enough information to determine whether any sign from the entire dataset is a bumpy road sign or not, the advantage of using the decision tree for only the subset of data that cannot be classified by the knowledge base is that in that subset less information is required for classification. In this experiment, bumpy road signs were the only white signs that were not classified by the knowledge base.

These results show that the proposed architecture is able to generate sensible explanations for its classification decisions, supporting hypothesis $H1$.

4.2 Comparison to baseline architecture

Thirty experimental trials were run to evaluate hypotheses $H1$ and $H2$. Specifically, the size of the training dataset was varied and the classification and question answering accuracy measured. A CNN-RNN architecture was used as the baseline for comparing the proposed architecture to one that only used deep networks. This is the most common base network used in state of the art architectures, although the state of the art version usually has additional features. Because these additional features could be applied to the proposed architecture as easily as to the baseline, comparison to just the baseline architecture was considered sufficient to show that the proposed architecture improves on state of the art performance for the type of domain that it was designed for (a specific area of knowledge with limited training data). The

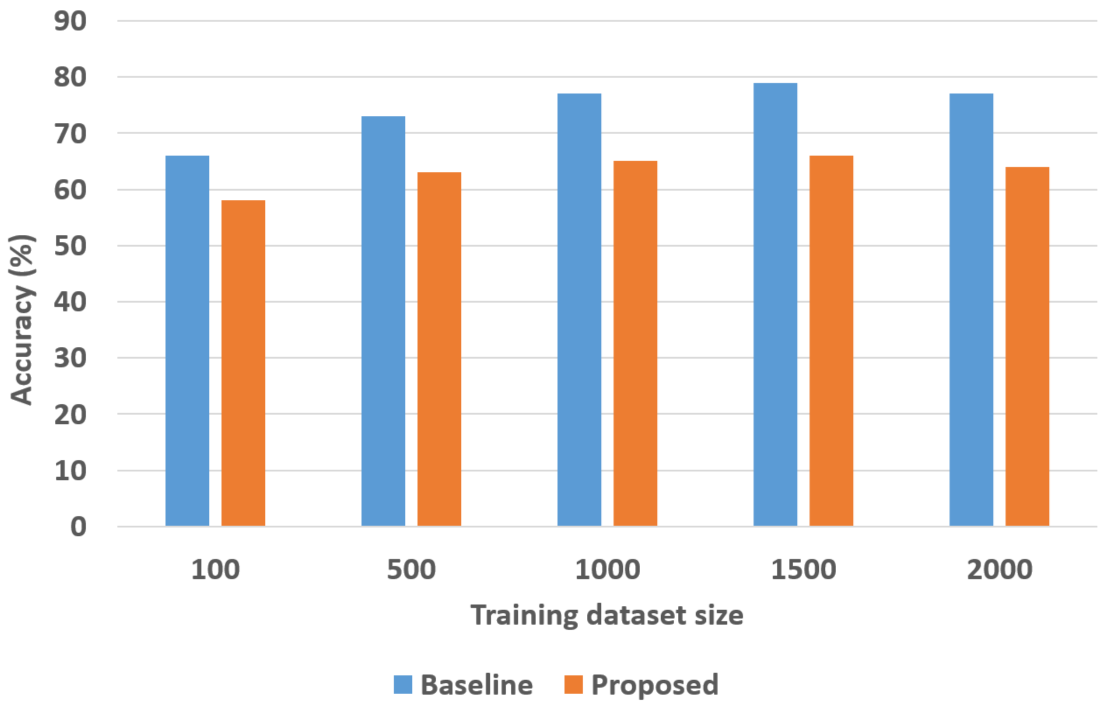

Figure 4.2: Classification accuracy as a function of number of training samples in SS domain.

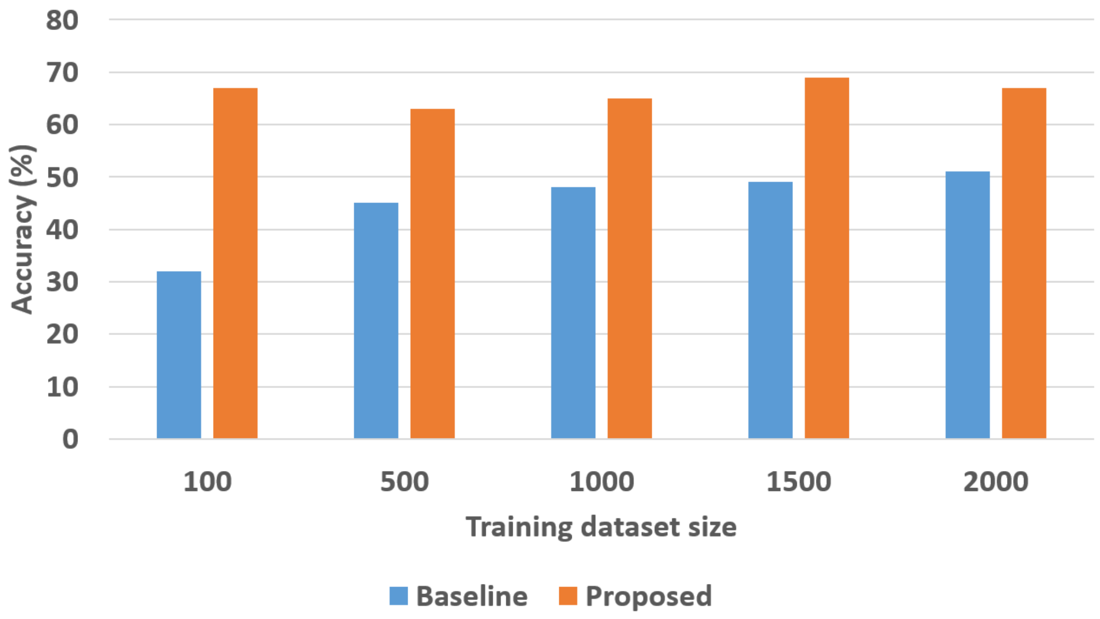

Figure 4.3: VQA accuracy as a function of number of training samples in SS domain.

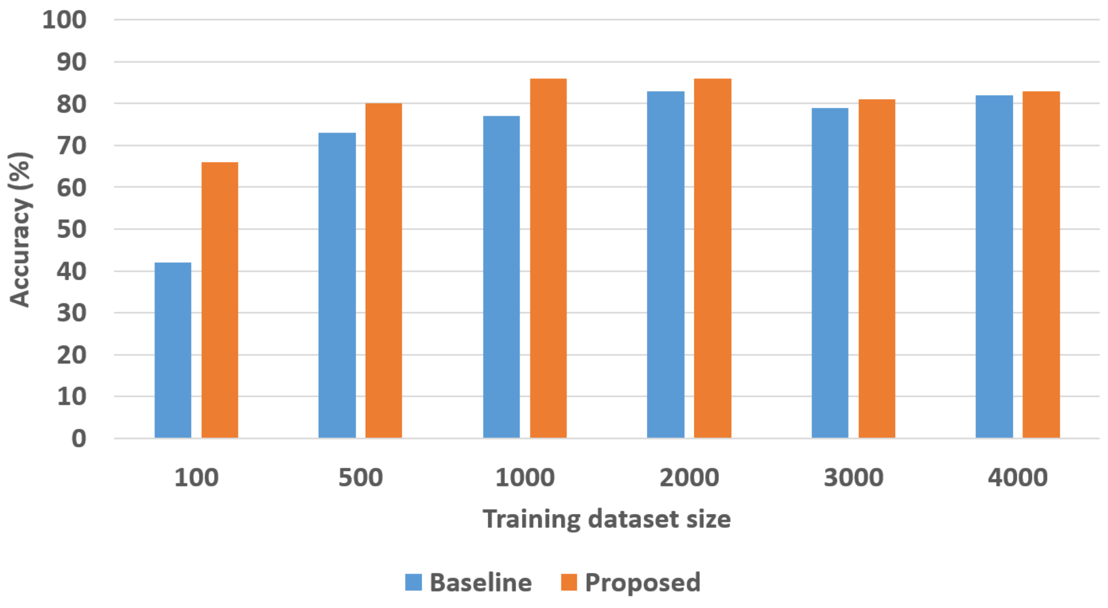

Figure 4.4: Classification accuracy as a function of number of training samples in TS domain.

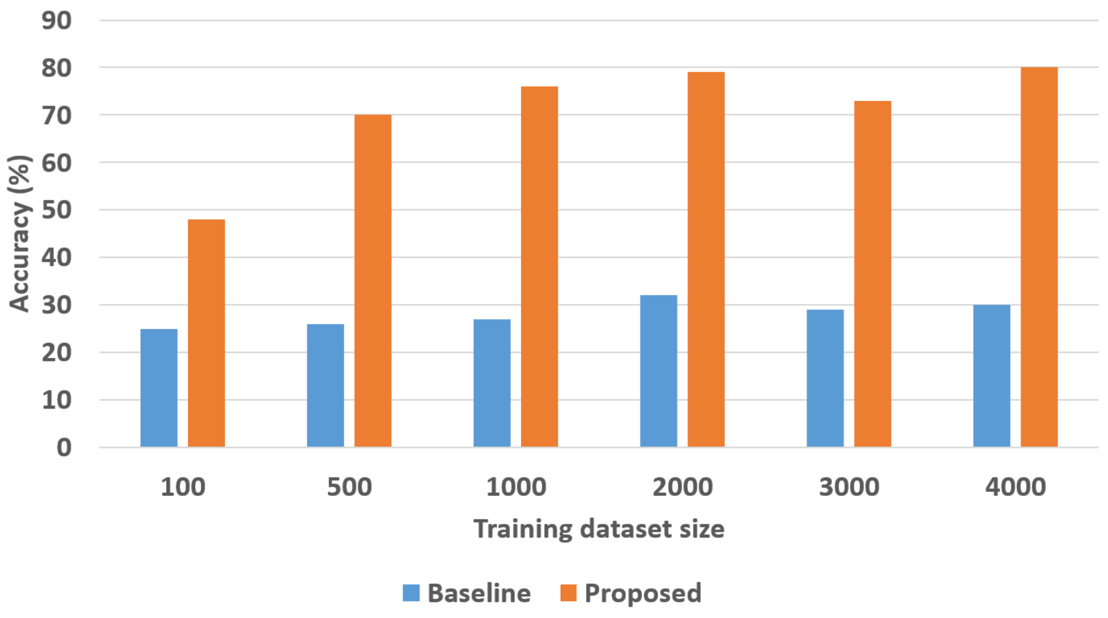

Figure 4.5: VQA accuracy as a function of number of training samples in TS domain.

results are summarized in Figures 4.2 and 4.3 for the SS domain, and in Figures 4.4 and 4.5 for the TS domain.

Interestingly, in the relatively simple SS domain, which has only two classes (stable and unstable), the baseline's classification performance was higher than that of the proposed architecture (see Figure 4.2). This is because this classification task was simple enough for the deep network to learn, even with limited training data. For the more complex task of VQA, however, there was insufficient training data for the data-hungry baseline architecture, and it performed poorly in both the SS and TS domains (see Figures 4.3 and 4.5). The proposed architecture performed well for VQA because only a subset of the questions in the dataset were answered by the RNN (the questions paired with images that were classified by the decision tree), which restricted the RNN's search space and allowed it to learn sufficiently even from limited training data.

The statistical significance of the observed VQA performance was calculated by running paired two-tailed t-tests. It was observed that the VQA performance of the proposed architecture was significantly better than that of the baseline architecture for all the dataset sizes tested. These results support hypothesis $H2$.

To further explore the observed results, a 'confidence value' was obtained from the logits layer of each CNN used to extract a feature from the input image. For each CNN, the confidence value was the largest probability assigned to any of the possible values of the corresponding feature, i.e., it was the probability assigned to the most likely value of the feature. These confidence values were considered to be a measure of the network's confidence in the corresponding features being a good representation of the image. If the confidence value for any feature was low (below 50%), the image features were only used to revise the decision tree (during training), or were processed using the decision tree (during testing). It was hypothesized that this approach would improve the accuracy of classification and question answering, but it did not make any significant difference in the experimental trials.

4.3 Architecture combination

In the RA domain the VQA and planning architectures were combined to create an implementation that had both functionalities. The simulated Turtlebot, equipped with classification, question answering and planning knowledge bases, was able to navigate through an office building, locate the intended recipient of a message, deliver the message, detect and reason about objects in its surroundings, and answer questions about the rooms it had visited. Average results from 30 trials showed VQA performance (after the VQA components of the combined architecture had been trained on 500 images) to be 82%, and planning performance is discussed in detail in Section 4.4. The capabilities of the combined architecture are shown in the following two execution traces. The first used the original planning architecture, while the second used a planning architecture that had been updated through axiom learning (more details in the following

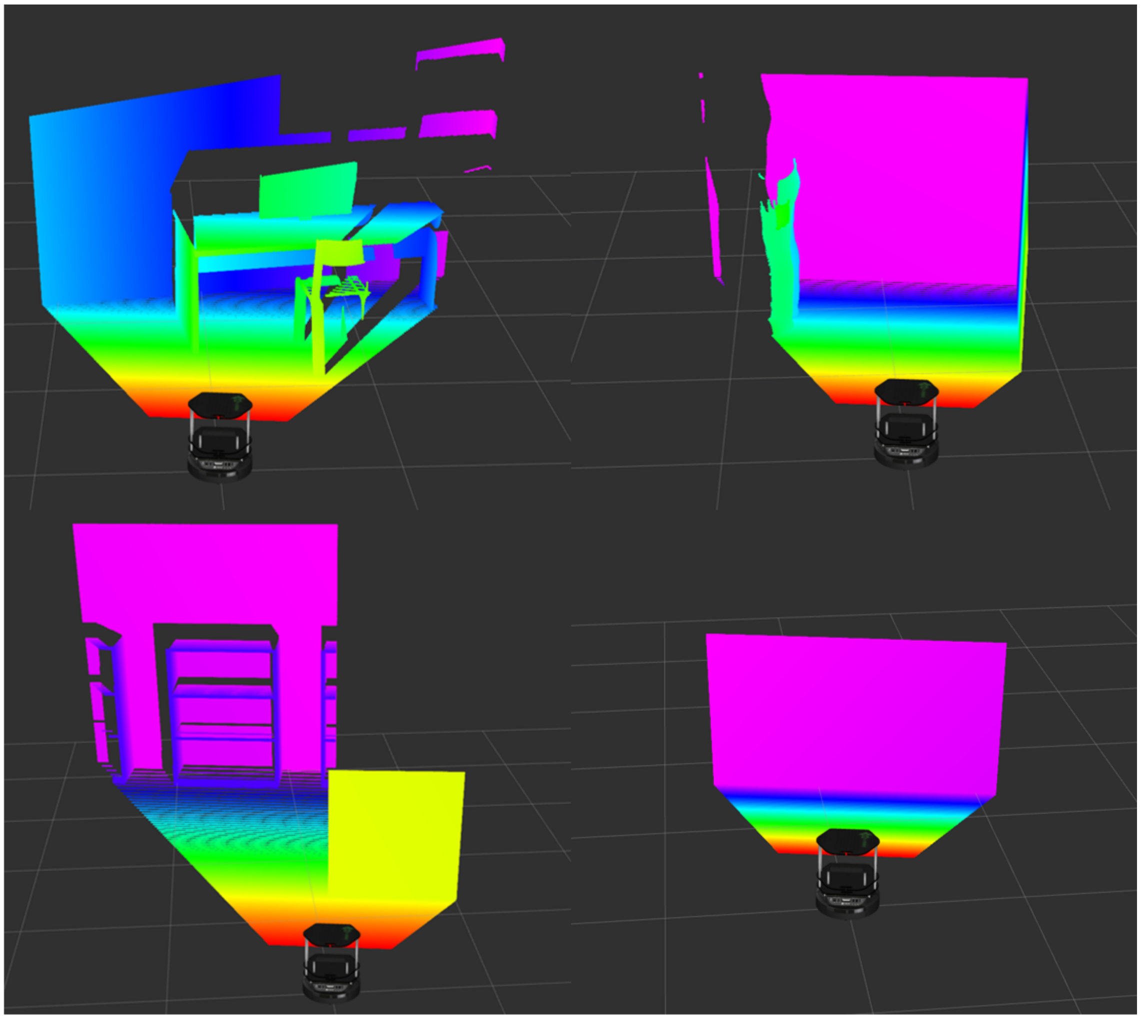

Figure 4.6: The images taken by the simulated Turtlebot after delivering a message in Bob's office.

section).

Figure 4.6 shows the images used for classification and question answering in the first execution trace. The robot's goal was to deliver a message from Sarah to Bob, and then return to Sarah to answer questions about the state of Bob's location. The robot's initial location was Sarah's office. The robot came up with a plan to move to the kitchen, deliver the message to Bob there, and then return to Sarah's office. However, when it moved to the kitchen it observed that Bob was not there, so it replanned to move to the library, deliver its message, and then return to Sarah's office via the kitchen. Again, the Turtlebot did not observe Bob in his expected location (the library), so it had to replan a second time. This time the robot's plan was to move to Bob's office, deliver its message, and then move back through the library and kitchen to Sarah's office. The third plan was successful, and after finding Bob in his office and delivering its message, the Turtlebot examined Bob's office by rotating and taking the four pictures seen in Figure 4.6. Having returned to Sarah's office, the robot had the following exchange:

Sarah's question: "is Bob's current location cluttered?"

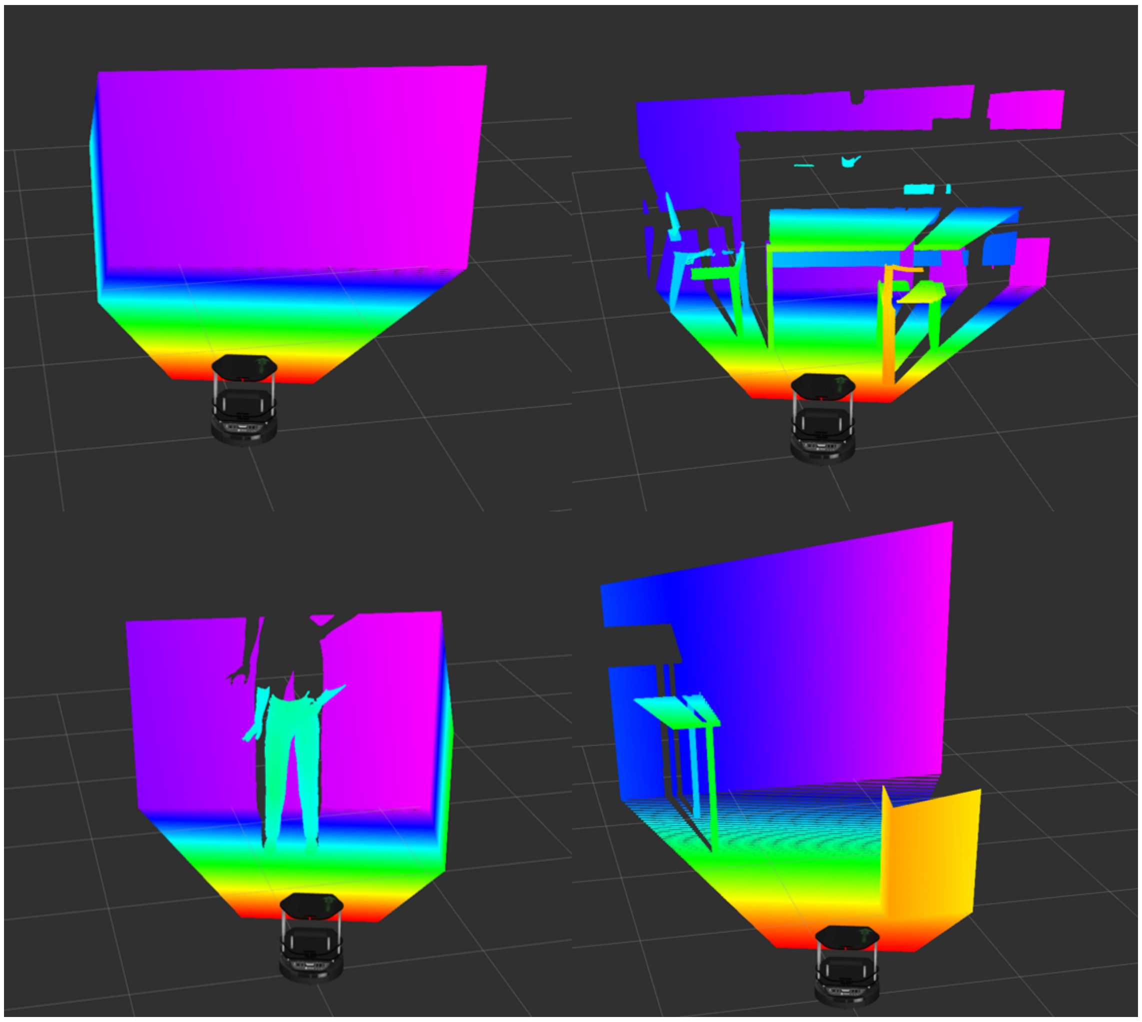
Figure 4.7: The images taken by the simulated Turtlebot after delivering a message in Sally's office.

Answer: "No."

Explanation:

"Bob was found in Bob's office.

The objects detected were Bob's chair, Bob's desk and Bob's computer.

The room was classified as uncluttered because it usually has three objects and three objects were detected."

It is interesting to note that though the bookcases in the library are visible through the doorway in one of the Turtlebot's images, it does not detect the bookcases as objects. This is because the Turtlebot's depth camera has coloured them dark blue to indicate that they are far away, and the CNNs for object detection have only been trained to detect close by objects as the rooms in the domain are quite small.

Figure 4.7 shows the images used for classification and question answering in the second execution trace. The robot's goal was to deliver a message from John to Sally, and then return to John to answer questions about the state of Sally's location. The robot's initial location was the kitchen. The robot

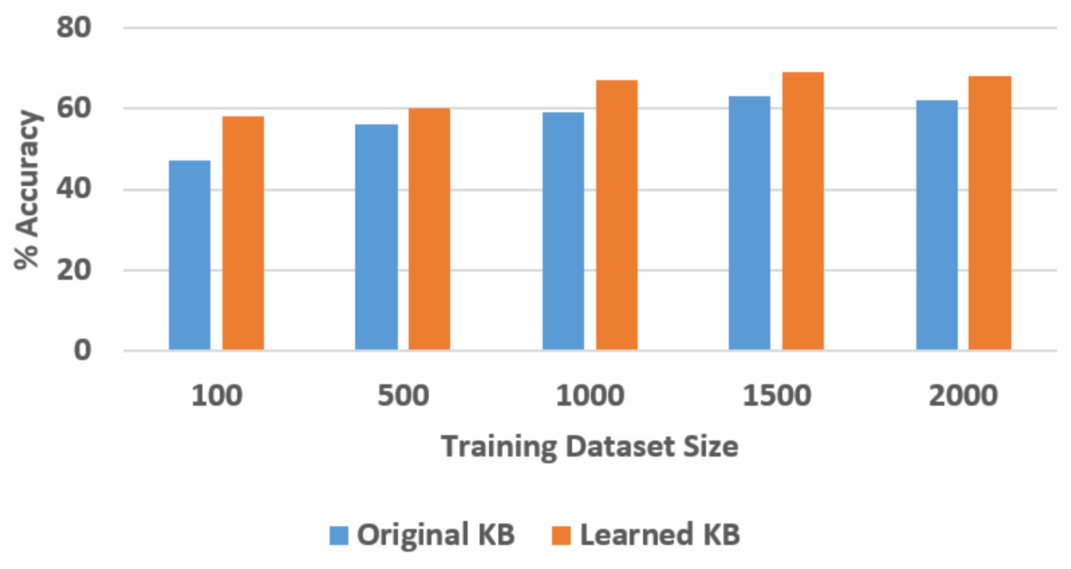

Figure 4.8: Classification performance in the stability domain before and after axiom learning.

came up with a plan to move to Sally's office, deliver the message and return to the kitchen. This plan was successful, and after finding Sally in her office and delivering the message the robot took images to examine the objects in the room. After returning to the kitchen it had the following exchange:

John's question: "is Sally's current location messy?"

Answer: "Yes."

Explanation:

"Sally was found in Sally's office.
The objects detected were Sally's chair, Sally's desk, Sally's computer, chair_one, and cup_one.
The room was classified as messy because chair_one belongs in the kitchen and cup_one belongs in the kitchen."

In this case the robot did not actually detect all the objects correctly. There was also a plate from the kitchen on Sally's desk, but it was too flat for the robot to be able to see it. Despite not detecting all of the objects the robot answered the question correctly.

These results support hypothesis $H3$.

4.4 Axiom learning

In complex domains it can be difficult to include all the relevant rules in a knowledge base when creating it by hand. In these domains the best approach is to add all the obvious rules to the knowledge base by hand, and then use axiom learning to add more complex rules that the user may have missed, resulting in a complete knowledge base that gives high performance.

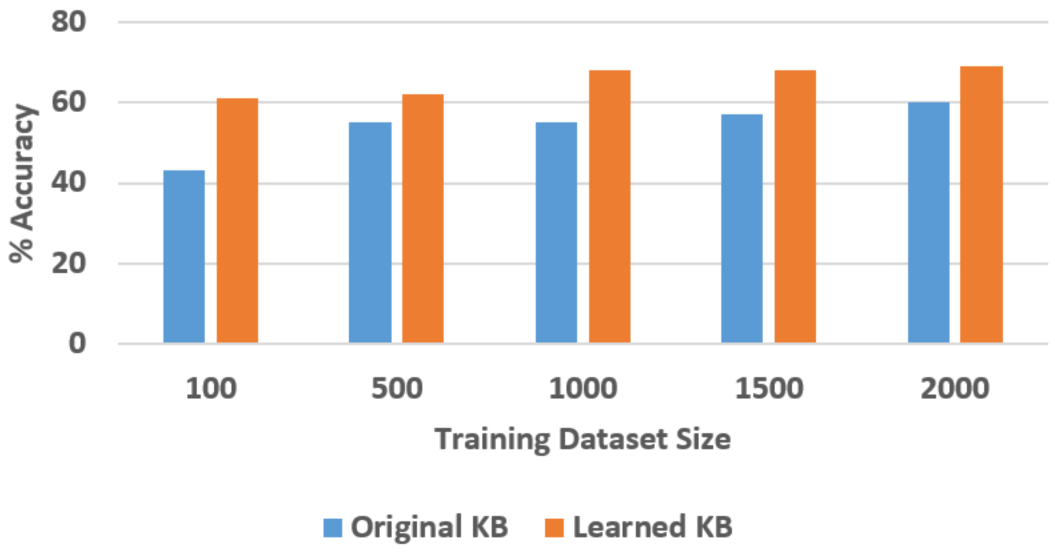

Figure 4.9: VQA performance in the stability domain before and after axiom learning.

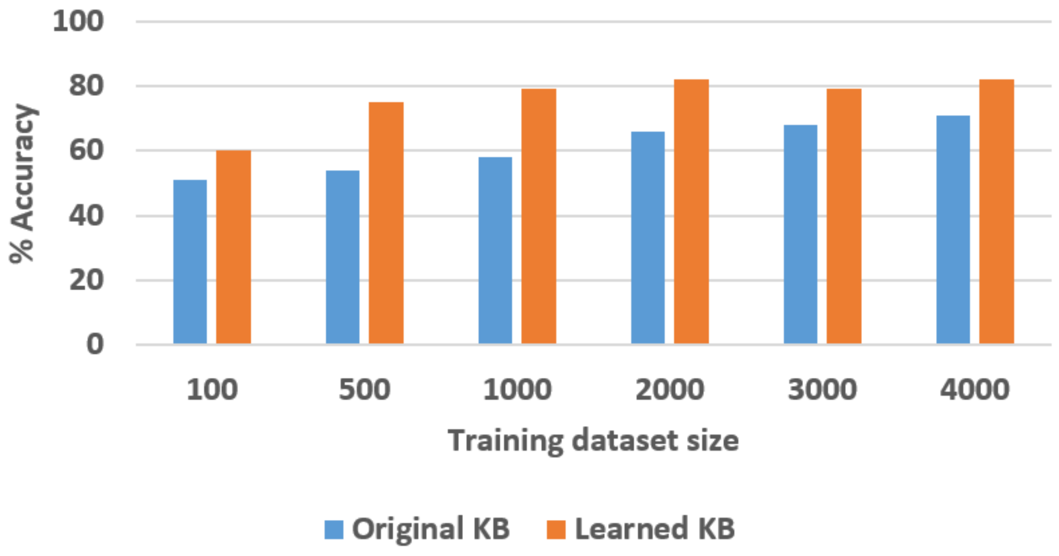

Figure 4.10: Classification performance in the traffic domain before and after axiom learning.

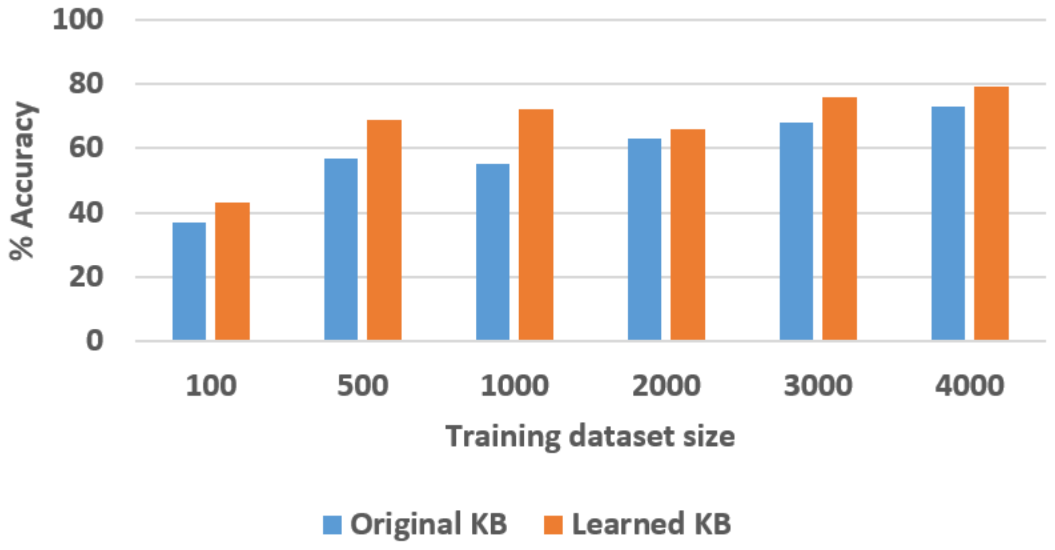

Figure 4.11: VQA performance in the traffic domain before and after axiom learning.

The SS and TS domains were too simple for axiom learning to make a difference, but the advantage of axiom learning in a complex domain could be seen by removing some of the axioms, thus simulating a complex domain where it was difficult for the user to find all the relevant axioms. In both these domains axiom learning was applied to the classification knowledge base after some of the classification axioms had been removed. Figures 4.8, 4.9, 4.10, and 4.11 show classification and VQA performance in the stability and traffic domains. Performance was tested before and after axiom learning, with 'Original KB' referring to the knowledge base before axiom learning had been performed, and 'Learned KB' referring to the knowledge base after the learned axioms had been added. These figures show the averaged results of ten trials. In each trial, one quarter of the classification axioms in the knowledge base were randomly selected to be removed, thus simulating a complex domain in which the human designer could not find all the relevant axioms. This handicapped knowledge base is what is refered to in the figures as 'Original KB'. The statistical significance of the observed classification and VQA performance was calculated by running paired two-tailed t-tests. It was observed that performance before axiom learning was significantly better than performance after axiom learning. An example of axiom learning and generalisation from the stability domain is given below.

The two automatically learned axioms

$$structure(stable) \leftarrow num\_blocks(3), \neg structure(narrow\_base), structure(lean). \tag{4.1}$$

and

$$structure(stable) \leftarrow num\_blocks(3), \neg structure(narrow\_base), \neg structure(lean). \tag{4.2}$$

generalise to

$$structure(stable) \leftarrow num\_blocks(3), \neg structure(narrow\_base). \tag{4.3}$$

Axiom learning	Tower height (boxes)	Planning time (seconds)
Before	2.5	155
After	3.6	124

Table 4.1: TB domain planning performance (averages of 30 trials) before and after axiom learning.

The complex and dynamic TB domain was used to test axiom learning for planning knowledge bases. The results can be seen in Table 4.1. Before axiom learning was used to learn stability rules the robot took a longer time to plan and came up with plans that resulted in shorter towers than after. These results indicate that after adding automatically learned rules to the knowledge base the robot was able to make more sensible planning decisions, coming up with stable box combinations that resulted in higher towers and fewer collapses. Restricting the number of block combinations that could be attempted also reduced planning time.

Some examples of the axioms it learned are:

$$holds(unstable\_placement(red\_box), I) \ \leftarrow \ holds(loc(green\_box, 1), I). \tag{4.4}$$

$$holds(unstable\_placement(blue\_box), I) \ \leftarrow \ holds(loc(yellow\_box, 1), I). \tag{4.5}$$

The first axiom states that an unstable placement would be the result of placing the red box when the green box is at tower level one (which is the base of the tower). The second axiom states that an unstable placement would be the result of placing the blue box when the yellow box is at the base of the tower. These axioms are sensible because they prohibit placing a block on top of a tower with a smaller block as its base, which would usually be unstable. With enough of these rules added to the knowledge base the robot will never come up with an action plan that results in a commonly unstable box combination.

The RA domain was also used to evaluate the effects of axiom learning. Its planning task was to find particular individuals and deliver messages to them. There were four employees in its simulated office world, and each one spent most of their time in their assigned workplace (their office). Axiom learning was used in this domain to learn statics that encode the location of each employee's workplace, e.g. *workplace(john,johns_office)*. These were included in the knowledge base to allow the simulated Turtlebot to more accurately predict where each person would be, and thus waste less time searching for them when trying to deliver a message. A default axiom was included to make use of the information:

$$holds(loc(P, L), 0) \ \leftarrow \ not\ default\_negated(P, L), \ workplace(P, L). \tag{4.6}$$

This axiom means that the Turtlebot will assume that a person's initial location is their workplace unless the default has been negated. There are two conditions that can cause the default to be negated:

$$default\_negated(P, L) \leftarrow obs(loc(P, L2), true, I),\ L! = L2. \tag{4.7}$$

$$default\_negated(P, L) \leftarrow obs(loc(P, L), false, I). \tag{4.8}$$

The first of these two axioms means that the default assumption should be ignored if the person in question is observed to be in a location other than their workplace, and the second axiom means that the default assumption should be ignored if the workplace is observed to not contain the person.

Axiom learning	Plans (per trial)	Actions (per trial)	Execution time (seconds per trial)	Planning time (seconds per trial)	Planning time (seconds per plan)
Before	3.5	8	273	17.23	5.08
After	1	5	184	3.88	3.88

Table 4.2: RA domain planning performance (averages of 30 trials) before and after axiom learning.

Thirty paired trials were run. For each paired trial the same goal and initial conditions were used to test planning performance before and after axiom learning had occurred. The results can be viewed in Table 4.2. Axiom learning was able to generate a static encoding the workplace of every employee included in the domain. Before axiom learning the Turtlebot often looked in the wrong location and had to replan, resulting in an average of 3.5 plans calculated per trial. After axiom learning the Turtlebot always went straight to the message recipient's workplace so no replanning was needed. This also resulted in fewer actions being executed per trial, as before axiom learning unnecessary actions would be executed while the Turtlebot searched rooms the message recipient wasn't in. Axiom learning improved both execution and planning time. Planning time was calculated as both per trial (this value was improved through the reduction in replanning) and per individual plan (this value was improved because the Turtlebot could assume each worker was in their office and thus didn't need to consider every possibility).

In each of the RA experiments the recipient of the message was in their office, which meant the Turtlebot was able to complete its message delivery tasks much more efficiently when it knew where each person's workplace was. This was considered a reasonable representation of employee behaviour as though the office workers would go to the kitchen during their lunch breaks and would take trips to the bathroom, during working (i.e. not on break) hours they would almost always be in their offices, and it was assumed that coworkers would mainly require the robot's message delivery services during working hours.

These results show that axiom learning can be used to automatically add any missed axioms to a knowledge base, improving performance. This supports hypothesis $H4$.

Conclusions

Deep learning is an important machine learning technique that can be used for a huge variety of applications. Currently, deep learning algorithms work well when enough training data is provided, but performance for complex tasks falls dramatically when the dataset size shrinks to a few thousand samples or less. The internal functioning of neural networks is difficult to understand, making deep learning an uninterpretable technique that some people are reluctant to trust. A new architecture has been designed to break the VQA task down into parts and apply deep learning only to the sub-tasks where they were really needed (image processing, answering questions for unexpected feature combinations). Non-monotonic logic and decision tree induction were used to provide interpretable outputs for classification and question answering. This resulted in an architecture that had minimal training data requirements, interpretable outputs, and high performance. Its modular design made it easy to combine with other architectures. Experimental results showed that the proposed architecture outperformed an architecture based on just deep networks when the size of the training dataset was small, and the proposed architecture provided intuitive explanations for classification decisions. Experiments with planning and axiom learning verified that in complex domains using axiom learning to add to a user-created knowledge base improved performance, and that the proposed architecture could be successfully combined with a planning architecture.

This architecture opens up multiple directions for future work. It would be interesting to explore the use of this architecture in other domains with datasets of increasing size and complexity (in terms of the features and the questions). It would also be useful to include different deep network structures in the architecture, using the explanatory answers to further understand the internal representation of these networks. The current axiom learning methodology applies to state constraints and statics, and an alternative methodology for executability conditions was referred to, but in future work a new methodology could be developed for causal laws. An additional improvement that could be made is to use online learning - in which testing data is used to iteratively update the training dataset - to improve the proposed architecture's performance over time.

Bibliography

[1] T. Zhang, D. Dai, T. Tuytelaars, and L. Van Gool, "Speech-based visual question answering," *Computing Research Repository,* vol. abs/1705.00464, 2017. [Online]. Available: arXiv, https://arxiv.org/abs/1705.00464. [Accessed Jan 17, 2019].

[2] Masuda, S. de la Puente, and X. Giro-i-Nieto, "Open-ended visual question-answering," B.Sc.Eng dissertation, Telecom BCN school, Technical University of Catalonia, Barcelona, Spain, 2016.

[3] A. Jiang, F. Wang, F. Porikli, and Y. Li, "Compositional memory for visual question answering," *Computing Research Repository,* vol. abs/1511.05676, 2015. [Online]. Available: arXiv, https://arxiv.org/abs/1511.05676. [Accessed Jan 17, 2019].

[4] S. Pandhre and S. Sodhani, "Survey of recent advances in visual question answering," *Computing Research Repository,* vol. abs/1709.08203, 2017. [Online]. Available: arXiv, https://arxiv.org/abs/1709.08203. [Accessed Jan 17, 2019].

[5] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A deep learning approach to visual question answering," *International Journal of Computer Vision*, vol. 125, pp. 110-135, Dec. 2017.

[6] T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. Zitnick, and P. Dollar, "Microsoft COCO: Common objects in context," in *Lecture Notes in Computer Science, vol. 8693, ECCV 2014, Zurich, Switzerland, September 6-12, 2014*, D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars, Eds. Springer, 2014.

[7] M. Malinowski and M. Fritz, "Towards a visual Turing challenge," in *Neural Information Processing Systems Workshop on Learning Semantics, NIPS 2014, Montreal, Canada, December 8-13, 2014.*

[8] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, "Stacked attention networks for image question answering," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, USA, June 26 - July 1, 2016*. pp. 21-29.

[9] J. Lu, J. Yang, D. Batra, and D. Parikh, "Hierarchical question-image co-Attention for visual question answering," in *30th Conference on Neural Information Processing Systems, NIPS 2016, Barcelona, Spain, December 5-10, 2016.*

[10] X. Lin and D. Parikh, “Active learning for visual question answering: An empirical study,” *Computing Research Repository,* vol. abs/1711.01732, 2017. [Online]. Available: arXiv, https://arxiv.org/abs/1711.01732. [Accessed Jan 17, 2019].

[11] P. Wang, Q. Wu, C. Shen, A. van den Hengel, and A. Dick, “Explicit knowledge-based reasoning for visual question answering,” in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence Main track, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. pp. 1290-1296.

[12] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. Shamma, M. Bernstein, and F. Li, “Visual genome: Connecting language and vision using crowdsourced dense image annotations,” *International Journal of Computer Vision*, vol. 123, pp. 32-73, Feb. 2016.

[13] D. Teney and A. van den Hengel, “Zero-shot visual question answering,” *Computing Research Repository,* vol. abs/1611.05546, 2016. [Online]. Available: arXiv, https://arxiv.org/abs/1611.05546. [Accessed Jan 17, 2019].

[14] A. Jabri, A. Joulin, and L. van der Maaten, “Revisiting visual question answering baselines,” in *Lecture Notes in Computer Science, vol. 9912, ECCV 2016, Amsterdam, The Netherlands, October 8-16, 2016*. B. Leibe, J. Matas, N. Sebe, M. Welling, Eds. Springer, 2016. pp. 727-739.

[15] Y. Goyal, T. Khot, D. Stay, D. Batra, and D. Parikh, “Making the V in VQA matter: Elevating the role of image understanding in visual question answering,” in *30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, Hawaii, July 21-26, 2017*. pp. 6325-6334.

[16] P. Zhang, Y. Goyal, D. Stay, D. Batra, and D. Parikh, “Yin and yang: Balancing and answering binary visual questions,” in *29th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, USA, June 26 - July 1, 2016*. pp. 5014-5022.

[17] A. Agrawal, D. Batra, D. Parikh, and A. Kembhavi, “Dont just assume; look and answer: Overcoming priors for visual question answering,” in *31st IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, Utah, USA, June 19-21, 2018.*

[18] A. ray, G. Christie, M. Bansal, D. Batra, and D. Parikh, “Question relevance in VQA: Identifying non-visual and false-premise questions,” in *2016 Conference on Empirical Methods on Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-5, 2016.*

[19] R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” in *2017 IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. pp. 618-626.

[20] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *AAAI Publications, Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. pp. 1507-1514.

[21] S. Tellex, R. A. Knepper, A. Li, T. M. Howard, D. Rus, and N. Roy, "Asking for help using inverse semantics," in *Proceedings of Robotics: Science and Systems 2014, Berkeley, USA, July 12-16, 2014*, D. Fox, L. E. Kavraki, H. Kurniawati Eds.

[22] M. R. Walter, S. Hemachandra, B. Homberg, S. Tellex, and S. Teller, "Learning semantic maps from natural language descriptions," in *Proceedings of Robotics: Science and Systems 2013, Berlin, Germany, June 24-28, 2013*, P. Newman, D. Fox, D. Hsu Eds.

[23] S. Tellex and D. Roy, "Towards surveillance video search by natural language query," in *Proceedings of the ACM International Conference on Image and Video, CIVR 2009, Santorini, Fira, Greece, July 8-19, 2009*.

[24] S. Tellex, T. Kollar, G. Shaw, N. Roy, and D. Roy, "Grounding spatial language for video search," in *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodel Interaction, Article No. 31, ICMI-MLMI 2010, Beijing, China, November 8-10, 2010.*

[25] K. Hsiao, S. Tellex, S. Vosoughi, R. Kubat, and D. Roy, "Object schemas for grounding language in a responsive robot," *Connection Science*, vol. 20, pp. 253-276, Dec. 2008.

[26] J. MacGlashan, M. Babes-Vroman, M. desJardins, M. L. Littman, S. Muresan, S. Squire, S. Tellex, D. Arumugam, and L. Yang, "Grounding English commands to reward functions," in in *Proceedings of Robotics: Science and Systems 2015, Rome, Italy, June 24-28, 2015*, L. E. Kavraki, D. Hsu, J. Buchli Eds.

[27] N. F. Rajani and R. J. Mooney, "Using explanations to improve ensembling of visual question answering systems," in *Proceedings of the IJCAI 2017 Workshop on Explainable Artifical Intelligence, IJCAI 2017, Melbourne, Australia, August 20, 2017*. pp. 43-47.

[28] S. Venugopalan, L. A. Hendricks, R. Mooney, and K. Saenko, "Improving LSTM-based video description with linguistic knowledge mined from text," in *The 2016 Conference on Empirical Methods on Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-5, 2016*. pp. 1961-1966.

[29] S. Venugopalan, M. Rohrbach, J. Donahue, R. J. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence - video to text," in *Internation Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 13-16, 2015*.

[30] J. Thomason, S. Zhang, R. Mooney, and P. Stone, "Learning to interpret natural language commands through human-robot dialog," in *Proceedings of the Twenty-Fourth International Joint Conference on Artifical Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. pp. 1923-1929.

[31] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, "Translating videos to natural language using deep recurrent neural networks," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2015, Denver, Colorado, USA, May 31 - June 5, 2015*. pp. 1494-1504.

[32] J. Thomason, S. Venugopalan, S. Guadarrama, K. Saenko, and R. Mooney, "Integrating language and vision to generate natural language descriptions of videos in the wild," in *Proceedings of the 25th Interntational Conference on Computational Linguistics: Technocal Papers, COLING 2014, Dublin, Ireland, August 23-29, 2014*. pp. 1218-1227.

[33] R. Timofte, K. Zimmermann, and L. van Gool, "Multi-view traffic sign detection, recognition, and 3D localisation," *Machine Vision and Applications*, vol. 25(3), pp. 633-647, April 2014.

[34] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

[35] E. Coumans and Y. Bai, *Official Python Interface for the Bullet Physics SDK specialized for Robotics Simulation and Reinforcement Learning*. [Online]. Available: https://pypi.org/project/pybullet/. [Accessed Jan. 19, 2019].

[36] M. Gelfond and Y. Kahl, *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.* Cambridge: Cambridge University Press, 2014.

[37] E. Balai, M. Gelfond, and Yuanlin Zhang, "SPARC - Sorted ASP with consistency restoring rules," in *Proceedings of Answer Set Programming and Other Computing Paradigms, 5th International Workshop, ASPOCP 2012, Budapest, Hungary, September 4, 2012*.

[38] M. Sridharan and B. Meadows, "Should I do that? using relational reinforcement learning and declarative programming to discover domain axioms," in *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2016, place, date, 2016*. pp. 252-259.

[39] U. Furbach, I. Glckner, H. Helbig, and B. Pelzer, "Logic-based question answering," *Knstliche Intelligenz*, vol. 24, pp. 5155, 2010.

[40] M. Wagner, H. Basevi, R. Shetty, W. Li, M. Malinowski, M. Fritz, and A. Leonardis, "Answering visual what-if questions: From actions to predicted scene descriptions," in *Visual Learning and*

Embodied Agents in Simulation Environments Workshop at ECCV, ECCV18-VLEASE, Munich, Germany, September 9th, 2018.

[41] M. T. Young and E. Amir, "Building knowledge about buildings," in *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century, Vancouver, BC, Canada, October 22-25, 1995.*

[42] H. O. Nyongesa and P. L. Rosin, "Neural-fuzzy applications in computer vision," *Journal of Intelligent and Robotic Systems*, vol. 29, pp. 309-315, 2000.

[43] R. Munoz-Salinas, E. Aguirre, M. Gar, and A. Gon, "Door-detection using computer vision and fuzzy logic," *WSEAS Transactions on Systems*, vol. 10(3), Jan. 2004.

[44] J. M. Keller and P. Gader, "Fuzzy logic and the principle of least commitment in computer vision," in *Proceedings of the AAAI 2006 Fall Symposium on Semantic Web and Collaborative Knowledge Acquisition, shortcut, Arlington, Virginia, USA, October 13-15, 2006.*

[45] M. Hu, Y. Zhao, and G. Zhai, "Active learning algorithm can establish classifier of blueberry damage with very small training dataset using hyperspectral transmittance data," *Chemometrics and Intelligent Laboratory Systems*, vol. 172, pp. 52-57, Jan. 2018.

[46] C. Lu and W. Li, "Ship classification in high-resolution SAR images via transfer learning with small training dataset," *Sensors*, vol. 19(1), pp. 63, Dec. 2018.

[47] Z. Huang, Z. Pan, and B. Lei, "Transfer learning with deep convolutional neural network for SAR target Classification with Limited Labeled Data," *Remote Sensing*, vol. 9, pp. 907, 2017.

[48] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, Miami, FL, USA, June 20-25, 2009.*

[49] Q. Zhang, Y. N. Wu, and S. Zhu, "Interpretable convolutional neural networks," 2017. [Online]. Available: arXiv, https://arxiv.org/abs/1710.00935. [Accessed Jan. 27, 2019].

[50] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *2017 IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. pp. 3449-3457.

[51] I. Sturm, S. Bach, W. Samek, and K. Muller, "Interpretable deep neural networks for single-trial EEG classification," *Journal of Neuroscience Methods*, vol. 274, pp. 141-145, 2016.

[52] J. Ijsselmuiden and R. Stiefelhagen, “Towards high-level human activity recognition through computer vision and temporal logic,” in *KI 2010: Advances in Artificial Intelligence, KI 2010, Karlsruhe, Germany, September 21-24, 2010*, R. Dillman, J. Beyerer, U.D. Hanebeck, T. Schultz Eds. Springer, 2010. pp. 426-435.

[53] R. Moller, B. Neumann, and M. Wessel, “Towards computer vision with description logics: some recent progress,” in *Workshop on Integration of Speech and Image Understanding, ICCV 1999, Corfu, Greece, September 21, 1999*. IEEE Computer Society. pp. 101-116.

[54] K. Price, T. Russ, R. MacGregor, “Knowledge representation for computer vision: The VEIL project,” in *ARPA Image Understanding Workshop, ARPA 1994, Monterey, California, USA, November 13-16, 1994*.

[55] S. R. Fiorini and A. Abel, “A review on knowledge-based computer vision,” 2010.

[56] M. D. Levine and W. Hong, “A knowledge-based approach to computer vision systems,” in *Proceedings of Graphics Interface and Vision Interface 1986, Vancouver, British Columbia, Canada, May 26-30, 1986*. pp. 260-265.

[57] G. van Rossum and F.L. Drake (eds), “Python reference manual”, PythonLabs, Virginia, USA, 2001. [Online]. Available: http://www.python.org. [Accessed Feb 1, 2019].

[58] B. A. Draper, A. Hanson, and E. M. Riseman, “Knowledge-directed vision: Control, learning and integration,” *Proceedings of the IEEE*, vol. 84(11), pp. 1625-1637, Dec. 1996.

[59] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, Nevada, USA, June 26 - July 1, 2016*. pp. 770-778.

[60] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9(8), pp. 1735-1780, Nov. 1997.

[61] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *Computing Research Repository,* vol. abs/1409.1556, 2014. [Online]. Available: arXiv, https://arxiv.org/abs/1409.1556. [Accessed Jan 29, 2019].

[62] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Man, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vigas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine

learning on heterogeneous systems," 2015. [Online]. Available: www.tensorflow.org. [Accessed Feb 1, 2019].

[63] S. Shalev-Shwartz and A. Shashua, "On the sample complexity of end-to-end training vs. semantic abstraction training," *Computing Research Repository,* vol. abs/1604.06915, 2016. [Online]. Available: arXiv, https://arxiv.org/abs/1604.06915. [Accessed Feb 1, 2019].

[64] E. Erdem, and V. Patoglu, "Applications of action languages to cognitive robotics," *Correct Reasoning*, E. Erdem, J. Lee, Y. Lierler, D. Pearce Eds. Springer, 2012. pp. 229-246.

[65] S. Zhang, M. Sridharan, and J. Wyatt, "Mixed logical inference and probabilistic planning for robots in unreliable worlds," *IEEE Transactions on Robotics*, vol. 31, pp. 699713, 2015.

[66] E. Erdem, M. Gelfond, and N. Leone, "Applications of answer set programming," *AI Magazine*, vol. 37, pp. 5368, 2016.

[67] M. Sridharan, M. Gelfond, S. Zhang, and J. Wyatt, "REBA: A refinement-based architecture for knowledge representation and reasoning in robotics," 2018. [Online]. Available: http://arxiv.org/abs/1508.03891. [Accessed Feb. 4, 2019].

[68] M. Sridharan and B. Meadows, "Knowledge representation and interactive learning of domain knowledge for human-robot collaboration," *Advances in Cognitive Systems*, vol. 7, pp. 6988, 2018.

[69] M. Balduccini and M. Gelfond, "Logic programs with consistency-restoring rules," in *AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium, Palo Alto, California, March 24-26, 2003*. pp. 918.

[70] H. Riley and M. Sridharan, "Non-monotonic logical reasoning and deep learning for explainable visual question answering," in *2018 International Conference on Human-Agent Interaction Southampton, HAI 2018, UK, December 15-18, 2018.*

[71] R. O. Duda, P. E. Hart, and D. G. Stork, *Patter Classification (Second Edition).* Wiley-Blackwell, 2000.

[72] Y. Bai, J. Fu, T. Zhao, and T. Mei, "Deep attention neural tensor network for visual question answering," in *2018 European Conference on Computer Vision, ECCV 2018, Munich, Germany, September 8-14, 2018.*

[73] I. Schwartz, A. Schwing, and T. Hazan, "High-Order Attention Models for Visual Question Answering," in *Advances in Neural Information Processing Systems 2018, NIPS 2018, Long Beach, USA, December 2-8, 2017*. pp. 36643674.

[74] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *2017 International Conference on Machine Learning, ICML 2017, Sydney, Australia, August 6-11, 2017.* pp. 18851894.

[75] M. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016, San Francisco, CA, USA, August 13-17, 2016.* pp. 11351144.

[76] Y. Gil, "Learning by experimentation: Incremental refinement of incomplete planning domains," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning, ICML 1994, New Brunswick, NJ, USA, July 10-13, 1994*. pp. 8795.

[77] R. P. Otero, "Induction of the effects of actions by monotonic methods," in *2003 International Conference on Inductive Logic Programming, ILP 2003, Szeged, Hungary, September 29 - October 1, 2003.* pp. 299310.

[78] M. Balduccini, "Learning action descriptions with A-Prolog: Action language C," in *2007 AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning, AAAI 2007 Spring Symposium, Palo Alto, California, March 26-28, 2007.*

[79] M. Law, A. Russo, and K. Broda, "The complexity and generality of learning answer set programs," *Artificial Intelligence*, vol. 259, pp. 110146, 2018.

[80] J. E. Laird, *The Soar Cognitive Architecture*. The MIT Press, 2012

[81] V. Sarathy and M. Scheutz, "A logic-based computational framework for inferring cognitive affordances," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 8, 2016.

[82] J. Y. Chai, Q. Gao, L. She, S. Yang, S. Saba-Sadiya, and G. Xu, "Language to action: Towards interactive task learning with physical agents," in *2018 International Joint Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13-19, 2018.*

[83] J. E. Laird, K. Gluck, J. Anderson, K. D. Forbus, O. C. Jenkins, C. Lebiere, D. Salvucci, M. Scheutz, A. Thomaz, G. Trafton, R. E. Wray, S. Mohan, and J. R. Kirk, "Interactive task learning," *IEEE Intelligent Systems* vol. 32, pp. 621, August 2017.

[84] H. Riley, "Github repository," 2019. [Online]. Available: https://github.com/hril230/masters_code. [Accessed Feb. 12, 2019].

[85] N. Koenig and A. Howard, "Design and use paradigms for gazebo, and open-source multi-robot simulator," in *2004 Intelligent Robots and Systems, IROS 2004, Sendai, Japan, September 28 - October 2, 2004.*

[86] A. Rasouli and J. K. Tsotsos, “The effect of color space selection on detectability and discriminability of colored objects,” 2017. [Online]. Available: http://arXiv.org/abs/1702.05421. [Accessed Feb. 19, 2019].