A Confidence-Based Interface for Neuro-Symbolic Visual Question Answering

Thomas Eiter, Nelson Higuera, Johannes Oetsch, Michael Pritz

Institute of Logic and Computation,
Vienna University of Technology (TU Wien), Austria
{eiter,higuera,oetsch,pritz}@kr.tuwien.ac.at

Abstract

We present a neuro-symbolic visual question answering (VQA) approach for the CLEVR dataset that is based on the combination of deep neural networks and answer-set programming (ASP), a logic-based paradigm for declarative problem solving. We provide a translation mechanism for the questions included in CLEVR to ASP programs. By exploiting choice rules, we consider deterministic and non-deterministic scene encodings. In addition, we introduce a confidence-based interface between the ASP module and the neural network which allows us to restrict the non-determinism to objects classified by the network with high confidence. Our experiments show that the non-deterministic scene encoding achieves good results even if the neural networks are trained rather poorly in comparison with the deterministic approach. This is important for building robust VQA systems if network predictions are less-than perfect.

1 Introduction

The capability of perceiving visual information and to reason about it is vital for many tasks. While humans have little trouble solving such problems, they often pose a hard challenge for machines; this motivates approaches for *visual question answering* (VQA) (Antol et al. 2015).

We consider VQA tasks, where the goal is to find the answer to a question using information from a scene. To this end, a system must understand the question, extract the relevant information from the corresponding scene, and perform some kind of reasoning. To examine the strengths and weaknesses of VQA systems, several data sets have been published (Malinowski and Fritz 2014; Antol et al. 2015; Ren, Kiros, and Zemel 2015; Zhu et al. 2016; Johnson et al. 2017; Sampat et al. 2021). We use the CLEVR dataset (Johnson et al. 2017) for our purposes, since it has detailed annotations describing the kind of reasoning each question requires.

Neuro-symbolic approaches combine deep learning with symbolic reasoning (Xu et al. 2018; Manhaeve et al. 2018; Yi et al. 2018; Yang, Ishay, and Lee 2020; Basu, Shakerin, and Gupta 2020; Mao et al. 2019). The basic idea is to separate perception and reasoning, i.e., deep learning models are used for perception (e.g., object detection or natural language processing), and logic-based inference is used for reasoning; as the semantics of the employed reasoning formalism is known, the way in which an answer is reached is transparent.

We present a neuro-symbolic VQA approach for the CLEVR dataset that is based on the combination of *deep neural networks* and *answer-set programming* (ASP) (Brewka, Eiter, and Truszczyński 2011),[1] a logic-based paradigm for declarative problem solving, where we focus on object detection and reasoning while we omit natural language processing for the VQA tasks. CLEVR provides in fact structured representations of the natural language questions, so-called *functional programs*. We encode functional programs as well as the output of a neural network in an ASP program, such that the answer sets of the program represent the answers to the given question w.r.t. to the scene information. More specifically, we use YOLOv3 (Redmon and Farhadi 2018) for bounding-box prediction and object classification. The scene encoding in the ASP program makes use of *non-deterministic choice rules* for the objects predicted with high confidence by the network. This means that we do not only consider the prediction with the highest score, but also reasonable alternatives with lower ones. This allows our approach to make up for mistakes made in object classification in the reasoning component as the constraints in the program exclude choices that do not lead to an answer. To determine what we regard as predictions of high confidence, we use statistical analysis on the distribution of network prediction values and disregard predictions that are below a threshold defined as a function of mean and standard deviation of the distribution.

We emphasise that our goal is not to achieve better results on CLEVR than related approaches, in fact the dataset can be considered solved, but rather to explore to what extent non-determinism helps when network predictions are wrong. Our experiments show that our system performs well on the CLEVR dataset and that the non-deterministic scene encoding achieves good results even if the neural networks are trained rather poorly in comparison with the deterministic approach where we only consider predictions with maximal scores. This is important for building robust VQA systems if predictions by the network are less-than perfect. Even well-trained networks can be negatively affected by noise or if settings like illumination change. Also, our method of incorporating neural network output into ASP programs leads

[1] https://anonymous.4open.science/r/submission/.

to a drastic performance improvement in terms of run time compared to a previous neuro-symbolic ASP approach (Yang, Ishay, and Lee 2020).

The remainder of this paper is organised as follows. We first discuss relevant related work in Section 2 and review ASP and CLEVR in Section 3. Our approach to VQA using ASP and confidence-thresholds is detailed in Section 4. Afterwards, we present an experimental evaluation of our approach in Section 5, and we conclude in Section 6.

2 Related Work

Purely deep-learning-based approaches (Yang et al. 2016; Lu et al. 2016; Jabri, Joulin, and Van Der Maaten 2016) led to significant advances in VQA. Some systems rely on attention mechanisms to focus on certain features of the image and question to retrieve the answer (Yang et al. 2016; Lu et al. 2016). Jabri, Joulin, and Van Der Maaten (2016) achieved good results by framing a VQA task as a classification problem. Some VQA systems are however suspected to not learn to reason but to exploit dataset biases to find answers, as described by Johnson et al. (2017).

Besides these purely data driven attempts, there are also systems which incorporate symbolic reasoning in combination with neural-based methods for VQA (Yi et al. 2018; Basu, Shakerin, and Gupta 2020; Mao et al. 2019). The system proposed by Yi et al. (2018) consists of a scene parser, which retrieves object level scene information from images, a question parser, which creates symbolic programs from natural language questions, and a program executor that runs these programs on the abstract scene representation. Our system is akin to this system, but we use ASP for scene representation as well as question encoding, and our program executor is an ASP solver. A similar system architecture appears in the approach by Mao et al. (2019) with the difference that scene and question parsing is jointly learned from image and question-answer pairs, whereas the components of Yi et al.'s system are trained separately. This means that annotated images are not necessary for training, which makes the system much more versatile. The approach of Basu, Shakerin, and Gupta (2020) builds like ours on ASP. They use object-level scene representations and parse natural language questions to obtain rules which represent the question. The answer to a question is given by the answer set for the image-question encoding which is combined with commonsense knowledge. However, their approach is not amenable to non-determinism for the scene encoding in order to deal with competing object classifications as we do.

Our non-deterministic scene encoding approach was inspired by the DeepProbLog (Manhaeve et al. 2018) and NeurASP (Yang, Ishay, and Lee 2020) frameworks. DeepProblog uses ProbLog, which is a probabilistic extension of Prolog, as the logical language, in which Manhaeve et al. introduced neural-annotated disjunctions to pass the output of the neural network as an input to the logic program. NeurASP uses the same bridging idea, now called neural atom, but uses ASP instead of ProbLog; notably, ProbLog is query-based, while ASP is model-based. Intuitively, neural atoms in NeurASP give rise to choice rules from the set of all random events and their respective outcomes for a specific problem. The neural network is used to predict confidence scores for the latter, which are treated as probabilities associated with respective atoms from which probabilities for answer sets are calculated. Our approach differs as we discard random event outcomes with confidence scores not exceeding a preset threshold.

3 Background

We next provide preliminaries on ASP and background on the CLEVR dataset.

3.1 Answer-Set Programming

Answer-set programming (ASP) is a declarative problem solving paradigm, where a problem is encoded as a logic program such that the models of the program, the *answer sets*, correspond to the solutions of the problem. Answer sets can be computed using dedicated ASP solvers.[2] We provide a short introduction to the most important ASP concepts and refer to background literature (Brewka, Eiter, and Truszczyński 2011; Gebser et al. 2012) for more details.

An *ASP program* is a set of *rules* of the form

$$a_1 \mid \ldots \mid a_k \leftarrow b_1, \ldots, b_m, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_n . \qquad (1)$$

where all a_i, b_j, c_l are atoms, not denotes *default negation*, and $k, m, n \geq 0$. We call $\{a_i, \ldots, a_k\}$ the head of r and $\{b_1, \ldots, b_m, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_n\}$ the body of r. Intuitively, a rule expresses that whenever all atoms $b_1, \ldots, b_m$ are true, and there is no evidence for any of the default negated atoms $c_1, \ldots, c_n$, then some atom in the head has to be true. If $m = n = 0$ and $k = 1$, then r is called a *fact* and $\leftarrow$ is omitted. A fact represents definite knowledge, since its body is always satisfied. A rule with an empty head is called *constraint* and is used to eliminate unwanted answer sets.

An *interpretation* I is a set of atoms. It satisfies a rule r if it contains at least one a_i from its head whenever it satisfies its body, i.e., $\{b_1, ..., b_m\} \subseteq I$ and $\{c_1, ..., c_n\} \cap I = \varnothing$ holds. Furthermore, I is a model of a program P if I satisfies each rule in P. An *answer-set* of P is a model of P where each atom can be derived in a consistent and non-circular way (Gelfond and Lifschitz 1991).

A program can have no, one, or more than one answer sets. For illustration, the simple program

$$a \leftarrow \mathit{not}\, b . \quad b \leftarrow \mathit{not}\, a .$$

has the answer sets $I_1 = \{a\}$ and $I_2 = \{b\}$. If we add the constraint $\leftarrow a.$ to the above program, the only remaining answer set is $\{b\}$ since $\{a\}$ is eliminated.

We will also use *choice rules* which are of the form

$$i\,\{a_1; \ldots; a_n\}\,j \leftarrow b_1, \ldots, b_m, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_n.$$

Such a rule says that when its body is satisfied, at least i and at most j atoms from $\{a_1, \ldots, a_n\}$ must be true in every answer set. For example, the following program

$$2\,\{a; b; c\}\,2 . \qquad \text{(also written } \{a; b; c\} = 2\text{)}$$

has the answer sets $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$.

[2] E.g., potassco.org, www.dlvsystem.com.

ASP also features weak constraints, sometimes called soft constraints, to solve optimisation problems. A *weak constraint* is a rule of form

$$\leftsquigarrow b_1, \ldots, b_m, \; not\; c_1, \ldots, \; not\; c_n. \; [w]$$

where w is an integer weight. Informally, whenever the body of a weak constraint is satisfied in an answer set, then a penalty of w is incurred; optimal answer sets are those minimising the total penalty of all weak constraints.

While the modelling language of ASP solvers contain variables, ASP is in essence a propositional formalism where variables are replaced by constant symbols in a preprocessing step called *grounding*, and a program with variables is effectively a short-hand for its ground version. For illustration, consider the following program, where variables start with an upper-case letter, and constant symbols are lower case:

$$p(a). \quad p(b).$$
$$q(X) \leftarrow p(X).$$

The last rule will be replaced by the two ground rules

$$q(a) \leftarrow p(a).$$
$$q(b) \leftarrow p(b).$$

3.2 The CLEVR Dataset

CLEVR (Johnson et al. 2017) is a dataset designed to test and diagnose the reasoning capabilities of VQA systems. It consists of pictures showing scenes with objects and questions related to them; there are about ten questions per image. The dataset was created with the goal of minimising biases in the data, since some VQA systems are suspected to exploit them to find answers instead of actually reasoning about the question and scene information (Johnson et al. 2017). The dataset can be downloaded from the project website.[3]

Each CLEVR image depicts a scene with objects in it. The objects differ by the values of their attributes, which are $size$ (big, small), $color$ (brown, blue, cyan, gray, green, purple, red, yellow), $material$ (metal, rubber), and $shape$ (cube, cylinder, sphere). Every image comes with a ground truth scene graph describing the scene depicted in it. Figure 1 contains three images from the CLEVR validation dataset with corresponding questions.

Questions in CLEVR are constructed using *functional programs* which represent the question in a structured format. These are symbolic templates for a question which are instantiated with the corresponding values. For each such question template, there are one or more natural language sentences to which they are mapped. For illustration, the question "How many large things are either cyan metallic cylinders or yellow blocks?" from Fig. 1 can be represented by the functional program shown in Fig. 2. There, function *scene()* returns the set of objects of the scene, the filter functions restrict a set of objects to subsets with respective properties, *union()* yields the union of two sets, and *count()* finally returns the number of elements of a set. A detailed description of functional programs in CLEVR can be found in the documentation of the dataset (Johnson et al. 2017).

[3] https://cs.stanford.edu/people/jcjohns/clevr/.

4 VQA with ASP and Confidence-Thresholds

Before going into detail, we present a bird's eye view of our approach to VQA for the CLEVR dataset, which consists of the following elements:

1. **Object detection**: we train neural networks for bounding-box prediction and object classification of the CLEVR scenes;
2. **Confidence thresholds**: we determine a threshold for network predictions that we consider to be of high confidence by statistical analysis on the distribution of prediction values of the neural networks;
3. **ASP encoding**: we translate CLEVR functional programs that represent questions as well as network predictions that pass confidence thresholds into ASP programs and use an ASP solver to compute the answers.

4.1 Object Detection

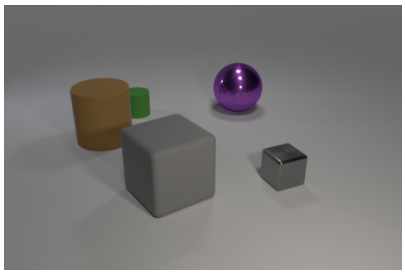
We use YOLOv3 (Redmon and Farhadi 2018) for bounding-box prediction and object classification, and assume the object detector's output is a matrix whose rows correspond to the bounding-box predictions in the input picture. More specifically, we assume each bounding-box prediction to be a vector of the form $(c_1, ..., c_n, x_1, y_1, x_2, y_2)$, where the pairs (x_1, y_1) and (x_2, y_2) give the top-left and bottom-right corner point of the bounding box, respectively. Furthermore, $c_1, ..., c_n$ are *class confidence scores* with $c_i \in [0, 1]$ for $1 \leq i \leq n$. As customary, higher confidence scores represent higher confidence of a correct prediction. Each c_i represents the score for a specific combination of object attributes $size$, $color$, $material$ and $shape$ and their respective values; we call this combination the *object class* of position i. For any object class c, let $\bar{c}$ be the list $size, shape, material, color$ of its attribute values. For example, assume c is the object class "large red metallic cylinder", then $\bar{c}$ = $\mathrm{large}, \mathrm{cylinder}, \mathrm{metallic}, \mathrm{red}$. Note that there are n = 96 object classes in CLEVR.

Every row of the prediction matrix has also its own bounding-box confidence score. The number of bounding-box predictions of the object detection system depends on the *bounding-box threshold*, which is a hyper-parameter used to filter out rows with a low confidence score. For example, if this threshold is set to 0.5, then all predictions with confidence score below 0.5 are discarded.

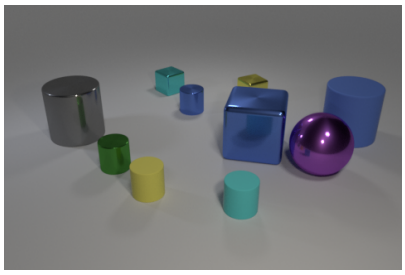
4.2 Confidence Thresholds

Given class confidence scores $c_1, ..., c_n$ from an object detection prediction, we would like to focus on classifications that have reasonable high confidence and discard ones with low confidence for the subsequent reasoning process. Using a fixed threshold hardly achieves this, since it does not take the distribution of confidence scores in the application area (or validation data for experiments) into consideration. Our approach solves this problem by fixing the threshold based on the mean and the standard deviation of prediction scores.

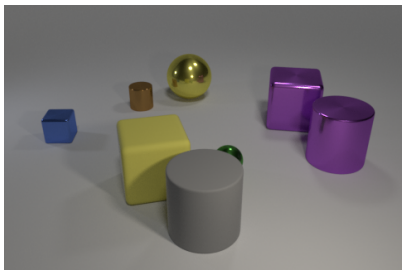
More formally, given a list of prediction matrices $\boldsymbol{X}^1, ..., \boldsymbol{X}^m$, where any $\boldsymbol{X}^{\boldsymbol{i}}$ is of dimension $N^i \times M$, we



Q: Is there a big brown object of the same shape as the green thing?
A: Yes



Q: How many large things are either cyan metallic cylinders or yellow blocks?
A: 0



Q: The tiny shiny cylinder has what color?
A: Brown

Figure 1: Three scenes and question-answer pairs from the CLEVR validation set.

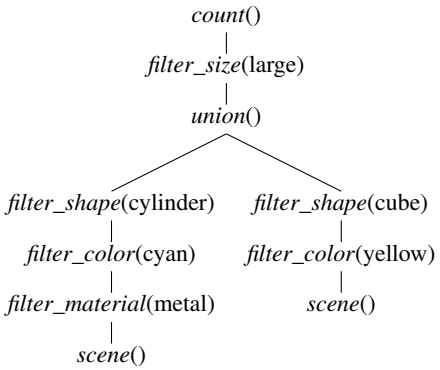


Figure 2: CLEVR functional program representing the question: "How many large things are either cyan metallic cylinders or yellow blocks?".

compute the mean μ and standard deviation σ for the maximum class confidence scores:

$$\mu = \frac{1}{\sum_{k=1}^{m} N^k} \sum_{k=1}^{m} \sum_{i=1}^{N^k} \max_{1 \le j \le n} (\boldsymbol{X}_{i,j}^k) \tag{2}$$

$$\sigma = \sqrt{\frac{1}{\sum_{k=1}^{m} N^k} \sum_{k=1}^{m} \sum_{i=1}^{N^k} (\max_{1 \le j \le n} (\boldsymbol{X}_{i,j}^k) - \mu)^2} \tag{3}$$

We suggest computing these values on the validation dataset used in training the object detector. Then, we define the *confidence threshold* θ that determines what is considered a confident class prediction as follows:

$$\theta = max(\mu - \alpha \cdot \sigma, 0) \tag{4}$$

We consider class predictions as sufficiently confident if their confidence score is not lower than the mean minus α many standard deviations. The value for α in Eqn. (4) is a parameter that must be provided, and it should depend on how well the network is trained. We observed that bigger values can be beneficial if the network is better trained even though the improvement might not be all too great. For poorly trained models, a big alpha value does not lead to improvements, and a smaller value is sufficient, as the standard deviation is high anyway.

4.3 ASP Encoding

To solve VQA tasks, we rely on ASP to infer the right answer given the neural network output and a confidence threshold. We outline the details in the following.

Question encoding. The first step of our approach is to translate the functional program that represents a natural language question into an ASP *fact representation*. We illustrate this for the question "How many large things are either cyan metallic cylinders or yellow blocks?" from Section 3.2. The functional program for this question from Fig. 2 is encoded by the following ASP facts:

$$\begin{aligned} end(8)\,. \\ count(8,7)\,. \\ filter\_large(7,6)\,. \\ union(6,3,5)\,. \\ filter\_cylinder(3,2)\,. \quad filter\_cube(5,4)\,. \\ filter\_cyan(2,1)\,. \quad filter\_yellow(4,0)\,. \\ filter\_metal(1,0)\,. \\ scene(0)\,. \end{aligned}$$

It should be intuitively clear how the structure of the functional program is encoded using indices that refer to output (first arguments) and input (remaining arguments) of the respective basic functions.

Scene encoding. Let $\boldsymbol{X}$ be a prediction matrix, θ be a confidence threshold that was determined as described in Section 4.2, and k, $1 \le k \le 96$, be a further integer parameter of the translation. Recall that the output of the basic CLEVR function $scene()$ corresponds to the objects detected in the scene which in turn correspond to the individual rows of $\boldsymbol{X}$.

For a row $\boldsymbol{X}_i$ with confidence class scores $c_1, ..., c_n$, set C_i contains every object class with score greater or equal than θ. If no such class exists, C_i contains the k object classes with highest class confidence scores. Intuitively, k is a fall-back parameter that ensures that some classes are selected in case all scores are low.

For every row $\boldsymbol{X}_i$ with bounding-box corners (x_1, y_1) and (x_2, y_2), as well as $C_i = \{c_1, \ldots c_l\}$, we construct a choice

rule of form

$$\{obj(O, i, \bar{c}_1, x_1, y_1, x_2, y_2); \\ \vdots \\ obj(O, i, \bar{c}_l, x_1, y_1, x_2, y_2)\} = 1 \leftarrow scene(O).$$

Every object with sufficiently high confidence score will thus be considered for computing the final answer in a non-deterministic way. Furthermore, for every $c \in C_i$, we add the weak constraint

$$\leftsquigarrow obj(O, i, \bar{c}, x_1, y_1, x_2, y_2)\ [w_c]$$

where the weight w_c is defined as $\lfloor 1000 - s \cdot 1000 \rfloor$, and s is the class confidence score for c in $\boldsymbol{X}_i$. This approach is inspired by the NeurASP implementation. It achieves that object selections are penalised by a weight which corresponds to the object's class confidence score. Resulting answer sets can thus be ordered according to the total confidence of the involved object predictions. We refer to this encoding as *non-deterministic scene encoding*, but we also consider the simpler special case of a *deterministic scene encoding* where each C_i is defined to contain only the single object class with the highest confidence score.

Encoding of the basic CLEVR functions. We next present encodings for the remaining CLEVR functions.

Filter rules: The CLEVR filter rules restrict sets of objects. We only present the rule for *filter_color*(yellow), the rules for the other colours, materials, and shapes are defined analogously:

$$obj(T, I, \ldots, \text{yellow}, \ldots) \leftarrow filter\_yellow(T, T_1), \\ obj(T_1, I, \ldots, \text{yellow}, \ldots).$$

Variables T, resp., T_1 are used to indicate output, resp., input, references, and I represents the object identifier. We omit arguments that are not relevant for the particular filter functions.

Count rule: Function $count()$ returns the number of elements of a given set. We encode it as follows:

$$int(T, V) \leftarrow count(T, T_1),\ \#count\{I : obj(T_1, I, ...)\} = V.$$

Here, $\#count$ is an ASP *aggregate function* that computes the numbers of object identifiers referenced by variable T_1.

Rules for set operations: The two set operation functions in CLEVR are intersection and union. We present respective ASP rules for each of them:

$$obj(T, I, \ldots) \leftarrow and(T, T_1, T_2),\ obj(T_1, I, \ldots), \\ obj(T_2, I, \ldots).$$
$$obj(T, I, \ldots) \leftarrow or(T, T_1, T_2),\ obj(T_1, I, \ldots).$$
$$obj(T, I, \ldots) \leftarrow or(T, T_1, T_2),\ obj(T_2, I, \ldots).$$

Uniqueness constraint: The CLEVR function $unique()$ is used to assert that there is exactly one input object, and, if this is the case, it is propagated to the output. We encode this in ASP using one rule for propagation and one constraint to eliminate any answer set if the uniqueness assumption is violated:

$$\leftarrow unique(T, T_1),\ obj(T_1, I, \ldots),\ obj(T_1, I', \ldots),\ I \neq I'.$$
$$obj(T, \ldots) \leftarrow unique(T, T_1),\ obj(T_1, \ldots).$$

Spatial-relation rules: A number of CLEVR functions allows to determine objects that are in a certain spatial relation with another object. We present the rule that allows to identify all objects that are left relative to a given reference, the rules for right, front, and behind, are defined analogously:

$$obj(T, I, \ldots) \leftarrow relate\_left(T, T_1, T_2), I \neq I', X_1 < X_1', \\ obj(T_1, I, \ldots, X_1, \ldots), obj(T_2, I', \ldots, X_1', \ldots).$$

Exist rule: The $exist()$ rule in CLEVR returns true if the referenced set of objects is not empty. Respective ASP rules look as follows:

$$bool(T, \text{true}) \leftarrow exist(T, T_1),\ obj(T_1, \ldots).$$
$$bool(T, \text{false}) \leftarrow exist(T, T_1),\ not\ bool(T, \text{true}).$$

Query rules: Query functions allow to return an attribute value of a referenced object. We present a rule to query for the size of an object, the rules for colour, material, and shape look similar:

$$size(T, Size) \leftarrow query\_size(T, T_1), obj(T_1, \ldots, Size, \ldots).$$

Same-attribute-relation rules: Similar to the spatial-relation functions, same-attribute-relation rules allow to select sets of objects if they agree on a specified attribute with a specified reference object. We illustrate the ASP encoding for the size attribute, the ones for colour, material and shape are defined with the necessary changes:

$$obj(T, I, \ldots) \leftarrow same\_size(T, T_1, T_2), \\ obj(T_1, I, \ldots, Size, \ldots), obj(T_2, I', \ldots, Size, \ldots), I \neq I'.$$

Integer-comparison rules: CLEVR supports the common relations for comparing integers like "equals", "less-than", and "greater-than". We present the ASP encoding for "equals":

$$bool(T, \text{true}) \leftarrow equal\_integer(T, T_1, T_2),\ int(T_1, V), \\ int(T_2, V).$$
$$bool(T, \text{false}) \leftarrow equal\_integer(T, T_1, T_2),\ not\ bool(T, \text{true}).$$

Attribute-comparison rules: To check if two objects have the same attributes, like size, colour, material, or shape, CLEVR provides attribute-comparisons rules. The one for size can be represented in ASP as follows, the others are defined analogously:

$$bool(T, \text{true}) \leftarrow equal\_size(T, T_1, T_2),\ size(T_1, V), \\ size(T_2, V).$$
$$bool(T, \text{false}) \leftarrow equal\_size(T, T_1, T_2),\ not\ bool(V, \text{true}).$$

In addition to the rules above, we also use rules to derive the $ans/1$ atom that extracts the final answer for the encoded CLEVR question from the output of the basic function at the root of the computation:

$$ans(V) \leftarrow end(T), size(T, V).$$
$$ans(V) \leftarrow end(T), color(T, V).$$
$$ans(V) \leftarrow end(T), material(T, V).$$
$$ans(V) \leftarrow end(T), shape(T, V).$$
$$ans(V) \leftarrow end(T), bool(T, V).$$
$$ans(V) \leftarrow end(T), int(T, V).$$
$$\leftarrow not\ ans(\_).$$

The last constraint enforces that a least one answer is derived.

Putting all together, to find an answer to a CLEVR question, we translate the corresponding functional program into its fact representation and join it with the rules presented above. Each answer set then corresponds to a CLEVR answer that is founded in a particular choice for scene objects. If the deterministic scene encoding is used, there will be at most one answer set, for the non-deterministic encoding, there can be multiple ones. No answer set may exist due to imperfect object recognition.

5 Experiments on the CLEVR Dataset

Recall that the parameters of our approach are (i) the bounding-box threshold for object detection, (ii) α for computing the confidence threshold as distance from the mean in terms of standard deviations, which is relevant for the non-deterministic scene encoding, and (iii) k as a fall-back parameter for object-class selection. We experimentally evaluated our approach on the CLEVR dataset to study the effects of different parameter settings. In particular, we study

- the effects of different bounding-box thresholds and training epochs for object detection,
- how the deterministic scene encoding compares to the non-deterministic one for question answering, and
- the run-time performance of our approach in comparison with the related VQA system NeurASP.

For computing the non-deterministic scene encodings, we set $\alpha = 1$ and $k = 2$.[4]

5.1 Object-Detection Evaluation

For object detection, we used an open source implementation of YOLOv3 (Redmon and Farhadi 2018).[5] The system was trained on 4000 CLEVR images with bounding box annotations, as suggested in related work (Yi et al. 2018). We used models trained for 25, 50 and 200 epochs, respectively, to obtain different levels of training for the neural networks. Regarding the bounding-box thresholds, we considered two settings, namely 0.25 and 0.50.

Table 1 summarises the results of the evaluation of how the differently well-trained networks perform for detecting the objects in the CLEVR scenes. We report on precision and recall, which are defined as usual in terms of true positives (TP), false positives (FP), and false negatives (FN). A TP is a prediction that indeed contains an object class with high confidence that is correct w.r.t. the scene annotations in CLEVR. A *FP* is a prediction that does not contain any correct object class with sufficiently high confidence. A *FN* is an object that exists according to the scene annotations, but there is no prediction with a corresponding object class of high confidence. Note that high confidence refers to above the confidence threshold for the non-deterministic setting and to the highest confidence score for the deterministic one.

As expected, our results show that the total number of FP and FN decreases for the better trained YOLOv3 models. Naturally, a low bounding-box threshold yields more FP detections, while the number of FN decreases. Setting the bounding-box threshold to a higher value usually leads to fewer FP but also more FN.

5.2 Question-Answering Evaluation

We used the ASP solver `clingo`[6] to compute answer sets. Table 2 sheds light on the impact of the training level and bounding-box thresholds of the models on question answering for the deterministic and non-deterministic scene encodings. Our system yields either correct, incorrect, or no answers to the CLEVR questions, and we report respective rates. The non-deterministic scene encoding outperformed the deterministic approach for all settings of training epochs and bounding-box thresholds. This suggests that the improvements in precision and recall seen in Table 1 for the non-deterministic encoding in comparison with the deterministic case carries over to question answering. It thus seems beneficial to consider more than one prediction of the object detection system when in doubt.

5.3 Comparison with NeurASP

As discussed in the related work section, NeurASP is closely related to our approach. There, neural network outputs are interpreted as probability distributions that are extended from atoms to the models of the program. A potential drawback is that calculating probabilities for models involves model counting which is an expensive operation. While NeurASP also embodies the idea of non-determinism for object classifications, it does not incorporate a mechanism to restrict object classes to ones with high confidence like in our approach.

The choice rules in NeurASP always contain all 96 CLEVR object classes, and their probabilities come from the YOLO network. We also consider a setup for NeurASP where only the best prediction is considered, while the probabilities of all other atoms are set to zero. In principle, our approach resembles the one of NeurASP for our ASP encoding by using weak constraints instead of probabilities. However, run-time performance and memory consumption are quite different. We could not run NeurASP on the entire set of questions due to memory problems and thus restricted our analysis to a smaller random sample. Table 3 summaries a comparison of our approach and NeurASP on 15000 CLEVR questions. NeurASP outperforms our approach in terms of correct answers, because it does not restrict the number of atoms for the choice rules as we do. However, this comes at a price as run-times are much longer, which can be explained by the inflation of the search space due to the unrestricted choice rules. Overall, the experiments further support our belief that non-determinism is useful for neuro-symbolic VQA systems and suitable mechanisms to restrict it do reasonable choices allows for more efficient implementations.

[4] All experiments were carried out on an Ubuntu (20.04.3 LTS) system with a 3.60GHz Intel CPU, 16GiB of RAM, and an NVIDIA GeForce GTX 1080 GPU with 8GB of memory installed.

[5] https://github.com/eriklindernoren/PyTorch-YOLOv3.

[6] https://potassco.org/clingo/.

	Training Epochs/Bounding-Box Threshold					
	25/0.25	50/0.25	200/0.25	25/0.50	50/0.50	200/0.50
Deterministic Scene Encoding						
true positives	69347	93841	96576	43240	86703	95442
false positives	13433	1651	406	1260	629	175
false negatives	28011	3517	782	54118	10655	1916
recall	71.23%	96.39%	99.20%	44.41%	89.06%	98.03%
precision	83.77%	98.27%	99.58%	97.17%	99.28%	99.82%
Non-Deterministic Scene Encoding						
true positives	86986	95645	96978	85396	95023	96666
false positives	10455	2061	604	10923	1566	389
false negatives	10372	1713	380	11962	2335	692
recall	89.35%	98.24%	99.61%	87.71%	97.60%	99.29%
precision	89.27%	97.89%	99.38%	88.66%	98.38%	99.60%

Table 1: Precision and recall for object detection on CLEVR scenes.

	Training Epochs/Bounding-Box Threshold					
	25/0.25	50/0.25	200/0.25	25/0.50	50/0.50	200/0.50
Deterministic Scene Encoding						
correct	65.11%	93.01%	96.90%	42.88%	82.97%	95.29%
wrong	17.04%	3.05%	1.08%	28.63%	8.53%	2.04%
no answer	17.85%	3.94%	2.02%	28.49%	8.50%	2.67%
Non-Deterministic Scene Encoding						
correct	80.27%	94.54%	97.23%	78.69%	93.95%	96.93%
wrong	12.64%	2.73%	1.08%	13.61%	3.12%	1.27%
no answer	7.09%	2.73%	1.70%	7.7%	2.93%	1.80%

Table 2: Results for question answering based on deterministic and non-deterministic scene encodings.

6 Conclusion

We introduced a neuro-symbolic VQA system for the CLEVR dataset based on ASP. We presented a translation from CLEVR questions in the form of functional programs to ASP rules, where we propose a non-deterministic and a deterministic approach to encode object-level scene information. We have shown that our system yields very good results on the CLEVR validation dataset. Question answering based on non-deterministic scene encodings has achieved better results compared to the deterministic approach, especially if the neural networks for object classification are trained poorly. This is important to ensure robustness of VQA systems if network predictions are not perfect.

For future work, it would be interesting to to learn the confidence threshold instead of relying on statistical measures. We also plan to apply our approach to other datasets, especially ones that do not use synthetic scenes, and to extend our system with natural-language processing.

Acknowledgments

This work was supported by funding from the Bosch Center for Artificial Intelligence at Renningen, Germany.

References

Antol, S.; Agrawal, A.; Lu, J.; Mitchell, M.; Batra, D.; Zitnick, C. L.; and Parikh, D. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, 2425–2433.

Basu, K.; Shakerin, F.; and Gupta, G. 2020. Aqua: Asp-based visual question answering. In *International Symposium on Practical Aspects of Declarative Languages*, 57–72. Springer.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM*, 54(12): 92–103.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gelfond, M.; and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4): 365–385.

Jabri, A.; Joulin, A.; and Van Der Maaten, L. 2016. Revisiting visual question answering baselines. In *European Conference on Computer Vision*, 727–739. Springer.

Johnson, J.; Hariharan, B.; van der Maaten, L.; Fei-Fei, L.; Zitnick, C. L.; and Girshick, R. B. 2017. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR 2017)*, 1988–1997. IEEE Computer Society.

Lu, J.; Yang, J.; Batra, D.; and Parikh, D. 2016. Hierarchical question-image co-attention for visual question answering.

	Training Epochs/Bounding-Box Threshold					
	25/0.25	50/0.25	200/0.25	25/0.50	50/0.50	200/0.50
NeurASP						
correct	86.09%	96.74%	98.53%	84.87%	96.17%	98.20%
wrong	13.88%	3.21%	1.45%	15.09%	3.77%	1.77%
no answer	0.03%	0.05%	0.03%	0.04%	0.06%	0.03%
runtime (s)	9149	4375	4364	8750	4234	4694
NeurASP (only best prediction)						
correct	75.63%	95.78%	98.33%	53.12%	88.21%	96.87%
wrong	23.51%	4.14%	1.63%	38.17%	11.30%	3.06%
no answer	0.87%	0.08%	0.03%	8.71%	0.49%	0.07%
runtime (s)	3114	3628	3575	1245	3174	3577
Deterministic Scene Encoding						
correct	64.39%	92.93%	97.01%	42.33%	82.79%	95.05%
wrong	17.60%	3.06%	1.11%	29.55%	8.65%	2.21%
no answer	18.01%	4.01%	1.88%	28.11%	8.56%	2.73%
runtime (s)	84	89	87	81	89	85
Non-Deterministic Scene Encoding						
correct	79.94%	94.55%	97.40%	78.23%	93.87%	97.01%
wrong	12.86%	2.79%	1.07%	13.91%	3.16%	1.35%
no answer	7.20%	2.65%	1.53%	7.87%	2.97%	1.64%
runtime (s)	111	107	107	109	105	105

Table 3: A comparison of NeurASP and our approach on a random sample of 15000 CLEVR questions.

Advances in Neural Information Processing Systems, 29: 289–297.

Malinowski, M.; and Fritz, M. 2014. A multi-world approach to question answering about real-world scenes based on uncertain input. *Advances in Neural Information Processing Systems*, 27: 1682–1690.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and Raedt, L. D. 2018. DeepProbLog: Neural Probabilistic Logic Programming. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 3753–3763. Red Hook, NY, USA: Curran Associates Inc.

Mao, J.; Gan, C.; Kohli, P.; Tenenbaum, J. B.; and Wu, J. 2019. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations*.

Redmon, J.; and Farhadi, A. 2018. YOLOv3: An Incremental Improvement. *CoRR*, abs/1804.02767.

Ren, M.; Kiros, R.; and Zemel, R. 2015. Exploring models and data for image question answering. *Advances in Neural Information Processing Systems*, 28: 2953–2961.

Sampat, S. K.; Kumar, A.; Yang, Y.; and Baral, C. 2021. CLEVR_HYP: A Challenge Dataset and Baselines for Visual Question Answering with Hypothetical Actions over Images. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Van den Broeck, G. 2018. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 5502–5511. PMLR.

Yang, Z.; He, X.; Gao, J.; Deng, L.; and Smola, A. 2016. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 21–29.

Yang, Z.; Ishay, A.; and Lee, J. 2020. NeurASP: Embracing Neural Networks into Answer Set Programming. In *IJCAI*, 1755–1762.

Yi, K.; Wu, J.; Gan, C.; Torralba, A.; Kohli, P.; and Tenenbaum, J. 2018. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Zhu, Y.; Groth, O.; Bernstein, M.; and Fei-Fei, L. 2016. Visual7w: Grounded question answering in images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4995–5004.