Visual Question Answering with DeepProbLog

Jorrit Willaert[1]
[1]Catholic University of Leuven, Leuven – Belgium
e-mail: jorrit.willaert@student.kuleuven.be

***Abstract* – This report focusses on Visual Question Answering, where a Neuro Symbolic AI approach with a knowledge base is compared with a purely neural network based approach. From the experiments, it follows that DeepProbLog, the framework used for the Neuro Symbolic AI approach, is able to achieve the same accuracy as the pure neural network based approach with almost 200 times less iterations. Clearly, the training is much more targeted, but however, comes at a cost. The algebraic operators internal to DeepProbLog are extremely costly and hence the actual training time is considerably slower. Another drawback of DeepProbLog is that no easy speedups can be achieved, since the algebraic operators only work on CPU's, and hence cannot benefit from accelerators such as GPU's.**

***Keywords* – Neuro Symbolic AI, Visual Question Answering, DeepProbLog, Problog, Convolutional Neural Networks**

1 INTRODUCTION

The Neuro Symbolic AI field is interested in building a bridge between the robustness of probabilistic knowledge, with the well-known popularity and proven strengths of deep neural networks. DeepProbLog [1] offers this ability, by using both the strengths of neural networks (i.e. system 1, typical subconscious tasks such as visual recognition, the processing of languages, ...), along with the strengths of rule-based probabilistic systems (i.e. system 2, slow, sequential thinking such as the derivation of a proof).

This paper elaborates on an application that requires both systems to be used, namely Visual Question Answering. System 1 will be required in order to gain an understanding of the image under investigation, with in particular their shapes and colors. System 2, on the other hand, will use this extracted information for deriving certain properties of objects[1], or even for capturing the relations[2] between the objects.

2 ORGANIZATION OF THE PAPER

This paper will first provide some necessary background in Section 3 on certain types of datasets that are typically used for Visual Question Answering purposes, along with introducing some results of other researchers. Then, Section 4 will dive deeper in the different components constituting the overall system. A main focus of this paper is to outline the advantages of using a Neuro-Symbolic AI approach (offered by the DeepProbLog framework), compared to a purely neural network based approach. These experiments are listed in Section 5. Finally, the main findings of this paper are repeated in Section 6.

3 LITERATURE SURVEY

The application focuses on Visual Question Answering (VQA), for which huge datasets are present, along with very sophisticated methods. The best known dataset for VQA is CLEVR [2], which contains 100k images accompanied by one million questions. An example image is given in Figure 1, while example questions are:

- Are there an equal number of large things and metal spheres?
- What size is the cylinder that is left of the brown metal thing that is left of the big sphere?
- How many objects are either small cylinders or metal things?

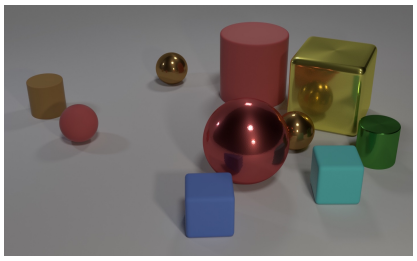

Fig. 1. A sample image from the CLEVR dataset [2]

Clearly, both system 1 and system 2 are actively used when answering these questions. One could wonder if neural networks alone could answer these questions without having an explicit system 2 encoding (i.e. the rule based knowledge base). Intuitively, it makes sense that if certain facts of the world are known[3], learning can proceed much more quickly[4]. Seen from an optimization viewpoint, errors made during prediction in this setup can be targeted exactly, which makes the optimization process more targeted as well, and hence more efficient. Finally, this paper also provides evidence for these statements, since in subsection 5.1, the comparison between a VQA implementation with DeepProbLog is made with a purely neural network based approach.

This paper is inspired on the CLEVR dataset, but uses however a much more simplified version. In essence, it is almost like the Sort-Of-CLEVR dataset [3]. This Sort-Of-CLEVR dataset contains images as in Figure 2, while asking questions such as:

[1]For example, finding the shape of the green object, or deriving if it is located on the left hand side of the image.

[2]Here, one could think of counting the number of circles in the image.

[3]Facts can be encoded, e.g. counting the number of spheres is simply a matter of detecting all the spheres in the image, after which a mathematical summation is a statement in the knowledge base.

[4]Not to say that learning might even be impossible if a lot of background knowledge is required.

- Non-relational questions: the shape, horizontal or vertical location of an object.
- Relational questions: shape of the closest/furthest object to the object under investigation, or the number of objects with the same shape.

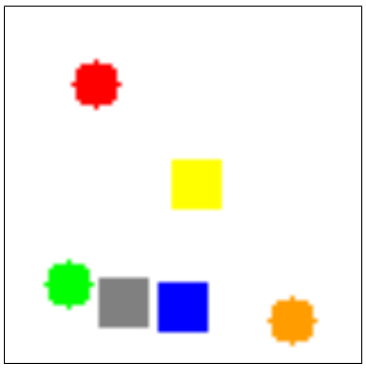

Fig. 2. A sample image from the CLEVR dataset [3]

As illustrated earlier, both system 1 and system 2 are required for these types of VQA's.

A. Santoro et al. conducted similar experiments with the Sort-of-CLEVR dataset as in this paper, where they were able to achieve an accuracy of 63% on relational questions with CNN's. In contrast, an accuracy of 94% for both relational and non-relational questions was achieved with CNN's, complemented with an RN. For them, augmenting the model with a relational module, such as an RN, turned out to be sufficient to overcome the hurdle of solving relational questions [3]. They were the only researchers who undertook experiments on a dataset resembling the one from this paper, and hence are the only reference point.

Finally, since this application uses DeepProbLog, quite some time was spent in digesting the DeepProbLog paper [1], along with understanding the examples provided in the code repository [4].

4 APPROACH

The implementation process involved three main parts:

1. Generation of data.
2. Linking the data and controlling the training process in pure Python code.
3. Creation of the logical part with DeepProbLog statements.

4.1 Generation of data

As mentioned in Section 3, the data used in this application is based on the Sort-Of-CLEVR dataset, with one extra simplification. Given that the logical part will have to decide whether an object is for example located on the left side of an image, the neural network will have to convey positional information to the logical part. Hence, each discrete position will have to be encoded by a possible outcome of the neural network. Therefore, objects may only be located at certain positions in a grid. In this paper, results on a grid of 2x2 and 6x6 are discussed.

The data generator that was used for the creation of the Sort-Of-CLEVR dataset has been modified in order to place objects in the mentioned grid positions [3]. An example of a generated image is given in Figure 3, where the difference with Figure 2 is the grid-layout.

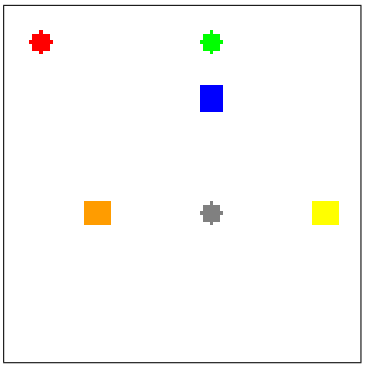

Fig. 3. A sample image from the dataset that has been used for this application

Each specified color will have an object located somewhere in the grid, of which the shape can be a square or a circle.

These images are accompanied with a question about a random object, which can be one of the following:

- Non-relational - What is the shape of this object?[5]
- Non-relational - Is this object located on the left side of the image?
- Non-relational - Is this object located on the bottom side of the image?
- Relational - How many objects have the same shape as this object?

These questions are encoded in a vector encoding, after which they are stored in a CSV file, along with the expected answers. A training and test dataset has been generated beforehand, in order to make the training process more efficient.

4.2 Controlling the training process

The overall training process is controlled via the Python API of DeepProbLog, along with general PyTorch implementations of the CNN's. First of all, CNN's are defined with PyTorch. A relatively simple network is used, where the input is given as a square RGB image of 100 pixels wide, which is transformed by the CNN into 72 output features for the 6x6 grid[6]. Each color that is present in the image has its accompanied CNN network, hence the 72 output features encode the possible positions of the object with that color, along with their shape, which can be either square or circular ($6 \cdot 6 \cdot 2 = 72$).

The final thing (besides the logical rule encodings) required before commencing the training process, are the data loaders. The most challenging part here is the transformation from the generated data to specific query mappings and their outcome.

4.3 Logical rule encodings

Once the CNN belonging to a specific color has determined the position and the shape of that object, logical rules can deduce whether this object is located on the left hand side of the image, on the bottom side, and how many objects have

[5]The shape will be either a square or a circle.

[6]For the 2x2 grid example, 8 output features are required.

the same shape. The logical rule program has been listed in Appendix 6.1.

5 EXPERIMENTS

The main focus of this paper is to outline the advantages of using a Neuro Symbolic AI approach (offered by the DeepProbLog framework), instead of a purely neural network based approach. Therefore, a neural network based approach had to be implemented. Not all details will be listed here, but the main idea is that the image has to be fused with the question, after which a prediction can be made[7]. The general structure of this network is given in Figure 4.

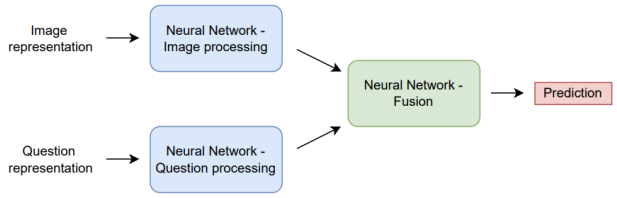

Fig. 4. Abstract representation of the purely neural network based architecture [5]

5.1 *COMPARISONS WITH PURE SYSTEM 1 APPROACHES*

1) Experiment - 2x2 grid: The loss curves of both the DeepProbLog approach, as well as the purely neural network based approach, are visualized respectively in Figure 5 and Figure 6.

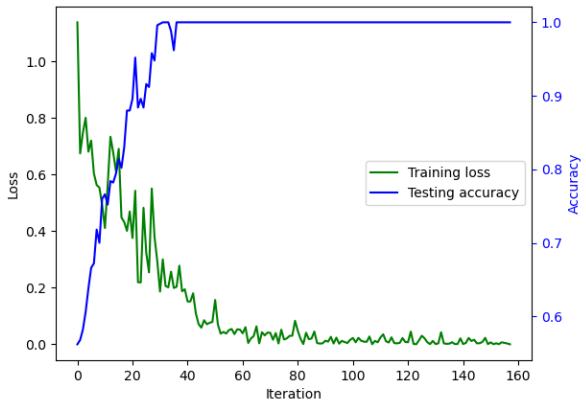

Fig. 5. DeepProbLog 2x2: Loss curve

An extremely important remark to be made is the difference between 'number of iterations' and 'number of epochs'. By the number of iterations, the number of forward and backward passes of a batch (with size 32) is meant, whereas the number of epochs denotes the number of times all the images of the training set are forward and backwardly passed. In this application, a training size of 10 000 was used, hence one epoch consists of 312.5 iterations.

From the loss curves, it is clear that both approaches seem

[7]Possible answers on the questions are: rectangle, circle, yes, no, 1, 2, 3, 4, 5 and 6. Hence, the neural network will have 10 output nodes.

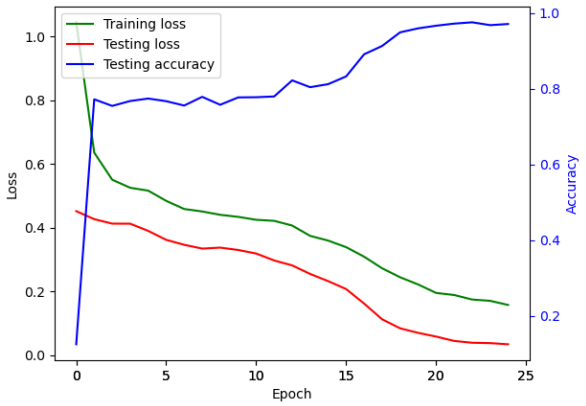

Fig. 6. Pure NN based approach 2x2: Loss curve

to converge to an accuracy of 100%. However, DeepProbLog only requires around 40 iterations, whereas the purely neural network based approach requires at least 7 800 iterations. This again demonstrates the value of Neuro Symbolic AI.

On the other hand, one has to consider the actual running times for those iterations. DeepProbLog takes around 10 minutes to finish its 160 iterations, while the purely neural network based approach only requires around 5 minutes to finish 7 800 iterations. Taking into consideration that the purely neural network based approach can be accelerated massively (by using GPU's), while DeepProbLog can't[8], it is clear that DeepProbLog trains much more targeted, but is computationally extremely heavy (at least for now).

2) Experiment - 6x6 grid: The loss curves for the 6x6 experiment of DeepProbLog and the neural network based approach are depicted in respectively Figure 7 and Figure 8.

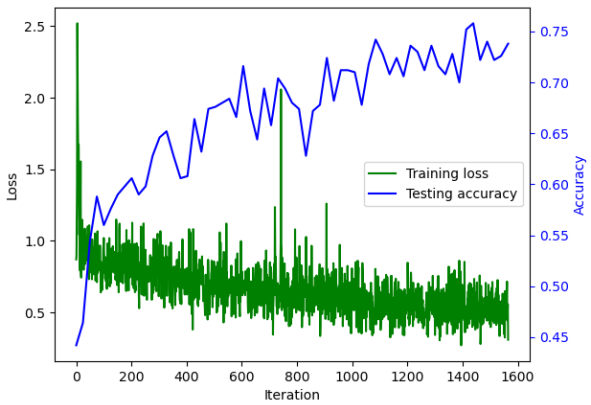

Fig. 7. DeepProbLog 6x6: Loss curve

In these experiments, the training time of the purely neural network approach was 20 minutes[9], while the training time of

[8]DeepProbLog offers the ability to send the CNN to a GPU for faster inference, however, the arithmetic operators (i.e. semirings) of DeepProbLog work on the CPU. These arithmetic operators possess by far the highest computational cost.

[9]For fair speed estimation, training has been conducted on the CPU.

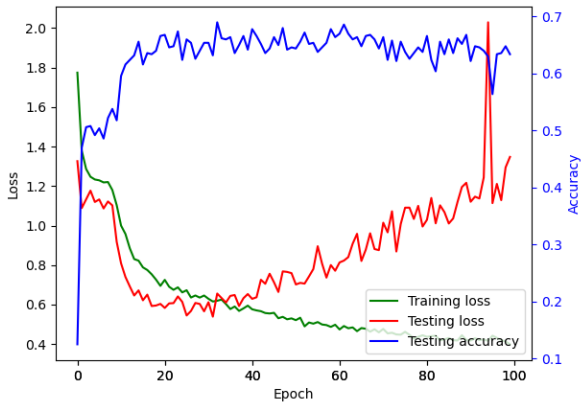

Fig. 8. Pure NN based approach 6x6: Loss curve

the DeepProbLog approach was a little under eight hours[10].

The most important observation here is that the purely neural network approach overfits quickly and also achieves a considerably lower accuracy (68% instead of almost 75% for DeepProbLog). Another important remark is the fact that both approaches are clearly not longer able to attain 100% accuracy. However, if the DeepProbLog network could train longer, it would be able to converge somewhat further.

In Figure 9 and Figure 10, the confusion matrices are depicted in order to show where typical mistakes are made.

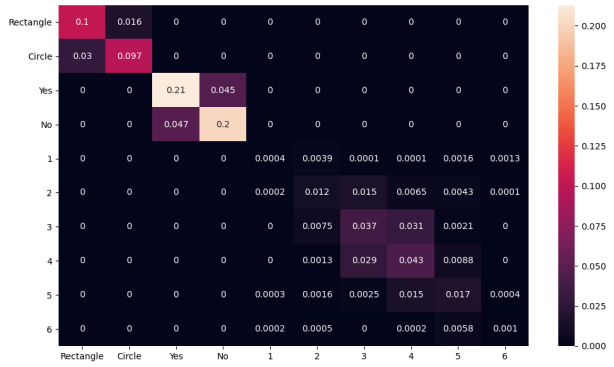

Fig. 9. The confusion matrix of the 6x6 dataset from DeepProbLog

DeepProbLog naturally will not make any 'nonsense' mistakes such as answering 'yes' to the question 'What shape has the object under investigation?', since the possible correct answers are encoded in the program. However, the purely neural network based approach has learned perfectly to link the possible answers to the questions.

Another observation is that DeepProbLog is much better in answering the questions 'What shape has the object under investigation?' and 'Is the object located on the left hand side (or on the bottom side) of the image?' then a purely neural network approach is. This makes sense, since DeepProbLog can use its knowledge base to derive these properties, once the

[10]However, it should be mentioned that approximately half of the time was spent on calculating the accuracy each time, since the whole testing set had to be forwarded through the network, while for the training iterations this is only one batch.

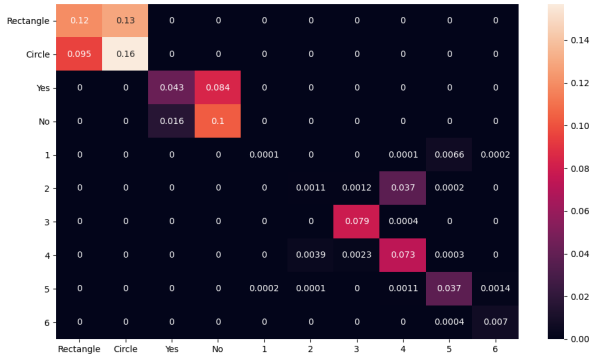

Fig. 10. The confusion matrix of the 6x6 dataset from the purely neural network approach

position (and shape) of the given object is determined. The observation that a pure neural network based approach has a much harder time to distinguish these cases, was also observed by A. Santoro et al. [3], where they achieved an accuracy of 52.3% on these questions with a pure neural network based approach. DeepProbLog was not able to achieve the accuracy of 94% that these researchers achieved, which may be due to the inherent training cost of the algebraic operators, as well as due to less resources and less hyperparameter tuning, among many other unknown variables.

Regarding the relational question: 'How many objects have the same shape as the object under investigation?', a lot more confusion in both approaches is going on. DeepProbLog will typically be able to come close to the correct answer, but may make some 'off-by-one' mistakes[11]. The purely neural network based approach does also in this perspective a little worse. It is also plausible that in this approach, the neural network may have observed that due to probabilistic reasons, there are likely three or four equal objects (inclusive the one under investigation), and hence prefers such an answer.

6 CONCLUSIONS

The strengths of the Neuro Symbolic AI field have been demonstrated in the context of Visual Question Answering. By using DeepProbLog, the chosen framework for the Neuro Symbolic AI task, it became clear that almost 200 times less iterations are required to achieve the same accuracy as a purely neural network based approach. However, due to the costly algebraic operators, the total training time of the DeepProbLog approach was considerably slower, compared to the purely neural network based approach.

Important to notice, however, is that DeepProbLog performs a lot better on the non-relational questions. This because the knowledge base can derive these properties much more accurately, while a purely neural network based approach has more difficulties with such derivations.

Hence, a lot of value can be seen in Neuro Symbolic AI approaches, despite the costly algebraic operators. Especially because for tasks where the knowledge base is much larger, these approaches can make the difference between being able to learn a certain task or not. Over time, speedups for these

[11]I.e. one CNN that is mispredicting the shape of its object.

algebraic operators could probably be developed, which opens the road to even more applications.

REFERENCES

[1] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, L. De Raedt, K. U. Leuven, "Neural Probabilistic Logic Programming in DeepProbLog", , jul 2019, doi:10.48550/arxiv.1907.08194, URL: https://arxiv.org/abs/1907.08194v2, 1907.08194.

[2] J. Johnson, L. Fei-Fei, B. Hariharan, C. L. Zitnick, L. Van Der Maaten, R. Girshick, "CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning", *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 1988–1997, 2017, doi: 10.1109/CVPR.2017.215, 1612.06890.

[3] K. Heecheol, "kimhc6028/relational-networks: Pytorch implementation of "A simple neural network module for relational reasoning" (Relational Networks)", URL: https://github.com/kimhc6028/relational-networks.

[4] R. Manhaeve, "ML-KULeuven/deepproblog: DeepProbLog is an extension of ProbLog that integrates Probabilistic Logic Programming with deep learning by introducing the neural predicate.", URL: https://github.com/ML-KULeuven/deepproblog.

[5] C. Theodoropoulos, "Information Retrieval and Search Engines [H02C8b] - Project Visual Question Answering", *Toledo*, pp. 1–5, 2022.

APPENDIX

6.1 *Logical rule encodings*

Note: it has been tried to use cuts (with the 'cut' library), given that the neural network is seemingly invoked again if the first of a dual predicate fails. However, this turned out to not yield any performance improvement and has been abandoned.

```
% Grid = 6 by 6

width(6).
size(N) :-
    width(Width),
    N is Width * Width.
states(N) :-
    size(Size),
    N is Size * 2.

nn(cnn_red,[X],Y_red, [0, 1, 2, 3, 4, 5, 6, 7,
 ↪  8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
 ↪  19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
 ↪  30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
 ↪  41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
 ↪  52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
 ↪  63, 64, 65, 66, 67, 68, 69, 70, 71]) ::
 ↪  detect_state(red, X, Y_red).
nn(cnn_green,[X], Y_green, [0, 1, 2, 3, 4, 5,
 ↪  6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
 ↪  18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
 ↪  29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
 ↪  40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
 ↪  51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
 ↪  62, 63, 64, 65, 66, 67, 68, 69, 70, 71]) ::
 ↪  detect_state(green, X, Y_green).
nn(cnn_blue,[X],Y_blue, [0, 1, 2, 3, 4, 5, 6,
 ↪  7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
 ↪  18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
 ↪  29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
 ↪  40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
 ↪  51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
 ↪  62, 63, 64, 65, 66, 67, 68, 69, 70, 71]) ::
 ↪  detect_state(blue, X, Y_blue).
nn(cnn_orange,[X], Y_orange, [0, 1, 2, 3, 4, 5,
 ↪  6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
 ↪  18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
 ↪  29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
 ↪  40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
 ↪  51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
 ↪  62, 63, 64, 65, 66, 67, 68, 69, 70, 71]) ::
 ↪  detect_state(orange, X, Y_orange).
nn(cnn_grey,[X], Y_grey, [0, 1, 2, 3, 4, 5, 6,
 ↪  7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
 ↪  18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
 ↪  29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
 ↪  40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
 ↪  51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
 ↪  62, 63, 64, 65, 66, 67, 68, 69, 70, 71]) ::
 ↪  detect_state(grey, X, Y_grey).
nn(cnn_yellow,[X], Y_yellow, [0, 1, 2, 3, 4, 5,
 ↪  6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
 ↪  18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
 ↪  29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
 ↪  40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
 ↪  51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
 ↪  62, 63, 64, 65, 66, 67, 68, 69, 70, 71]) ::
 ↪  detect_state(yellow, X, Y_yellow).

detect_same_states(Shape, Img, Count) :-
    check_state(red, Shape, Img, Return_red),
    check_state(green, Shape, Img,
     ↪  Return_green),
    check_state(blue, Shape, Img, Return_blue),
    check_state(orange, Shape, Img,
     ↪  Return_orange),
    check_state(grey, Shape, Img, Return_grey),
    check_state(yellow, Shape, Img,
     ↪  Return_yellow),
    Count is Return_red + Return_green +
     ↪  Return_blue + Return_orange +
     ↪  Return_grey + Return_yellow.

check_state(Color, Shape, Img, Return) :-
    detect_state(Color, Img, Y_color),
    to_shape(Y_color, Shape_color),
    Shape =:= Shape_color,
```

```
    Return = 1.
check_state(Color, Shape, Img, Return) :-
    detect_state(Color, Img, Y_color),
    to_shape(Y_color, Shape_color),
    Shape =\= Shape_color,
    Return = 0.

% NON-BINARY QUESTIONS
% Question about the shape
shape(Img, Color, Rectangle) :-
    shape_is_rectangle(Img, Color),
    Rectangle = 1.
shape(Img, Color, Rectangle) :-
    \+ shape_is_rectangle(Img, Color),
    Rectangle = 0.

shape_is_rectangle(Img, Color) :-
    detect_state(Color, Img, Y_color),
    size(Size),
    Y_color < Size.

% Question about the horizontal side
horizontal_side(Img, Color, Left) :-
    position_is_left(Img, Color),
    Left = 1.
horizontal_side(Img, Color, Left) :-
    \+ position_is_left(Img, Color),
    Left = 0.

position_is_left(Img, Color) :-
    detect_state(Color, Img, Y_color),
    position(Y_color, Pos),
    left_side(Pos).

% Question about the vertical side
vertical_side(Img, Color, Bottom) :-
    position_is_bottom(Img, Color),
    Bottom = 1.
vertical_side(Img, Color, Bottom) :-
    \+ position_is_bottom(Img, Color),
    Bottom = 0.

position_is_bottom(Img, Color) :-
    detect_state(Color, Img, Y_color),
    position(Y_color, Pos),
    bottom_side(Pos).


% BINARY QUESTION
% Question about the number of same shapes like
 ↪  that one
number_of_shapes(Img, Color, Count) :-
    detect_state(Color, Img, Y_red),
    to_shape(Y_red, Shape),
    detect_same_states(Shape, Img, Count).

to_shape(Y_color, Shape) :-
    size(Size),
    Y_color < Size,
    Shape = 0. % Shape = 0 --> Rectangle
to_shape(Y_color, Shape) :-
    size(Size),
    Y_color >= Size,
    Shape = 1. % Shape = 1 --> Circle


position(Out, Pos) :-
    size(Size),
    Pos is Out mod Size.

left_side(Out) :-
    to_coor(Out, X, Y),
    width(Width),
    X < (Width // 2).

bottom_side(Out) :-
    to_coor(Out, X, Y),
    width(Width),
    Y >= (Width // 2). % Be careful. Y axis
     ↪  runs down!

to_coor(Out, X, Y) :-
    width(Width),
    X is Out mod Width,
    Y is Out // Width.
```