Supplementary Material

1. Examples of Question Graph Generation

Following are some examples of questions and their corresponding generated graphs. Every graph is given both in the format of a string sequence (as produced by the sequence-to-sequence model) and as the equivalent question graph. Each example is taken from a different data source. The examples are given from simplest graph (a single node) to larger graphs.

The special tokens used in the string sequences of the graphs are:

<*NewNode*>: Indicates a new node, followed by the node index and the object class (c in the graph figures) strings. The following strings relate to fields of this node until the next <*NewNode*>.

<p>: Indicates a required property, followed by the required property.

<F>: Indicates a queried property (f in the graph figures), followed by the queried property type.

<N>: Indicates a queried property of a set of objects (g in the graph figures), followed by the queried property type.

<*rd*>: Indicates a relation with a child node, followed by the required relation and the index of the corresponding child node.

<*rp*>: Indicates a relation with a parent node, followed by the required relation and the index of the corresponding parent node.

<*nodeType*>: Indicates the node type field, followed by the node type ('regular' or 'superNode'). Note that the default value is 'regular', so the field is provided only for super nodes.

<*nodes*>: Indicates an included sub node, followed by the index of the included sub node. This field is only relevant and provided for super nodes.

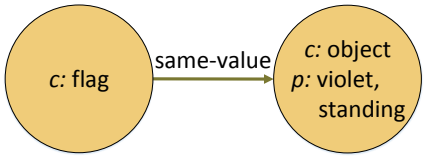
<*is_plural*>: Indicates whether the object is given in plural form, followed by the corresponding boolean value (used only for phrasing the 'full' answers). The default is *'false'*.

The graph symbols (in the diagrams): c: object class, p: property, f: queried property, g: queried property of a set of objects

- **VQA dataset:**
  The following example is taken from the VQA [1] validation set. The corresponding question graph is generated by the 'VG-Enhanced' question-to-graph model.
  Question:
  Where is the green frog?
  Graph String Sequence:
  <*NewNode*>, 1, frog, <*p*>, green, <*F*>, location
  Question Graph:

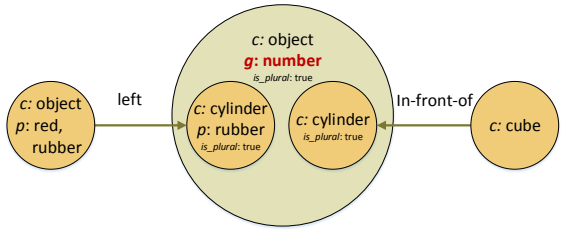
  Note: 'Where' was mapped to the location property; the question is mapped to a single node.

- **Extended CLEVR data:**
  The following example is based on a CLEVR question, modified by replacing CLEVR visual elements with corresponding ones, sampled from the 'Extended' set. The corresponding question graph is generated by the 'Extended-Enhanced' question-to-graph model.
  Question:
  Does the flag have the same value as the violet standing thing?
  Graph String Sequence:
  <*NewNode*>, 1, flag, <*rd*>, same value, 2, <*NewNode*>, 2, object, <*p*>, violet, <*p*>, standing, <*rp*>, same value, 1
  Question Graph:



  Note:
  - To generate a graph, the question must be syntactically correct, but it is not required to have a visual sense or even to have a reasonable meaning.
  - The relation in the question graph includes property comparison; this type of relations (for properties with corresponding visual estimators) is handled by the answering procedure.

- **CLEVR data:**
  The following example is taken from the CLEVR validation set [5]. The corresponding question graph is generated by the 'None-Basic' question-to-graph model (which includes mapping of concepts to their synonyms).
  Question:
  What number of objects are matte cylinders that are on the left side of the red matte object or cylinders in front of the block?
  Graph String Sequence:
  <*NewNode*>, 1, object, <*p*>, red, <*p*>, rubber, <*rd*>, left, 2, <*NewNode*>, 2, cylinder, <*p*>, rubber, <*is_plural*>, true, <*rp*>, left, 1, <*NewNode*>, 3, cube, <*rd*>, front, 4, <*NewNode*>, 4, cylinder, <*is_plural*>, true, <*rp*>, front, 3, <*NewNode*>, 5, object, <*nodeType*>, superNode, <*is_plural*>, true, <*nodes*>, 2, <*nodes*>, 4, <*N*>, number
  Question Graph:



  Note: the large circle is a super node, which handles the 'OR' connective; 'matte' and 'block' were mapped to their corresponding basic terms: 'rubber' and 'cube', respectively.

2. Answering Procedure Algorithm

The answering procedure is a recursive procedure guided by the question graph to provide an answer. The input is the question graph and the image, and the output is the produced answer, possibly combined with elaborations and explanations. The procedure follows the graph nodes and edges, invokes the relevant basic procedures and integrates all the information to provide the answer. A schematic description of the procedure is given in Algorithm 1 below.

The first step is an instance segmentation, carried out by applying mask R-CNN [3] to the image. Next, a recursive function ($getGraphAnswer$) is invoked for node handling (starting at a root node of a subgraph, if available, and at an arbitrary node otherwise). It runs procedures from the set of basic procedures, which activate visual estimators to check the requirements (properties, relations), and fetch required information (queried property). The retrieved objects from the mask R-CNN that fulfill the requirements are paired with the corresponding question objects (a 'working memory' module is used to store and share this information), so that subsequent tests will be applied to the correct objects. The number of required objects is set according to quantifiers (*e.g.* 'all', 'three') or by the need to evaluate a property that depends on the entire object set (*e.g.* 'how many'). If a node is a 'sub node', i.e. included in a 'super node', all valid objects are added as optional objects for its corresponding 'super node'. Once all the node's object tests are completed (not including set requirements), the same function ($getGraphAnswer$) is invoked for the next node, determined by a DFS traversal. After the node and the following nodes in the recursion are tested and validated in the image, tests are applied, if needed, to verify the node's set requirements. Examples for such tests are counting the objects in the set and quantity comparisons. Note that tests of relations may include property comparisons, *e.g. 'same size'*. The recursive process is terminated either when the full graph is grounded in the image as required or when no alternatives are left to check, which may happen after a partial check (*e.g.* no objects of a required class were detected).

The last stage is generating the final answer. Partial answers are provided by the basic procedures invoked during the graph traversal, in the form of text sequences. The final answer is produced from these partial answers, mostly based on selecting one of them. The provided answers can be configured to be short or full. Short answers include only the required information, which can be "yes", "no", object class, a particular property, a number, or reporting a failure such as "unknown class: 'logo'". The full answers are based on the partial answers, which use fixed, short templates, for example "Yes, there is a $<c_1>$ $<r>$ a $<c_2>$", where $<r>$ is the relation, and $<c_1>$ and $<c_2>$ are the participating object classes. Additional explanations and elaborations are added (if relevant) to the full answer based on available intermediate results, as well as additional processing for checking alternatives.

An important aspect of the answering procedure is that it depends *only on the abstract structure of the question graph*, and not on the particular object classes, properties and relations. The particular visual elements are parameters given to the procedure and used to invoke the corresponding visual estimators. Hence, graphs with the same abstract structure, such as the example in Figure 1, induce the same answering procedure. This allows generalization to totally different domains that include novel objects, properties and relations.

The utilized visual estimators perform real-world tasks, corresponding to the genuine question's requirements. Hence, the intermediate results of the answering procedure can be exploited for other related tasks. Objects are grounded in the image according to the requirements of the question (see markings of intermediate results on images in Figures 2, 3, 4 and in the main paper). This means that the referring expressions comprehension task [11, 4, 10] is carried out during the answering process, where the referring ex-

Algorithm 1: Answering procedure

Input: question_graph, image
Result: Answer to question
initialization: run instance segmentation (mask R-CNN [3]),
$workMem.current\_node = first\_root\_node$;
$\boldsymbol{Run}[success, answer] = getGraphAnswer$:
begin
  $current\_node = workMem.current\_node$;
  Node parameters: $\boldsymbol{p}$: properties, $\boldsymbol{r}$: relations, f: queried property, g: queried property of a set, $\boldsymbol{obj}$: candidate objects[a];
  for *obj in* $\boldsymbol{obj}$ **do**
    if ¬ *empty(*$\boldsymbol{p}$*)* **then**
      for p *in* $\boldsymbol{p}$ **do**
        $[success, answer]$ = $is\_p(obj)$;
        if $\neg success$ **then** $break$ **end**
      end
      if ¬ *success* **then**
        if *#possible_objs <#required_objs*[b] **then**
          return $[success, answer]$
        else
          continue
        end
      end
    end
    if $\neg empty(f)$ **then** $answer = get\_f(obj)$; **end**
    if *empty(*$\boldsymbol{r}$*)* **then**
      if *exist(next_root_node)* **then**
        $workMem.current\_node = next\_root\_node$;
        Run $[success, answer]$ = $getGraphAnswer$;
      end
    else
      for r *in* $\boldsymbol{r}$ **do**
        for $child\_obj$ *in* $\boldsymbol{child\_objs}$[c] **do**
          $[success, answer]$ = $is\_r(child\_obj, obj)$;
          if *success* **then**
            $workMem.current\_node = next\_node$[d];
            Run $[success, answer]$ = $getGraphAnswer$;
            if $success \wedge (\#success\_child\_objs == \#required\_child\_objs$[b]$)$ **then** $break$ **end**
          end
          if $\neg success \wedge (\#possible\_child\_objs < \#required\_child\_objs$[b]$)$ **then** $break$ **end**
        end
        if $\neg success$ **then** $break$ **end**
      end
    end
    if $success$ **then** $break$ **end**
  end
  if $success \wedge \neg empty(g)$ **then** $answer = get\_g(valid\_objs)$; **end**
  if $success \wedge$ *comp_num_en* $\wedge$ *is_checked(comp_node)* **then**
    $answer = comp\_num$[e]$(valid\_objs, comp\_objs, comp\_type)$;
  end
  if $success \wedge is\_sub\_node$ **then**
    $save\_for\_super\_node(success\_objs)$; **end**
  return $[success, answer]$
end

[a] According to instance segmentation and previous checks. If 'super node': candidate objects are from 'sub nodes'
[b] According to quantifiers
[c] Candidate objects for child nodes
[d] Either child node or next unvisited root node of a subgraph
[e] Compare number of valid objects between nodes ('same', 'fewer', 'more')

pressions are the questions (but also parts of the questions). Performing the grounding using standard referring expressions instead of the questions can be carried out by simple adaptations. Also, saved results of the checked objects can be used to enable follow-up questions and carrying out the visual dialog task [2, 7].

Any estimator can be integrated in our system for any required object class, property, relation or related prediction. In addition to the main visual estimators mentioned in the paper for the conducted experiments, our system employs general rule based estimators for spatial relations and properties (*e.g.* the relation 'on' and the property 'the left').

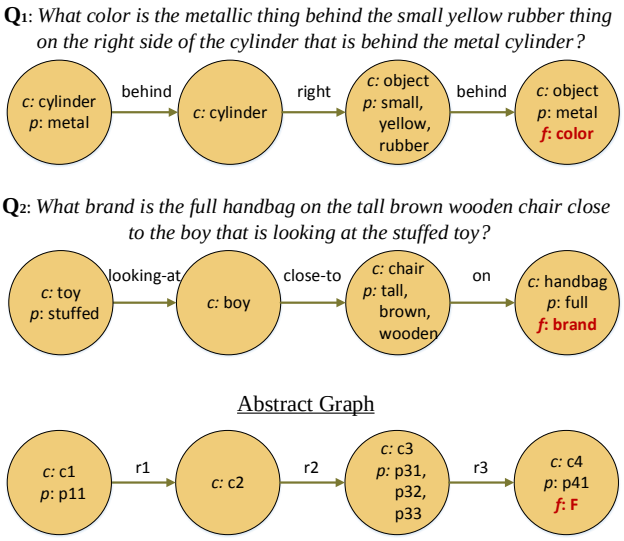


Figure 1. Both questions in the two top panels share the same abstract structure, represented by the graph in the bottom panel, where visual elements are parameters. All questions represented by this abstract graph share the same answering procedure, allowing generalization to entirely new domains with novel visual elements.

3. Extended Domain Experiment

In this experiment we test methods trained on CLEVR data (for UnCoRd, we use the 'None-Basic' question-to-graph mapper) using questions with a new property type. We use the 'stackability' properties ('stackable', 'nonstackable'), which refer to the ability to put objects on top of each other (a property used by humans in the CLEVR-Humans dataset [6]). Our approach can handle such extensions in a straightforward manner, using two steps:

1. A question modification step, where the new property in the question is replaced with another property (that does not appear in the question, but was used in the original dataset). In the corresponding graph, produced by the mapper, this property is replaced back to the original term. Note that this step is required as we use the simplest mapper (no exploitation of extended training for fairness).

2. Add a 'stackability' classifier. Two classification options were tested: a trained classifier (same architecture as our CLEVR property classifiers, with a classification accuracy of 99.98%) and a direct inference as an inherent property of the object class, without any training (that is, objects with flat bottom and top such as cubes and cylinders are 'stackable' where spheres are 'nonstackable').

We compare our method to TbD [12], which is trained with question-answer examples and hence it is not possible to plug-in a property classifier. However, to have a fair comparison, we apply finetuning to the system, but limited to a binary classification of the new property, rather than full QA training. The first step was performed in a similar manner to step (1) above, by replacing each new property in the question with a 'known' property and then back to the new one in the generated program. This is less straight-forward than for our method, as replacements are required for various program elements used by TbD that are related to the property (*e.g.* multiple modules such as '*filter_stackability[stackable]*', '*query_stackability*'). The second step requires questions-answers training (or actually programs-answers training). As this method composes a network from modules, we added new modules for the novel 'stackability' concepts, and trained them using simple existence questions (*e.g.* 'Is there a stackable object?'). Modified weights in the model were only the ones of the new modules and of the last classification layer that are connected to the novel answers: 'stackable' and 'nonstackable'. This allows the method to learn the function required for the new modules (corresponding to training a property classifier in our method). The TbD method includes multiple modules related to the new property, and it is not possible to activate all of them during the binary training. Nevertheless, we apply this procedure (which is more demanding and less natural than the simple classifier plug-in for the UnCoRd method) as it provides a fair comparison with a corresponding method to the one used to extend UnCoRd.
The training was performed using one existence question for each CLEVR training set image. The answering accuracy for the corresponding validation set (similar type of existence questions for the new property on CLEVR validation images) is 99.95%. This shows that the model is capable of learning the new property and use it on questions of the type used in training, but not necessarily generalize to new questions.
Evaluation of the methods was performed using 10,000 new questions, generated for CLEVR validation images using CLEVR questions templates with added 'stackability' terms (for all types of questions). The results are summarized in Table 1, including results of the original TbD method without modifications (with random weights for 'stackability' modules and other related weights).
The results show that the UnCoRd model maintains its near perfect results while the extensibility for the TbD fails and its performance drops considerably. As other end-to-end methods, question answering is treated as a multi-class classification problem. This means that by this approach even if the modules could have been trained properly, the weights for the last classification layers require an additional appropriate tuning. Since the used questions for TbD training were limited to binary existence questions with 'yes'/'no' answers, no data was given to properly tune the classification weights of the novel answers: 'stackable' and 'nonstackable'. This results with the TbD providing these novel answers improperly and unrelated to the actual questions, which may hide the actual impact of training the 'stackability' modules. Our method does not suffer from this issue, since the simple plugging-in of a 'stackability' classifier handles the integration of 'stackability' concepts in all types of questions, including with the novel answers. Examining and comparing performance without the effect of the novel answers was performed by removing the questions that query the 'stackability' property and excluding the novel answers from TbD optional predictions. The results for this test are given in Table 2.

Method	Exist	Count	Compare Numbers	Query Attribute	Compare Attribute	Overall
TbD-st	63.7	39.2	63.8	37.9	54.7	45.6
TbD-rand_st	21.8	26.7	30.9	24.6	16.7	24.1
UnCoRd-st_tr	99.5	98.9	**99.8**	99.4	99.3	99.3
UnCoRd-st_by_cl	**99.9**	**99.4**	**99.8**	**99.8**	**99.8**	**99.7**

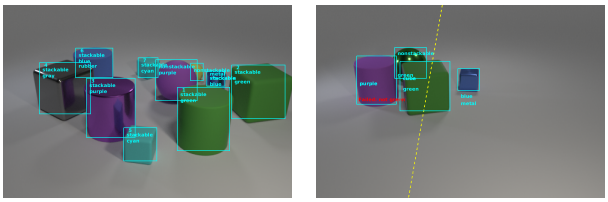
Table 1. Accuracy of answering questions with 'stackability' properties on CLEVR validation images.
TbD-st: TbD with binary trained 'stackability' modules
TbD-rand_st: TbD with random 'stackability' weights
UnCoRd-st_tr: UnCoRd with trained 'stackability' classifier
UnCoRd-st_by_cl: UnCoRd with 'stackability' inferred by the object class

Method	Exist	Count	Compare Numbers	Query Attribute	Compare Attribute	Overall
TbD-st-no_st_ans	66.6	46.4	74.0	72.0	66.2	63.3
TbD-rand_st-no_st_ans	69.2	45.2	73.2	73.0	63.9	63.5
UnCoRd-st_tr	99.5	98.9	**99.8**	99.4	99.3	99.3
UnCoRd-st_by_cl	**99.9**	**99.4**	**99.8**	**99.9**	**99.8**	**99.7**

Table 2. Accuracy of answering questions with 'stackability' properties without novel answers ('stackable', 'nonstackable'). Naming corresponds to Table 1, where the additional suffix **-no_st_ans** represents cancelling the prediction of the novel answers.

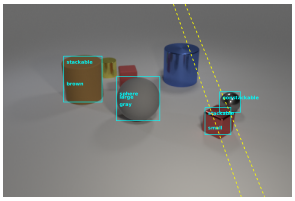
For this test TbD results improve but are still much inferior to the UnCoRd method, which remained practically the same. It is interesting to see that the TbD results with trained 'stackability' modules are not better than with untrained modules. The results imply that tuning the novel modules by one specific task (one type of questions) does not guide the modules in a general direction of performing their designated tasks. Explicit training is required for other question types to increase their performance. The

modules have no independent meaning as real 'stackability' classifiers. Our method uses real classifiers and does not share this limitation. Also, the fact that TbD with untrained 'stackability' modules does not fail completely demonstrates that each module has a limited effect in tuning the final results of the full network. As each question include at least one 'stackability' concept (average of 1.4 'stackability' concepts per question), the performance is different than expected if a real visual estimator was replaced by a random results generator. As all modules in the TbD method receive the same image features as an input (in addition to the previous module's output), this may imply that the modules have a limited (but not negligible) effect on the generated features passed to the classification layers.
Some examples of questions and answers with the 'stackability' concepts are shown in Figure 2 (UnCoRd's answers were the same for both 'stackability' classification options).

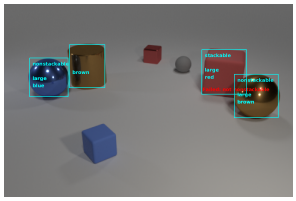


Q: What number of other objects are there of the same stackability as the blue metallic object? *GT: 7*
UnCoRd A: 7
TbD-st A: 5
TbD-st-no_st_ans A: 5

Q: There is a green thing that is on the left side of the block that is in front of the blue thing; what stackability is it? *GT: nonstackable*
UnCoRd A: nonstackable
TbD-st A: nonstackable
TbD-st-no_st_ans A: no



Q: Are there any large gray spheres left of the nonstackable object that is on the right side of the small stackable thing in front of the brown stackable object? *GT: yes*
UnCoRd A: yes
TbD-st A: no
TbD-st-no_st_ans A: no



Q: There is another nonstackable object that is the same size as the blue nonstackable thing; what is its color? *GT: brown*
UnCoRd A: brown
TbD-st A: nonstackable
TbD-st-no_st_ans A: gray

Figure 2. Examples for questions and answers on CLEVR images with the 'stackability' properties.

The results of the performed extensibility tests demonstrate the limitations in extending end-to-end VQA models, even the ones designed as a composition of modules. Compositional neural network were proposed to break the question answering task into separate tasks carried out by separate modules that are composed according to a program. However, as demonstrated above, these modules cannot be trained independently of the question answering task and cannot handle novel reasoning combinations, unseen in training. Contrary to this limitation, our method is extended naturally with novel concepts that can be used in all question types. This emphasizes the advantages of our method in extending its scope beyond the training domain.

4. Commonsense Utilization

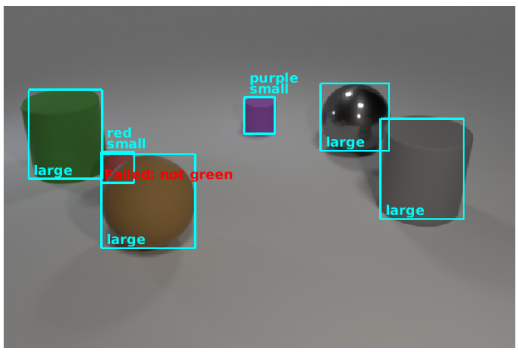
In general question answering, there is an important role to prior common knowledge, such as connections between categories of objects. To obtain general commonsense knowledge we use ConceptNet database (version 5) [15]. The source of information for this database is the internet (as well as results from additional databases). It allows querying for concepts and relations between them of the form:

concept1 - *relation* $\rightarrow$ concept2 (*e.g.* horse - *IsA* $\rightarrow$ animal)

The query is performed by providing two of the triplet [relation, concept1, concept2] and querying for the third. These common knowledge relations provide complement capabilities for answering 'real world' questions in which such common knowledge is assumed. We currently use ConceptNet mainly to extend understanding of objects' classes (*e.g.* superordinate classes and similar classes).

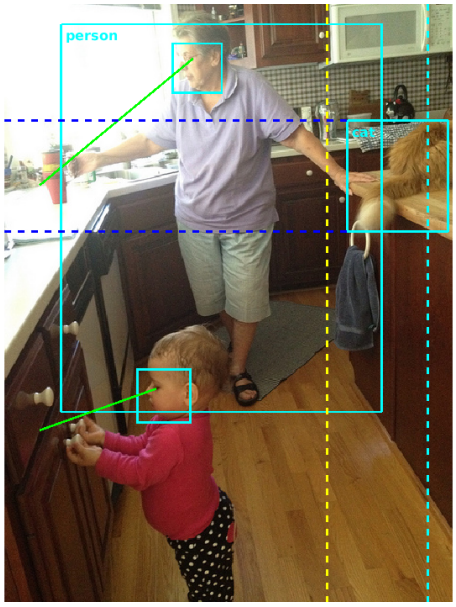
5. Examples for Answer Elaboration

Following are examples for images and corresponding questions, with answers provided by the UnCoRd system. The answers were configured to be full, and consequently provide explanations and seek alternatives, when an answer is not grounded in the image. Figure 3 demonstrates a failed grounding, and in Figure 4, alternatives are examined and reported. Note that the domains in the two examples are completely different; in Figure 3 image and question are from the CLEVR domain, while in Figure 4 the domain is real-world images and categories.



Q: Are any tiny green metal things visible?
A: There are no valid green objects (failed due to 4 large objects, a purple object and a red object)

Figure 3. An example of UnCoRd answer in the 'full' mode, explaining a failed grounding. The question-to-graph model was 'Min-Enhanced'.



Q: Is the man looking at a cat?
A: Couldn't find any object of class: man (found subclasses: a woman and a girl).
There are no people (superordinate class) looking at a cat.
Existing alternative relations (to man-looking_at-cat): 'person to the left of a cat'

Figure 4. An example for UnCoRd 'full' answer, demonstrating reporting alternatives. The question-to-graph model was 'Min-Enhanced'. The used visual estimators are mask R-CNN [3] (for the 80 classes of COCO dataset [9]), age and gender classification [8], face detection [13] and gaze estimation [14]. When a subordinate class of a person (in this case 'man') fails, the more general class 'person' is tested and proposed as an alternative. In addition, alternative relations are tested when the requested relation fails.

References

[1] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *International Conference on Computer Vision (ICCV)*, 2015. 1

[2] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, Jose M. F. Moura, Devi Parikh, and Dhruv Batra. Visual dialog. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 3

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017. 2, 3, 6

[4] Ronghang Hu, Huazhe Xu, Marcus Rohrbach, Jiashi Feng, Kate Saenko, and Trevor Darrell. Natural language object retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[5] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017. 2

[6] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *ICCV*, 2017. 3

[7] Satwik Kottur, José MF Moura, Devi Parikh, Dhruv Batra, and Marcus Rohrbach. Clevr-dialog: A diagnostic dataset for multi-round reasoning in visual dialog. *arXiv preprint arXiv:1903.03166*, 2019. 3

[8] Gil Levi and Tal Hassner. Age and gender classification using convolutional neural networks. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) workshops*, June 2015. 6

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014*. 2014. 6

[10] Runtao Liu, Chenxi Liu, Yutong Bai, and Alan L. Yuille. Clevr-ref+: Diagnosing visual reasoning with referring expressions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2

[11] Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L. Yuille, and Kevin Murphy. Generation and comprehension of unambiguous object descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[12] David Mascharka, Philip Tran, Ryan Soklaski, and Arjun Majumdar. Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4942–4950, 2018. 4

[13] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool. Face detection without bells and whistles. In *ECCV*, 2014. 6

[14] Adria Recasens*, Aditya Khosla*, Carl Vondrick, and Antonio Torralba. Where are they looking? In *Advances in Neural Information Processing Systems (NIPS)*, 2015. * indicates equal contribution. 6

[15] Robert Speer and Catherine Havasi. Conceptnet 5: A large semantic network for relational knowledge. In *The People's Web Meets NLP: Collaboratively Constructed Language Resources*, pages 161–176. Springer Berlin Heidelberg, 2013. 5