COMPOSITIONAL VISUAL INTELLIGENCE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Justin Johnson
August 2018

© 2018 by Justin Johnson. All Rights Reserved.

Re-distributed by Stanford University under license with the author.

This work is licensed under a Creative Commons Attribution-Noncommercial 3.0 United States License.
http://creativecommons.org/licenses/by-nc/3.0/us/

This dissertation is online at: http://purl.stanford.edu/kp451bm8485

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Fei-Fei Li, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Noah Goodman

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Chris Re

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

The field of computer vision has made enormous progress in the last few years, largely due to convolutional neural networks. Despite success on traditional computer vision tasks, our systems are still a long way from the general visual intelligence of people. An important facet of visual intelligence is composition - understanding of the whole derives from an understanding of the parts. To achieve the goal of compositional visual intelligence, we must explore new computer vision tasks, create new datasets, and develop new models that exploit compositionality. In this dissertation I will discuss my work on three different computer vision tasks involving language, where embracing compositionality helps us build systems with richer visual intelligence.

I will first discuss image captioning: traditional systems generate short sentences describing images, but by decomposing images into regions and descriptions into phrases we can that generate two types of richer descriptions: dense captions and paragraphs. Second, I will discuss visual question answering: existing datasets consist primarily of short, simple questions; to study more complex questions requiring compositional reasoning, we introduce a new benchark dataset where existing methods fall short. We then propose an explicitly compositional model for visual question answering that internally converts questions to functional programs, and executes these programs by composing neural modules. Third, I will discuss text-to-image: existing systems can retrieve or generate simple images of a single object conditioned on text descriptions, but struggle with more complex descriptions. By replacing freeform natural language with compositional scene graphs of objects and relationships, we can retrieve and generate complex images containing multiple objects.

Acknowledgments

I owe enormous thanks to my advisor, Fei-Fei Li. Her guidance over the past six years has helped me grow as a researcher and as a scientist. From her I learned not only technical skills but also a taste for selecting research problems and the importance of effective communication. She challenged me to step back and see the big picture, to place each bit of work into a larger intellectual context, and to dream about the exciting next steps that lie just over the horizon.

Thank you to my coauthors, all of whom helped shape my research: Alexandre Alahi, Lamberto Ballan, Michael Bernstein, Agrim Gupta, Ross Girshick, Bharath Hariharan, Judy Hoffman, Russell Kaplan, Andrej Karpathy, Jonathan Krause, Ranjay Krishna, Li-Jia Li, Laurens van der Maaten, Silvio Savarese, Michael Stark, Joseph Suarez, Jiren Zhu, and Larry Zitnick.

Thank you to my thesis committee: Noah Goodman, Christopher Manning, Christopher Ré, and Dan Yamins for insightful feedback and discussions.

I am thankful for the support of several funding agencies, including the Office of Naval Research Multidisciplinary University Research Initiative (ONR MURI) and the Brown Institute for Media Innovation.

I am thankful to all members of the Stanford Vision Lab, past and present, for countless interesting discussions and for generally making life more fun during my PhD: Chris Baldassano, Shyamal Buch, Jia Deng, Alireza Fathi, Jim Fan, Animesh Garg, Timnit Gebru, Michelle Greene, Albert Haque, De-An Huang, Marius Cătălin Iordan, Armand Joulin, Joseph Lim, Zelun Luo, Ajay Mandlekar, Juan Carlos Niebles, Guido Pusiol, Vignesh Ramanathan, Olga Russakovsky, Kevin Tang, Danfei Xu, Bangpeng Yao, and Yuke Zhu. I owe special thanks to Andrej Karpathy for teaching

me much of what I know about deep learning and effective teaching, and to Serena Yeung for solidarity in teaching and job hunting,

During my PhD I had wonderful internships at Yahoo! Research, Google Cloud AI, and Facebook AI Research; I am thankful to Li-Jia Li, Lu Jiang, Ross Girshick, Bharath Hariharan, Laurens van der Maaten, and Larry Zitnick for making these experiences so positive.

I must also thank my family. From a young age my parents Jay and Michelle instilled in me the importance of education and a fascination for science and technology. I would not be where I am today without their unconditional love and support along every step of the way. I'm thankful to my sister Sara, my grandparents, and my many aunts and uncles for their continual encouragement. I'm grateful for my dogs Cody and Chelsea, who know nothing of computer science but never fail to brighten my day. Finally my wife Joy has been my constant cheerleader and my best friend.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The past several years have been a time of fantastic progress in computer vision. The combination of large-scale image datasets [35, 138, 193], powerful programmable GPU hardware [164], and deep convolutional neural networks [118] has led to significant advances in fundamental problems like image classification [75, 202, 215] and object detection [61, 62, 191]. In some settings, image classification systems have even been shown to outperform humans [76, 193].

Although computer vision systems perform well on particular problems, they still lack rich, general visual intelligence that is a hallmark of the human visual experience. For example, consider the two images in Figure 1.1. When processed by a modern convolutional neural network trained for image classification [75], each of these two images receive the same label: *african elephant*. This label is not incorrect – indeed, elephants are the primary focus in both images. For the image in Figure 1.1(a), this single category label is a passable summary of the image's semantic content.

On the other hand, the simple category label of *african elephant* misses most of the important semantics of the image in Figure 1.1(b). Fully understanding this image requires a *compositional* view of visual intelligence: rather than trying to assign a single category label to the image, we should instead understand the individual

(a) (b)

Figure 1.1: Two images which are both labeled as *africal elephant* by an image classification system [75].

components of the image, and piece together our understanding of the whole by composing our understanding of the parts.

Fully capturing the meaning of the image in Figure 1.1(b) requires understanding *objects* (two elephants and a crocodile), *attributes* of objects (elephant 2 is small and scared), and *relationships* between objects (one elephant is behind the other; the big elephant is fighting the crocodile). In addition to the visual evidence present in the image, we might also need general *knowledge* about the world (crocodiles can eat elephants; children look like their parents but smaller), which can be combined with visual evidence via *reasoning* to infer facts about the image (one elephant is the parent of the other; the crocodile wants to eat the elephant).

A natural way to express our compositional understanding of images is through *language*, due to its inherently compositional structure [159]. For example, having understood all relevant components of the image in Figure 1.1(b), we might describe the image as *"A mother elephant defending her child"*.

If we are to build artificial systems with rich compositional visual intelligence mediated by language as described above, then we must move beyond traditional computer vision tasks like image classification and object detection. To this end, my goal in this dissertation is to develop novel computer vision tasks, datasets, and methods for expressing compositional visual intelligence with language. In each task, we will see that models exploiting the compositional structure of the problem allows us to perform richer forms of visual reasoning than had been previously possible.

1.2 Dissertation outline

This dissertation is organized into three major sections, each focusing on a different task involving vision and language where compositional reasoning is necessary.

1.2.1 Image Captioning

The first section, comprising Chapter 2 and Chaper 3, is concerned with the task of *image captioning*, where a system receives an image as input and must produce natural-language text describing that image. Due to the inherently compositional structure of language, generating good descriptions requires a compositional understanding of images. Prior work on image captioning has mostly focused on generating short, single-sentence descriptions of images [27, 40, 104, 151, 224]; by exploiting compositional structure we generate richer forms of captions.

Chapter 2 introduces the task of *dense captioning*, where a system must detect regions of interest in an image and describe each detected region with natural language. Densely describing an image with many region descriptions allows our system to give much richer descriptions than are possible with a single sentence. At the same time, working with image regions forces our model to reason explicitly about the presence and locations of objects in images.

Chapter 3 is concerned with the task of *paragraph captioning*, where a system must output a multi-sentence paragraph describing the contents of an input image. Similar to dense captioning, generating descriptive paragraphs allows our system to describe images in much more detail than is possible with a single sentence. Unlike dense captions, which are a set of potentially disjointed descriptions of regions, a descriptive paragraph gives a single cohesive description of the entire input image. Our approach for paragraph captioning leverages compositionality on both the input image and the linguistic output: it detects and reasons about regions of the input image, and produces natural language output with a hierarchical recurrent network that decomposes paragraphs into a hierarchy of sentences and words.

1.2.2 Question Answering

The second section, comprising Chapter 4 and Chapter 5, is concerned with the task of *visual question answering*, where a system receives as input an image and a question about that image, and must produce an answer to the question. Prior work on visual question answering [7, 146, 259] has mostly focused on short, factual questions about images. In contrast, we want to build systems that can answer longer, more complex questions that require not only perceiving the contents of the input image but also performing some type of *reasoning* to arrive at the answer.

Chapter 4 introduces the CLEVR dataset for evaluating compositional visual reasoning. CLEVR consists of high-quality rendered images of simple shapes with different sizes, colors, and materials. Each image is accompanied by automatically generated complex questions that require both perception and reasoning, such as *"How many red things are the same color as the thing next to the sphere?"* We evaluate several prior methods for visual question answering and show that they struggle to perform the compositional reasoning necessary to succeed on CLEVR.

Chapter 5 proposes a new model for visual question answering which is able to outperform prior work by explicitly leveraging compositionality. Rather than representing questions as a sequence of words, we instead model them as *functional programs*, where a program describes the sequence of primitive steps of reasoning that a system must take in order to answer the question. Our system first converts questions into these functional programs, then uses a Neural Modular Network [4] to execute these programs on images. This formulation allows the model to learn independent modules specialized for each primitive reasoning step, which can be composed to answer the complex compositional questions on CLEVR.

1.2.3 Text to Image

The first two sections have addressed tasks where systems receive images as input. In contrast the third section, comprising Chapters 6 and 7, considers the problem of *text to image* where a system receives a textual description of a scene as input, and must produce a corresponding image as output. By leveraging compositionality, we

are able to handle complex descriptions with many objects and relationships.

Chapter 6 introduces the notion of a *scene graph*, which is a compositional data structure for describing scene semantics. The vertices in a scene graph are *objects* that appear in the scene; object may be modified by semantic *attributes*, edges in the graph give semantic *relationships* between objects. A scene graph thus makes explicit much of the compositional information that might be expressed by descriptive text. We show how a scene graph can be used as a query to retrieve matching images from an image collection.

Chapter 7 addresses the task of *scene graph to image generation*. Like above, the system receives as input a scene graph describing the contents of a hypothetical scene; however rather than retrieving a matching image from an image collection, instead we synthesize a novel image from scratch which matches the scene graph. Prior work generating images from natural language text gives compelling results for simple descriptions [186, 187, 189, 253], but struggles to faithfully generate images from complex descriptions. In contrast, our system exploits the compositional nature of the task by conditioning the generation on compositional scene graphs rather than raw text; this allows us to generate complex images with many objects and relationships.

1.3 Previously published work

Much of the technical content of this dissertation has previously been published; these publications are [98] (Chapter 2), [115] (Chapter 3), [96] (Chapter 4), [97] (Chapter 5), [99] (Chapter 6), and [95] (Chapter 7).

During my PhD I have had the pleasure to work on a number of other topics including recurrent neural networks [105], image annotation [94], human trajectory prediction [74], artistic style transfer [73, 93], and robust digital watermarking [258]. However these topics are beyond the scope of this dissertation and will not be discussed in the chapters to follow.

Chapter 2

Dense Captioning

2.1 Introduction

Our ability to effortlessly point out and describe all aspects of an image relies on a strong semantic understanding of a visual scene and all of its elements. However, despite numerous potential applications, this ability remains a challenge for our state of the art visual recognition systems. In the last few years there has been significant progress in image classification [118, 193, 215, 252], where the task is to assign one label to an image. Further work has pushed these advances along two orthogonal directions: First, rapid progress in object detection [62, 200, 216] has identified models that efficiently identify and label multiple salient regions of an image. Second, recent advances in image captioning [26, 27, 40, 106, 151, 224, 238] have expanded the complexity of the label space from a fixed set of categories to sequence of words able to express significantly richer concepts.

However, despite encouraging progress along the label density and label complexity axes, these two directions have remained separate. In this work we take a step towards unifying these two inter-connected tasks into one joint framework. First, we introduce the dense captioning task (see Figure 2.1), which requires a model to predict a set of descriptions across regions of an image. Object detection is hence recovered as a special case when the target labels consist of one word, and image captioning is

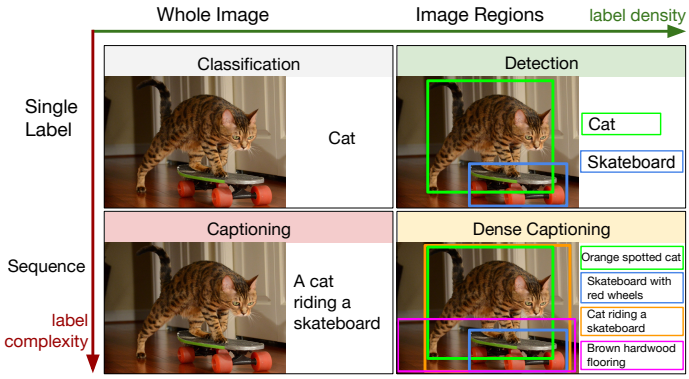


Figure 2.1: We address the Dense Captioning task (bottom right) by generating dense, rich annotations with a single forward pass.

recovered when all images consist of one region that spans the full image.

Additionally, we develop a Fully Convolutional Localization Network (FCLN) to address the dense captioning task. Our model is inspired by recent work in image captioning [27, 40, 106, 151, 224]: it is composed of a Convolutional Neural Network followed by a Recurrent Neural Network language model. Drawing on work in object detection [191], our second core contribution is to introduce a new dense localization layer. This layer is fully differentiable and can be inserted into any neural network that processes images to enable region-level training and predictions. Internally, the localization layer predicts a set of regions of interest in the image and uses bilinear interpolation [70, 86] to smoothly extract the activations inside each region.

We experiment on the large-scale Visual Genome dataset [116], which contains $\approx$ 94K images and $\approx$ 4.1M region captions. Our results show both performance and speed improvements over approaches based on previous state of the art. Our code and data are publicly available to support further progress in this area.

2.2 Related Work

Our work draws on recent work in object detection, image captioning, and soft spatial attention that allows downstream processing of particular regions in the image.

Object Detection. Our core visual processing module is a Convolutional Neural Network (CNN) [118, 128], which has emerged as a powerful model for visual recognition tasks [193]. The first application of these models to dense prediction tasks was introduced in R-CNN [62], where each region of interest was processed independently. Further work has focused on processing all regions with only single forward pass of the CNN [61, 77], and on eliminating explicit region proposal methods by directly predicting the bounding boxes either in the image coordinate system [43, 216], or in a fully convolutional [140] and hence position-invariant settings [185, 191, 200]. Most related to our approach is the work of Ren et al. [191] who develop a region proposal network (RPN) that regresses from anchors to regions of interest. However, they adopt a 4-step optimization process, while our approach does not require training pipelines. Additionally, we replace their RoI pooling mechanism with a differentiable, spatial soft attention mechanism [70, 86]. In particular, this change allows us to backpropagate through the region proposal network and train the whole model jointly.

Image Captioning. Several pioneering approaches have explored the task of describing images with natural language [9, 48, 90, 119, 123, 170, 203, 204]. More recent approaches based on neural networks have adopted Recurrent Neural Networks (RNNs) [79, 226] as the core architectural element for generating captions. These models have previously been used in language modeling [11, 68, 155, 213], where they are known to learn powerful long-term interactions [105]. Several recent approaches to Image Captioning [27, 40, 45, 106, 111, 151, 224] rely on a combination of RNN language model conditioned on image information, possibly with soft attention mechanisms [29, 238]. Similar to our work, Karpathy and Fei-Fei [106] run an image captioning model on regions but they do not tackle the joint task of detection and description in one model. Our model is end-to-end and designed in such a way that the prediction for each region is a function of a larger image context, which we show also leads to stronger performance. Finally, the metrics we develop for the dense captioning task are inspired by metrics developed for image captioning [26, 36, 223].

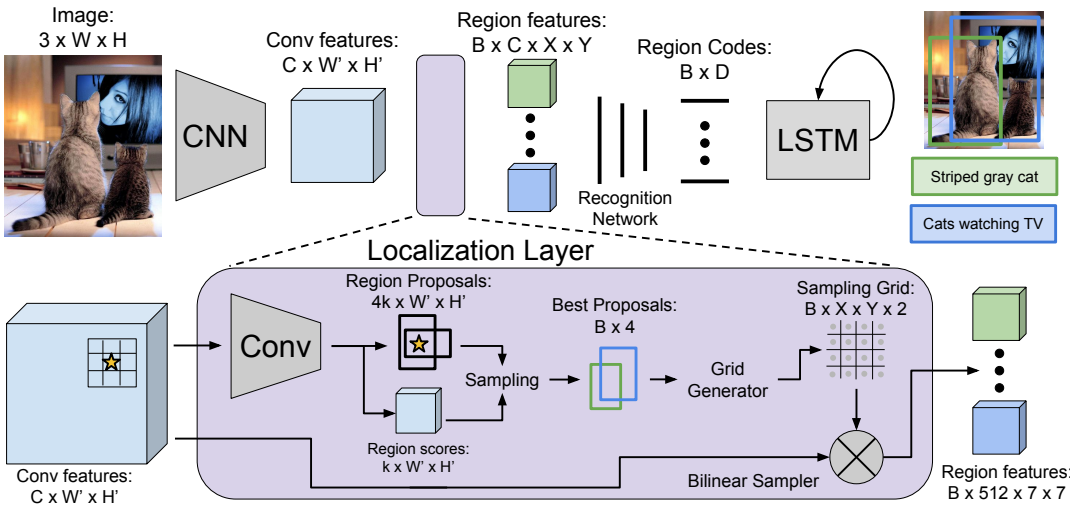


Figure 2.2: Model overview. An input image is first processed a CNN. The Localization Layer proposes regions and smoothly extracts a batch of corresponding activations using bilinear interpolation. These regions are processed with a fully-connected recognition network and described with an RNN language model. The model is trained end-to-end with gradient descent.

2.3 Model

Our goal is to design an architecture that jointly localizes regions of interest and then describes each with natural language. The primary challenge is to develop a model that supports end-to-end training with a single step of optimization, and both efficient and effective inference. Our proposed architecture (see Figure 2.2) draws on architectural elements present in recent work on object detection, image captioning and soft spatial attention to simultaneously address these design constraints.

In Section 2.3.1 we first describe the components of our model. Then in Sections 2.3.2 and 2.3.3 we address the loss function and the details of training and inference.

2.3.1 Model Architecture

2.3.1.1 Convolutional Network

We use the VGG-16 architecture [202] for its state-of-the-art performance [193]. It consists of 13 layers of 3×3 convolutions interspersed with 5 layers of 2×2 max

pooling. We remove the final pooling layer, so an input image of shape $3 \times W \times H$ gives rise to a tensor of features of shape $C \times W' \times H'$ where $C = 512$, $W' = \lfloor \frac{W}{16} \rfloor$, and $H' = \lfloor \frac{H}{16} \rfloor$. The output of this network encodes the appearance of the image at a set of uniformly sampled image locations, and forms the input to the localization layer.

2.3.1.2 Fully Convolutional Localization Layer

The localization layer receives an input tensor of activations, identifies spatial regions of interest and smoothly extracts a fixed-sized representation from each region. Our approach is based on that of Faster R-CNN [191], but we replace their RoI pooling mechanism [61] with bilinear interpolation [86], allowing our model to propagate gradients backward through the coordinates of predicted regions. This modification opens up the possibility of predicting affine or morphed region proposals instead of bounding boxes [86],but we leave these extensions to future work.

Inputs/outputs The localization layer accepts a tensor of activations of size $C \times W' \times H'$. It then internally selects B regions of interest and returns three output tensors giving information about these regions:

1. **Region Coordinates**: A matrix of shape $B \times 4$ giving bounding box coordinates for each output region.
2. **Region Scores**: A vector of length B giving a confidence score for each output region. Regions with high confidence scores are more likely to correspond to ground-truth regions of interest.
3. **Region Features**: A tensor of shape $B \times C \times X \times Y$ giving features for output regions; is represented by an $X \times Y$ grid of C-dimensional features.

Convolutional Anchors Similar to Faster R-CNN [61], our localization layer predicts region proposals by regressing offsets from a set of translation-invariant anchors. In particular, we project each point in the $W' \times H'$ grid of input features back into the $W \times H$ image plane, and consider k anchor boxes of different aspect ratios centered at

this projected point. For each of these k anchor boxes, the localization layer predicts a confidence score and four scalars regressing from the anchor to the predicted box coordinates. These are computed by passing the input feature map through a 3×3 convolution with 256 filters, a rectified linear nonlinearity, and a 1×1 convolution with $5k$ filters. This results in a tensor of shape $5k \times W' \times H'$ containing scores and offsets for all anchors.

Box Regression We use the parameterization of [61] to regress from anchors to region proposals. Given an anchor box with center (x_a, y_a), width w_a, and height h_a, our model predicts scalars (t_x, t_y, t_w, t_h) giving normalized offsets and log-space scale transforms, so that the output region has center (x, y) and shape (w, h) given by:

$$x = x_a + t_x w_a \qquad y = y_a + t_y h_a \tag{2.1}$$

$$w = w_a \exp(t_w) \qquad h = h_a \exp(t_h) \tag{2.2}$$

Box Sampling Processing a typical image of size $W = 720, H = 540$ with $k = 12$ anchor boxes gives rise to 17,280 region proposals. Since running the recognition network and the language model for all proposals would be prohibitively expensive, it is necessary to subsample them.

At training time, we follow the approach of [191] and sample a minibatch containing $B = 256$ boxes with at most $B/2$ positive regions and the rest negatives. A region is positive if it has an intersection over union (IoU) of at least 0.7 with some ground-truth region; in addition, the predicted region of maximal IoU with each ground-truth region is positive. A region is negative if it has IoU < 0.3 with all ground-truth regions. Our sampled minibatch contains $B_P \leq B/2$ positive regions and $B_N = B - B_P$ negative regions, sampled uniformly without replacement from the set of all positive and all negative regions respectively.

At test time we subsample using greedy non-maximum suppression (NMS) based on the predicted proposal confidences to select the $B = 300$ most confident propoals.

The coordinates and confidences of the sampled proposals are collected into tensors of shape $B \times 4$ and B respectively, and are output from the localization layer.

Bilinear Interpolation. After sampling, we are left with region proposals of varying sizes and aspect ratios. In order to interface with the full-connected recognition network and the RNN language model, we must extract a fixed-size feature representation for each variably sized region proposal.

To solve this problem, Fast R-CNN [61] proposes an RoI pooling layer where each region proposal is projected onto the $W' \times H'$ grid of convolutional features and divided into a coarse $X \times Y$ grid aligned to pixel boundaries by rounding. Features are max-pooled within each grid cell, resulting in an $X \times Y$ grid of output features.

The RoI pooling layer is a function of two inputs: convolutional features and region proposal coordinates. Gradients can be propagated backward from output features to input features, but not to input proposal coordinates. To overcome this limitation, we replace the RoI pooling layer with with bilinear interpolation [70, 86].

Concretely, given an input feature map U of shape $C \times W' \times H'$ and a region proposal, we interpolate the features of U to produce an output feature map V of shape $C \times X \times Y$. After projecting the region proposal onto U we follow [86] and compute a *sampling grid* G of shape $X \times Y \times 2$ associating each element of V with real-valued coordinates into U. If $G_{i,j} = (x_{i,j}, y_{i,j})$ then $V_{c,i,j}$ should be equal to U at $(c, x_{i,j}, y_{i,j})$; however since $(x_{i,j}, y_{i,j})$ are real-valued, we convolve with a sampling kernel k and set

$$V_{c,i,j} = \sum_{i'=1}^{W} \sum_{j'=1}^{H} U_{c,i',j'} k(i' - x_{i,j}) k(j' - y_{i,j}). \tag{2.3}$$

We use bilinear sampling, corresponding to the kernel $k(d) = \max(0, 1 - |d|)$. The sampling grid is a linear function of the proposal coordinates, so gradients can be propagated backward into predicted region proposal coordinates. Running bilinear interpolation to extract features for all sampled regions gives a tensor of shape $B \times C \times X \times Y$, forming the final output from the localization layer.

2.3.1.3 Recognition Network

The recognition network is a fully-connected network that processes region features from the localization layer. Features from each region are flattened into a vector

and passed through two full-connected layers, each using rectified linear units and regularized using Dropout. For each region this produces a code of dimension $D = 4096$ that compactly encodes its visual appearance. The codes for all positive regions are collected into a matrix of shape $B \times D$ and passed to the RNN language model.

In addition, we allow the recognition network one more chance to refine the confidence and position of each proposal region. It outputs a final scalar confidence of each proposed region and four scalars encoding a final spatial offset to be applied to the region proposal. These two outputs are computed as a linear transform from the D-dimensional code for each region. The final box regression uses the same parameterization as Section 2.3.1.2.

2.3.1.4 RNN Language Model

Following previous work [27, 40, 106, 151, 224], we use the region codes to condition an RNN language model [68, 155, 213]. Concretely, given a training sequence of tokens $s_1, \ldots, s_T$, we feed the RNN $T + 2$ word vectors $x_{-1}, x_0, x_1, \ldots, x_T$, where $x_{-1} = \text{CNN}(I)$ is the region code encoded with a linear layer and followed by a ReLU non-linearity, x_0 corresponds to a special START token, and x_t encode each of the tokens s_t, $t = 1, \ldots, T$. The RNN computes a sequence of hidden states h_t and output vectors y_t using a recurrence formula $h_t, y_t = f(h_{t-1}, x_t)$ (we use the LSTM [79] recurrence). The vectors y_t have size $|V| + 1$ where V is the token vocabulary, and where the additional one is for a special END token. The loss function on the vectors y_t is the average cross entropy, where the targets at times $t = 0, \ldots, T-1$ are the token indices for s_{t+1}, and the target at $t = T$ is the END token. The vector y_{-1} is ignored. Our tokens and hidden layers have size 512.

At test time we feed the visual information x_{-1} to the RNN. At each time step we sample the most likely next token and feed it to the RNN in the next time step, repeating the process until the special END token is sampled.

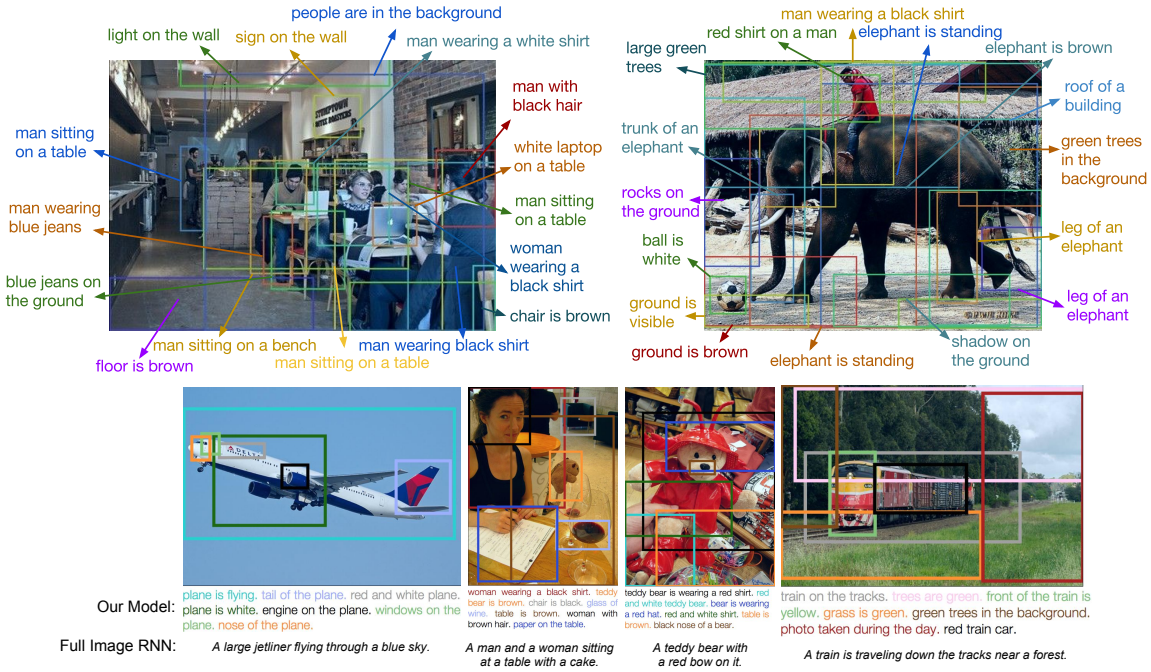


Figure 2.3: Example captions generated and localized by our model on test images. We render the top few most confident predictions. On the bottom row we additionally contrast the amount of information our model generates compared to the Full image RNN.

2.3.2 Loss function

During training our ground truth consists of positive boxes and descriptions. Our model predicts positions and confidences of sampled regions twice: in the localization layer and again in the recognition network. We use binary logistic lossses for the confidences trained on sampled positive and negative regions. For box regression, we use a smooth L1 loss in transform coordinate space similar to [191]. The fifth term in our loss function is a cross-entropy term at every time-step of the language model.

We normalize all loss functions by the batch size and sequence length in the RNN. We searched over an effective setting of the weights between these contributions and found that a reasonable setting is to use a weight of 0.1 for the first four criterions, and a weight of 1.0 for captioning.

2.3.3 Training and optimization

We train the full model end-to-end in a single step of optimization. We initialize the CNN with weights pretrained on ImageNet [193] and all other weights from a gaussian with standard deviation of 0.01. We use stochastic gradient descent with momentum 0.9 to train the weights of the convolutional network, and Adam [108] to train the other components of the model. We use a learning rate of 1×10^{-6} and set $\beta_1 = 0.9, \beta_2 = 0.99$. We begin fine-tuning the layers of the CNN after 1 epoch, and for efficiency we do not fine-tune the first four convolutional layers of the network.

Our training batches consist of a single image that has been resized so that the longer side has 720 pixels. Our implementation uses Torch7 [33] and [169]. One mini-batch runs in approximately 300ms on a Maxwell Titan X GPU and it takes about three days of training for the model to converge.

2.4 Experiments

Dataset Existing datasets relating images and natural language either only include full image captions [26, 246], or ground words of captions in regions but do not provide individual region captions [179]. We perform experiments using the Visual Genome (VG) region captions dataset [116] [1] This dataset contains 94,313 images and 4,100,413 snippets of text (43.5 per image), each grounded to a region of an image. Images were taken from the intersection of MS COCO and YFCC100M [220], and annotations were collected on Amazon Mechanical Turk by asking workers to draw a bounding box on the image and describe its content in text. Example captions include "cats play with toys hanging from a perch", "newspapers are scattered across a table", "woman pouring wine into a glass", "mane of a zebra", and "red light".

Preprocessing We collapse words that appear less than 15 times into a special `<UNK>` token, giving a vocabulary of 10,497 words. We strip referring phrases such as "there is...", or "this seems to be a". For efficiency we discard all annotations with more than 10 words (7% of annotations). We also discard all images that have fewer

[1] Dataset can be downloaded at `http://visualgenome.org`.

	Language (METEOR)			**DenseCap** (AP)			**Test runtime** (ms)			
Region source	EB	RPN	GT	EB	RPN	GT	Prop.	CNN	RNN	Total
Image RNN [106]	0.173	0.197	0.209	2.42	4.27	*14.11*	210ms	2950ms	**10ms**	3170ms
Region RNN [106]	0.221	0.244	0.272	1.07	4.26	*21.90*	210ms	2950ms	**10ms**	3170ms
FCLN on EB [61]	**0.264**	**0.296**	0.293	4.88	3.21	*26.84*	210ms	**140ms**	**10ms**	360ms
Ours (FCLN)	**0.264**	0.273	**0.305**	**5.24**	**5.39**	***27.03***	**90ms**	**140ms**	**10ms**	**240ms**

Table 2.1: Dense captioning evaluation on the test set of 5,000 images. The language metric is METEOR (high is good), our dense captioning metric is Average Precision (AP, high is good), and the test runtime performance for a 720×600 image with 300 proposals is given in milliseconds on a Maxwell Titan X GPU (ms, low is good). "Prop." is the time to compute region proposals, and "CNN" includes the time for the recognition network. EB, RPN, and GT correspond to EdgeBoxes [260], Region Proposal Network [191], and ground truth boxes respectively, used at test time. Numbers in GT columns (italic) serve as upper bounds assuming perfect localization.

than 20 or more than 50 annotations to reduce the variation in the number of regions per image. We are left with 87,398 images; we assign 5,000 each to val/test splits and the rest to train.

For test time evaluation we also preprocess the ground truth regions in the validation/test images by merging heavily overlapping boxes into single boxes with several reference captions. For each image we iteratively select the box with the highest number of overlapping boxes (based on IoU with threshold of 0.7), and merge these together (by taking the mean) into a single box with multiple reference captions. We then exclude this group and repeat the process.

2.4.1 Dense Captioning

In the dense captioning task the model receives a single image and produces a set of regions, each annotated with a confidence and a caption.

Evaluation metrics Intuitively, we would like our model to produce both well-localized predictions (as in object detection) and accurate descriptions (as in image captioning). Inspired by evaluation metrics in object detection [44, 138] and image captioning [223], we propose to measure the mean Average Precision (AP) across a

range of thresholds for both localization and language accuracy. For localization we use intersection over union (IoU) thresholds .3, .4, .5, .6, .7. For language we use METEOR score thresholds 0, .05, .1, .15, .2, .25. We adopt METEOR since this metric was found to be most highly correlated with human judgments in settings with a low number of references [223]. We measure the average precision across all pairwise settings of these thresholds and report the mean AP.

To isolate the accuracy of language in the predicted captions without localization we also merge ground truth captions across each test image into a bag of references sentences and evaluate predicted captions with respect to these references without taking into account their spatial position.

Baseline models Following Karpathy and Fei-Fei [106], we train only the Image Captioning model (excluding the localization layer) on individual, resized regions. We refer to this approach as a *Region RNN model*. To investigate the differences between captioning trained on full images or regions we also train the same model on full images and captions from MS COCO (*Full Image RNN model*).

At test time we consider three sources of region proposals. First, to establish an upper bound we evaluate the model on ground truth boxes (GT). Second, similar to [106] we use an external region proposal method to extract 300 boxes for each test image. We use EdgeBoxes [260] (EB) due to their strong performance and speed. Finally, EdgeBoxes have been tuned to obtain high recall for objects, but our regions data contains a wide variety of annotations around groups of objects, stuff, etc. Therefore, as a third source of test time regions we follow Faster R-CNN [191] and train a separate Region Proposal Network (RPN) on the VG regions data. This corresponds to training our full model except without the RNN language model.

As the last baseline we reproduce the approach of Fast R-CNN [61], where the region proposals during training are fixed to EdgeBoxes instead of being predicted by the model (*FCLN on EB*). The results of this experiment can be found in Table 2.1. We now highlight the main takeaways.

Discrepancy between region and image statistics Focusing on the first two rows of Table 2.1, Region RNN obtains consistently stronger results on METEOR alone, supporting the difference in the language statistics between regions and images. Note that these models were trained on nearly the same images, but one on full image captions and the other on region captions. However, despite differences in language, the two models reach comparable performance on the final metric.

RPN outperforms external region proposals In all cases we obtain performance improvements when using the RPN network instead of EB regions. The only exception is the FCLN model that was only trained on EB boxes. Our hypothesis is that this reflects people's tendency of annotating regions more general than those containing objects. The RPN network can learn these distributions from the raw data, while the EdgeBoxes method was designed for high recall on objects. In particular, note that this also allows our model (FCLN) to outperform the FCLN on EB baseline, which is constrained to EdgeBoxes during training (5.24 vs. 4.88 and 5.39 vs. 3.21). This is despite the fact that their localization-independent language scores are comparable, which suggests that our model achieves improvements specifically due to better localization. Finally, the noticeable drop in performance of the FCLN on EB model when evaluating on RPN boxes (5.39 down to 3.21) also suggests that the EB boxes have particular visual statistics, and that the RPN boxes are likely out of sample for the FCLN on EB model.

Our model outperforms individual region description Our final model performance is listed under the RPN column as 5.39 AP. In particular, note that in this one cell of Table 2.1 we report the performance of our full joint model instead of our model evaluated on the boxes from the independently trained RPN network. Our performance is quite a bit higher than that of the Region RNN model, even when the region model is evaluated on the RPN proposals (5.93 vs. 4.26). We attribute this improvement to the fact that our model can take advantage of visual information from the context outside of the test regions.

	Ranking				Localization			
	R@1	R@5	R@10	MR	IoU@0.1	IoU@0.3	IoU@0.5	MIoU
Image RNN [106]	0.10	0.30	0.43	13	-	-	-	-
EB + Image RNN [106]	0.11	0.40	0.55	9	0.348	0.156	0.053	0.020
Region RNN [61]	0.18	0.43	0.59	7	0.460	0.273	0.108	0.077
Ours (FCLN)	**0.27**	**0.53**	**0.67**	**5**	**0.560**	**0.345**	**0.153**	**0.137**

Table 2.2: Results for image retrieval experiments. We evaluate ranking using recall at k (R@K, higher is better) and median rank of the target image (MR, lower is better). We evaluate localization using ground-truth region recall at different IoU thresholds (IoU@t, higher is better) and median IoU (MIoU, higher is better). Our method outperforms baselines at both ranking and localization.

Qualitative results We show example predictions of the dense captioning model in Figure 2.3. The model generates rich snippet descriptions of regions and accurately grounds the captions in the images. For instance, note that several parts of the elephant are correctly grounded and described ("trunk of an elephant", "elephant is standing", and both "leg of an elephant"). The same is true for the airplane, where the tail, engine, nose and windows are correctly localized. Common failure cases include repeated detections (e.g. the elephant is described as standing twice).

Runtime evaluation Our model is efficient at test time: a 720×600 image is processed in 240ms. This includes running the CNN, computing $B = 300$ region proposals, and sampling from the language model for each region.

Table 2.1 (right) compares the test-time runtime performance of our model with baselines that rely on EdgeBoxes. Regions RNN is slowest since it processes each region with an independent forward pass of the CNN; with a runtime of 3170ms it is more than 13× slower than our method.

FCLN on EB extracts features for all regions after a single forward pass of the CNN. Its runtime is dominated by EdgeBoxes; it is $\approx 1.5\times$ slower than our method.

Our method takes 88ms to compute region proposals, of which nearly 80ms is spent running NMS to subsample regions in the Localization Layer. This time can be drastically reduced by using fewer proposals: using 100 region proposals reduces our total runtime to 166ms.

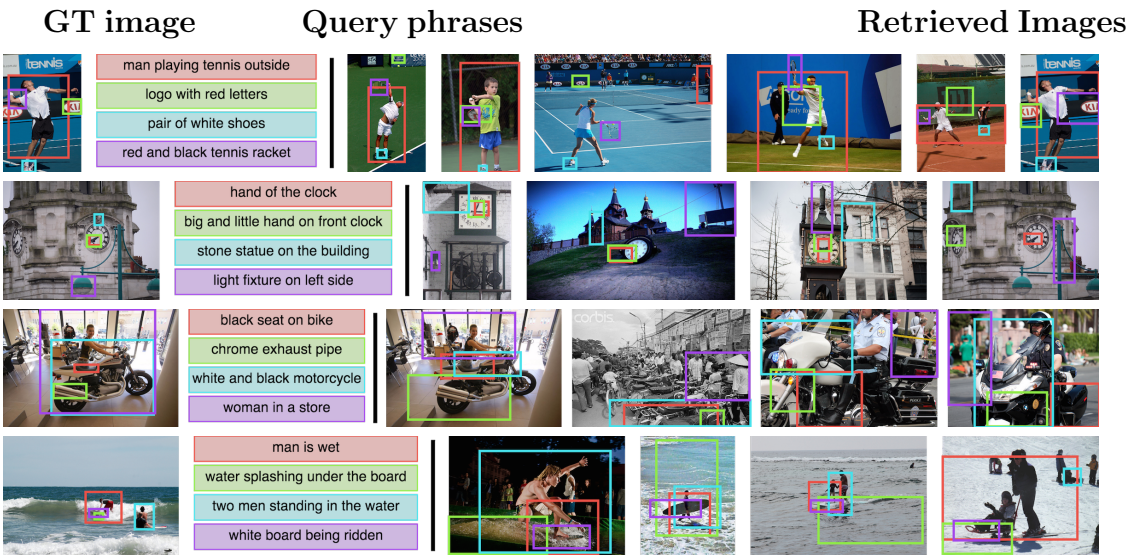


Figure 2.4: Example image retrieval results using our dense captioning model. From left to right, each row shows a grund-truth test image, ground-truth region captions describing the image, and the top images retrieved by our model using the text of the captions as a query. Our model is able to correctly retrieve and localize people, animals, and parts of both natural and man-made objects.

2.4.2 Image Retrieval using Regions and Captions

In addition to generating novel descriptions, our dense captioning model can support image retrieval using natural-language queries, and can localize these queries in retrieved images. We evaluate our model's ability to correctly retrieve images and accurately localize textual queries.

Experiment setup We use 1000 random images from the VG test set and generate 100 test queries by repeatedly sampling four random captions from some image and then expect the model to correct retrieve the source image for each query.

Evaluation To evaluate ranking, we report the fraction of queries for which the correct source image appears in the top k positions for $k \in \{1, 5, 10\}$ (recall at k) and the median rank of the correct image across all queries.

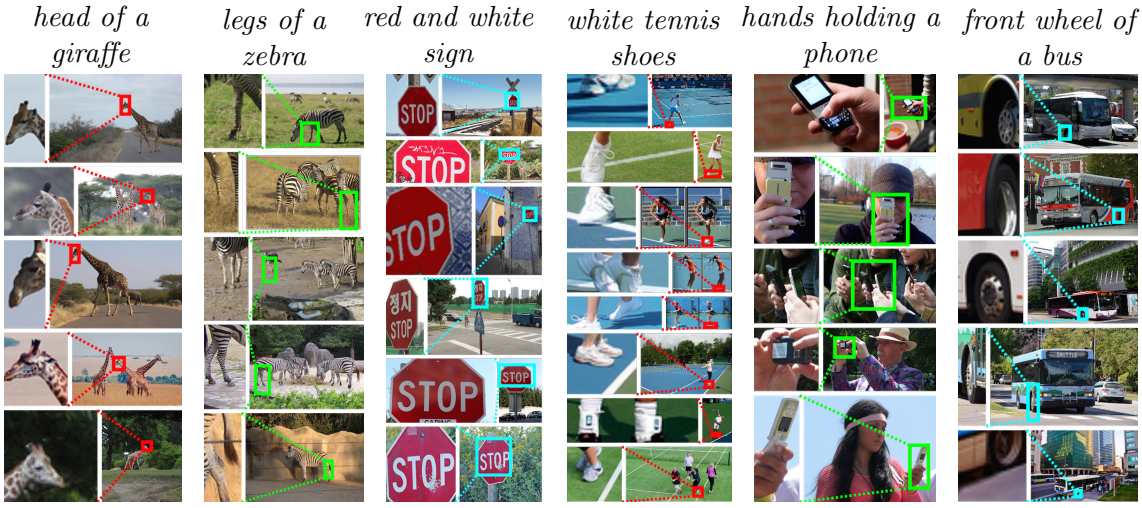


Figure 2.5: Example results for open world detection. We use our dense captioning model to localize arbitrary pieces of text in images, and display the top detections on the test set for several queries.

To evaluate localization, for each query caption we examine the image and ground-truth bounding box from which the caption was sampled. We compute IoU between this ground-truth box and the model's predicted grounding for the caption. We then report the fraction of query caption for which this overlap is greater than a threshold t for $t \in \{0.1, 0.3, 0.5\}$ (recall at t) and the median IoU across all query captions.

Models We compare the ranking and localization performance of full model with baseline models from Section 2.4.1.

For the Full Image RNN model trained on MS COCO, we compute the probability of generating each query caption from the entire image and rank test images by mean probability across query captions. Since this does not localize captions we only evaluate its ranking performance.

The Full Image RNN and Region RNN methods are trained on full MS COCO images and ground-truth VG regions respectively. In either case, for each query and test image we generate 100 region proposals using EdgeBoxes and for each query caption and region proposal we compute the probability of generating the query caption from the region. Query captions are aligned to the proposal of maximal

probability, and images are ranked by the mean probability of aligned caption / region pairs. The process for the full FCLN model is similar, but uses the top 100 proposals from the localization layer rather than EdgeBoxes proposals.

Discussion Figure 2.4 shows examples of ground-truth images, query phrases describing those images, and images retrieved from these queries using our model. Our model is able to localize small objects ("hand of the clock", "logo with red letters"), object parts, ("black seat on bike", "chrome exhaust pipe"), people ("man is wet") and some actions ("man playing tennis outside").

Quantitative results comparing our model against the baseline methods is shown in Table 2.2. The relatively poor performance of the Full Image RNN model (Med. rank 13 vs. 9,7,5) may be due to mismatched statistics between its train and test distributions: the model was trained on full images, but in this experiment it must match region-level captions to whole images (Full Image RNN) or process image regions rather than full images (EB + Full Image RNN).

The Region RNN model does not suffer from a mismatch between train and test data, and outperforms the Full Image RNN model on both ranking and localization. Compared to Full Image RNN, it reduces the median rank from 9 to 7 and improves localization recall at 0.5 IoU from 0.053 to 0.108.

Our model outperforms the Region RNN baseline for both ranking and localization under all metrics, further reducing the median rank from 7 to 5 and increasing localization recall at 0.5 IoU from 0.108 to 0.153.

The baseline uses EdgeBoxes which was tuned to localize objects, but not all query phrases refer to objects. Our model achieves superior results since it learns to propose regions from the training data.

Open-world Object Detection Using the retrieval setup described above, our dense captioning model can also be used to localize arbitrary pieces of text in images. This enables "open-world" object detection, where instead of committing to a fixed set of object classes at training time we can specify object classes using natural language at test-time. We show example results for this task in Figure 2.5, where we display

the top detections on the test set for several phrases.

Our model can detect animal parts ("head of a giraffe", "legs of a zebra") and also understands some object attributes ("red and white sign", "white tennis shoes") and interactions between objects ("hands holding a phone"). The phrase "front wheel of a bus" is a failure case: the model correctly identifies wheels of buses, but cannot distinguish between the front and back wheel.

2.5 Discussion

In this chapter we introduced the dense captioning task, which requires a model to simultaneously localize and describe regions of an image. To address this task we developed the FCLN architecture, which supports end-to-end training and efficient test-time performance. Our FCLN architecture is based on recent CNN-RNN models developed for image captioning but includes a novel, differentiable localization layer that can be inserted into any neural network to enable spatially-localized predictions. Our experiments in both generation and retrieval settings demonstrate the power and efficiency of our model with respect to baselines related tp previous work, and qualitative experiments show visually pleasing results. In future work we would like to relax the assumption of rectangular proposal regions and to discard test-time NMS in favor of a trainable spatial suppression layer.

Chapter 3

Descriptive Image Paragraphs

3.1 Introduction

Vision is the primary sensory modality for human perception, and language is our most powerful tool for communicating with the world. Building systems that can simultaneously understand visual stimuli and describe them in natural language is therefore a core problem in both computer vision and artificial intelligence as a whole. With the advent of large datasets pairing images with natural language descriptions [80, 116, 138, 246] it has recently become possible to generate novel sentences describing images [27, 40, 104, 151, 224]. While the success of these methods is encouraging, they all share one key limitation: *detail*. By only describing images with a single high-level sentence, there is a fundamental upper-bound on the quantity and quality of information approaches can produce.

One recent alternative to sentence-level captioning is the task of dense captioning [98], which overcomes this limitation by detecting many regions of interest in an image and describing each with a short phrase. By extending the task of object detection to include natural language description, dense captioning describes images in considerably more detail than standard image captioning. However, this comes at a cost: descriptions generated for dense captioning are not coherent, they do not form a cohesive whole describing the entire image.

Sentences:
A girl is eating donuts with a boy in a restaurant
A boy and girl sitting at a table with doughnuts.
Two kids sitting a coffee shop eating some frosted donuts
Two children sitting at a table eating donuts.
Two children eat doughnuts at a restaurant table.

Paragraph: Two children are sitting at a table in a restaurant. The children are one little girl and one little boy. The little girl is eating a pink frosted donut with white icing lines on top of it. The girl has blonde hair and is wearing a green jacket with a black long sleeve shirt underneath. The little boy is wearing a black zip up jacket and is holding his finger to his lip but is not eating. A metal napkin dispenser is in between them at the table. The wall next to them is white brick. Two adults are on the other side of the short white brick wall. The room has white circular lights on the ceiling and a large window in the front of the restaurant. It is daylight outside.

Figure 3.1: Paragraphs are longer, more informative, and more linguistically complex than sentence captions. Here we show an image (left top) with its sentence captions from MS COCO [138] (left bottom) and the paragraph used in this work (right).

In this chapter we address the shortcomings of both traditional image captioning and the recently-proposed dense image captioning by introducing the task of generating paragraphs that richly describe images (Fig. 3.1). Paragraph generation combines the strengths of these tasks but does not suffer from their weaknesses – like traditional captioning, paragraphs give a coherent natural language description for images, but like dense captioning, they can do so in fine-grained detail.

Generating paragraphs for images is challenging, requiring both fine-grained image understanding and long-term language reasoning. To overcome these challenges, we propose a model that decomposes images and paragraphs into their constituent parts: We break images into semantically meaningful pieces by detecting objects and other regions of interest, and we reason about language with a hierarchical recurrent neural network, decomposing paragraphs into their corresponding sentences. In addition, we also demonstrate for the first time the ability to transfer visual and linguistic knowledge from large-scale region captioning [116], which we show has the ability to improve paragraph generation.

To validate our method, we collected a dataset of image and paragraph pairs, which complements the whole-image and region-level annotations of MS COCO [138] and Visual Genome [116]. To validate the complexity of the paragraph generation task, we performed a linguistic analysis of our collected paragraphs, comparing them to sentence-level image captioning. We compare our approach with numerous baselines, showcasing the benefits of hierarchical modeling for generating descriptive paragraphs.

The rest of this chapter is organized as follows: Sec. 3.2 overviews related work in image captioning and hierarchical RNNs, Sec. 3.3 introduces the paragraph generation task, describes our newly-collected dataset, and performs a simple linguistic analysis on it, Sec. 3.4 details our model for paragraph generation, Sec. 3.5 contains experiments, and Sec. 3.6 concludes with discussion.

3.2 Related Work

Image Captioning Building connections between visual and textual data has been a longstanding goal in computer vision. One line of work treats the problem as a ranking task, using images to retrieve relevant captions from a database and vice-versa [48, 80, 106]. Due to the compositional nature of language, it is unlikely that any database will contain all possible image captions; therefore another line of work focuses on generating captions directly. Early work uses handwritten templates to generate language [119] while more recent methods train recurrent neural network language models conditioned on image features [27, 40, 104, 151, 224, 245] and sample from them to generate text. Similar methods have also been applied to generate captions for videos [40, 243, 247].

A handful of approaches to image captioning reason not only about whole images but also image regions. Xu et al. [238] generate captions using a recurrent network with attention, so that the model produces a distribution over image regions for each word. In contrast to their work, which uses a coarse grid as image regions, we use semantically meaningful regions of interest. Karpathy and Fei-Fei [104] use a ranking loss to align image regions with sentence fragments but do not do generation with

the model. Johnson et al. [98] introdue the task of dense captioning, which detects and describes regions of interest, but these descriptions are independent and do not form a coherent whole.

There has also been some pioneering work on video captioning with multiple sentences [192]. While videos are a natural candidate for multi-sentence description generation, image captioning cannot leverage strong temporal dependencies, adding extra challenge.

Hierarchical Recurrent Networks In order to generate a paragraph description, a model must reason about long-term linguistic structures spanning multiple sentences. Due to vanishing gradients, recurrent neural networks trained with stochastic gradient descent often struggle to learn long-term dependencies. Alternative recurrent architectures such as long-short term memory (LSTM) [79] help alleviate this problem through a gating mechanism that improves gradient flow. Another solution is a *hierarchical* recurrent network, where the architecture is designed such that different parts of the model operate on different time scales.

Early work applied hierarchical recurrent networks to simple algorithmic problems [42]. The Clockwork RNN [113] uses a related technique for audio signal generation, spoken word classification, and handwriting recognition; a similar hierarchical architecture was also used in [19] for speech recognition. In these approaches, each recurrent unit is updated on a fixed schedule: some units are updated on every timestep, while other units might be updated every other or every fourth timestep. This type of hierarchy helps reduce the vanishing gradient problem, but the hierarchy of the model does not directly reflect the hierarchy of the output sequence.

More related to our work are hierarchical architectures that directly mirror the hierarchy of language. Li et al. [131] introduce a hierarchical autoencoder, and Lin et al. [137] use different recurrent units to model sentences and words. Most similar to our work is Yu et al. [247], who generate multi-sentence descriptions for cooking videos using a different hierarchical model. Due to the less constrained non-temporal setting in our work, our method has to learn in a much more generic fashion and has been made simpler as a result, relying more on learning the interplay between

	COCO [138] (Sentences)	Ours (Paragraphs)
Description Length	11.30	67.50
Sentence Length	11.30	11.91
Diversity	19.01	70.49
Nouns	33.45%	25.81%
Adjectives	27.23%	27.64%
Verbs	10.72%	15.21%
Pronouns	1.23%	2.45%

Table 3.1: Statistics of paragraph descriptions, compared with sentence-level captions used in prior work. Description and sentence lengths are represented by the number of tokens present, diversity is the inverse of the average CIDEr score between sentences of the same image, and part of speech distributions are aggregated from Penn Treebank [152] part of speech tags.

sentences. Additionally, our method reasons about semantic regions in images, which both enables the transfer of information from these regions and leads to more interpretability in generation.

3.3 Paragraphs are Different

To what extent does describing images with paragraphs differ from sentence-level captioning? To answer this question, we collected a novel dataset of paragraph annotations, comparised of 19,551 MS COCO [138] and Visual Genome [116] images, where each image has been annotated with a paragraph description. Annotations were collected on Amazon Mechanical Turk, using U.S. workers with at least 5,000 accepted HITs and an acceptance rate of 98% or greater[1], and were additionally subject to automatic and manual spot checks on quality. Fig. 3.1 demonstrates an example, comparing our collected paragraph with the five corresponding sentence-level captions from MS COCO. Though it is clear that the paragraph is longer and more descriptive than any one sentence, we note further that a single paragraph can be more detailed than *all five* sentence captions, even when combined. This occurs because of redundancy in sentence-level captions – while each caption might use

[1]Available at `http://cs.stanford.edu/people/ranjaykrishna/im2p/index.html`

slightly different words to describe the image, since all sentence captions have the goal of describing the image as a whole, they are fundamentally limited in terms of both diversity and their total information.

We quantify these observations along with various other statistics of language in Tab. 3.1. For example, we find that each paragraph is roughly six times as long as the average sentence caption, and the individual sentences in each paragraph are of comparable length as sentence-level captions. To examine the issue of sentence diversity, we compute the average CIDEr [223] similarity between COCO sentences for each image and between the individual sentences in each collected paragraph, defining the final diversity score as 100 minus the average CIDEr similarity. Viewed through this metric, the difference in diversity is striking – sentences within paragraphs are substantially more diverse than sentence captions, with a diversity score of 70.49 compared to only 19.01. This quantifiable evidence demonstrates that sentences in paragraphs provide significantly more information about images.

Diving deeper, we performed a simple linguistic analysis on COCO sentences and our collected paragraphs, comprised of annotating each word with a part of speech tag from Penn Treebank via Stanford CoreNLP [150] and aggregating parts of speech into higher-level linguistic categories. A few common parts of speech are given in Tab. 3.1. As a proportion, paragraphs have somewhat more verbs and pronouns, a comparable frequency of adjectives, and somewhat fewer nouns. Given the nature of paragraphs, this makes sense – longer descriptions go beyond the presence of a few salient objects and include information about their properties and relationships. We also note but do not quantify that paragraphs exhibit higher frequencies of more complex linguistic phenomena, *eg.* coreference occurring in Fig. 3.1, wherein sentences refer to either "two children", "one little girl and one little boy", "the girl", or "the boy." We belive that these types of long-range phenomena are a fundamental property of descriptive paragraphs with human-like language and cannot be adequately explored with sentence-level captions.

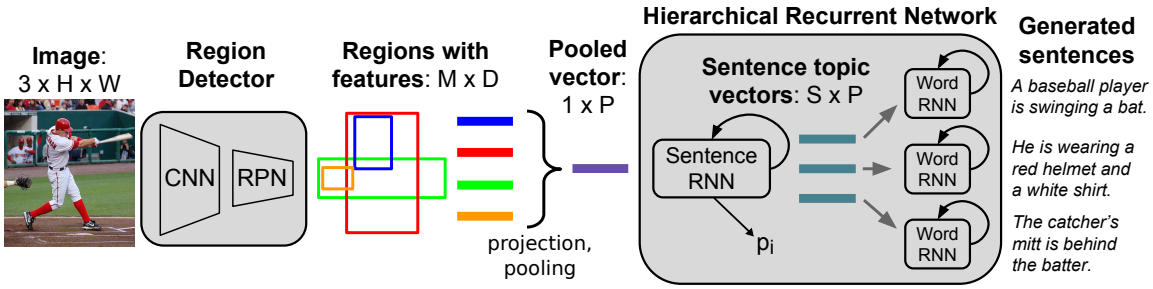


Figure 3.2: Overview of our model. Given an image (left), a region detector (comprising a convolutional network and a region proposal network) detects regions of interest and produces features for each. Region features are projected to $\mathbb{R}^P$, pooled to give a compact image representation, and passed to a hierarchical recurrent neural network language model comprising a sentence RNN and a word RNN. The sentence RNN determines the number of sentences to generate based on the halting distribution p_i and also generates sentence topic vectors, which are consumed by each word RNN to generate sentences.

3.4 Method

Overview Our model takes an image as input, generating a natural-language paragraph describing it, and is designed to take advantage of the compositional structure of both images and paragraphs. Fig. 3.2 provides an overview. We first decompose the input image by detecting objects and other regions of interest, then aggregate features across these regions to produce a pooled representation richly expressing the image semantics. This feature vector is taken as input by a hierarchical recurrent neural network composed of two levels: a sentence RNN and a word RNN. The sentence RNN receives the image features, decides how many sentences to generate in the resulting paragraph, and produces an input topic vector for each sentence. Given this topic vector, the word RNN generates the words of a single sentence. We also show how to transfer knowledge from a dense image captioning [98] task to our model for paragraph generation.

3.4.1 Region Detector

The region detector receives an input image of size $3 \times H \times W$, detects regions of interest, and produces a feature vector of dimension $D = 4096$ for each region. Our region detector follows [98, 191]; we provide a summary here for completeness: The image is resized so that its longest edge is 720 pixels, and is then passed through a convolutional network initialized from the 16-layer VGG network [202]. The resulting feature map is processed by a region proposal network [191], which regresses from a set of anchors to propose regions of interest. These regions are projected onto the convolutional feature map, and the corresponding region of the feature map is reshaped to a fixed size using bilinear interpolation and processed by two fully-connected layers to give a vector of dimension D for each region.

Given a dataset of images and ground-truth regions of interest, the region detector can be trained in an end-to-end fashion as in [191] for object detection and [98] for dense captioning. Since paragraph descriptions do not have annotated groundings to regions of interest, we use a region detector trained for dense image captioning on the Visual Genome dataset [116], using the publicly available implementation of [98]. This produces $M = 50$ detected regions.

One alternative worth noting is to use a region detector trained strictly for object detection, rather than dense captioning. Although such an approach would capture many salient objects in an image, its paragraphs would suffer: an ideal paragraph describes not only objects, but also scenery and relationships, which are better captured by dense captioning task that captures *all* noteworthy elements of a scene.

3.4.2 Region Pooling

The region detector produces a set of vectors $v_1, \ldots, v_M \in \mathbb{R}^D$, each describing a different region in the input image. We wish to aggregate these vectors into a single pooled vector $v_p \in \mathbb{R}^P$ that compactly describes the content of the image. To this end, we learn a projection matrix $W_{pool} \in \mathbb{R}^{P \times D}$ and bias $b_{pool} \in \mathbb{R}^P$; the pooled vector v_p is computed by projecting each region vector using W_{pool} and taking an elementwise maximum, so that $v_p = \max_{i=1}^{M}(W_{pool} v_i + b_{pool})$. While alternative approaches

for representing collections of regions, such as spatial attention [238], may also be possible, we view these as complementary to the model proposed in this chapter; furthermore we note recent work [180] which has proven max pooling sufficient for representing any continuous set function, giving motivation that max pooling does not, in principle, sacrifice expressive power.

3.4.3 Hierarchical Recurrent Network

The pooled region vector $v_p \in \mathbb{R}^P$ is given as input to a hierarchical neural language model composed of two modules: a *sentence RNN* and a *word RNN*. The sentence RNN is responsible for deciding the number of sentences S that should be in the generated paragraph and for producing a P-dimensional *topic vector* for each of these sentences. Given a topic vector for a sentence, the word RNN generates the words of that sentence. We adopt the standard LSTM architecture [79] for both the word RNN and sentence RNN.

As an alternative to this hierarchical approach, one could instead use a non-hierarchical language model to directly generate the words of a paragraph, treating the end-of-sentence token as another word in the vocabulary. Our hierarchical model is advantageous because it reduces the length of time over which the recurrent networks must reason. Our paragraphs contain an average of 67.5 words (Tab. 3.1), so a non-hierarchical approach must reason over dozens of time steps, which is extremely difficult for language models. However, since our paragraphs contain an average of 5.7 sentences, each with an average of 11.9 words, both the paragraph and sentence RNNs need only reason over much shorter time-scales, making learning an appropriate representation much more tractable.

Sentence RNN The sentence RNN is a single-layer LSTM with hidden size $H = 512$ and initial hidden and cell states set to zero. At each time step, it receives the pooled region vector v_p as input, and in turn produces a sequence of hidden states $h_1, \ldots, h_S \in \mathbb{R}^H$, one for each sentence in the paragraph. Each hidden state h_i is used in two ways: First, a linear projection from h_i and a logistic classifier produce a distribution p_i over the two states $\{\texttt{CONTINUE} = 0, \texttt{STOP} = 1\}$ which determine

whether the ith sentence terminates the paragraph. Second, the hidden state h_i is fed to a two-layer fully-connected network to produce the topic vector $t_i \in \mathbb{R}^P$ for the ith sentence of the paragraph, which is the input to the word RNN.

Word RNN The word RNN is a two-layer LSTM with hidden size $H = 512$, which, given a topic vector $t_i \in \mathbb{R}^P$ from the sentence RNN, is responsible for generating the words of a sentence. We follow the input formulation of [224]: the first and second inputs to the RNN are the topic vector and a special `START` token, and subsequent inputs are learned embedding vectors for the words of the sentence. At each timestep the hidden state of the last LSTM layer is used to predict a distribution over the words in the vocabulary, and a special `END` token signals the end of a sentence. After each Word RNN has generated the words of their respective sentences, these sentences are finally concatenated to form the generated paragraph.

3.4.4 Training and Sampling

Training data consists of pairs (x, y), with x an image and y a ground-truth paragraph description for that image, where y has S sentences, the ith sentence has N_i words, and y_{ij} is the jth word of the ith sentence. After computing the pooled region vector v_p for the image, we unroll the sentence RNN for S timesteps, giving a distribution p_i over the $\{\texttt{CONTINUE}, \texttt{STOP}\}$ states for each sentence. We feed the sentence topic vectors to S copies of the word RNN, unrolling the ith copy for N_i timesteps, producing distributions p_{ij} over each word of each sentence. Our training loss $\ell(x, y)$ for the example (x, y) is a weighted sum of two cross-entropy terms: a *sentence loss* ℓ_{sent} on the stopping distribution p_i, and a *word loss* ℓ_{word} on the word distribution p_{ij}:

$$\ell(x, y) = \lambda_{sent} \sum_{i=1}^{S} \ell_{sent}(p_i, \mathbf{I}[i = S]) \tag{3.1}$$

$$+ \lambda_{word} \sum_{i=1}^{S} \sum_{j=1}^{N_i} \ell_{word}(p_{ij}, y_{ij}) \tag{3.2}$$

To generate a paragraph for an image, we run the sentence RNN forward until

the stopping probability $p_i(\texttt{STOP})$ exceeds a threshold $T_{\texttt{STOP}}$ or after S_{MAX} sentences, whichever comes first. We then sample sentences from the word RNN, choosing the most likely word at each timestep and stopping after choosing the `STOP` token or after N_{MAX} words. We set the parameters $T_{\texttt{STOP}} = 0.5$, $S_{MAX} = 6$, and $N_{MAX} = 50$ based on validation set performance.

3.4.5 Transfer Learning

Transfer learning has become pervasive in computer vision. For tasks such as object detection [191] and image captioning [40, 104, 224, 238], it has become standard practice not only to process images with convolutional neural networks, but also to initialize the weights of these networks from weights that had been tuned for image classification, such as the 16-layer VGG network [202]. Initializing from a pre-trained convolutional network allows a form of knowledge transfer from large classification datasets, and is particularly effective on datasets of limited size. Might transfer learning also be useful for paragraph generation?

We propose to utilize transfer learning in two ways. First, we initialize our region detection network from a model trained for dense image captioning [98]; although our model is end-to-end differentiable, we keep this sub-network fixed during training both for efficiency and also to prevent overfitting. Second, we initialize the word embedding vectors, recurrent network weights, and output linear projection of the word RNN from a language model that had been trained on region-level captions [98], fine-tuning these parameters during training to be better suited for the task of paragraph generation. Parameters for tokens not present in the region model are initialized from the parameters for the `UNK` token. This initialization strategy allows our model to utilize linguistic knowledge learned on large-scale region caption datasets [116] to produce better paragraph descriptions, and we validate the efficacy of this strategy in our experiments.

	METEOR	CIDEr	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Sentence-Concat	12.05	6.82	31.11	15.10	7.56	3.98
Template	14.31	12.15	37.47	21.02	12.30	7.38
DenseCap-Concat	12.66	12.51	33.18	16.92	8.54	4.54
Image-Flat ([104])	12.82	11.06	34.04	19.95	12.20	7.71
Regions-Flat-Scratch	13.54	11.14	37.30	21.70	13.07	8.07
Regions-Flat-Pretrained	14.23	12.13	38.32	22.90	14.17	**8.97**
Regions-Hierarchical (ours)	**15.95**	**13.52**	**41.90**	**24.11**	**14.23**	8.69
Human	19.22	28.55	42.88	25.68	15.55	9.66

Table 3.2: Results for generating paragraphs. Our Region-Hierarchical method is compared with six baselines and human performance along six language metrics.

3.5 Experiments

In this section we describe our paragraph generation experiments on the collected data described in Sec. 3.3, which we divide into 14,575 training, 2,487 validation, and 2,489 testing images.

3.5.1 Baselines

Sentence-Concat: To demonstrate the difference between sentence-level and paragraph captions, this baseline samples and concatenates five sentence captions from a model [104] trained on MS COCO captions [138]. The first sentence uses beam search (beam size = 2) and the rest are sampled. The motivation for this is as follows: the image captioning model first produces the sentence that best describes the image as a whole, and subsequent sentences use sampling in order to generate a diverse range of sentences, since the alternative is to repeat the same sentence from beam search. We have validated that this approach works better than using either only beam search or only sampling, as the intent is to make the strongest possible comparison at a task-level to standard image captioning. We also note that, while Sentence-Concat is trained on MS COCO, all images in our dataset are also in MS COCO, and our descriptions were also written by users on Amazon Mechanical Turk.

Image-Flat: This model uses a flat representation for both images and language, and is equivalent to the standard image captioning model NeuralTalk [104]. It takes the whole image as input, and decodes into a paragraph token by token. We use the publically available implementation of [104], which uses the 16-layer VGG network [202] to extract CNN features and projects them as input into an LSTM [79], training the whole model jointly end-to-end.

Template: This method represents a very different approach to generating paragraphs, similar in style to an open-world version of more classical methods like BabyTalk [119], which converts a structured representation of an image into text via a handful of manually specified templates. The first step of our template-based baseline is to detect and describe many regions in a given target image using a pretrained dense captioning model [98], which produces a set of region descriptions tied with bounding boxes and detection scores. The region descriptions are parsed into a set of subjects, verbs, objects, and various modifiers according to part of speech tagging and a handful of TokensRegex [21] rules, which we find suffice to parse the vast majority ($\geq 99\%$) of the fairly simplistic and short region-level descriptions.

Each parsed word is scored by the sum of its detection score and the log probability of the generated tokens in the original region description. Words are then merged into a coherent graph representing the scene, where each node combines all words with the same text and overlapping bounding boxes. Finally, text is generated using the top $N = 25$ scored nodes, prioritizing `subject-verb-object` triples first in generation, and representing all other nodes with existential "there is/are" statements.

DenseCap-Concat: This baseline is similar to Sentence-Concat, but instead concatenates DenseCap [98] predictions as separate sentences in order to form a paragraph. The intent of analyzing this method is to disentangle two key parts of the Template method: captioning and detection (DenseCap), and heuristic recombination into paragraphs. We combine the top $n = 14$ outputs of DenseCap to form DenseCap-Concat's output based on validation CIDEr+METEOR.

Other Baselines: "Regions-Flat-Scratch" uses a flat language model for decoding and initializes it from scratch. The language model input is the projected and pooled region-level image features. "Regions-Flat-Pretrained" uses a pre-trained language model. These baselines are included to show the benefits of decomposing the image into regions and pre-training the language model.

3.5.2 Implementation Details

All baseline neural language models use two layers of LSTM [79] units with 512 dimensions. The feature pooling dimension P is 1024, and we set $\lambda_{sent} = 5.0$ and $\lambda_{word} = 1.0$ based on validation set performance. Training is done via stochastic gradient descent with Adam [108], implemented in Torch. Of critical note is that model checkpoint selection is based on the best combined METEOR and CIDEr score on the validation set – although models tend to minimize validation loss fairly quickly, it takes much longer training for METEOR and CIDEr scores to stop improving.

3.5.3 Main Results

We present our main results at generating paragraphs in Tab. 3.2, which are evaluated across six language metrics: CIDEr [223], METEOR [36], and BLEU-{1,2,3,4} [172]. The Sentence-Concat method performs poorly, achieving the lowest scores across all metrics. Its lackluster performance provides further evidence of the stark differences between single-sentence captioning and paragraph generation. Surprisingly, the hard-coded template-based approach performs reasonably well, particularly on CIDEr, METEOR, and BLEU-1, where it is competitive with some of the neural approaches. This makes sense: the template approach is provided with a strong prior about image content since it receives region-level captions [98] as input, and the many expletive "there is/are" statements it makes, though uninteresting, are safe, resulting in decent scores. However, its relatively poor performance on BLEU-3 and BLEU-4 highlights the limitation of reasoning about regions in isolation – it is unable to produce much text relating regions to one another, and further suffers from a lack of "connective tissue" that transforms paragraphs from a series of disconnected thoughts into a

Sentence-Concat	Template	Regions-Hierarchical
A red double decker bus parked in a field. A double decker bus that is parked at the side of two and a road. A blue bus in the middle of a grand house. A new camera including a pinstripe boys and red white and blue outside. A large blue double decker bus with a front of a picture with its passengers in the.	There is a yellow and white bus, and a front wheel of a bus. There is a clear and blue sky, and a front wheel of a bus. There is a bus, and windows. There is a number on a train, and a white and red sign. There is a tire of a truck.	There are two buses driving in the road. There is a yellow bus on the road with white lines painted on it. It is stopped at the bus stop and a person is passing by it. In front of the bus there is a black and white bus.
A man riding a horse drawn carriage down a street. Post with two men ride on the back of a wagon with large elephants. A man is on top of a horse in a wooden track. A person sitting on a bench with two horses in a street. The horse sits on a garage while he looks like he is traveling in.	People are riding a horse, and a man in a white shirt is sitting on a bench. People are sitting on a bench, and there is a wheel of a bicycle. There is a building with windows, and an blue umbrella. There are parked wheels, and a wheel. There is a brick.	A man is riding a carriage on a street. Two people are sitting on top of the horses. The carriage is made of wood. The carriage is black. The carriage has a white stripe down the side. The building in the background is a tan color.
Two giraffes standing in a fenced in area. A big giraffe is is reading a tree. A giraffe sniffing the ground with its head. A couple of giraffe standing next to each other. Two giraffes are shown behind a fence and a fence.	Giraffes are standing in a field, and there is a standing giraffe. Tall green trees behind a fence are behind a fence, and there is a neck of a giraffe. There is a green grass, and a giraffe. There is a trunk of a tree, and a brown fence. there is a tree trunk, and white letters.	A giraffe is standing next to a tree. There is a pole with some green leaves on it to the right. There is a white and black brick building behind the fence. there are a bunch of trees and bushes as well.
A young girl is playing with a frisbee. Man on a field with an orange frisbee. A woman holds a frisbee on a bench on a sunny day. A young girl is holding a green green frisbee. A girl throwing a frisbee in a park.	A girl is holding a tennis racket, and there is a green and brown grass. There is a pink shirt on a woman, and the background. The woman with a hair is wearing blue shorts, and there are red flowers. There are trees, and a blue frisbee in an air.	A woman in a red shirt and a black short short sleeve red shorts is holding a yellow frisbee. She is wearing a green shirt and white pants. She is wearing a pink shirt and short sleeve skirt. In her hand she is holding a white frisbee and a hand can be seen through it. Behind her are two white chairs. In the background is a large green and white building.

Figure 3.3: Example paragraph generation results for our model (Regions-Hierarchical) and the Sentence-Concat and Template baselines. The first three rows are positive results and the last row is a failure case.

coherent whole. DenseCap-Concat scores worse than Template on all metrics except CIDEr, illustrating the necessity of Template's caption parsing and recombination.

Image-Flat, trained on the task of paragraph generation, outperforms Sentence-Concat, and the region-based reasoning of Regions-Flat-Scratch improves results further on all metrics. Pre-training results in improvements on all metrics, and our full model, Regions-Hierarchical, achieves the highest scores among all methods on every metric except BLEU-4. One hypothesis for the mild superiority of Regions-Flat-Pretrained on BLEU-4 is that it is better able to reproduce words immediately at the end and beginning of sentences more exactly due to their non-hierarchical structure, providing a slight boost in BLEU scores.

To make these metrics more interpretable, we performed a human evaluation by collecting an additional paragraph for 500 randomly chosen images, with results in the last row of Tab. 3.2. As expected, humans produce superior descriptions to any automatic method, performing better on all language metrics considered. Of particular note is the large gap between humans our the best model on CIDEr and METEOR, which are both designed to correlate well with human judgment [36, 223].

Finally, we note that we have also tried the SPICE evaluation metric [3], which has shown to correlate well with human judgements for sentence-level image captioning. Unfortunately, SPICE does not seem well-suited for evaluating long paragraph descriptions – it does not handle coreference or distinguish between different instances of the same object category. These are reasonable design decisions for sentence-level captioning, but is less applicable to paragraphs. In fact, human paragraphs achieved a considerably lower SPICE score than automated methods.

3.5.4 Qualitative Results

We present qualitative results from our model and the Sentence-Concat and Template baselines in Fig. 3.3. Some interesting properties of our model's predictions include its use of coreference in the first example ("a bus", "it", "the bus") and its ability to capture relationships between objects in the second example. Also of note is the order in which our model chooses to describe the image: the first sentence tends to be fairly high level, middle sentences give some details about scene elements mentioned earlier in the description, and the last sentence often describes something in the background, which other methods are not able to capture. Anecdotally, we observed that this follows the same order with which most humans tended to describe images.

The failure case in the last row highlights another interesting phenomenon: even though our model was wrong about the semantics of the image, calling the girl "a woman", it has learned that "woman" is consistently associated with female pronouns ("she", "she", "her hand", "behind her").

It is also worth noting the general behavior of the two baselines. Paragraphs from Sentence-Concat tend to be repetitive in sentence structure and are often simply inaccurate due to the sampling required to generate multiple sentences. On the other hand, the Template baseline is largely accurate, but has uninteresting language and lacks the ability to determine which things are most important to describe. In contrast, Regions-Hierarchical stays relevant and furthermore exhibits more interesting patterns of language.

	Average Length	Std. Dev. Length	Diversity	Nouns	Verbs	Pronouns	Vocab Size
Sentence-Concat	56.18	4.74	34.23	32.53	9.74	0.95	2993
Template	60.81	7.01	45.42	23.23	11.83	0.00	422
Regions-Hierarchical	70.47	17.67	40.95	24.77	13.53	2.13	1989
Human	67.51	25.95	69.92	25.91	14.57	2.42	4137

Table 3.3: Language statistics of test set predictions. Part of speech statistics are given as percentages, and diversity is calculated as in Section 3.3. "Vocab Size" indicates the number of unique tokens output across the entire test set, and human numbers are calculated from ground truth. Note that the diversity score for humans differs slightly from the score in Tab. 3.1, which is calculated on the entire dataset.

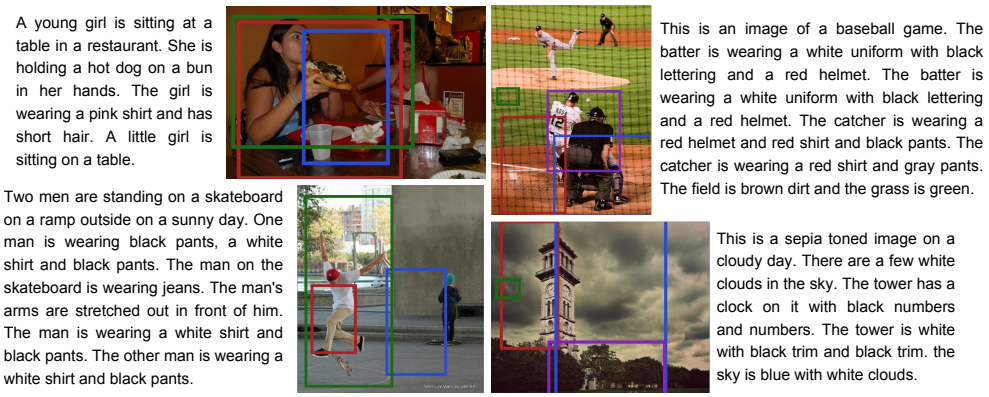


Figure 3.4: Examples of paragraph generation from only a few regions. Since only a small number of regions are used, this data is out of sample for the model, but it is still able to focus on the regions of interest while ignoring the rest of the image.

3.5.5 Paragraph Language Analysis

To shed a quantitative light on the linguistic phenomena generated, in Tab. 3.3 we show statistics of the language produced by a representative spread of methods.

Our hierarchical approach generates text of similar average length and variance as human descriptions, with Sentence-Concat and the Template approach somewhat shorter and less varied in length. Sentence-Concat is also the least diverse method, though all automatic methods remain far less diverse than human sentences, indicating ample space for improvement. According to this diversity metric, the Template approach is actually the most diverse automatic method, which may be attributed to how the method is hard-coded to sequentially describe each region in the scene in

turn, regardless of importance or how interesting such an output may be (see Fig. 3.3). While both our hierarchical approach and the Template method produce text with similar portions of nouns and verbs as human paragraphs, only our approach was able to generate a reasonable quantity of pronouns. Our hierarchical method also had a much wider vocabulary compared to the Template approach, though Sentence-Concat, trained on hundreds of thousands of MS COCO [138] captions, is a bit larger.

3.5.6 Generating Paragraphs from Fewer Regions

As an exploratory experiment in order to highlight the interpretability of our model, we investigate generating paragraphs from a smaller number of regions than the $M = 50$ used in the rest of this work. Instead, we only give our method access to the top few detected regions as input, with the hope that the generated paragraph focuses only on those particularly regions, preferring not to describe other parts of the image. The results for a handful of images are shown in Fig. 3.4. Although the input is extremely out of sample compared to the training data, the results are still quite reasonable – the model generates paragraphs describing the detected regions without much mention of objects or scenery outside of the detections. Taking the top-right image as an example, despite a few linguistic mistakes, the paragraph generated by our model mentions the batter, catcher, dirt, and grass, which all appear in the top detected regions, but does not pay heed to the pitcher or the umpire in the background.

3.6 Discussion

In this chapter we have introduced the task of describing images with long, descriptive paragraphs, and presented a hierarchical approach for generation that leverages the compositional structure of both images and language. We have shown that paragraph generation is different from traditional image captioning and have tailored our model to suit these differences. Experimentally, we have demonstrated the advantages of our approach over prior methods and shown how region-level knowledge can be effectively transferred to paragraph captioning. We have also demonstrated the benefits of our

model in interpretability, generating descriptive paragraphs using only a few image regions. We anticipate further opportunities for knowledge transfer at the intersection of vision and language, and project that visual and lingual compositionality will continue to lie at the heart of effective paragraph generation.

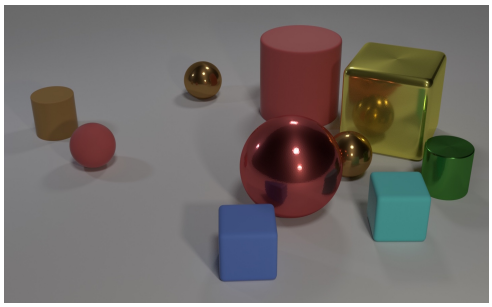
Chapter 4

A Dataset for Compositional Visual Reasoning

4.1 Introduction

A long-standing goal of artificial intelligence research is to develop systems that can reason and answer questions about visual information. Recently, several datasets have been introduced to study this problem [7, 55, 116, 146, 190, 248, 259]. Each of these Visual Question Answering (VQA) datasets contains challenging natural language questions about images. Correctly answering these questions requires perceptual abilities such as recognizing objects, attributes, and spatial relationships as well as higher-level skills such as counting, performing logical inference, making comparisons, or leveraging commonsense world knowledge [184]. Numerous methods have attacked these problems [4, 5, 54, 143, 241], but many show only marginal improvements over strong baselines [7, 85, 257]. Unfortunately, our ability to understand the limitations of these methods is impeded by the inherent complexity of the VQA task. Are methods hampered by failures in recognition, poor reasoning, lack of commonsense knowledge, or something else?

The difficulty of understanding a system's competences is exemplified by *Clever Hans*, a 1900s era horse who appeared to be able to answer arithmetic questions.



Q: Are there an equal number of large things and metal spheres?
Q: What size is the cylinder that is left of the brown metal thing that is left of the big sphere? **Q:** There is a sphere with the same size as the metal cube; is it made of the same material as the small red sphere?
Q: How many objects are either small cylinders or metal things?

Figure 4.1: A sample image and questions from CLEVR. Questions test aspects of visual reasoning such as attribute identification, counting, comparison, multiple attention, and logical operations.

Careful observation revealed that Hans was correctly "answering" questions by reacting to cues read off his human observers [177]. Statistical learning systems, like those used for VQA, may develop similar "cheating" approaches to superficially "solve" tasks without learning the underlying reasoning processes [209, 210]. For instance, a statistical learner may correctly answer the question *"What covers the ground?"* not because it understands the scene but because biased datasets often ask questions about the *ground* when it is snow-covered [1, 254]. How can we determine whether a system is capable of sophisticated reasoning and not just exploiting biases of the world, similar to Clever Hans?

In this chapter we propose a *diagnostic dataset* for studying the ability of VQA systems to perform visual reasoning. We refer to this dataset as the Compositional Language and Elementary Visual Reasoning diagnostics dataset (CLEVR; pronounced as *clever* in homage to Hans). CLEVR contains 100k rendered images and about one million automatically-generated questions, of which 853k are unique. It has challenging images and questions that test visual reasoning abilities such as counting, comparing, logical reasoning, and storing information in memory, as illustrated in Figure 4.1.

We designed CLEVR with the explicit goal of enabling detailed analysis of visual reasoning. Our images depict simple 3D shapes; this simplifies recognition and allows us to focus on *reasoning skills*. We ensure that the information in each image is *complete and exclusive* so that external information sources, such as commonsense knowledge, cannot increase the chance of correctly answering questions. We minimize

question-conditional bias via rejection sampling within families of related questions, and avoid degenerate questions that are seemingly complex but contain simple shortcuts to the correct answer. Finally, we use structured ground-truth representations for both images and questions: images are annotated with ground-truth object positions and attributes, and questions are represented as *functional programs* that can be executed to answer the question (see Section 4.3). These representations facilitate in-depth analyses not possible with traditional VQA datasets.

These design choices also mean that while images in CLEVR may be visually simple, its questions are complex and require a range of reasoning skills. For instance, factorized representations may be required to generalize to unseen combinations of objects and attributes. Tasks such as counting or comparing may require short-term memory [79] or attending to specific objects [143, 241]. Questions that combine multiple subtasks in diverse ways may require compositional systems [4, 5] to answer.

We use CLEVR to analyze a suite of VQA models and discover weaknesses that are not widely known. For example, we find that current state-of-the-art VQA models struggle on tasks requiring *short term memory*, such as comparing the attributes of objects, or *compositional reasoning*, such as recognizing novel attribute combinations. These observations point to novel avenues for further research.

Finally, we stress that accuracy on CLEVR is not an end goal in itself: a handcrafted system with explicit knowledge of the CLEVR universe might work well, but will not generalize to real-world settings. Therefore CLEVR should be used *in conjunction* with other VQA datasets in order to study the reasoning abilities of general VQA systems.

The CLEVR dataset, as well as code for generating new images and questions, is publicly available at `https://cs.stanford.edu/people/jcjohns/clevr/`.

4.2 Related Work

In recent years, a range of benchmarks for visual understanding have been proposed, including datasets for image captioning [24, 48, 138, 246], referring to objects [107], relational graph prediction [116], and visual Turing tests [60, 147]. CLEVR, our

diagnostic dataset, is most closely related to benchmarks for visual question answering [7, 55, 116, 146, 190, 218, 248, 259], as it involves answering natural-language questions about images. The two main differences between CLEVR and other VQA datasets are that: (1) CLEVR minimizes biases of prior VQA datasets that can be used by learning systems to answer questions correctly without visual reasoning and (2) CLEVR's synthetic nature and detailed annotations facilitate in-depth analyses of reasoning abilities that are impossible with existing datasets.

Prior work has attempted to mitigate biases in VQA datasets in simple cases such as yes/no questions [60, 254], but it is difficult to apply such approaches to more complex questions without a high-quality semantic representation of both questions and answers. In CLEVR, this semantic representation is provided by the functional program underlying each image-question pair, and biases are largely eliminated via sampling. Winograd schemas [130] are another approach for controlling bias in question answering: these questions are carefully designed to be ambiguous based on syntax alone and require commonsense knowledge. Unfortunately this approach does not scale gracefully: the first phase of the 2016 Winograd Schema Challenge consists of just 60 hand-designed questions. CLEVR is also related to the bAbI question answering tasks [227] in that it aims to diagnose a set of clearly defined competences of a system, but CLEVR focuses on visual reasoning whereas bAbI is purely textual.

We are also not the first to consider synthetic data for studying (visual) reasoning. SHRDLU performed simple, interactive visual reasoning with the goal of moving specific objects in the visual scene [230]; this study was one of the first to demonstrate the brittleness of manually programmed semantic understanding. The pioneering DAQUAR dataset [148] contains both synthetic and human-written questions, but they only generate 420 synthetic questions using eight text templates. VQA [7] contains 150,000 natural-language questions about abstract scenes [261], but these questions do not control for question-conditional bias and are not equipped with functional program representations. CLEVR is similar in spirit to the SHAPES dataset [4], but it is more complex and varied both in terms of visual content and question variety and complexity: SHAPES contains 15,616 total questions with just 244 unique questions while CLEVR contains nearly a million questions of which 853,554 are unique.

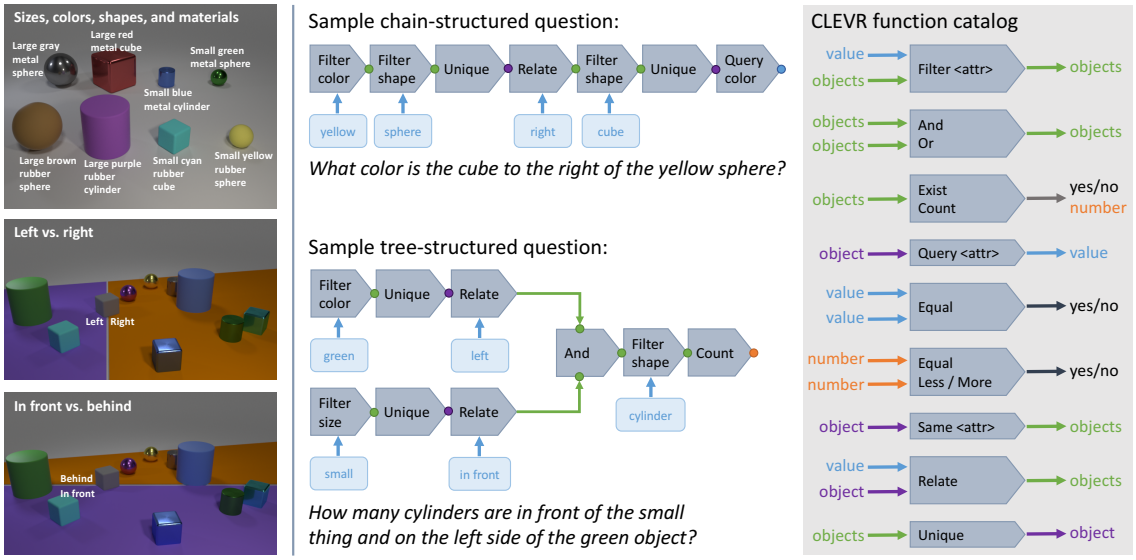


Figure 4.2: A field guide to the CLEVR universe. **Left:** Shapes, attributes, and spatial relationships. **Center:** Examples of questions and their associated functional programs. **Right:** Catalog of basic functions used to build questions. See Section 4.3 for details.

4.3 The CLEVR Diagnostic Dataset

CLEVR provides a dataset that requires complex reasoning to solve and that can be used to conduct rich diagnostics to better understand the visual reasoning capabilities of VQA systems. This requires tight control over the dataset, which we achieve by using synthetic images and automatically generated questions. The images have associated ground-truth object locations and attributes, and the questions have an associated machine-readable form. These ground-truth structures allow us to analyze models based on, for example: question type, question topology (chain *vs.* tree), question length, and various forms of relationships between objects. Figure 4.2 gives a brief overview of the main components of CLEVR, which we describe in detail below.

Objects and relationships. The CLEVR universe contains three object shapes (cube, sphere, and cylinder) that come in two absolute sizes (small and large), two

materials (shiny "metal" and matte "rubber"), and eight colors. Objects are spatially related via four relationships: "left", "right", "behind", and "in front". The semantics of these prepositions are complex and depend not only on relative object positions but also on camera viewpoint and context. We found that generating questions that invoke spatial relationships with semantic accord was difficult. Instead we rely on a simple and unambiguous definition: projecting the camera viewpoint vector onto the ground plane defines the "behind" vector, and one object is behind another if its ground-plane position is further along the "behind" vector. The other relationships are similarly defined. Figure 4.2 (left) illustrates the objects, attributes, and spatial relationships in CLEVR. The CLEVR universe also includes one non-spatial relationship type that we refer to as the *same-attribute relation*. Two objects are in this relationship if they have equal attribute values for a specified attribute.

Scene representation. Scenes are represented as collections of objects annotated with shape, size, color, material, and position on the ground-plane. A scene can also be represented by a *scene graph* [99, 116], where nodes are objects annotated with attributes and edges connect spatially related objects. A scene graph contains all ground-truth information for an image and could be used to replace the vision component of a VQA system with *perfect sight*.

Image generation. CLEVR images are generated by randomly sampling a scene graph and rendering it using Blender [14]. Every scene contains between three and ten objects with random shapes, sizes, materials, colors, and positions. When placing objects we ensure that no objects intersect, that all objects are at least partially visible, and that there are small horizontal and vertical margins between the image-plane centers of each pair of objects; this helps reduce ambiguity in spatial relationships. In each image the positions of the lights and camera are randomly jittered.

Question representation. Each question in CLEVR is associated with a *functional program* that can be *executed* on an image's scene graph, yielding the answer

Split	Images	Questions	Unique questions	Overlap with train
Total	100,000	999,968	853,554	-
Train	70,000	699,989	608,607	-
Val	15,000	149,991	140,448	17,338
Test	15,000	149,988	140,352	17,335

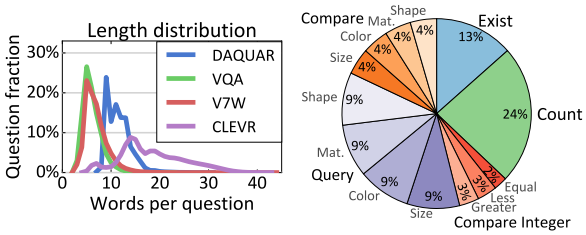


Figure 4.3: **Left:** Statistics for CLEVR; the majority of questions are unique and few questions from the val and test sets appear in the training set. **Center:** Comparison of question lengths for different VQA datasets; CLEVR questions are generally much longer. **Right:** Distribution of question types in CLEVR.

to the question. Functional programs are built from simple basic functions that correspond to elementary operations of visual reasoning such as *querying* object attributes, *counting* sets of objects, or *comparing* values. As shown in Figure 4.2, complex questions can be represented by compositions of these simple building blocks. Full details about each basic function can be found in Appendix A.

As we will see in Section 4.4, representing questions as functional programs enables rich analysis that would be impossible with natural-language questions. A question's program tells us exactly which reasoning abilities are required to solve it, allowing us to compare performance on questions requiring different types of reasoning.

We categorize questions by *question type*, defined by the outermost function in the question's program; for example the questions in Figure 4.2 have types *query-color* and *exist*. Figure 4.3 shows the number of questions of each type.

Question families. We must overcome several key challenges to generate a VQA dataset using functional programs. Functional building blocks can be used to construct an infinite number of possible functional programs, and we must decide which program structures to consider. We also need a method for converting functional programs to natural language in a way that minimizes question-conditional bias. We solve these problems using *question families*.

A question family contains a template for constructing functional programs and several text templates providing multiple ways of expressing these programs in natural language. For example, the question *"How many red things are there?"* can be formed

by instantiating the text template *"How many <C> <M> things are there?"*, binding the parameters <*C*> and <*M*> (with types "color" and "material") to the values *red* and `nil`. The functional program *count(filter_color(red, scene()))* for this question can be formed by instantiating the associated program template

count(filter_color(<C>, filter_material(<M>, scene())))

with the same values, using the convention that functions taking a `nil` input are removed after instantiation.

CLEVR contains a total of 90 question families, each with a single program template and an average of four text templates. Text templates were generated by manually writing one or two templates per family and then crowdsourcing question rewrites. To further increase language diversity we use a set of synonyms for each shape, color, and material. With up to 19 parameters per template, a small number of families can generate a huge number of unique questions; Figure 4.3 shows that of the nearly one million questions in CLEVR, more than 853k are unique. CLEVR can easily be extended by adding new question families.

Question generation. Generating questions is conceptually simple: we choose a question family, select values for each of its template parameters, execute the resulting program on the image's scene graph to find the answer, and use one of the text templates from the question family to generate the final natural-language question.

However, many combinations of values give rise to questions which are either *ill-posed* or *degenerate*. The question *"What color is the cube to the right of the sphere?"* would be *ill-posed* if there were many cubes right of the sphere, or *degenerate* if there were only one cube in the scene since the reference to the sphere would then be unnecessary. Avoiding such ill-posed and degenerate questions is critical to ensure the correctness and complexity of our questions.

A naïve solution is to randomly sample combinations of values and reject those which lead to ill-posed or degenerate questions. However, the number of possible configurations for a question family is exponential in its number of parameters, and most of them are undesirable. This makes brute-force search intractable for our complex question families.

Instead, we employ a depth-first search to find valid values for instantiating question families. At each step of the search, we use ground-truth scene information to prune large swaths of the search space which are guaranteed to produce undesirable questions; for example we need not entertain questions of the form *"What color is the <S> to the <R> of the sphere"* for scenes that do not contain spheres.

Finally, we use rejection sampling to produce an approximately uniform answer distribution for each question family; this helps minimize question-conditional bias since all questions from the same family share linguistic structure.

4.4 VQA Systems on CLEVR

4.4.1 Models

VQA models typically represent images with features from pretrained CNNs and use word embeddings or recurrent networks to represent questions and/or answers. Models may train recurrent networks for answer generation [55, 148, 232], multiclass classifiers over common answers [7, 143, 144, 190, 257, 259], or binary classifiers on image-question-answer triples [54, 85, 201]. Many methods incorporate attention over the image [54, 201, 237, 241, 259] or question [143]. Some methods incorporate memory [235] or dynamic network architectures [4, 5].

Experimenting with all methods is logistically challenging, so we reproduced a representative subset of methods: baselines that do not look at the image (Q-type mode, LSTM), a simple baseline (CNN+BoW) that performs near state-of-the-art [85, 257], and more sophisticated methods using recurrent networks (CNN+LSTM), sophisticated feature pooling (CNN+LSTM+MCB), and spatial attention (CNN+LSTM+SA).[1] These are described in detail below.

Q-type mode: Similar to the "per Q-type prior" method in [7], this baseline predicts the most frequent training-set answer for each question's type.

LSTM: Similar to "LSTM Q" in [7], the question is processed with learned word

[1] We performed initial experiments with dynamic module networks [5] but its parsing heuristics did not generalize to the complex questions in CLEVR so it did not work out-of-the-box; see Appendix A.

embeddings followed by a word-level LSTM [79]. The final LSTM hidden state is passed to a multi-layer perceptron (MLP) that predicts a distribution over answers. This method uses no image information so it can only model question-conditional bias.

CNN+BoW: Following [257], the question is encoded by averaging word vectors for each word in the question and the image is encoded using features from a convolutional network (CNN). The question and image features are concatenated and passed to a MLP which predicts a distribution over answers. We use word vectors trained on the GoogleNews corpus [154]; these are not fine-tuned during training.

CNN+LSTM: As above, images and questions are encoded using CNN features and final LSTM hidden states, respectively. These features are concatenated and passed to an MLP that predicts an answer distribution.

CNN+LSTM+MCB: Images and questions are encoded as above, but instead of concatenation, their features are pooled using compact multimodal pooling (MCB) [54, 56].

CNN+LSTM+SA: Again, the question and image are encoded using a CNN and LSTM, respectively. Following [241], these representations are combined using one or more rounds of soft spatial attention and the final answer distribution is predicted with an MLP.

Human: We used Mechanical Turk to collect human responses for 5500 random questions from the test set, taking a majority vote among three workers.

Implementation details. Our CNNs are ResNet-101 models pretrained on ImageNet [75] that are not finetuned; images are resized to 224×224 prior to feature extraction. CNN+LSTM+SA extracts features from the last layer of the `conv4` stage, giving $14 \times 14 \times 1024$-dimensional features. All other methods extract features from the final average pooling layer, giving 2048-dimensional features. LSTMs use one or two layers with 512 or 1024 units per layer. MLPs use ReLU functions and dropout [207]; they have one or two hidden layers with between 1024 and 8192 units per layer. All models are trained using Adam [108].

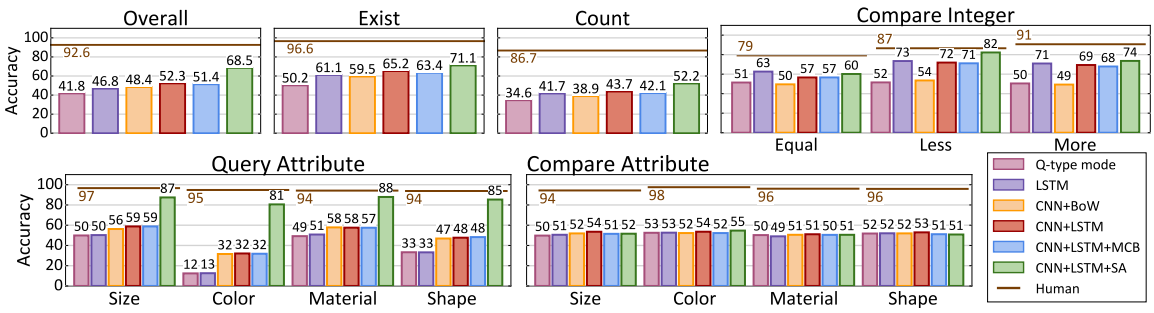


Figure 4.4: Accuracy per question type of the six VQA methods on the CLEVR dataset (higher is better). Figure best viewed in color.

Experimental protocol. CLEVR is split into train, validation, and test sets (see Figure 4.3). We tuned hyperparameters (learning rate, dropout, word vector size, number and size of LSTM and MLP layers) independently per model based on the validation error. All experiments were designed on the validation set; after finalizing the design we ran each model once on the test set. *All experimental findings generalized from the validation set to the test set.*

4.4.2 Analysis by Question Type

We can use the program representation of questions to analyze model performance on different forms of reasoning. We first evaluate performance on each question type, defined as the outermost function in the program. Figure 4.4 shows results and detailed findings are discussed below.

Querying attributes: Query questions ask about an attribute of a particular object (*eg. "What color is the thing right of the red sphere?"*). The CLEVR world has two sizes, eight colors, two materials, and three shapes. On questions asking about these different attributes, Q-type mode and LSTM obtain accuracies close to 50%, 12.5%, 50%, and 33.3% respectively, showing that the dataset has minimal question-conditional bias for these questions. CNN+LSTM+SA substantially outperforms all other models on these questions; its attention mechanism may help it focus on the target object and identify its attributes.

Comparing attributes: Attribute comparison questions ask whether two objects have the same value for some attribute (*eg. "Is the cube the same size as the sphere?"*). The only valid answers are "yes" and "no". Q-Type mode and LSTM achieve accuracies close to 50%, confirming there is no dataset bias for these questions. Unlike attribute-query questions, attribute-comparison questions require a limited form of memory: models must identify the attributes of two objects and keep them in memory to compare them. Interestingly, none of the models are able to do so: all models have an accuracy of approximately 50%. This is also true for the CNN+LSTM+SA model, suggesting that its attention mechanism is not capable of attending to two objects at once to compare them. This illustrates how CLEVR can reveal limitations of models and motivate follow-up research, *eg.*, augmenting attention models with explicit memory.

Existence: Existence questions ask whether a certain type of object is present (*eg.*, *"Are there any cubes to the right of the red thing?"*). The 50% accuracy of Q-Type mode shows that both answers are *a priori* equally likely, but the LSTM result of 60% does suggest a question-conditional bias. There may be correlations between question length and answer: questions with more filtering operations (*eg.*, "large red cube" *vs.* "red cube") may be more likely to have "no" as the answer. Such biases may be present even with uniform answer distributions per question family, since questions from the same family may have different numbers of filtering functions. CNN+LSTM(+SA) outperforms LSTM, but its performance is still quite low.

Counting: Counting questions ask for the number of objects fulfilling some conditions (*eg. "How many red cubes are there?"*); valid answers range from zero to ten. Images have three and ten objects and counting questions refer to subsets of objects, so ensuring a uniform answer distribution is very challenging; our rejection sampler therefore pushes towards a uniform distribution for these questions rather than enforcing it as a hard constraint. This results in a question-conditional bias, reflected in the 35% and 42% accuracies achieved by Q-type mode and LSTM. CNN+LSTM(+MCB) performs on par with LSTM, suggesting that CNN features contain little information relevant to counting. CNN+LSTM+SA performs slightly better, but at 52% its absolute performance is low.

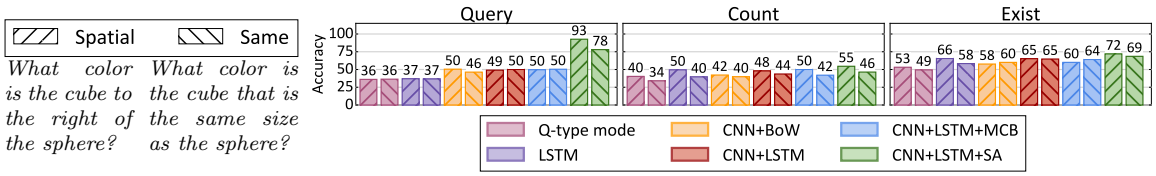


Figure 4.5: Accuracy on questions with a single *spatial relationship vs.* a single *same-attribute* relationship. For *query* and *count* questions, models generally perform worse on questions with *same-attribute* relationships. Results on *exist* questions are mixed.

Integer comparison: Integer comparison questions ask which of two object sets is larger (*eg. "Are there fewer cubes than red things?"*); this requires counting, memory, and comparing integer quantities. The answer distribution is unbiased (see Q-Type mode) but a set's size may correlate with the length of its description, explaining the gap between LSTM and Q-type mode. CNN+BoW performs at chance: BoW mixes the words describing each set, making it impossible for the learner to discriminate between them. CNN+LSTM+SA outperforms LSTM on "less" and "more" questions, but no model outperforms LSTM on "equal" questions. Most models perform better on "less" than "more" due to asymmetric question families.

4.4.3 Analysis by Relationship Type

CLEVR questions contain two types of relationships: *spatial* and *same-attribute* (see Section 4.3). We can compare the relative difficulty of these two types by comparing model performance on questions with a single spatial relationship and questions with a single same-attribute relationship; results are shown in Figure 4.5. On query-attribute and counting questions we see that same-attribute questions are generally more difficult; the gap between CNN+LSTM+SA on spatial and same-relate query questions is particularly large (93% *vs.* 78%). Same-attribute relationships may require a model to keep attributes of one object "in memory" for comparison, suggesting again that models augmented with explicit memory may perform better on these questions.

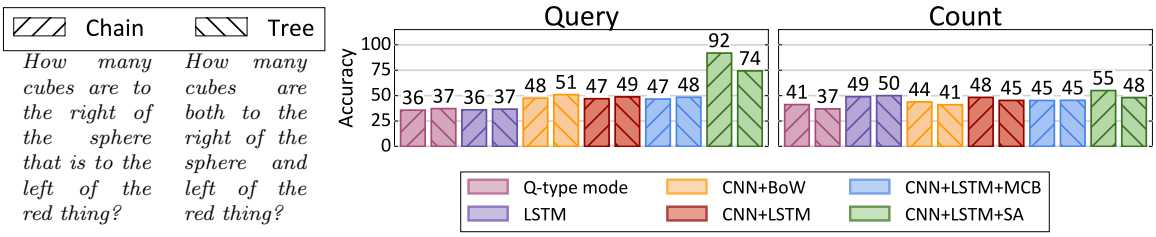


Figure 4.6: Accuracy on questions with two spatial relationships, broken down by question topology: chain-structured questions *vs.* tree-structured questions joined with a logical AND operator.

4.4.4 Analysis by Question Topology

We next evaluate model performance on different question topologies: *chain-structured* questions *vs. tree-structured* questions with two branches joined by a logical AND (see Figure 4.2). In Figure 4.6, we compare performance on chain-structured questions with two spatial relationships vs. tree-structured questions with one relationship along each branch. On query questions, CNN+LSTM+SA shows a large gap between chain and tree questions (92% *vs.* 74%); on count questions, CNN+LSTM+SA slightly outperforms LSTM on chain questions (55% *vs.* 49%) but no method outperforms LSTM on tree questions. Tree questions may be more difficult since they require models to perform two subtasks in parallel before fusing their results.

4.4.5 Effect of Question Size

Intuitively, longer questions should be harder since they involve more reasoning steps. We define a question's *size* to be the number of functions in its program, and in Figure 4.7 we show accuracy on query-attribute questions as a function of question size.[2] Surprisingly accuracy appears unrelated to question size.

However, many questions can be correctly answered even when some subtasks are not solved correctly. For example, the question in Figure 4.7 (top) can be answered correctly without identifying the correct large blue cylinder, because all large objects

[2]We exclude questions with same-attribute relations since their max size is 10, introducing unwanted correlations between size and difficulty. Excluded questions show the same trends (see Appendix A).

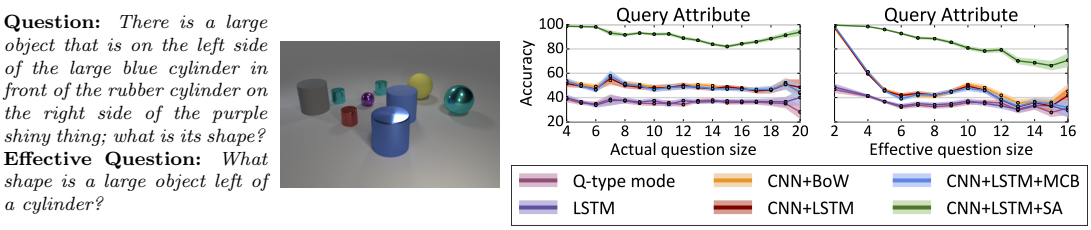


Figure 4.7: **Left**: Many questions can be answered correctly without correctly solving all subtasks. For a given question and scene we can prune functions from the question's program to generate an *effective question* which is shorter but gives the same answer. **Right**: Accuracy on query questions *vs.* actual and effective question size. Accuracy decreases with effective question size but not with actual size. Shaded area shows a 95% confidence interval.

left of a cylinder are cylinders.

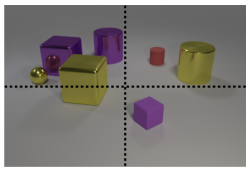
To quantify this effect, we define the *effective question* of an image-question pair: we prune functions from the question's program to find the smallest program that, when executed on the scene graph for the question's image, gives the same answer as the original question.[3] A question's *effective size* is the size of its effective question. Questions whose effective size is smaller than their actual size need not be degenerate. The question in Figure 4.7 is not degenerate because the entire question is needed to resolve its object references (there are two blue cylinders and two rubber cylinders), but it has a small effective size since it can be correctly answered without resolving those references.

In Figure 4.7 (bottom), we show accuracy on query questions as a function of effective question size. The error rate of all models increases with effective question size, suggesting that models struggle with long reasoning chains.

4.4.6 Spatial Reasoning

We expect that questions with more spatial relationships should be more challenging since they require longer chains of reasoning. The top set of plots in Figure 4.8

[3]Pruned questions may be *ill-posed* (Section 4.3) so they are executed with modified semantics; see Appendix A for details.



Question: *There is a purple cube that is in front of the yellow metal sphere; what material is it?*
Absolute question: *There is a purple cube in the front half of the image; what material is it?*

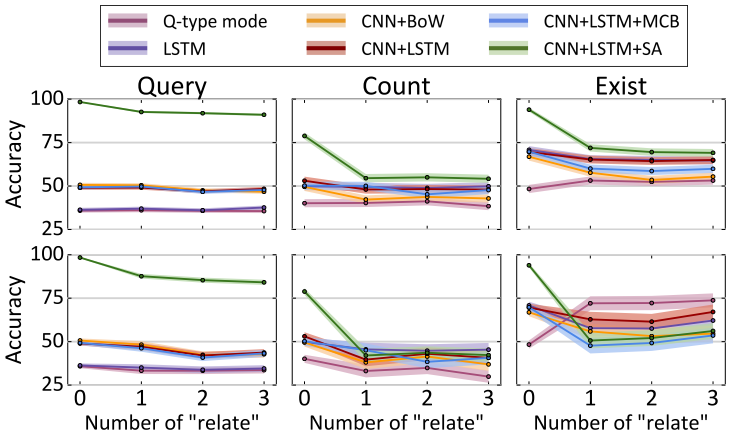


Figure 4.8: **Left**: Some questions can be correctly answered using *absolute* definitions for spatial relationships; for example in this image there is only one purple cube in the bottom half of the image. **Right**: Accuracy of each model on *chain-structured* questions as a function of the number of spatial relationships in the question, separated by question type. Top row shows all chain-structured questions; bottom row excludes questions that can be correctly answered using absolute spatial reasoning.

shows accuracy on chain-structured questions with different numbers of relationships.[4] Across all three question types, CNN+LSTM+SA shows a significant drop in accuracy for questions with one or more spatial relationship; other models are largely unaffected by spatial relationships.

Spatial relationships require reasoning about relative object positions. However, as shown in Figure 4.8, some questions can be answered using *absolute spatial reasoning*. In this question the purple cube can be found by simply looking in the bottom half of the image; reasoning about its position relative to the metal sphere is unnecessary.

Questions only requiring absolute spatial reasoning can be identified by modifying the semantics of spatial relationship functions in their programs: instead of returning sets of objects related to the input object, they ignore their input object and return the set of objects in the half of the image corresponding to the relationship. A question only requires absolute spatial reasoning if executing its program with these modified semantics does not change its answer.

[4]We restrict to chain-structured questions to avoid unwanted correlations between question topology and number of relationships.

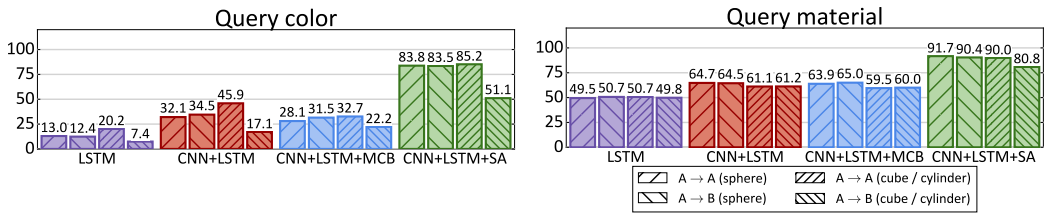


Figure 4.9: In *Condition A* all cubes are gray, blue, brown, or yellow and all cylinders are red, green, purple, or cyan; in *Condition B* color palettes are swapped. We train models in Condition A and test in both conditions to assess their generalization performance. We show accuracy on "query color" and "query material" questions, separating questions by shape of the object being queried.

The bottommost plots of Figure 4.8 show accuracy on chain-structured questions with different number of relationships, *excluding* questions that can be answered with absolute spatial reasoning. On query questions, CNN+LSTM+SA performs worse when absolute spatial reasoning is excluded; on count questions no model outperforms LSTM, and on exist questions no model outperforms Q-type mode. These results suggest that models have not learned the semantics of spatial relationships.

4.4.7 Compositional Generalization

Practical systems should perform well on images and questions that contain novel combinations of attributes not seen during training. To do so models might need to learn disentangled representations for attributes, *eg.* learning separate representations for color and shape instead of memorizing all possible color/shape combinations.

We retrain models on Condition A and compare their performance when testing on Condition A (A $\rightarrow$ A) or Condition B (A $\rightarrow$ B). In Figure 4.9 we show accuracy on query-color and query-material questions, separating questions asking about spheres (which are the same in A and B) and cubes/cylinders (which change from A to B).

Between A$\rightarrow$A and A$\rightarrow$B, all models perform about the same when asked about the color of spheres, but perform worse when asked about the color of cubes or cylinders; CNN+LSTM+SA drops from 85% to 51%. Models seem to learn strong biases about object colors and cannot overcome these biases when conditions change.

When asked about the material of cubes and cylinders, CNN+LSTM+SA shows a

smaller gap between A→A and A→B (90% vs 81%); other models show no gap. Having seen metal cubes and red metal objects during training, models can understand the material of red metal cubes.

4.5 Discussion

This chapter has introduced CLEVR, a dataset designed to aid diagnostic evaluation of visual question answering (VQA) systems by minimizing dataset bias and providing rich ground-truth representations for both images and questions. Our experiments show that CLEVR facilitates in-depth analysis not possible with other VQA datasets: our question representations allow us to slice the dataset along different axes (question type, relationship type, question topology, *etc.*), and comparing performance along these axes allows us to better understand the reasoning capabilities of VQA systems. Our analysis has revealed several key shortcomings of current VQA systems:

- **Short-term memory**: All systems we tested performed poorly in situations requiring short-term memory, including attribute comparison and integer equality questions (Section 4.4.2), same-attribute relationships (Section 4.4.3), and tree-structured questions (Section 4.4.4). Attribute comparison questions are of particular interest, since models can successfully *identity* attributes of objects but struggle to *compare* attributes.
- **Long reasoning chains**: Systems struggle to answer questions requiring long chains of nontrivial reasoning, including questions with large effective sizes (Section 4.4.5) and count and existence questions with many spatial relationships (Section 4.4.6).
- **Spatial Relationships**: Models fail to learn the true semantics of spatial relationships, instead relying on absolute image position (Section 4.4.6).
- **Disentangled Representations**: By training and testing models on different distributions (Section 4.4.7) we argue that models do not learn representations that disentangle object attributes; they seem to learn strong biases from the training data and cannot overcome these biases when conditions change.

Our study also shows cases where current VQA systems are successful. In particular, spatial attention [241] allows models to focus on objects and identify their attributes even on questions requiring multiple steps of reasoning.

These observations present clear avenues for future work on VQA. We plan to use CLEVR to study models with explicit short-term memory, facilitating comparisons between values [67, 100, 228, 235]; explore approaches that encourage learning disentangled representations [10]; and investigate methods that compile custom network architectures for different patterns of reasoning [4, 5]. We hope that diagnostic datasets like CLEVR will help guide future research in VQA and enable rapid progress on this important task.

Chapter 5

Inferring and Exceuting Programs for Visual Reasoning

5.1 Introduction

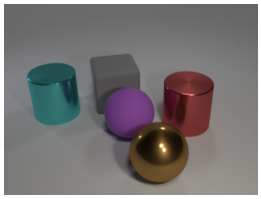
In many applications, computer-vision systems need to answer sophisticated queries by *reasoning* about the visual world (Figure 5.1). To deal with novel object interactions or object-attribute combinations, visual reasoning needs to be *compositional*: without ever having seen a "person touching a bike", the model should be able to understand the phrase by putting together its understanding of "person", "bike" and "touching". Such compositional reasoning is a hallmark of human intelligence, and allows people to solve a plethora of problems using a limited set of basic skills [125].

In contrast, modern approaches to visual recognition learn a mapping directly from inputs to outputs; they do not explicitly formulate and execute compositional plans. Direct input-output mapping works well for classifying images [118] and detecting objects [61] for a small, fixed set of categories. However, it fails to outperform strong baselines on tasks that require the model to understand an exponentially large space of objects, attributes, actions, and interactions, such as visual question answering (VQA) [7, 259]. Instead, models that learn direct input-output mappings tend to learn dataset biases but not reasoning [38, 85, 96].

How many chairs are at the table?

Is there a pedestrian in my lane?

Is the person with the blue hat touching the bike in the back?



Is there a matte cube that has the same size as the red metal object?

Figure 5.1: Compositional reasoning is a critical component needed for understanding the complex visual scenes encountered in applications such as robotic navigation, autonomous driving, and surveillance. Current models fail to do such reasoning [96].

In this chapter, we argue that to successfully perform complex reasoning tasks, it might be necessary to explicitly incorporate compositional reasoning in the model structure. Specifically, we investigate a new model for visual question answering that consists of two parts: a *program generator* and an *execution engine*. The *program generator* reads the question and produces a plan or *program* for answering the question by composing functions from a function dictionary. The *execution engine* implements each function using a small neural module, and executes the resulting module network on the image to produce an answer. Both the program generator and the modules in the execution engine are neural networks with generic architectures; they can be trained separately when ground-truth programs are available, or jointly in an end-to-end fashion.

Our model builds on prior work on neural module networks that incorporate compositional reasoning [4, 5]. Prior module networks do not generalize well to new problems, because they rely on a hand-tuned program generator based on syntactic parsing, and on hand-engineered modules. By contrast, our model does not rely on such heuristics: we only define the function vocabulary and the "universal" module architecture by hand, learning everything else.

We evaluate our model on the CLEVR dataset [96], which has proven to be challenging for state-of-the-art VQA models. The CLEVR dataset contains ground-truth

programs that describe the compositional reasoning required to answer the given questions. We find that with only a small amount of reasoning supervision (9000 ground truth programs which is 2% of those available), our model outperforms state-of-the-art non-compositional VQA models by ~20 percentage points on CLEVR. We also show that our model's compositional nature allows it to generalize to novel questions by composing modules in ways that are not seen during training.

Though our model works well on the algorithmically generated questions in CLEVR, the true test is whether it can answer questions asked by humans in the wild. We collect a new dataset of human-posed free-form natural language questions about CLEVR images. Many of these questions have out-of-vocabulary words and require reasoning skills that are absent from our model's repertoire. Nevertheless, when finetuned on this dataset without additional program supervision, our model learns to compose its modules in novel but intuitive ways to best answer new types of questions. The result is an interpretable mapping of free-form natural language to programs, and a ~9 point improvement in accuracy over the best competing models.

5.2 Related Work

Our work is related to to prior research on visual question answering, reasoning-augmented models, semantic parsers, and (neural) program-induction methods.

Visual question answering (VQA) is a popular proxy task for gauging the quality of visual reasoning systems [102, 232]. Like the CLEVR dataset, benchmark datasets for VQA typically comprise a set of questions on images with associated answers [7, 116, 146, 218, 259]; both questions and answers are generally posed in natural language. Many systems for VQA employ a very similar architecture [7, 54, 56, 143, 148, 149, 235]: they combine an RNN-based embedding of the question with a convolutional network-based embedding of an image in a classification model over possible answers. Recent work has questioned whether such systems are capable of developing visual reasoning capabilities: (1) very simple baseline models were found to perform competitively on VQA benchmarks by exploiting biases in the data [66, 85, 254] and (2) experiments on CLEVR, which was designed to control such biases,

revealed that current systems do not learn to reason about spatial relationships or to learn disentangled representations [96].

Our model aims to address these problems by explicitly constructing an intermediate program defining the reasoning process required to answer the question. We show that our model succeeds on several kinds of reasoning where other models fail.

Reasoning-augmented models add components to neural network models to facilitate the development of reasoning processes in such models. For example, models such as neural Turing machines [67, 69], memory networks [212, 228], and stack-augmented recurrent networks [100] add explicit memory components to neural networks to facilitate learning of reasoning processes that involve long-term memory. While long-term memory is likely to be a crucial component of intelligence, it is not a prerequisite for reasoning, especially the kind of reasoning that is required for answering questions about images.[1] Therefore, we do not consider memory-augmented models in this study.

Module networks are an example of reasoning-augmented models that use a syntactic parse of a question to determine the architecture of the network [4, 5, 81]. The final network is composed of trained neural modules that execute the "program" produced by the parser. The main difference between our models and existing module networks is that we replace hand-designed off-the-shelf syntactic parsers [112], which perform very poorly on complex questions such as those in CLEVR [96], by a learnt program generator that can adapt to the task at hand.

Semantic parsers attempt to map natural language sentences to logical forms. Often, the goal is to answer natural language questions using a knowledge base [135]. Recent approaches to semantic parsing involve a learnt *programmer* [134]. However, the semantics of the program and the execution engine are fixed and known a priori, while we learn both the program generator and the execution engine.

Program-induction methods learn programs from input-output pairs by fitting the parameters of a neural network to predict the output that corresponds to a

[1] Memory is likely indispensable in more complex settings such as visual dialogues or SHRDLU [34, 230].

particular input value. Such models can take the form of a feedforward scoring function over operators in a domain-specific language that can be used to guide program search [8], or of a recurrent network that decodes a vectorial program representation into the actual program [103, 122, 161, 249–251]. The recurrent networks may incorporate compositional structure that allows them to learn new programs by combining previously learned sub-programs [188].

Our approach differs from prior work on program induction in the type of input-output pairs that are used and the way the domain-specific language is implemented. Prior work on neural program interpreters considers simple algorithms such as sorting of a list of integers; by contrast, we consider inputs that comprise an image and an associated question (in natural language). Program induction approaches also assume knowledge of the low-level operators such as arithmetic operations. In contrast, we use a learnt execution engine and assume minimal prior knowledge.

5.3 Method

We develop a learnable compositional model for visual question answering. Our model takes as input an image x and a visual question q about the image. The model selects an answer $a \in \mathcal{A}$ to the question from a fixed set $\mathcal{A}$ of possible answers. Internally, the model predicts a program z representing the reasoning steps required to answer the question. The model then executes the predicted program on the image, producing a distribution over answers.

To this end, we organize our system into two components: a *program generator*, $z = \pi(q)$, which predicts programs from questions, and an *execution engine*, $a = \phi(x, z)$, which executes a program z on an image x to predict an answer a. Both the program generator and the execution engine are neural networks that are learned from data. In contrast to prior work [4, 5], we do not manually design heuristics for generating or executing the programs.

We present learning procedures both for settings where (some) ground-truth programs are available during training, and for settings without ground-truth programs. In practice, our models need *some* program supervision during training, but we find

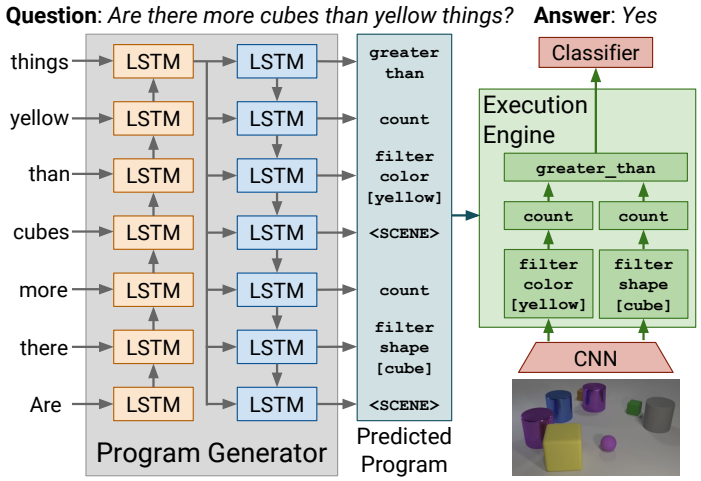


Figure 5.2: System overview. The **program generator** is a sequence-to-sequence model which inputs the question as a sequence of words and outputs a program as a sequence of functions, where the sequence is interpreted as a prefix traversal of the program's abstract syntax tree. The **execution engine** executes the program on the image by assembling a neural module network [4] mirroring the structure of the predicted program.

that the program generator requires very few of such programs in order to learn to generalize (see Figure 5.3).

5.3.1 Programs

Like all programming languages, our programs are defined by *syntax* giving rules for building valid programs, and *semantics* defining the behavior of valid programs. We focus on learning semantics for a fixed syntax. Concretely, we fix the syntax by pre-specifying a set $\mathcal{F}$ of functions f, each of which has a fixed arity $n_f \in \{1, 2\}$. Because we are interested in visual question answering, we include in the vocabulary a special constant `Scene`, which represents the visual features of the image. We represent valid programs z as *syntax trees* in which each node contains a function $f \in \mathcal{F}$, and in which each node has as many children as the arity of the function f.

			Compare Integer			Query				Compare				
Method	Exist	Count	Equal	Less	More	Size	Color	Mat.	Shape	Size	Color	Mat.	Shape	Overall
Q-type mode	50.2	34.6	51.4	51.6	50.5	50.1	13.4	50.8	33.5	50.3	52.5	50.2	51.8	42.1
LSTM	61.8	42.5	63.0	73.2	71.7	49.9	12.2	50.8	33.2	50.5	52.5	49.7	51.8	47.0
CNN+LSTM	68.2	47.8	60.8	74.3	72.5	62.5	22.4	59.9	50.9	56.5	53.0	53.8	55.5	54.3
CNN+LSTM+SA [241]	68.4	57.5	56.8	74.9	68.2	90.1	83.3	89.8	87.6	52.1	55.5	49.7	50.9	69.8
CNN+LSTM+SA+MLP	77.9	59.7	60.3	83.7	76.7	85.4	73.1	84.5	80.7	72.3	71.2	70.1	69.7	73.2
Human† [96]	96.6	86.7	79.0	87.0	91.0	97.0	95.0	94.0	94.0	94.0	98.0	96.0	96.0	92.6
Ours-strong (700K prog.)	**97.1**	**92.7**	**98.0**	**99.0**	**98.9**	**98.8**	**98.4**	**98.1**	**97.3**	**99.8**	**98.5**	**98.9**	**98.4**	**96.9**
Ours-semi (18K prog.)	95.3	90.1	93.9	97.1	97.6	98.1	97.1	97.7	96.6	99.0	97.6	98.0	97.3	95.4
Ours-semi (9K prog.)	89.7	79.7	85.2	76.1	77.9	94.8	93.3	93.1	89.2	97.8	94.5	96.6	95.1	88.6

Table 5.1: Question answering accuracy (higher is better) on the CLEVR dataset for baseline models, humans, and three variants of our model. The strongly supervised variant of our model uses all 700K ground-truth programs for training, whereas the semi-supervised variants use 9K and 18K ground-truth programs, respectively. †Human performance is measured on a 5.5K subset of CLEVR questions.

5.3.2 Program generator

The program generator $z = \pi(q)$ predicts programs z from natural-language questions q that are represented as a sequence of words. We use a prefix traversal to serialize the syntax tree, which is a non-sequential discrete structure, into a sequence of functions. This allows us to implement the program generator using a standard LSTM sequence-to-sequence model; see [214] for details.

When decoding at test time, we take the argmax function at each time step. The resulting sequence of functions is converted to a syntax tree; this is straightforward since the arity of each function is known. Some generated sequences do not correspond to prefix traversals of a tree. If the sequence is too short (some functions do not have enough children) then we pad the sequence with `Scene` constants. If the sequence is too long (some functions have no parents) then unused functions are discarded.

5.3.3 Execution engine

Given a predicted program z and an input image x, the execution engine executes the program on the image, $a = \phi(x, z)$, to predict an answer a. The execution engine is implemented as a neural module network [4]: the program z is used to assemble a question-specific neural network composed from a set of modules. For each function

$f \in \mathcal{F}$, the execution engine maintains a neural network module m_f. Given a program z, the execution engine creates a neural network $m(z)$ by mapping each function f to its corresponding module m_f in the order defined by the program: the outputs of the "child modules" are used as input into their corresponding "parent module".

Our modules use a generic architecture, in contrast to [4]. A module of arity n receives n features maps of shape $C \times H \times W$ and produces a feature map of shape $C \times H \times W$. Each unary module is a standard residual block [75] with two 3×3 convolutional layers. Binary modules concatenate their inputs along the channel dimension, project from $2C$ to C channels using a 1×1 convolution, and feed the result to a residual block. The `Scene` module takes visual features as input ($conv4$ features from ResNet-101 [75] pretrained on ImageNet [193]) and passes these features through four convolutional layers to output a $C \times H \times W$ feature map.

Using the same architecture for all modules ensures that every valid program z corresponds to a valid neural network which inputs the visual features of the image and outputs a feature map of shape $C \times H \times W$. This final feature map is flattened and passed into a multilayer perceptron classifier that outputs a distribution over possible answers.

5.3.4 Training

Given a VQA dataset containing (x, q, z, a) tuples with ground truth programs z, we can train both the program generator and execution engine in a supervised manner. Specifically, we can (1) use pairs (q, z) of questions and corresponding programs to train the program generator, which amounts to training a standard sequence-to-sequence model; and (2) use triplets (x, z, a) of the image, program, and answer to train the execution engine, using backpropagation to compute the required gradients (as in [4]).

Annotating ground-truth programs for free-form natural language questions is expensive, so in practice we may have few or no ground-truth programs. To address this problem, we opt to train the program generator and execution engine jointly on (x, q, a) triples *without ground-truth programs*. However, we cannot backpropagate

through the argmax operations in the program generator. Instead we replace the argmaxes with sampling and use REINFORCE [229] to estimate gradients on the outputs of the program generator; the reward for each of its outputs is the negative zero-one loss of the execution engine, with a moving-average baseline.

In practice, joint training using REINFORCE is difficult: the program generator needs to produce the right program without understanding what the functions mean, and the execution engine has to produce the right answer from programs that may not accurately implement the question asked. We propose a more practical *semi-supervised learning* approach. We first use a *small* set of ground-truth programs to train the program generator, then fix the program generator and train the execution engine using predicted programs on a large dataset of (x, q, a) triples. Finally, we use REINFORCE to jointly finetune the program generator and execution engine. Crucially, ground-truth programs are *only* used to train the initial program generator.

5.4 Experiments

We evaluate our model on the recent CLEVR dataset [96]. Standard VQA methods perform poorly on this dataset, showing that it is a challenging benchmark. All questions are equipped with ground-truth programs, allowing for experiments with varying amounts of supervision.

We first perform experiments using strong supervision in the form of ground-truth programs. We show that in this strongly supervised setting, the combination of program generator and execution engine works much better on CLEVR than alternative methods. Next, we show that this strong performance is maintained when a small number of ground-truth programs, which capture only a fraction of question diversity, is used for training. Finally, we evaluate the ability of our models to perform compositional generalization, as well as generalization to free-form questions posed by humans. Code reproducing the results of our experiments is available from `https://github.com/facebookresearch/clevr-iep`.

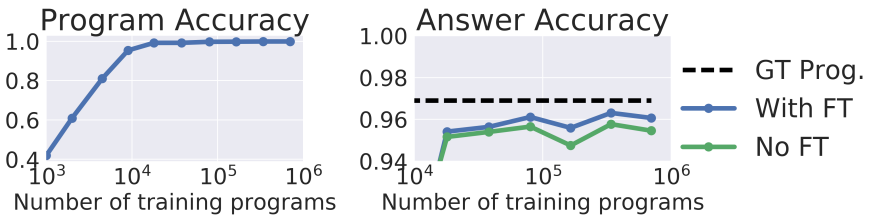


Figure 5.3: Accuracy of predicted programs (left) and answers (right) as we vary the number of ground-truth programs. Blue and green give accuracy before and after joint finetuning; the dashed line shows accuracy of our strongly-supervised model.

5.4.1 Baselines

In the previous chapter we tested several VQA models on CLEVR; these methods are reproduced here as baselines:

Q-type mode: This baseline predicts the most frequent answer for each of the question types in CLEVR.

LSTM: Similar to [7, 148], questions are processed with learned word embeddings followed by a word-level LSTM [79]. The final LSTM hidden state is passed to a multi-layer perceptron (MLP) that predicts a distribution over answers. This method uses no image information, so it can only model question-conditional biases.

CNN+LSTM: Images and questions are encoded using convolutional network (CNN) features and final LSTM hidden states, respectively. These features are concatenated and passed to a MLP that predicts an answer distribution.

CNN+LSTM+SA [241]: Questions and images are encoded using a CNN and LSTM as above, then combined using two rounds of soft spatial attention; a linear transform of the attention output predicts the answer.

CNN+LSTM+SA+MLP: Replaces the linear transform with an MLP for better comparison with the other methods.

The models that are most similar to ours are neural module networks [4, 5]. Unfortunately, neural module networks use a hand-engineered, off-the-shelf parser to produce programs, and this parser fails[2] on the complex questions in CLEVR [96]. Therefore, we were unable to include module networks in our experiments.

[2]See Appendix B for example parses of CLEVR questions.

5.4.2 Strongly and semi-supervised learning

We first experiment with a model trained using full supervision: we use the ground-truth programs for all questions in CLEVR to train both the program generator and the execution engine separately. The question answering accuracy of the resulting model on CLEVR is shown in Table 5.1 (Ours-strong). The results show that using strong supervision, our model can achieve near-perfect accuracy on CLEVR (even outperforming Mechanical Turk workers).

In practice, ground-truth programs may not not available for all questions. We use the semi-supervised training process described in Section 5.3.4 to determine how many ground-truth programs are needed to match full supervision. First, the program generator is trained in a supervised manner using a small number of questions and programs; next, the execution engine is trained on *all* CLEVR questions, using predicted rather than ground-truth programs. Finally, both components are jointly finetuned *without* ground-truth programs. Table 5.1 shows the accuracy of semi-supervised models trained with 9K and 18K ground-truth programs (Ours-semi).

The results show that 18K ground-truth programs are sufficient to train a model that performs almost on par with a fully supervised model (that used all 700K programs for training). This strong performance is not due to the program generator simply remembering all programs: the total number of unique programs in CLEVR is approximately 450K. This implies that after observing only a small fraction ($\leq 4\%$) of all possible programs, the model is able to understand the underlying structure of CLEVR questions and use that understanding to generalize to new questions.

Figure 5.3 analyzes how the accuracy of the predicted programs and the final answer vary with the number of ground-truth programs used. We measure the accuracy of the program generator by deserializing the function sequence produced by the program generator, and marking it as correct if it matches the ground-truth program exactly.[3] Our results show that with about 20K ground-truth programs, the program generator achieves near perfect accuracy, and the final answer accuracy is almost as good as strongly-supervised training. Training the execution engine using programs

[3]This may underestimate the true accuracy, since unequal programs can be equivalent.

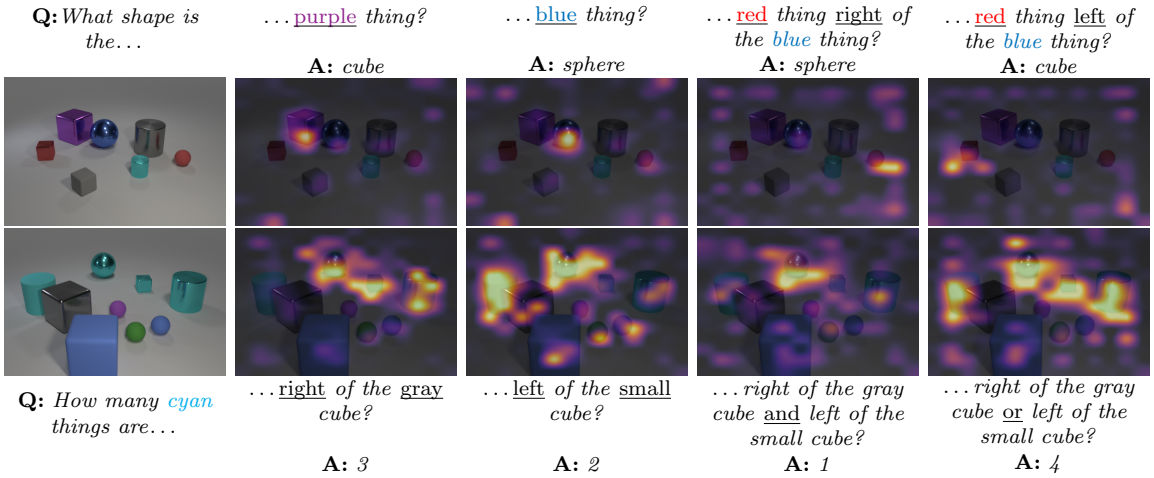


Figure 5.4: Visualizations of the norm of the gradient of the sum of the predicted answer scores with respect to the final feature map. From left to right, each question adds a module to the program; the new module is *underlined* in the question. The visualizations illustrate which objects the model attends to when performing the reasoning steps for question answering. Images are from the validation set.

predicted from the program generator rather than ground-truth programs degrades accuracy by about 3 points, but some of that loss is recovered after joint finetuning.

5.4.3 What do the modules learn?

To obtain additional insight into what the modules in the execution engine have learned, we visualized the parts of the image that are being used to answer different questions; see Figure 5.4. Specifically, the figure displays the norm of the gradient of the sum of the predicted answer scores (softmax inputs) with respect to the final feature map. This visualization reveals several important aspects of our model.

First, it clearly attends to the correct objects even for complicated referring expressions involving spatial relationships, intersection and union of constraints, *etc.*

Second, the examples show that changing a *single* module (swapping *purple*/*blue*, *left*/*right*, *and*/*or*) results in drastic changes in both the predicted answer and model attention, demonstrating that the individual modules do in fact perform their intended functions. Modules learn specialized functions such as localization and set operations without explicit supervision of their outputs.

Method	Train A		Finetune B	
	A	B	A	B
LSTM	55.2	50.9	51.5	54.9
CNN+LSTM	63.7	57.0	58.3	61.1
CNN+LSTM+SA+MLP	80.3	68.7	75.7	75.8
Ours (18K prog.)	**96.6**	**73.7**	**76.1**	**92.7**

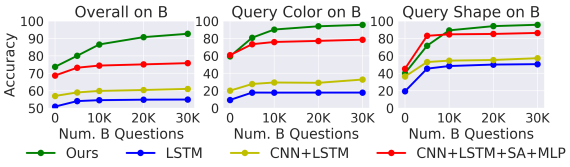


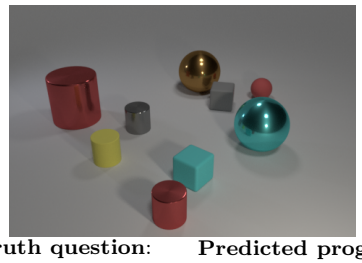
Figure 5.5: Question answering accuracy on the CLEVR-CoGenT dataset (higher is better). **Top:** We train models on Condition A, then test them on both Condition A and Condition B. We then finetune these models on Condition B using 3K images and 30K questions, and again test on both Conditions. Our model uses 18K programs during training on Condition A, and does not use any programs during finetuning on Condition B. **Bottom:** We investigate the effects of using different amounts of data when finetuning on Condition B. We show overall accuracy as well as accuracy on color-query and shape-query questions.

5.4.4 Generalizing to new attribute combinations

In the previous chapter we introduced the CLEVR-CoGenT dataset for investigating the ability of VQA models to perform compositional generalization. The dataset contains data in two different conditions: in Condition A, all cubes are gray, blue, brown, or yellow and all cylinders are red, green, purple, or cyan; in Condition B, cubes and cylinders swap color palettes. In Section 4.4.7 we showed that VQA models trained on data from Condition A performed poorly on data from Condition B, suggesting the models are not well capable of generalizing to new conditions.

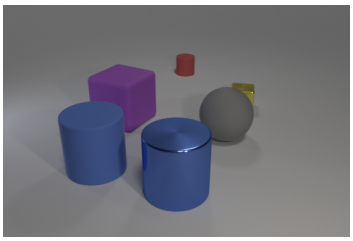
We performed experiments with our model on CLEVR-CoGenT: Figure 5.5 reports accuracy of the semi-supervised variant of our model trained on data from Condition A and evaluated on data from Condition B. Although the resulting model outperforms all baselines in Condition B, it still appears to suffer from the problems discussed in Section 4.4.7. A more detailed analysis revealed that our model does not outperform the CNN+LSTM+SA baseline for questions about an object's shape or color. This is not surprising: if the model never sees red cubes, it has no incentive to learn that the attribute "red" refers to the color and not to the shape.

We also performed experiments in which we used a small amount of training data *without ground-truth programs* from condition B for finetuning. We varied the amount of data from condition B that is available for finetuning. As shown in Figure 5.5,



Ground-truth question: *Is the number of matte blocks in front of the small yellow cylinder greater than the number of red rubber spheres to the left of the large red shiny cylinder?*
Program length: 20
A: *yes* ✓

Predicted program (translated): *Is the number of matte blocks in front of the small yellow cylinder greater than the number of large red shiny cylinders?*
Program length: 15
A: *no* ✗



Ground-truth question: *How many objects are big rubber objects that are in front of the big gray thing or large rubber things that are in front of the large rubber sphere?*
Program length: 16
A: *1* ✓

Predicted program (translated): *How many objects are big rubber objects in front of the big gray thing or large rubber spheres?*
Program length: 12
A: *2* ✗

Figure 5.6: Examples of long questions where the program and answer were predicted incorrectly when the model was trained on short questions, but both program and answer were correctly predicted after the model was finetuned on long questions. Above each image we show the ground-truth question and its program length; below, we show a manual English translation of the predicted program and answer before finetuning on long questions.

our model learns the new attribute combinations from only ∼10K questions (∼1K images), and outperforms similarly trained baselines across the board.[4] We believe that this is because the model's compositional nature allows it to quickly learn new semantics of attributes such as "red" from little training data.

5.4.5 Generalizing to new question types

Our experiments in Section 5.4.2 showed that relatively few ground-truth programs are required to train our model effectively. Due to the large number of unique programs in CLEVR, it is impossible to capture all possible programs with a small set of ground-truth programs; however, due to the synthetic nature of CLEVR questions, it is possible that a small number of programs could cover all possible program structures. In real-world scenarios, models should be able to generalize to questions with novel program structures without observing associated ground-truth programs.

[4]Note that this finetuning hurts performance on condition *A*. Joint finetuning on both conditions will likely alleviate this issue.

Method	Train Short		Finetune Both	
	Short	Long	Short	Long
LSTM	46.4	48.6	46.5	49.9
CNN+LSTM	54.0	52.8	54.3	54.2
CNN+LSTM+SA+MLP	74.2	**64.3**	74.2	67.8
Ours (25K prog.)	**95.9**	55.3	**95.6**	**77.8**

Table 5.2: Question answering accuracy on short and long CLEVR questions. **Left columns**: Models trained only on short questions; our model uses 25K ground-truth short programs. **Right columns**: Models trained on both short and long questions. Our model is trained on short questions then finetuned on the entire dataset; no ground-truth programs are used during finetuning.

To test this, we divide CLEVR questions into two categories based on their ground-truth programs: *short* and *long*. CLEVR questions are divided into *question families*, where all questions in the same family share the same program structure. A question is *short* if its family has a mean program length less than 16; otherwise it is *long*.[5]

We train the program generator and execution engine on short questions in a semi-supervised manner using 18K short programs, and test the resulting model on both short and long questions. This experiment tests the ability of our model to generalize from short to long chains of reasoning. Results are shown in Table 5.2.

The results show that when evaluated on long questions, our model trained on short questions underperforms the CNN+LSTM+SA model trained on the same set. Presumably, this result is due to the program generator learning a bias towards short programs. Indeed, Figure 5.6 shows that the program generator produces programs that refer to the right objects but that are too short.

We can undo this short-program bias by jointly finetuning the program generator and execution engine on the combined set of short and long questions, *without ground-truth programs*. To pinpoint the problem of short-program bias in the program generator, we leave the execution engine fixed during finetuning; it is only used to compute REINFORCE rewards for the program generator. After finetuning, our model substantially outperforms baseline models that were trained on the entire dataset, showing that the short-program bias has been unlearned; see Table 5.2.

[5]Partitioning at the family level rather than the question level allows for better separation of program structure between short and long questions.

Method	Train CLEVR	Train CLEVR, finetune human
LSTM	27.5	36.5
CNN+LSTM	37.7	43.2
CNN+LSTM+SA+MLP	50.4	57.6
Ours (18K prog.)	**54.0**	**66.6**

Table 5.3: Question answering accuracy on the CLEVR-Humans test set of four models after training on just the CLEVR dataset (**left**) and after finetuning on the CLEVR-Humans dataset (**right**).

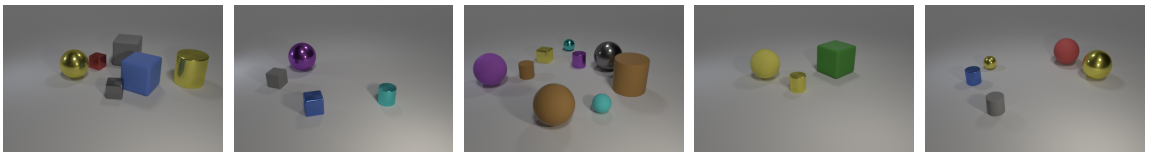
5.4.6 Generalizing to human-posed questions

The fact that questions in the CLEVR benchmark were generated algorithmically may favor some approaches over others. In particular, natural language tends to be more ambiguous than algorithmically generated questions. We performed an experiment to assess the extent to which models trained on CLEVR can be finetuned to answer human questions. To this end, we collected a new dataset of natural-language questions and answers for CLEVR images.

The CLEVR-Humans Dataset. Inspired by VQA [7], workers on Amazon Mechanical Turk were asked to write questions about CLEVR images that would be *hard for a smart robot to answer*; workers were primed with questions from CLEVR and restricted to answers in CLEVR. We filtered questions by asking three workers to answer each question, and removed questions that a majority of workers could not correctly answer. We collected one question per image; after filtering, we obtained 17,817 training, 7,202 validation, and 7,145 test questions on CLEVR images. The data is available from the first author's website.

The human questions are more challenging than synthetic CLEVR questions because they exhibit more linguistic variety. Unlike existing VQA datasets, the CLEVR-Humans questions do not require common-sense knowledge: they focus entirely on visual reasoning abilities, which makes them a good testbed for evaluating reasoning.

Figure 5.7 shows example human questions. Some are rewordings of synthetic CLEVR questions; others are answerable using the same basic functions as CLEVR but potentially with altered semantics for those skills. For example, people use spatial



Q: *Is there a blue box in the items?* **A:** *yes*

Predicted Program:
```
exist
filter_shape[cube]
filter_color[blue]
scene
```
Predicted Answer: ✓ *yes*

Q: *What shape object is farthest right?* **A:** *cylinder*

Predicted Program:
```
query_shape
unique
relate[right]
unique
filter_shape[cylinder]
filter_color[blue]
scene
```
Predicted Answer: ✓ *cylinder*

Q: *Are all the balls small?* **A:** *no*

Predicted Program:
```
equal_size
query_size
unique
filter_shape[sphere]
scene
query_size
unique
filter_shape[sphere]
filter_size[small]
scene
```
Predicted Answer: ✓ *no*

Q: *Is the green block to the right of the yellow sphere?* **A:** *yes*

Predicted Program:
```
exist
filter_shape[cube]
filter_color[green]
relate[right]
unique
filter_shape[sphere]
filter_color[yellow]
scene
```
Predicted Answer: ✓*yes*

Q: *Two items share a color, a material, and a shape; what is the size of the rightmost of those items?* **A:** *large*

Predicted Program:
```
count
filter_shape[cube]
same_material
unique
filter_shape[cylinder]
scene
```
Predicted Answer: ✗ *0*

Figure 5.7: Examples of questions from the CLEVR-Humans dataset, along with predicted programs and answers from our model. Question words that do not appear in CLEVR questions are underlined. Some predicted programs exactly match the semantics of the question (green); some programs closely match the question semantics (yellow), and some programs appear unrelated to the question (red).

relationships "left", "right", *etc.* differently than their meanings in CLEVR questions. Finally, some questions require skills not needed for answering synthetic questions.

Results. We train our model on CLEVR, and then finetune *only the program generator* on the CLEVR-Humans training set to adapt it to the additional linguistic variety; we do not adapt the execution engine due to the limited quantity of data. No ground-truth programs are available during finetuning. The embeddings in the sequence-to-sequence model of question words that do not appear in CLEVR synthetic questions are initialized randomly before finetuning.

During finetuning, our model learns to reuse the reasoning skills it has already mastered in order to answer the linguistically more diverse natural-language questions. As shown in Figure 5.7, it learns to map novel words (*"box"*) to known modules. When human questions are not expressible using CLEVR functions, our model still learns to produce reasonable programs closely approximating the question's intent. Our model

often fails on questions that cannot be reasonably approximated using our model's module inventory, such as the rightmost example in Figure 5.7. Quantitatively, the results in Table 5.3 show that our model outperforms all baselines on the CLEVR-Humans test set both with and without finetuning.

5.5 Discussion

Our results show that our model is able to generalize to novel scenes and questions and can even infer programs for free-form human questions using its learned modules. While these results are encouraging, there still are many questions that cannot be reasonably approximated using our fixed set of modules. For example, the question *"What color is the object with a unique shape?"* requires a model to identify unique shapes, for which no module is currently available. Adding new modules to our model is straightforward due to our generic module design, but automatically identifying and learning new modules without program supervision is still an open problem. One path forward is to design a Turing-complete set of modules; this would allow for all programs to be expressed without learning new modules. For example, by adding ternary operators (`if/then/else`) and loops (`for/do`), the question *"What color is the object with a unique shape?"* can be answered by *looping* over all shapes, counting the objects with that shape, and returning it *if* the count is one. These control-flow operators could be incorporated into our framework: for example, a loop could apply the same module to an input set and aggregate the results. We emphasize that learning such programs with limited supervision is an open research challenge, which we leave to future work.

This work in this chapter fits into a long line of work on incorporating symbolic representations into (neural) machine learning models [8, 17, 134, 188]. We have shown that explicit program representations can make it easier to compose programs to answer novel questions about images. Our generic program representation, learnable program generator and universal design for modules makes our model much more flexible than neural module networks [4, 5] and thus more easily extensible to new problems and domains.

Chapter 6

Image Retrieval with Scene Graphs

Retrieving images by describing their contents is one of the most exciting applications of computer vision. An ideal system would allow people to search for images by specifying not only objects ("man", "boat") but also structured *relationships* ("man on boat") and *attributes* ("boat is white") involving these objects. Unfortunately current systems fail for these types of queries because they do not utilize the structured nature of the query, as shown in Fig. 6.1.

To solve this problem, a computer vision system must explicitly represent and reason about *objects*, *attributes*, and *relationships* in images, which we refer to as *detailed semantics*. Recently Zitnick et al. have made important steps toward this goal by studying *abstract scenes* composed of clip-art [52, 261, 262]. They show that perfect recognition of detailed semantics improves image understanding and retrieval.

Bringing this level of semantic reasoning to *real-world scenes* is an important goal, but doing so involves two main challenges: *(1)* interactions between objects in a scene can be highly complex, going beyond simple pairwise relations, and *(2)* the assumption of a closed universe where all classes are known beforehand does not hold.

In order to address these challenges, this chapter proposes a novel framework for detailed semantic image retrieval, based on a conditional random field (CRF [124]) model of visual scenes. Our model draws inspiration from recent work in computer graphics that uses graph-based formulations to compare [51] and generate [22] scenes. We use the notion of a *scene graph* to represent the detailed semantics of a scene.

Results for the query on a popular image search engine.

Expected results for the query.

Figure 6.1: Image search using a complex query like "man holding fish and wearing hat on white boat" returns unsatisfactory results; Ideal results include correct *objects* ("man", "boat"), *attributes* ("boat is white") and *relationships* ("man on boat").

Our scene graphs capture the detailed semantics of visual scenes by explicitly modeling objects, attributes of objects, and relationships between objects. Our model performs semantic image retrieval using scene graphs as queries. Replacing textual queries with scene graphs allows our queries to describe the semantics of the desired image in precise detail without relying on unstructured text. This formulation is related to a number of methods for object and scene recognition using context [37, 72]. But by using scene graphs, we can model multiple modes of interaction between pairs of objects while traditional CRF models are more restricted, and encode a fixed relation given two nodes (e.g. think of "dog [eating, or playing, or being on the right of] a keyboard" in a scene graph versus. "dog *on the right of* a keyboard" in a CRF).

Specifically, our contributions in this chapter are:

- We introduce a CRF model (Section 6.4) for semantic image retrieval using scene graph (Section 6.2) queries. Our model outperforms baseline models that reason only about objects, and simple content-based image retrieval methods based on low-level visual features (Section 6.5), addressing challenge (1) above.

- We introduce a novel dataset (Sect. 6.3) of $5,000$ human-annotated scene graphs grounded to images that use an open-world vocabulary to describe images in great detail, addressing challenge (2) above.

We set out to demonstrate the importance and utility of modeling detailed semantics using a scene graph representation for an image retrieval task. But as our experiments demonstrate, an advantage of this representation is its ability to offer deeper and detailed insights into images in a general framework. Our model can perform semantically meaningful image retrieval based on an entire scene (Section 6.5.1), or only parts of scenes (Section 6.5.2), and localizes specific objects (Section 6.5.3). This differentiates our intention and system from traditional content-based image retrieval work, which is not the focus of this chapter.

6.1 Related Work

Image retrieval. Content-based image retrieval methods typically use low-level visual feature representations [18, 176], indexing [32, 88, 89, 208, 255], efficient sub-image search [127], object-category based retrieval [12, 13, 221], and other methods of modeling image similarity [181] to retrieve images based on their contents.

Text-based scene representations. There has been much recent interest in models that can jointly reason about images and natural language descriptions, ranging from Flickr tags [132] over sentences with canonical structure [119, 158, 262] to unaligned text corpora [106, 203, 204, 246]. While these models achieve impressive results in scene classification and object recognition using only weak supervision, they are typically limited in terms of expressiveness.

In contrast, our scene graphs are a structured representation of visual scenes. Each node is explicitly grounded in an image region, avoiding the inherent referential uncertainty of text-based representations. We currently use strong supervision in the form of a crowd-sourced data set (Section 6.3), but ultimately envision a system that learns novel scene graphs via active learning, like NEIL [28] or LEVAN [39].

Structured scene representations. More structured representations of visual scenes explicitly encode properties such as attributes [46, 47, 50, 126, 173], object co-occurrence [153], or spatial relationships between pairs of objects [30, 31, 37, 195].

Our scene graphs generalize these representations, since they allow us to express each in a unified way (Section 6.2). Concretely, our CRF (Section 6.4) learns models for particular relations between pairs of objects, similar in spirit to [72] or [242]. However, we consider a much larger, open vocabulary of objects and relationships.

Graph-structured representations have attained wide-spread use in computer graphics to efficiently represent compositional scenes [171]. Fisher et al. [51] use graph kernels to compare 3D scenes, and Chang et al. [22] generate novel 3D scenes from natural language descriptions using a scene graph representation. Parse graphs obtained in scene parsing [239, 256] are typically the result of applying a grammar designed for a particular domain (such as indoor scenes [256]), in contrast to our generic scene graphs.

Recent work by Lin et al. [136] constructs semantic graphs from text queries using hand-defined rules to transform parse trees and uses these semantic graphs to retrieve videos in the context of autonomous driving. Their system is constrained to the six object classes from KITTI [59] while our system uses an open-world vocabulary; our scene graphs also tend to be more complex than their semantic graphs.

Zitnick et al. have studied detailed semantics using abstract scenes built from clip-art aligned to a corpus of sentences [52, 261, 262].

Our approach extends this work as follows: first, we explicitly address real-world scenes, replacing the idealistic setting of perfect object and attribute recognition [261] by uncertain detections. Second, our scene graphs go beyond pairwise relations: we model and discover meaningful higher-order interactions between multiple objects and attributes (these sub-graphs can be seen as a generalization of visual phrases [31, 195]). Third, our CRF for scene graph grounding (Section 6.4) can ground a query scene graph to an unannotated test image.

Real-world scene datasets. Datasets have been a driving force in computer vision, ranging from classification [233] to object recognition and segmentation [35, 44, 49, 194]. More recently, there has been a shift away from iconic images toward datasets depicting objects in the context of entire scenes, e.g., SUN09 [30], PASCAL-Context [160], and the COCO [138] datasets. Our novel dataset of real-world scene

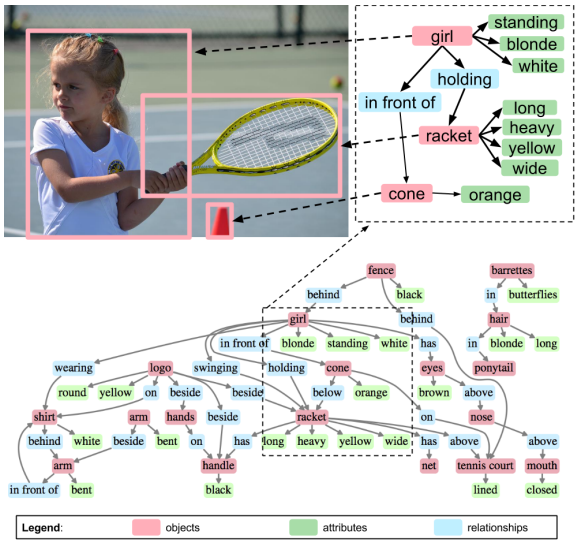


Figure 6.2: An example of a scene graph (bottom) and a grounding (top). The scene graph encodes objects ("girl"), attributes, ("girl is blonde"), and relationships ("girl holding racket"). The grounding associates each object of the scene graph to a region of an image. The image, scene graph, and grounding are drawn from our *real-world scene graphs* dataset (Section 6.3).

graphs advances in this direction by adding attributes and relationships. In contrast to prior work, our dataset with $5,000$ images focuses on depth rather than breadth (Section 6.3), resulting in a level of semantic detail that has not been achieved before. While this level of detail is similar in spirit to the Lotus Hill dataset [244], our dataset is based on an open universe assumption and is freely available.

6.2 Scene Graphs

To retrieve images containing particular semantic content, we need a formalized way to represent scene contents. This representation must be powerful enough to describe the rich variety of scenes that can exist, without being too cumbersome. To this end, we define two abstractions: a *scene graph*, which is a way of describing a scene, and a *scene graph grounding*, which is a concrete association of a scene graph to an image.

6.2.1 Definition

A *scene graph* is a data structure that describes the contents of a scene. A scene graph encodes object instances, attributes of objects, and relationships between objects.

This simple formulation is powerful enough to describe visual scenes in great detail because it places no restriction on the types of objects, attributes, and relationships that can be represented. Figure 6.2 (bottom) shows an example of a scene graph. In this example we see that object instances may be people ("girl"), places ("tennis court"), things ("shirt"), or parts of other objects ("arm"). Attributes can describe color ("cone is orange"), shape ("logo is round"), and pose ("arm is bent"). Relationships can encode geometry ("fence behind girl"), actions ("girl swinging racket"), and object parts ("racket has handle").

Formally, given a set of object classes $\mathcal{C}$, a set of attribute types $\mathcal{A}$, and a set of relationship types $\mathcal{R}$, we define a scene graph G to be a tuple $G = (O, E)$ where $O = \{o_1, \ldots, o_n\}$ is a set of objects and $E \subseteq O \times \mathcal{R} \times O$ is a set of edges. Each object has the form $o_i = (c_i, A_i)$ where $c_i \in \mathcal{C}$ is the class of the object and $A_i \subseteq \mathcal{A}$ are the attributes of the object.

6.2.2 Grounding a scene graph in an image

A scene graph on its own is not associated to an image; it merely describes a scene that could be depicted by an image. However a scene graph can be *grounded* to an image by associating each object instance of the scene graph to a region in an image. Figure 6.2 (top) shows an example of part of a scene graph grounded to an image.

Formally, we represent an image by a set of candidate bounding boxes B. A grounding of a scene graph $G = (O, E)$ is then a map $\gamma : O \to B$. For ease of notation, for $o \in O$ we frequently write $\gamma(o)$ as γ_o.

Given a scene graph and an image, there are many possible ways of grounding the scene graph to the image. In Section 6.4 we formulate a method for determining the best grounding of a scene graph to an image.

6.2.3 Why scene graphs?

An obvious alternative choice for representing the content of scenes is natural language. However, in order to represent visual scenes at the level of detail shown in Figure 6.2, a full paragraph of description would be necessary:

A blonde white girl is standing in front of an orange cone on a lined tennis court and is holding a long heavy yellow wide racket that has a black handle. The girl is wearing a white shirt; there is a bent arm in front of the shirt and another bent arm beside the first. There is a round yellow logo on the shirt, and the logo is beside hands that are on the handle of the racket. There is a black fence behind the girl, and the girl has brown eyes above a closed mouth. There are butterflies barrettes in long blonde hair, and the hair is in a ponytail.

To make use of such a description for image retrieval, we would need to resolve co-references in the text [101, 129, 183], perform relationship extraction to convert the unstructured text into structured tuples [163], and ground the entities of the tuples into image regions described by the text [117]. Such pipelines are challenging even in constrained settings [117], and would not scale to text of the detail shown above.

We can avoid these complexities by working directly with grounded scene graphs. We find that with careful user interface design, non-expert workers can quickly construct grounded scene graphs of arbitrary complexity. More details can be found in Section 6.3.

6.3 Dataset

To use scene graphs as queries for image retrieval, we need many examples of scene graphs grounded to images. To our knowledge no such dataset exists. To this end, we introduce a novel dataset of *real-world scene graphs*.

6.3.1 Data collection

We manually selected 5,000 images from the intersection of the YFCC100m [220] and Microsoft COCO [138] datasets, allowing our dataset to build upon rather than compete with these existing datasets.

	Full dataset	Experiments Section 6.5	COCO 2014 [138]	ILSVRC 2014 (Det) [193]	Pascal VOC [44]
Object classes	**6,745**	**266**	80	200	20
Attribute types	3,743	145	-	-	-
Relationship types	1,310	68	-	-	-
Object instances	93,832	69,009	**886,284**	534,309	27,450
Attribute instances	110,021	94,511	-	-	-
Relationship instances	112,707	109,535	-	-	-
Instances per obj. class	13.9	259.4	**11,087.5**	2,672.5	1,372
Instances per attr. type	29.4	651.8	-	-	-
Instances per rel. type	86.0	1,610.8	-	-	-
Objects per image	**18.8**	**13.8**	7.2	1.1	2.4
Attributes per image	22.0	18.9	-	-	-
Relationships per image	22.5	21.9	-	-	-
Attributes per object	1.2	1.0	-	-	-
Relationships per object	2.4	2.3	-	-	-

Table 6.1: Aggregate statistics for our *real-world scene graphs* dataset, for the full dataset and the restricted sets of object, attribute, and relationship types used in experiments.

For each of these images, we use Amazon's Mechanical Turk (AMT) to produce a human-generated scene graph. For each image, three workers write (object, attribute) and (object, relationship, object) tuples using an open vocabulary to describe the image, and draw bounding boxes for all objects. Bounding boxes for objects are corrected and verified using a system similar to [211], and all tuples are verified by other workers.

6.3.2 Analysis and statistics

Our full dataset of 5,000 images contains over 93,000 object instances, 110,000 instances of attributes, and 112,000 instances of relationships. For our experiments (Section 6.5), we consider only object classes and attribute types that appear at least 50 times in our training set and relationship types that occur at least 30 times in our training set. Even when we consider only the most common categories, as shown in Table 6.1, the dataset is still very rich with a mean of 13.8 objects, 18.9 attributes and 21.9 relationships per image.

A comparison of our dataset and other popular datasets can be found in Table 6.1.

Compared to other datasets we prioritize detailed annotations of individual images over sheer quantity of annotated images. Our dataset contains significantly more labeled object instances per image than existing datasets, and also provides annotated attributes and relationships for individual object instances; these types of annotations are simply not available in other datasets. In addition, our decision to use an open vocabulary allows annotators to label the most meaningful features of each image instead of being constrained to a fixed set of predefined classes.

In contrast to previous work that looks at individual relations between pairs of objects in isolation [195], the deeply connected nature of our scene graph representation allows us to study object interactions of arbitrary complexity. Figure 6.3 shows examples of scene-subgraphs that occur multiple times in our dataset, ranging from simple ("kite") to complex ("man on skateboard wearing red shirt"). These subgraphs also showcase the rich diversity expressed by our scene graphs. Attributes can for example encode object state ("umbrella is open"), material ("wooden table") and color ("red shirt", "black helmet"). Relationships can encode geometry ("lamp on table") as well as actions ("man riding motorcycle").

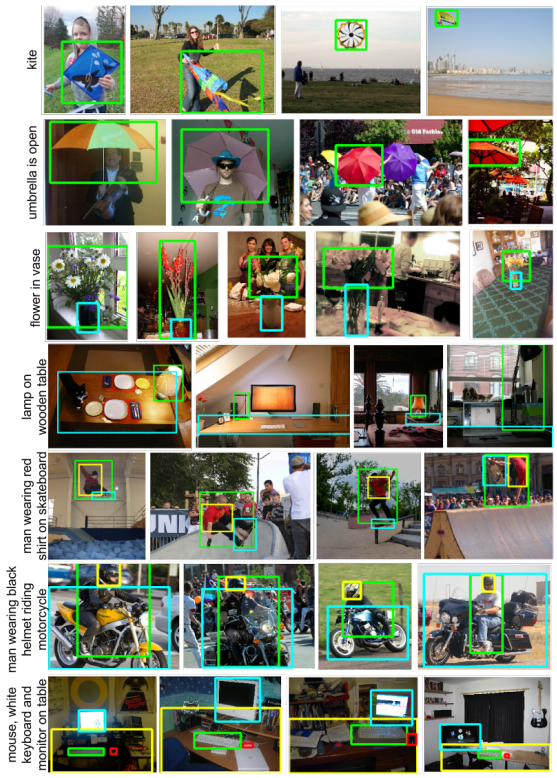


Figure 6.3: Examples of scene sub-graphs of increasing complexity (top to bottom) from our dataset, with attributes and up to 4 different objects.

Figure 6.4 uses our dataset to construct an *aggregate scene graph*, revealing the deeply connected nature of objects in the visual world. For example buses are frequently large and red, have black tires, are on streets, and have signs; signs can also appear on walls, which often occur behind men, who often wear white shirts and

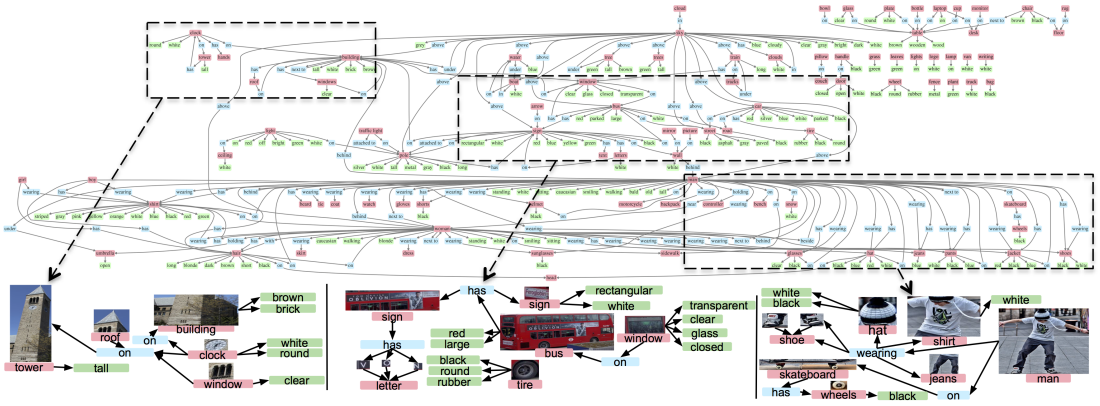


Figure 6.4: An *aggregate* scene graph computed using our entire dataset. We visualize the 150 most frequently occurring (object, relationship, object) and (object, attribute) tuples. We also provide 3 examples of scene graphs grounded in images that contribute to the sub-graphs within the dashed rectangles of the aggregated graph. Best viewed with magnification.

jeans. The high density of the aggregate scene graph around the "man", "woman", "sky", and "building" nodes suggest that these objects are prominent elements of the visual world, but for different reasons: sky and building occur in nearly every image, while the attributes and relationships of people in scenes carries a huge semantic weight.

6.4 Image Retrieval by Scene Graph Grounding

We wish to use a scene graph as a query to retrieve images portraying scenes similar to the one described by the graph. To do so, we need to measure the agreement between a query scene graph and an unannotated test image. We assume that this agreement can be determined by examining the best possible grounding of the scene graph to the image.

To this end we construct a conditional random field (CRF [124]) that models the distribution over all possible groundings. We perform maximum a posteriori (MAP) inference to find the most likely grounding; the likelihood of this MAP solution is taken as the score measuring the agreement between the scene graph and the image.

6.4.1 CRF formulation

Reusing notation from Section 6.2, let $G = (O, E)$ be a scene graph, B be a set of bounding boxes in an image, and γ a grounding of the scene graph to the image. Each object $o \in O$ gives rise to a variable γ_o in the CRF, where the domain of γ_o is B and setting $\gamma_o = b \in B$ corresponds to grounding object o in the scene graph to box b in the image. We model the distribution over possible groundings as

$$P(\gamma \mid G, B) = \prod_{o \in O} P(\gamma_o \mid o) \prod_{(o,r,o') \in E} P(\gamma_o, \gamma_{o'} \mid o, r, o'). \tag{6.1}$$

We use Bayes' rule to rewrite the term $P(\gamma_o \mid o)$ as $P(o \mid \gamma_o)P(\gamma_o)/P(o)$. Assuming uniform priors over bounding boxes and object classes, $P(\gamma_o)$ and $P(o)$ are constants and can be ignored during MAP inference. Therefore our final objective is

$$\gamma^* = \arg\max_{\gamma} \prod_{o \in O} P(o \mid \gamma_o) \prod_{(o,r,o') \in E} P(\gamma_o, \gamma_{o'} \mid o, r, o'). \tag{6.2}$$

Unary potentials. The term $P(o \mid \gamma_o)$ in Equation 6.2 is a unary potential modeling how well the appearance of the box γ_o agrees with the known object class and attributes of the object o. If $o = (c, A)$ then we decompose this term as

$$P(o \mid \gamma_o) = P(c \mid \gamma_o) \prod_{a \in A} P(a \mid \gamma_o). \tag{6.3}$$

The terms $P(c \mid \gamma_o)$ and $P(a \mid \gamma_o)$ are simply the probabilities that the bounding box γ_o shows object class c and attribute a. To model these probabilities, we use R-CNN [62] to to train detectors for each of the $|\mathcal{C}| = 266$ and $|\mathcal{A}| = 145$ object classes and attribute types. We apply Platt scaling [178] to convert the SVM classification scores for each object class and attribute into probabilities.

Binary potentials. The term $P(\gamma_o, \gamma_{o'} \mid o, r, o')$ in Equation 6.2 is a binary potential that models how well the pair of bounding boxes $\gamma_o, \gamma_{o'}$ express the tuple (o, r, o'). Let $\gamma_o = (x, y, w, h)$ and $\gamma_{o'} = (x', y', w', h')$ be the coordinates of the bounding boxes in the image. We extract features $f(\gamma_o, \gamma_{o'})$ encoding their relative position and scale:

$$f(\gamma_o, \gamma_{o'}) = \Big((x - x')/w,\ (y - y')/h,\ w'/w,\ h'/h\Big) \tag{6.4}$$

Suppose that the objects o and o' have classes c and c' respectively. Using the training data, we train a Gaussian mixture model (GMM) to model $P(f(\gamma_o, \gamma_{o'}) \mid c, r, c')$. If there are fewer than 30 instances of the tuple (c, r, c') in the training data then we instead fall back on an object agnostic model $P(f(\gamma_o, \gamma_{o'}) \mid r)$. In either case, we use Platt scaling to convert the value of the GMM density function evaluated at $f(\gamma_o, \gamma_{o'})$ to a probability $P(\gamma_o, \gamma_{o'} \mid o, r, o')$.

6.4.2 Implementation details

We compared several methods for generating candidate boxes for images, including Objectness [2], Selective Search (SS [222]), and Geodesic Object Proposals (GOP [114]). We found that SS achieves the highest object recall on our dataset; however we use GOP for all experiments as it provides the best trade-off between object recall ($\approx 70\%$ vs $\approx 80\%$ for SS) and number of regions per image (632 vs 1720 for SS). We perform approximate inference using off-the-shelf belief propagation [6].

6.5 Experiments

We perform image retrieval experiments using two types of scene graphs as queries. First, we use full ground-truth scene graphs as queries; this shows that our model can effectively make sense of extremely precise descriptions to retrieve images. Second, we jump to the other end of the query complexity spectrum and use extremely simple scene graphs as queries; this shows that our model is flexible enough to retrieve relevant images when presented with more open-ended and human-interpretable queries.

In addition, we directly evaluate the groundings found by our model and show that our model is able to take advantage of scene context to improve object localization.

Setup. We randomly split our dataset into $4,000$ training and $1,000$ test images. Our final vocabulary for object classes and attribute types is selected by picking all

		Rand	SIFT [141]	GIST [166]	CNN [118]	SG-obj [62]	SG-obj-attr	SG-obj-attr-rel
(a)	Med r	420	-	-	-	28	17.5	**14**
	R@1	0	-	-	-	0.113	0.127	**0.133**
	R@5	0.007	-	-	-	0.260	**0.340**	0.307
	R@10	0.027	-	-	-	0.347	0.420	**0.433**
(b)	Med r	94	64	57	36	17	12	**11**
	R@1	0	0	0.008	0.017	0.059	0.042	**0.109**
	R@5	0.034	0.084	0.101	0.050	0.269	0.294	**0.303**
	R@10	0.042	0.168	0.193	0.176	0.412	**0.479**	**0.479**
(c)	Med IoU	-	-	-	-	0.014	0.026	**0.067**
	R@0.1	-	-	-	-	0.435	0.447	**0.476**
	R@0.3	-	-	-	-	0.334	0.341	**0.357**
	R@0.5	-	-	-	-	0.234	0.234	**0.239**

Table 6.2: Quantitative results in entire scene retrieval ((a), Section 6.5.1), partial scene retrieval ((b), Section 6.5.2), and object localization ((c), Section 6.5.3).

terms mentioned ≥ 50 times in the training set, and our vocabulary for relationship types is the set of relationships appearing at least ≥ 30 times in the training set. Statistics of our dataset when restricted to this vocabulary are shown in the second column of Table 6.1. In our experiments we compare the following methods:

- **SG-obj-attr-rel**: Our model as described in Section 6.4. Includes unary object and attribute potentials and binary relationship potentials.
- **SG-obj-attr**: Our model, using only object and attribute potentials.
- **SG-obj**: Our model, using only object potentials. Equivalent to R-CNN [62] since the object class potentials are rescaled R-CNN detection scores.
- **CNN** [118]: L_2 distance between the last layer features extracted using the reference model from [91].
- **GIST** [166]: L_2 distance between the GIST descriptors of the query image and each test image (see Sect. 6.5.2).
- **SIFT** [141]: See Sect. 6.5.2.
- **Random**: A random permutation of the test images.

6.5.1 Full scene graph queries

We evaluate the performance of our model using the most complex scene graphs available to us – full human-generated scene graphs from our dataset. As argued in Section 6.2, these large scene graphs roughly correspond to a paragraph of text describing an image in great detail. Examples of query scene graphs and their rough textual equivalents are shown in Figure 6.5 (top).

Concretely, we select an image I_q and its associated human-annotated scene graph G_q from our test set. We use the graph G_q as a query to rank all of the test images, and record the rank of the query image I_q. We repeat this process using 150 randomly selected images from our test set and evaluate the ranking of the query image over all 150 trials. Table 6.2 (a) gives the results in the form of recall at rank k (higher is better) and median rank of the query image I_q (lower is better). Figure 6.7 (a) plots the recall over k. Note that the GIST, SIFT, and CNN baselines are meaningless here, as they would always rank the query image highest.

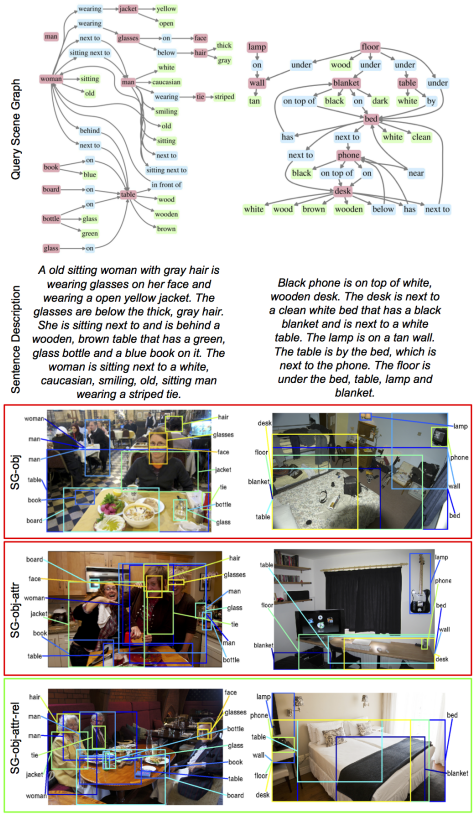


Figure 6.5: Example results for retrieval using full scene-graph queries. **Top**: Example query graphs. **Middle**: Rough text equivalents of the queries. **Bottom**: Top-1 retrieval results with groundings for our 3 methods. In both cases SG-obj-attr-rel succeeds in ranking the correct image at rank 1.

Results. In Table 6.2 (a), we observe that the detailed semantics encoded in our scene graphs greatly increases the performance for entire scene retrieval. Compared to SG-obj, SG-obj-attr-rel decreases the median rank of the query image from 28 by half to 14. Recall at k shows similar results, where SG-obj-attr-rel increases recall over SG-obj by 2% (R@1), 4.7% (R@5), and 8.6% (R@10), respectively. This performance

improvement increases for larger values of k (Fig.6.7 (a), blue vs red curve), to around 15% at 30. SG-obj-attr outperforms SG-obj, indicating that attributes are useful, and is in turn outperformed by SG-obj-attr-rel.

Figure 6.5 shows corresponding qualitative results: on the left, our full model successfully identifies and localizes the "old woman with a jacket, sitting next to a sitting man with a striped tie". On the right, even though some objects are misplaced, SG-obj-attr-rel is able to correctly place the "dark blanket on the bed", while SG-obj-attr incorrectly grounds the blanket to a dark region of the test image.

6.5.2 Small scene graph queries

We have shown that our model is able to handle the complexities of full scene graph queries. Here we show that our model can also be used to retrieve meaningful results given simpler, more human-interpretable scene graph queries.

We mine our dataset for re-occurring subgraphs containing two objects, one relationship, and one or two attributes, such as "sitting man on bench" and "smiling man wearing hat." We retain only subgraphs that occur ≥ 5 times in our test set, resulting in 598 scene subgraphs. We randomly selected 119 to be used as queries. For each query, we find the set of test images $I_1, \ldots, I_\ell$ that include it. We hold out I_1 from the test set and use the subgraph to rank the remaining 999 test images.

For the baseline methods we use the image I_1 rather than the graph to rank the test images. For the SIFT baseline, for each SIFT descriptor from I_1, we compute its 100 nearest neighbors among the descriptors from all other test images, and sort the test images by the number of these neighbors they contain. For the GIST and CNN baselines, we rank the test images based on the L_2 distance between the descriptor for I_1 and the descriptor for the test image.

We adapt the metrics from [80]; specifically, for each method we report the median rank of the highest ranked true positive image $I_2, \ldots, I_\ell$ across all queries and the recall at k for various values of k. Results are shown in Table 6.2 (b) and Figure 6.7 (b). Examples of retrieved images can be seen in Figure 6.6 (a,b), for the two queries mentioned above.

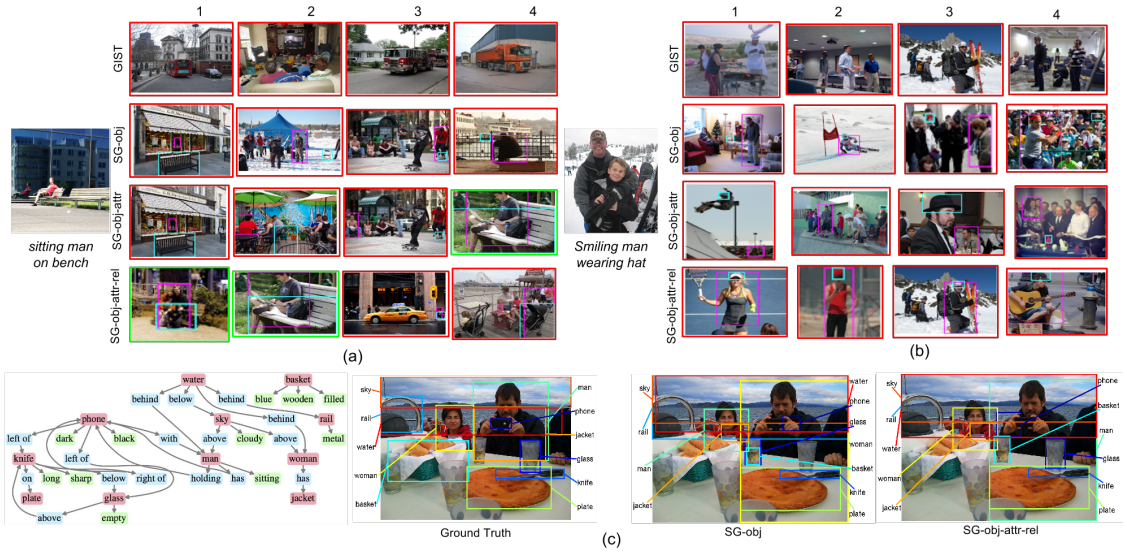


Figure 6.6: Top-*4* retrieval results returned by different methods using two different partial scene graph queries (a, b). Differences in fully automatic scene graph grounding when applying these methods to a particular test image (c).

Results. In Table 6.2 (b), we see that our full model SG-obj-attr-rel again outperforms SG-obj by a large margin, reducing the median rank from 17 to 11. SG-obj-attr comes close, with a median rank of 12. In terms of recall at k, SG-obj-attr-rel again dominates, improving over SG-obj by 5% (R@1), 3.4% (R@5), and 6.7% (R@10), respectively. This gap increases up to around 12% at 40 (Figure 6.7 (b)).

Qualitatively, Figure 6.6 (a) shows that SG-obj-attr-rel retrieves correct results for "sitting man on bench." In comparison, the first result returned by SG-obj and SG-obj-attr contains both a man and a bench that are correctly localized, but the man is not sitting on the bench. Figure 6.6 (b) shows that although SG-obj-attr-rel retrieves incorrect results for "smiling man wearing hat", it fails gracefully by returning images of a smiling woman wearing a hat, a man wearing a hat who is not smiling, and a smiling man wearing a helmet.

6.5.3 Object localization

We have shown that our scene graph representation improves image retrieval results; here, we show that it can also aid in localizing individual objects. For each image

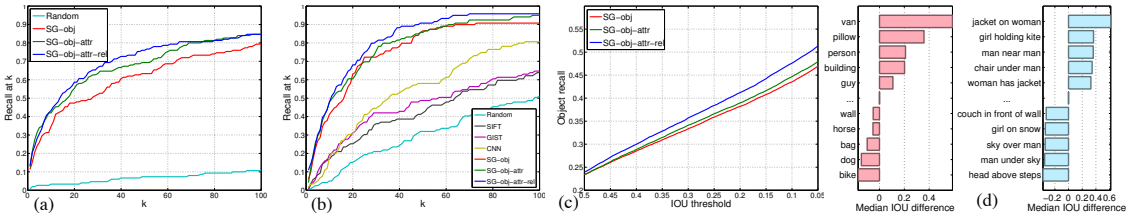


Figure 6.7: (a) Retrieval performance for entire scenes, (b) for partial scenes. (c) Object localization performance for entire scenes. (d) Increase in localization performance of our full model SG-obj-attr-rel vs SG-obj for individual objects (left) and objects participating in a relation (right). In (d), positive values indicate the SG-obj-attr-rel performs better than SG-obj.

I and its corresponding ground-truth (GT) scene graph G from our test set, we use each model to generate a grounding of G to I. For each object in G, we compute the intersection over union (IoU) between its GT position in the image and its position in the grounding generated by our model.

Figure 6.6 (c) gives an example scene graph and its computed grounding under SG-obj-attr-rel and SG-obj. SG-obj labels "man" as "woman" and vice versa, but SG-obj-attr-rel is able to correct this error. Note that the scene graph does not specify any direct relationships between the man and the woman; instead, SG-obj-attr-rel must rely on the relationships between the two people and other objects in the scene ("man holding phone", "woman has jacket").

To quantitatively compare our models, we report the median IoU across all object instances in all test images, (Med IoU) and the fraction of object instances with IoU above various thresholds (IoU@t). Table 6.2 shows that SG-obj-attr-rel outperforms both SG-obj and SG-obj-attr on all metrics. Interestingly, comparing SG-obj-attr-rel to SG-obj shows a nearly five-fold increase in median IoU. The generally low values for median IoU highlight the difficulty of the automatic scene graph grounding task. Figure 6.7 (c) shows that SG-obj-attr-rel performs particularly well compared to the baseline models at IoU thresholds below 0.5.

To gain more insight into the performance of our model, for each object instance in the test set we compute the difference in IoU with the GT between the grounding found by SG-obj-attr-rel and the grounding found by SG-obj. For each object class

that occurs at least 4 times in the test set, we compute the median difference in IoU between the two methods, and perform the same analysis for (object, relationship, object) tuples, where for each such tuple in the test set we compute the mean IoU of the objects referenced by the tuple. The top and bottom 5 object classes and tuples are visualized in Figure 6.7 (d).

This figure suggests that the context provided by SG-obj-attr-rel helps to localize rare objects (such as "van" and "guy") and objects with large appearance variations (such as "pillow" and "building"). SG-obj-attr-rel also gives large gains for tuples with well-defined spatial constraints ("jacket on woman", "chair under man"). Tuples encoding less well-defined relationships ("man under sky", "girl on snow") may suffer in performance as the model penalizes valid configurations not seen at training time.

6.6 Discussion

In this chapter we have used scene graphs as a novel representation for detailed semantics in visual scenes, and introduced a novel dataset of scene graphs grounded to real-world images. We have used this representation and dataset to construct a CRF model for semantic image retrieval using scene graphs as queries. We have shown that this model outperforms methods based on object detection and low-level visual features. We believe that semantic image retrieval is one of many exciting applications of our scene graph representation and dataset. In the next chapter we will see how scene graphs can be used to *generate* novel images rather than retrieving them from a fixed database.

Chapter 7

Image Generation from Scene Graphs

7.1 Introduction

What I cannot create, I do not understand

– Richard Feynman

The act of *creation* requires a deep understanding of the thing being created: chefs, novelists, and filmmakers must understand food, writing, and film at a much deeper level than diners, readers, or moviegoers. If our computer vision systems are to truly understand the visual world, they must be able not only *recognize* images but also to *generate* them.

Aside from imparting deep visual understanding, methods for generating realistic images can also be practically useful. In the near term, automatic image generation can aid the work of artists or graphic designers. One day, we might replace image and video search engines with algorithms that generate customized images and videos in response to the individual tastes of each user.

As a step toward these goals, there has been exciting recent progress on *text to image synthesis* [186, 187, 189, 253], combining recurrent networks and Generative

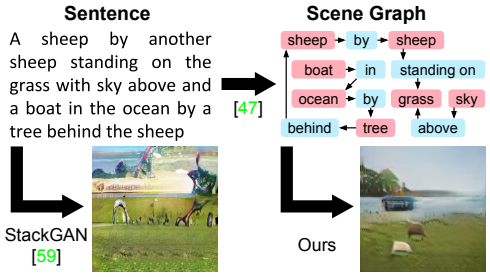


Figure 7.1: State-of-the-art methods for generating images from sentences, such as StackGAN [253], struggle to faithfully depict complex sentences with many objects. We overcome this limitation by generating images from *scene graphs*, allowing our method to reason explicitly about objects and their relationships.

Adversarial Networks [64] to generate images from natural language descriptions.

These methods can give stunning results on limited domains, such as fine-grained descriptions of birds or flowers. However as shown in Figure 7.1, leading methods for generating images from sentences struggle on complex sentences with many objects.

A sentence is a linear structure, with one word following another; however as shown in Figure 7.1, the information conveyed by a complex sentence can often be more explicitly represented as a *scene graph* of objects and their relationships. Scene graphs are a powerful structured representation for both images and language; they have been used for semantic image retrieval [99] and for evaluating [3] and improving [139] image captioning; methods have also been developed for converting sentences to scene graphs [199] and for predicting scene graphs from images [142, 162, 236, 240].

In this chapter we aim to generate complex images with many objects and relationships by conditioning our generation on scene graphs, allowing our model to reason explicitly about objects and their relationships.

This new task brings new challenges. We must be able to process scene graph inputs; for this we use a *graph convolution network* which passes information along graph edges. After processing the graph, we must bridge the gap between the symbolic graph-structured input and the two-dimensional image output; to this end we construct a *scene layout* by predicting bounding boxes and segmentation masks for all objects in the graph. Having predicted a layout, we must generate an image which respects it; for this we use a *cascaded refinement network* (CRN) [25] which processes

the layout at increasing spatial scales. Finally, we must ensure that our generated images are realistic and contain recognizable objects; we thus train adversarially against a pair of *discriminator* networks operating on image patches and generated objects. All components of the model are learned jointly in an end-to-end manner.

We experiment on two datasets: Visual Genome [116], which provides human annotated scene graphs, and COCO-Stuff [16] where we construct synthetic scene graphs from ground-truth object positions. On both datasets we show qualitative results demonstrating our method's ability to generate complex images which respect the objects and relationships of the input scene graph, and perform comprehensive ablations to validate each component of our model.

Automated evaluation of generative images models is a challenging problem unto itself [219], so we also evaluate our results with two user studies on Amazon Mechanical Turk. Compared to StackGAN [253], a leading system for text to image synthesis, users find that our results better match COCO captions in 68% of trials, and contain 59% more recognizable objects.

7.2 Related Work

Generative Image Models fall into three recent categories: Generative Adversarial Networks (GANs) [64, 182] jointly learn a *generator* for synthesizing images and a *discriminator* classifying images as real or fake; Variational Autoencoders [109] use variational inference to jointly learn an *encoder* and *decoder* mapping between images and latent codes; autoregressive approaches [167, 168] model likelihoods by conditioning each pixel on all previous pixels.

Conditional Image Synthesis conditions generation on additional input. GANs can be conditioned on category labels by providing labels as an additional input to both generator and discriminator [58, 157] or by forcing the discriminator to predict the label [165]; we take the latter approach.

Reed et al. [187] generate images from text using a GAN; Zhang et al. [253] extend this approach to higher resolutions using multistage generation. Related to our approach, Reed et al. generate images conditioned on sentences and keypoints

using both GANs [186] and multiscale autoregressive models [189]; in addition to generating images they also predict locations of unobserved keypoints using a separate generator and discriminator operating on keypoint locations.

Chen and Koltun [25] generate high-resolution images of street scenes from ground-truth semantic segmentation using a cascaded refinement network (CRN) trained with a perceptual feature reconstruction loss [57, 93]; we use their CRN architecture to generate images from scene layouts.

Related to our layout prediction, Chang et al. have investigated text to 3D scene generation [20, 23]; other approaches to image synthesis include stochastic grammars [92], probabalistic programming [120], inverse graphics [121], neural de-rendering [231], and generative ConvNets [234].

Scene Graphs represent scenes as directed graphs, where nodes are objects and edges give relationships between objects. Scene graphs have been used for image retrieval [99] and to evaluate image captioning [3]; some work converts sentences to scene graphs [199] or predicts grounded scene graphs for images [142, 162, 236, 240]. Most work on scene graphs uses the Visual Genome dataset [116], which provides human-annotated scene graphs.

Deep Learning on Graphs. Some methods learn embeddings for graph nodes given a single large graph [71, 175, 217] similar to word2vec [156] which learns embeddings for words given a text corpus. These differ from our approach, since we must process a new graph on each forward pass.

More closely related to our work are Graph Neural Networks (GNNs) [63, 65, 198] which generalize recursive neural networks [53, 205, 206] to operate on arbitrary graphs. GNNs and related models have been applied to molecular property prediction [41], program verification [133], modeling human motion [87], and premise selection for theorem proving [225]. Some methods operate on graphs in the spectral domain [15, 78, 110] though we do not take this approach.

7.3 Method

Our goal is to develop a model which takes as input a *scene graph* describing objects and their relationships, and which generates a realistic image corresponding to the graph. The primary challenges are threefold: first, we must develop a method for processing the graph-structured input; second, we must ensure that the generated images respect the objects and relationships specified by the graph; third, we must ensure that the synthesized images are realistic.

We convert scene graphs to images with an *image generation network* f, shown in Figure 7.2, which inputs a scene graph G and noise z and outputs an image $\hat{I} = f(G, z)$.

The scene graph G is processed by a *graph convolution network* which gives embedding vectors for each object; as shown in Figures 7.2 and 7.3, each layer of graph convolution mixes information along edges of the graph.

We respect the objects and relationships from G by using the object embedding vectors from the graph convolution network to predict bounding boxes and segmentation masks for each object; these are combined to form a *scene layout*, shown in Figure 7.2, which acts as an intermediate between the graph and the image domains.

The output image $\hat{I}$ is generated from the layout using a *cascaded refinement network* (CRN) [25], shown in the right half of Figure 7.2; each of its modules processes the layout at increasing spatial scales, eventually generating the image $\hat{I}$.

We generate realistic images by training f adversarially against a pair of *discriminator* networks D_{img} and D_{obj} which encourage the image $\hat{I}$ to both appear realistic and to contain realistic, recognizable objects.

Each of these components is described in more detail below; Appendix C describes the exact architecures used in our experiments.

Scene Graphs. The input to our model is a *scene graph* [99] describing objects and relationships between objects. Given a set of object categories $\mathcal{C}$ and a set of relationship categories $\mathcal{R}$, a scene graph is a tuple (O, E) where $O = \{o_1, \ldots, o_n\}$ is a set of objects with each $o_i \in \mathcal{C}$, and $E \subseteq O \times \mathcal{R} \times O$ is a set of directed edges of the form (o_i, r, o_j) where $o_i, o_j \in O$ and $r \in \mathcal{R}$.

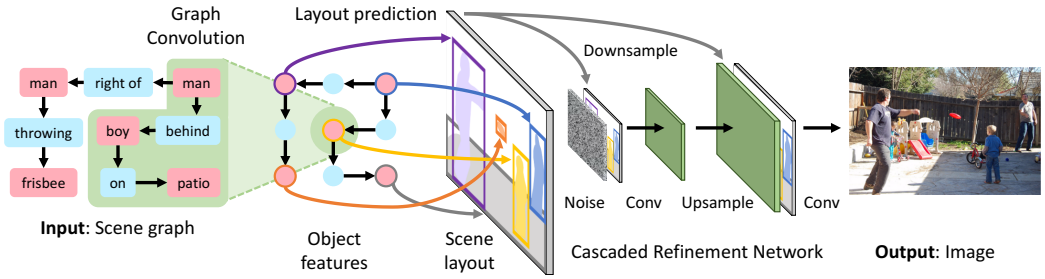


Figure 7.2: Overview of our image generation network f for generating images from scene graphs. The input to the model is a *scene graph* specifying objects and relationships; it is processed with a *graph convolution network* (Figure 7.3) which passes information along edges to compute embedding vectors for all objects. These vectors are used to predict bounding boxes and segmentation masks for objects, which are combined to form a *scene layout* (Figure 7.4). The layout is converted to an image using a *cascaded refinement network* (CRN) [25]. The model is trained adversarially against a pair of *discriminator networks*. During training the model observes ground-truth object bounding boxes and (optionally) segmentation masks, but these are predicted by the model at test-time.

As a first stage of processing, we use a learned embedding layer to convert each node and edge of the graph from a categorical label to a dense vector, analogous to the embedding layer typically used in neural language models.

Graph Convolution Network. To process scene graphs in an end-to-end manner, we need a neural network module which operates natively on graphs. To this end we use a *graph convolution network* composed of several *graph convolution layers*.

A traditional 2D convolution layer takes as input a spatial grid of feature vectors and produces as output a new spatial grid of vectors, where each output vector is a function of a local neighborhood of its corresponding input vector; in this way a convolution aggregates information across local neighborhoods of the input. A single convolution layer can operate on inputs of arbitrary shape through the use of *weight sharing* across all neighborhoods in the input.

Our graph convolution layer performs a similar function: given an input graph with vectors of dimension D_{in} at each node and edge, it computes new vectors of dimension D_{out} for each node and edge. Output vectors are a function of a neighborhood of their corresponding inputs, so that each graph convolution layer propagates

information along edges of the graph. A graph convolution layer applies the same function to all edges of the graph, allowing a single layer to operate on graphs of arbitrary shape.

Concretely, given input vectors $v_i, v_r \in \mathbb{R}^{D_{in}}$ for objects $o_i \in O$ and edges $(o_i, r, o_j) \in E$, we compute output vectors $v_i', v_r' \in \mathbb{R}^{D_{out}}$ for all nodes and edges using three functions g_s, g_p, and g_o, which take as input the triple of vectors (v_i, v_r, v_j) for an edge and output new vectors for the subject o_i, predicate r, and object o_j.

To compute the output vectors v_r' for edges we simply set $v_r' = g_p(v_i, v_r, v_j)$. Updating object vectors is more complex, since an object may participate in many relationships; as such the output vector v_i' for an object o_i should depend on all vectors v_j for objects to which o_i is connected via graph edges, as well as the vectors v_r for those edges. To this end, for each edge starting at o_i we use g_s to compute a *candidate vector*, collecting all such candidates in the set V_i^s; we similarly use g_o to compute a set of candidate vectors V_i^o for all edges terminating at o_i. Concretely,

$$V_i^s = \{g_s(v_i, v_r, v_j) : (o_i, r, o_j) \in E\} \tag{7.1}$$

$$V_i^o = \{g_o(v_j, v_r, v_i) : (o_j, r, o_i) \in E\}. \tag{7.2}$$

The output vector v_i' for object o_i is computed as $v_i' = h(V_i^s \cup V_i^o)$ where h is a symmetric function pooling an input set of vectors to a single output vector. A computational graph for a single graph convolution layer is shown in Figure 7.3.

In our implementation, the functions g_s, g_p, and g_o are computed with a single network which concatenates its three input vectors, feeds them to a multilayer perceptron (MLP), and computes three output vectors using fully-connected output heads. The pooling function h averages its input vectors and feeds the result to a MLP.

Scene Layout. Processing the input scene graph with a series of graph convolution layers gives an embedding vector for each object which aggregates information across all objects and relationships in the graph.

In order to generate an image, we must move from the graph domain to the image domain. To this end, we use the object embedding vectors to compute a *scene layout* which gives the coarse 2D structure of the image to generate; we compute the scene

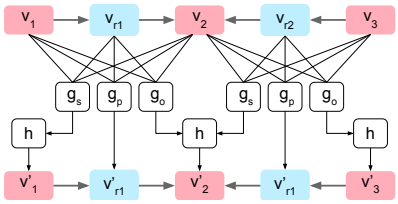


Figure 7.3: Computational graph illustrating a single graph convolution layer. The graph consists of three objects o_1, o_2, and o_3 and two edges (o_1, r_1, o_2) and (o_3, r_2, o_2). Along each edge, the three input vectors are passed to functions g_s, g_p, and g_o; g_p directly computes the output vector for the edge, while g_s and g_o compute *candidate vectors* which are fed to a symmetric pooling function h to compute output vectors for objects.

layout by predicting a segmentation mask and bounding box for each object using an *object layout network*, shown in Figure 7.4.

The object layout network receives an embedding vector v_i of shape D for object o_i and passes it to a *mask regression network* to predict a soft binary mask $\hat{m}_i$ of shape $M \times M$ and a *box regression network* to predict a bounding box $\hat{b}_i = (x_0, y_0, x_1, y_1)$. The mask regression network consists of several convolution and upsampling layers terminating in a sigmoid nonlinearity so that elements of the mask lies in the range $(0, 1)$; the box regression network is a MLP.

We multiply the embedding vector v_i elementwise with the mask $\hat{m}_i$ to give a masked embedding of shape $D \times M \times M$ which is then warped to the position of the bounding box using bilinear interpolation [86] to give an object layout. The scene layout is then the sum of all object layouts.

During training we use ground-truth bounding boxes b_i to compute the scene layout; at test-time we instead use predicted bounding boxes $\hat{b}_i$.

Cascaded Refinement Network. Given the scene layout, we must synthesize an image that respects the object positions given in the layout. For this task we use a Cascaded Refinement Network [25] (CRN). A CRN consists of a series of convolutional refinement modules, with spatial resolution doubling between modules; this allows generation to proceed in a coarse-to-fine manner.

Each module receives as input both the scene layout (downsampled to the input resolution of the module) and the previous module's output. These inputs are concatenated channelwise and passed to a pair of 3×3 convolution layers; the output

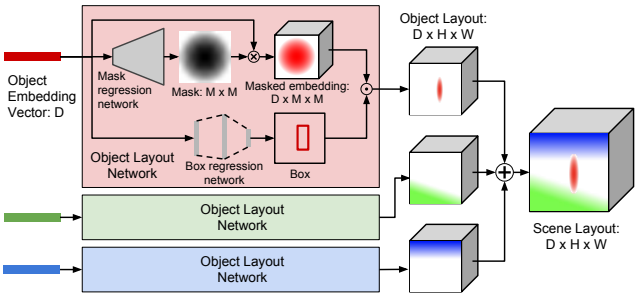


Figure 7.4: We move from the graph domain to the image domain by computing a *scene layout*. The embedding vector for each object is passed to an object layout network which predicts a layout for the object; summing all object layouts gives the scene layout. Internally the object layout network predicts a soft binary segmentation mask and a bounding box for the object; these are combined with the embedding vector using bilinear interpolation to produce the object layout.

is upsampled using nearest-neighbor interpolation before being passed to the next module. The first module takes Gaussian noise $z \sim p_z$ as input, and the output from the last module is passed to two final convolution layers to produce the output image.

Discriminators. We generate realistic output images by training the image generation network f adversarially against a pair of *discriminator* networks D_{img} and D_{obj}. A discriminator D attempts to classify its input x as real or fake by maximizing the objective [64]

$$\mathcal{L}_{GAN} = \mathop{\mathbb{E}}_{x \sim p_{\text{real}}} \log D(x) + \mathop{\mathbb{E}}_{x \sim p_{\text{fake}}} \log(1 - D(x)) \tag{7.3}$$

where $x \sim p_{\text{fake}}$ are outputs from the generation network f. At the same time, f attempts to generate outputs which will fool the discriminator by minimizing $\mathcal{L}_{GAN}$.[1]

The patch-based *image discriminator* D_{img} ensures that the overall appearance of generated images is realistic; it classifies a regularly spaced, overlapping set of image patches as real or fake, and is implemented as a fully convolutional network, similar to the discriminator used in [84].

The *object discriminator* D_{obj} ensures that each object in the image appears realistic; its input are the pixels of an object, cropped and rescaled to a fixed size using bilinear interpolation [86]. In addition to classifying each object as real or fake, D_{obj}

[1] In practice, to avoid vanishing gradients f typically maximizes the surrogate objective $\mathbb{E}_{x \sim p_{\text{fake}}} \log D(x)$ instead of minimizing $\mathcal{L}_{GAN}$ [64].

also ensures that each object is recognizable using an *auxiliary classifier* [165] which predicts the object's category; both D_{obj} and f attempt to maximize the probability that D_{obj} correctly classifies objects.

Training. We jointly train the generation network f and discriminators D_{obj} and D_{img}. The generation network is trained to minimize the weighted sum of six losses:

- *Box loss* $\mathcal{L}_{box} = \sum_{i=1}^{n} \|b_i - \hat{b}_i\|_1$ penalizing the L_1 difference between ground-truth and predicted boxes
- *Mask loss* $\mathcal{L}_{mask}$ penalizing differences between ground-truth and predicted masks with pixelwise cross-entropy; not used for models trained on Visual Genome
- *Pixel loss* $\mathcal{L}_{pix} = \|I - \hat{I}\|_1$ penalizing the L_1 difference between ground-truth generated images
- *Image adversarial loss* $\mathcal{L}_{GAN}^{img}$ from D_{img} encouraging generated image patches to appear realistic
- *Object adversarial loss* $\mathcal{L}_{GAN}^{obj}$ from the D_{obj} encouraging each generated object to look realistic
- *Auxiliarly classifier loss* $\mathcal{L}_{AC}^{obj}$ from D_{obj}, ensuring that each generated object can be classified by D_{obj}

Implementation Details. We augment all scene graphs with a special *image* object, and add special *in image* relationships connecting each true object with the *image* object; this ensures that all scene graphs are connected.

We train all models using Adam [108] with learning rate 10^{-4} and batch size 32 for 1 million iterations; training takes about 3 days on a single Tesla P100. For each minibatch we first update f, then update D_{img} and D_{obj}.

We use ReLU for graph convolution; the CRN and discriminators use discriminators use LeakyReLU [145] and batch normalization [83]. Full details about our architecture can be found in Appendix C, and code reproducing our experiments is available at `https://github.com/google/sg2im`.

7.4 Experiments

We train our model to generate 64×64 images on the Visual Genome [116] and COCO-Stuff [16] datasets. In our experiments we aim to show that our method generates images of complex scenes which respect the objects and relationships of the input scene graph.

7.4.1 Datasets

COCO. We experiment on the 2017 COCO-Stuff dataset [16], which augments a subset of COCO [138] with additional stuff categories. The dataset annotates 40K train and 5K val images with bounding boxes and segmentation masks for 80 *thing* categories (people, cars, *etc.*) and 91 *stuff* categories (sky, grass, *etc.*).

We use these annotations to construct synthetic scene graphs based on the 2D image coordinates of the objects, using six mutually exclusive geometric relationships: *left of*, *right of*, *above*, *below*, *inside*, and *surrounding*.

We ignore objects covering less than 2% of the image, and use images with 3 to 8 objects; we divide the COCO-Stuff 2017 val set into our own val and test sets, leaving us with 24,972 train, 1024 val, and 2048 test images.

Visual Genome. We experiment on Visual Genome [116] version 1.4 (VG) which comprises 108,077 images annotated with scene graphs. We divide the data into 80% train, 10% val, and 10% test; we use object and relationship categories occurring at least 2000 and 500 times respectively in the train set, leaving 178 object and 45 relationship types. We ignore small objects, and use images with between 3 and 30 objects and at least one relationship; this leaves us with 62,565 train, 5,506 val, and 5,088 test images with an average of ten objects and five relationships per image.

Visual Genome does not provide segmentation masks, so we omit the mask prediction loss for models trained on VG.

Figure 7.5: Examples of 64 × 64 generated images using graphs from the test sets of Visual Genome (left four columns) and COCO (right four columns). For each example we show the input scene graph and a manual translation of the scene graph into text; our model processes the scene graph and predicts a layout consisting of bounding boxes and segmentation masks for all objects; this layout is then used to generate the image. We also show some results for our model using ground-truth rather than predicted scene layouts. Some scene graphs have duplicate relationships, shown as double arrows. For clarity, we omit masks for some stuff categories such as sky and water.

7.4.2 Qualitative Results

Figure 7.5 shows example scene graphs from the Visual Genome and COCO test sets and generated images using our method, as well as predicted object bounding boxes and segmentation masks.

These examples show that our method can generate scenes with multiple objects, and even multiple instances of the same object type: for example Figure 7.5 (a) shows two sheep, (d) shows two busses, (g) contains three people, and (i) shows two cars.

These examples also show that our method generates images which respect the relationships of the input graph; for example in (i) we see one broccoli *left of* a second broccoli, with a carrot *below* the second broccoli; in (j) the man is *riding* the horse, and both the man and the horse have *legs* which have been properly positioned.

Figure 7.5 also shows examples of images generated by our method using ground-truth rather than predicted object layouts. In some cases our predicted layouts can vary significantly from the ground-truth layouts. For example in (k) the graph does not specify the position of the bird and our method renders it standing on the ground, but in the ground-truth layout the bird is flying in the sky. Our model is sometimes bottlenecked by layout prediction, such as (n) where using the ground-truth rather than predicted layout significantly improves the image quality.

In Figure 7.6 we demonstrate our model's ability to generate complex images by starting with simple graphs on the left and progressively building up to more complex graphs. From this example we can see that object positions are influenced by the relationships in the graph: in the top sequence adding the relationship *car below kite* causes the car to shift to the right and the kite to shift to the left so that the relationship is respected. In the bottom sequence, adding the relationship *boat on grass* causes the boat's position to shift.

7.4.3 Ablation Study

We demonstrate the necessity of all components of our model by comparing the image quality of several ablated versions of our model, shown in Table 7.1; see Appendix C for example images from ablated models.

Figure 7.6: Images generated by our method trained on Visual Genome. In each row we start from a simple scene graph on the left and progressively add more objects and relationships moving to the right. Images respect relationships like *car below kite* and *boat on grass*.

We measure image quality using *Inception score*[2] [196] which uses an ImageNet classification model [193, 215] to encourage recognizable objects within images and diversity across images. We test several ablations of our model:

No gconv omits graph convolution, so boxes and masks are predicted from initial object embedding vectors. It cannot reason jointly about the presence of different objects, and can only predict one box and mask per category.

No relationships uses graph convolution layers but ignores all relationships from the input scene graph except for trivial *in image* relationships; graph convolution allows this model to jointly about objects. Its poor performance demonstrates the utility of the scene graph relationships.

No discriminators omits both D_{img} and D_{obj}, relying on the pixel regression loss $\mathcal{L}_{pix}$ to guide the generation network. It tends to produce overly smoothed images.

[2]Defined as $\exp(\mathbb{E}_{\hat{I}} KL(p(y|\hat{I})\|p(y)))$ where the expectation is taken over generated images $\hat{I}$ and $p(y|\hat{I})$ is the predicted label distribution.

Method	Inception	
	COCO	VG
Real Images (64×64)	16.3 ± 0.4	13.9 ± 0.5
Ours (No gconv)	4.6 ± 0.1	4.2 ± 0.1
Ours (No relationships)	3.7 ± 0.1	4.9 ± 0.1
Ours (No discriminators)	4.8 ± 0.1	3.6 ± 0.1
Ours (No D_{obj})	5.6 ± 0.1	5.0 ± 0.2
Ours (No D_{img})	5.6 ± 0.1	$\mathbf{5.7 \pm 0.3}$
Ours (Full model)	$\mathbf{6.7 \pm 0.1}$	5.5 ± 0.1
Ours (GT Layout, no gconv)	7.0 ± 0.2	6.0 ± 0.2
Ours (GT Layout)	$\mathbf{7.3 \pm 0.1}$	$\mathbf{6.3 \pm 0.2}$
StackGAN [253] (64×64)	$\mathbf{8.4 \pm 0.2}$	-

Table 7.1: Ablation study using Inception scores. On each dataset we randomly split our test-set samples into 5 groups and report mean and standard deviation across splits. On COCO we generate five samples for each test-set image by constructing different synthetic scene graphs. For StackGAN we generate one image for each of the COCO test-set captions, and downsample their 256×256 output to 64×64 for fair comparison with our method.

No $\mathbf{D_{obj}}$ and **No $\mathbf{D_{img}}$** omit a discriminator. On both datasets, using any discriminator leads to significant improvements over models using $\mathcal{L}_{pix}$ alone. On COCO the two discriminators are complimentary, and combining them in our full model leads to large improvements. On VG, omitting D_{img} does not degrade performance.

In addition to ablations, we also compare with two **GT Layout** versions of our model which omit the $\mathcal{L}_{box}$ and $\mathcal{L}_{mask}$ losses, and use ground-truth bounding boxes during both training and testing; on COCO they also use ground-truth segmentation masks, similar to Chen and Koltun [25]. These methods give an upper bound to our model's performance in the case of perfect layout prediction.

Omitting graph convolution degrades performance even when using ground-truth layouts, suggesting that scene graph relationships and graph convolution have benefits beyond simply predicting object positions.

7.4.4 Object Localization

In addition to looking at images, we can also inspect the bounding boxes predicted by our model. One measure of box quality is high agreement between predicted and ground-truth boxes; in Table 7.2 we show the object recall of our model at two

	R@0.3		R@0.5		σ_x		σ_{area}	
	COCO	VG	COCO	VG	COCO	VG	COCO	VG
Ours (No gconv)	46.9	20.2	20.8	6.4	0	0	0	0
Ours (No rel.)	21.8	16.5	7.6	6.9	**0.1**	**0.1**	**0.2**	**0.1**
Ours (Full)	**52.4**	**21.9**	**32.2**	**10.6**	**0.1**	**0.1**	**0.2**	**0.1**

Table 7.2: Statistics of predicted bounding boxes. R@t is object recall with an IoU threshold of t, and measures agreement with ground-truth boxes. σ_x and σ_{area} measure box variety by computing the standard deviation of box x-positions and areas within each object category and then averaging across categories.

intersection-over-union thresholds.

Another measure for boxes is *variety*: predicted boxes for objects should vary in response to the other objects and relationships in the graph. Table 7.2 shows the mean per-category standard deviations of box position and area.

Without graph convolution, our model can only learn to predict a single bounding box per object category. This model achieves nontrivial object recall, but has no variety in its predicted boxes, as $\sigma_x = \sigma_{area} = 0$.

Using graph convolution without relationships, our model can jointly reason about objects when predicting bounding boxes; this leads to improved variety in its predictions. Without relationships, this model's predicted boxes have less agreement with ground-truth box positions.

Our full model with graph convolution and relationships achieves both variety and high agreement with ground-truth boxes, indicating that it can use the relationships of the graph to help localize objects with greater fidelity.

7.4.5 User Studies

Automatic metrics such as Inception scores or box statistics give a coarse measure of image quality; the true measure of success is human judgement of the generated images. For this reason we performed two user studies on Mechanical Turk to evaluate our results.

We are unaware of any previous end-to-end methods for generating images from scene graphs, so we compare our method with StackGAN [253], a state-of-the art method for generating images from sentence descriptions.

Which image matches the caption better?

User choice	332 / 1024 (32.4%)	**692** / 1024 **(67.6%)**

Figure 7.7: We performed a user study to compare the semantic interpretability of our method against StackGAN [253]. **Top:** We use StackGAN to generate an image from a COCO caption, and use our method to generate an image from a scene graph constructed from the COCO objects corresponding to the caption. We show users the caption and both images, and ask which better matches the caption. **Bottom:** Across 1024 val image pairs, users prefer the results from our method by a large margin.

Despite the different input modalities between our method and StackGAN, we can compare the two on COCO, which in addition to object annotations also provides captions for each image. We use our method to generate images from synthetic scene graphs built from COCO object annotations, and StackGAN[3] to generate images from COCO captions for the same images. Though the methods receive different inputs, they should generate similar images due to the correspondence between COCO captions and objects.

In user studies we downsample StackGAN images to 64×64 to compensate for differing resolutions between their method and ours. We repeat all trials with three workers and randomize order in all trials.

Caption Matching. We measure semantic interpretability by showing users a COCO caption, an image generated by StackGAN from that caption, and an image generated by our method from a scene graph built from the COCO objects corresponding to the caption. We ask users to select the image that better matches the caption. An example image pair and results are shown in Figure 7.7.

[3]We use the pretrained COCO model provided by the authors at `https://github.com/hanzhanggit/StackGAN-Pytorch`

Which objects are present? motorcycle, person, clouds

Thing recall	470 / 1650 (28.5%)	**772** / 1650 **(46.8%)**
Stuff Recall	1285 / 3556 (36.1%)	**2071** / 3556 **(58.2%)**

Figure 7.8: We performed a user study to measure the number of recognizable objects in images from our method and from StackGAN [253]. **Top:** We use StackGAN to generate an image from a COCO caption, and use our method to generate an image from a scene graph built from the COCO objects corresponding to the caption. For each image, we ask users which COCO objects they can see in the image. **Bottom:** Across 1024 val image pairs, we measure the fraction of *things* and *stuff* that users can recognize in images from each method. Our method produces more objects.

This experiment is biased toward StackGAN, since the caption may contain information not captured by the scene graph. Even so, a majority of workers preferred our results in 67.6% of image pairs, demonstrating that compared to StackGAN our method more frequently generates complex, semantically meaningful images.

Object Recall. This experiment measures the number of recognizable objects in each method's images. In each trial we show an image from one method and a list of COCO objects and ask users to identify which objects appear in the image. An example and results are snown in Figure 7.8.

We compute the fraction of objects that a majority of users believed were present, dividing the results into *things* and *stuff*. Both methods achieve higher recall for stuff than things, and our method achieves significantly higher object recall, with 65% and 61% relative improvements for thing and stuff recall respectively.

This experiment is biased toward our method since the scene graph may contain objects not mentioned in the caption, but it demonstrates that compared to StackGAN, our method produces images with more recognizable objects.

7.5 Discussion

In this chapter we have developed an end-to-end method for generating images from scene graphs. Compared to leading methods which generate images from text descriptions, generating images from structured scene graphs rather than unstructured text allows our method to reason explicitly about objects and relationships, and generate complex images with many recognizable objects.

Chapter 8

Conclusions

8.1 Overview

In this dissertation we have examined several tasks involving vision and language where compositionality plays a central role. In each of these tasks we have seen that building models to exploit compositionality allow a richer form of visual intelligence.

In Chapters 2 and 3 we discussed image captioning, and we presented models which reason compositionally by breaking images into regions. Doing so allowed us to generate new kinds of descriptions: dense captions and multisentence descriptive paragraphs. Each of these two novel description types can more richly describe input images than standard single-sentence captions.

In Chapters 4 and 5 we discussed visual question answering, with a focus on long, complex questions requiring visual reasoning. We introduced the CLEVR dataset as a benchmark to evaluate visual reasoning, and showed that many existing models for visual question answering struggle to reason. We then introduced an explicitly compositional model that converts natural-language questions to functional programs, which are then executed on the image. We showed that our model outperforms prior work on the CLEVR dataset, and can generalize better to novel situations.

In Chapters 6 and 7 we discussed two related text-to-image problems: image retrieval and image generation conditioned on linguistic descriptions. For both of

these tasks we showed that modeling descriptions as compositional scene graphs of objects and semantic relationships allowed us to build models that could retrieve and generate complex images with many objects and relationships.

8.2 Future Directions

In this dissertation we have taken some initial steps toward incorporating compositionality into modern computer vision systems. We have shown a few concrete examples of tasks involving vision and language where compositionality is important. However some important questions remain unanswered, and there are many more potential areas where compositional reasoning could be applied.

8.2.1 Explicitly and implicitly compositional models

The models presented in this dissertation are fairly *explicit* in their compositional representations: for captioning, our models (in Chapters 2 and 3) make explicit decisions about where to look in images; for question answering, our model (in Chapter 5) forms an explicit functional program; for text-to-image retrieval and generation, our models (in Chapters 6 and 7) work on scene graphs that make explicit the objects and relationships of the scene. It is intuitively obvious that tasks like captioning, question answering, and image synthesis require some form of compositional reasoning; however must we represent compositionality so explicitly in our models?

Since we initially published the compositional model from Chapter 5, followup work by other groups has shown very strong performance on the CLEVR benchmarks with end-to-end models that do not explicitly form functional programs [82, 174, 197]. Although they are not as explicitly compositional as the model we presented in Chapter 5, their strong performance on CLEVR suggests that they are capable of some form of compositional reasoning, but it is *implicit*.

These results raise some interesting questions. To what degree can an end-to-end model perform compositional reasoning? How is compositionality learned and represented in the weights and activations of a blackbox neural network model? Can

we discover some guiding theoretical principles about when we should prefer explicitly compositional models or end-to-end models which represent composition implicitly?

8.2.2 Video understanding

One application area where compositionality may prove useful is video understanding. In particular, the *temporal* structure of videos adds a new dimension along which to exploit compositionality.

In images, we have seen an object-centric approach to compositionality: images can be understood by recognizing objects, identifying semantic attributes of objects, and reasoning about relationships between objects. These same notions could also be applied to video understanding. However in videos we must also account for *change*: the set of visible or relevant objects as well as their attributes and relationships may change over the course of a video. A first-order task in video understanding might be to recognize all of the objects, attributes, and relationships over time – this might amount to converting a raw video input into some sort of temporal scene graph.

A more involved task in video understanding might be not only to recognize changes that happen as a video unfolds, but also to explain *why* those changes happened. For example a glass of water may change state from full to empty *because* the glass fell over *because* the table it was sitting on shook *because* a person ran into the table. In contrast to the object-centric compositional representations we have seen thus far, explaining change in this way requires a compositional understanding of videos in terms of *events* connected through time via *causal links*.

Much more work is needed in order to operationalize these ideas. What is the right way to represent compositional structure in video? What sort of datasets do we need to collect or construct to study compositionality in video? As we move from recognition to causal explanation, how do we evaluate when our methods succeed?

Appendix A

A Dataset for Compositional Reasoning: Supplemental Material

A.1 Basic Functions

As described in Section 4.3 and shown in Figure 4.2, each question in CLEVR is associated with a functional program built from a set of basic functions. In this section we detail the semantics of these basic functional building blocks.

Data Types. Our basic functional building blocks operate on values of the following types:

- `Object`: A single object in the scene.
- `ObjectSet`: A set of zero or more objects in the scene.
- `Integer`: An integer between 0 and 10 (inclusive).
- `Boolean`: Either `yes` or `no`.
- Value types:
  - `Size`: One of `large` or `small`.
  - `Color`: One of `gray`, `red`, `blue`, `green`, `brown`, `purple`, `cyan`, or `yellow`.
  - `Shape`: One of `cube`, `sphere`, or `cylinder`.

  - `Material`: One of `rubber` or `metal`.

- `Relation`: One of `left`, `right`, `in front`, or `behind`.

Basic Functions. The functional program representations of questions are built from the following set of basic building blocks. Each of these functions takes the image's scene graph as an additional implicit input.

- **scene** ($\emptyset \to$ `ObjectSet`)
  Returns the set of all objects in the scene.
- **unique** (`ObjectSet` $\to$ `Object`)
  If the input is a singleton set, then return it as a standalone `Object`; otherwise raise an exception and flag the question as *ill-posed* (See Section 3).
- **relate** (`Object` $\times$ `Relation` $\to$ `ObjectSet`)
  Return all objects in the scene that have the specified spatial relation to the input object. For example if the input object is a red cube and the input relation is `left`, then return the set of all objects in the scene that are left of the red cube.
- **count** (`ObjectSet` $\to$ `Integer`)
  Returns the size of the input set.
- **exist** (`ObjectSet` $\to$ `Boolean`)
  Returns `yes` if the input set is nonempty and `no` if it is empty.
- **Filtering functions**: These functions filter the input objects by some attribute, returning the subset of input objects that match the input attribute. For example calling `filter_size` with the first input `small` will return the set of all small objects in the second input.
  - **filter_size** (`ObjectSet` $\times$ `Size` $\to$ `ObjectSet`)
  - **filter_color** (`ObjectSet` $\times$ `Color` $\to$ `ObjectSet`)
  - **filter_material** (`ObjectSet` $\times$ `Material` $\to$ `ObjectSet`)
  - **filter_shape** (`ObjectSet` $\times$ `Shape` $\to$ `ObjectSet`)
- **Query functions**: These functions return the specified attribute of the input object; for example calling `query_color` on a red object returns `red`.

- **query_size** (`Object` → `Size`)
- **query_color** (`Object` → `Color`)
- **query_material** (`Object` → `Material`)
- **query_shape** (`Object` → `Shape`)

- **Logical operators**:
  - **AND** (`ObjectSet` × `ObjectSet` → `ObjectSet`)
    Returns the intersection of the two input sets.
  - **OR** (`ObjectSet` × `ObjectSet` → `ObjectSet`)
    Returns the union of the two input sets.

- **Same-attribute relations**: These functions return the set of objects that have the same attribute value as the input object, not including the input object. For example calling `same_shape` on a cube returns the set of all cubes in the scene, excluding the query cube.
  - **same_size** (`Object` → `ObjectSet`)
  - **same_color** (`Object` → `ObjectSet`)
  - **same_material** (`Object` → `ObjectSet`)
  - **same_shape** (`Object` → `ObjectSet`)

- **Integer comparison**: Checks whether the two integer inputs are equal, or whether the first is less than or greater than the second, returning either `yes` or `no`.
  - **equal_integer** (`Integer` × `Integer` → `Boolean`)
  - **less_than** (`Integer` × `Integer` → `Boolean`)
  - **greater_than** (`Integer` × `Integer` → `Boolean`)

- **Attribute comparison**: These functions return `yes` if their inputs are equal and `no` if they are not equal.
  - **equal_size** (`Size` × `Size` → `Boolean`)
  - **equal_material** (`Material` × `Material` → `Boolean`)
  - **equal_color** (`Color` × `Color` → `Boolean`)
  - **equal_shape** (`Shape` × `Shape` → `Boolean`)

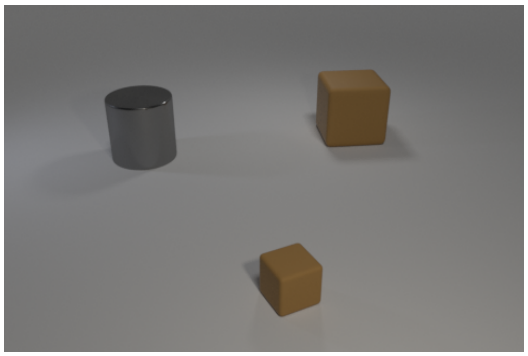


Figure A.1: For the above image and the question "What color is the cube behind the cylinder?", the effective question is "What color is the cube?" (see text for details).

A.2 Effective Question Size

In Section 4.5 we note that some questions can be correctly answered without correctly resolving all intermediate object references, and define a question's *effective question* to quantitatively measure this effect.

For any question we can compute its *effective question* by pruning functions from the question's program; the effective question is the smallest such pruned program that, when executed on the scene graph for the question's image, gives the same answer as the original question.

Some pruned questions may be *ill-posed*, meaning that some object references do not refer to a unique object. For example, consider the question *"What color is the cube behind the cylinder?"*; its associated program is

query_color(unique(filter_shape(cube,
relate(behind, unique(filter_shape(cylinder, scene()))))))

Imagine executing this program on the scene shown in Figure A.1. The innermost *filter_shape* gives a set containing the cylinder, the *relate* returns a set containing just the large cube in the back, the outer *filter_shape* does nothing, and the *query_color* returns `brown`.

This question is not *ill-posed* (Section 3) because the reference to *"the cube"* cannot be resolved without the rest of the question; however this question's effective size is less than its actual size because the question can be correctly answered without

resolving this object reference correctly.

To compute the effective question, we attempt to prune functions from this program. Starting from the innermost function and working out, whenever we find a function whose input type is `Object` or `ObjectSet`, we construct a pruned question by replacing that function's input with a *scene* function and executing it. The smallest such pruned program that gives the same answer as the original program is the effective question.

Pruned questions may be ill-posed, so we execute them with modified semantics. The output type of the *unique* function is changed from `Object` to `ObjectSet`, and it simply returns its input set. All functions taking an `Object` input are modified to take an `ObjectSet` input instead by mapping the original function over its input set and flattening the resulting set; thus the *relate* functions return the set of objects in the scene that have the specified relationship with any of the input objects, and the *query* functions return sets of values rather than single values.

Therefore for this example question we consider the following sequence of pruned programs. First we prune the inner *filter_shape* function:

query_color(unique(filter_shape(cube,relate(behind, unique(scene()))))))

The *relate* function now returns the set of objects which are behind some object, so it returns the large cube and the cylinder (since it is behind the small cube). The *filter_shape* function removes the cylinder, and the *query_color* returns a singleton set containing `brown`.

Next we prune the inner *unique* function:

query_color(unique(filter_shape(cube,relate(behind, scene()))))

Since *unique* computes the identity for pruned questions, execution is the same as above.

Next we prune the *relate* function:

query_color(unique(filter_shape(cube, scene())))

Now the *filter_shape* returns the set of both cubes, but since both are brown the *query_color* still returns a singleton set containing `brown`.

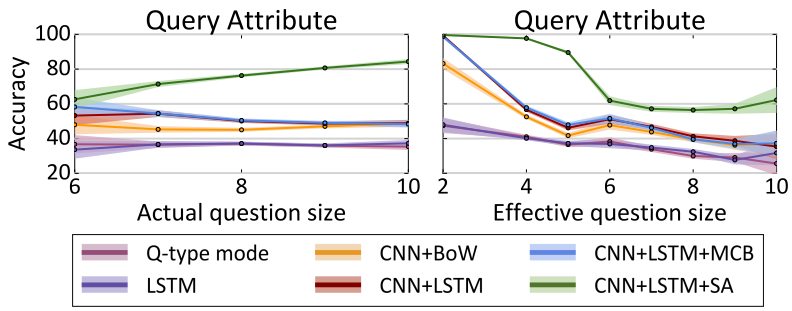


Figure A.2: Accuracy on query questions *vs.* actual and effective question size, restricting to questions with a *same-attribute* relationship. Figure 7 shows the same plots for questions without a *same-attribute* relationship. For both groups of questions we see that accuracy decreases as effective question size increases.

Next we prune the *filter_shape* function:

query_color(unique(scene()))

Now the *query_color* receives the set of all three input objects, so it returns a set containing `brown` and `gray`, which is different from the original question.

The effective question is therefore:

query_color(unique(filter_shape(cube, scene())))

A.2.1 Accuracy vs Question Size

Figure 4.7 in the main text shows model accuracy on *query-attribute* questions as a function of actual and effective question size, excluding questions with *same-attribute* relationships. Questions with same-attribute relationships have a maximum question size of 10 but questions without same-attribute relationships have a maximum size of 20; combining these questions thus leads to unwanted correlations between question size and difficulty.

In Figure A.2 we show model accuracy *vs.* actual and effective question size for questions with same-attribute relationships. Similar to Figure 4.7, we see that model accuracy either remains constant or increases as actual question size increases, but all models show a clear decrease in accuracy as effective question size increases.

A.3 Dynamic Module Networks

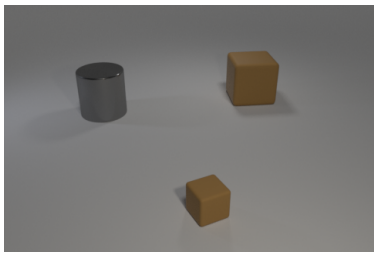
Module networks [4, 5] are a novel approach to visual question answering where a set of differentiable *modules* are used to assemble a custom network architecture to answer each question. Each module is responsible for performing a specific function such as *finding* a particular type of object, *describing* the current object of attention, or performing a *logical and* operation to merge attention masks. This approach seems like a natural fit for the rich, compositional questions in CLEVR; unfortunately we found that parsing heuristics tuned for the VQA dataset did not generalize to the longer, more complex questions in CLEVR.

Dynamic module networks [5] generate network architectures by performing a dependency parse of the question, using a set of heuristics to compute a set of *layout fragments*, combining these fragments to create *candidate layouts*, and ranking the candidate layouts using an MLP.

For some questions, the heuristics are unable to produce any layout fragments; in this case, the system uses a simple default network architecture as a fallback for answering that question. On a random sample of 10,000 questions from the VQA dataset [7], we found that dynamic module networks resorted to default architecture for 7.8% of questions; on a random sample of 10,000 questions from CLEVR, the default network architecture was used for 28.9% of questions. This suggests that the same parsing heuristics used for VQA do not apply to the questions in CLEVR; therefore the method of [5] did not work out-of-the box on CLEVR.

A.4 Example images and questions

The remaining pages show randomly selected images and questions from CLEVR. Each question is annotated with its answer, question type, and size. Recall from Section 3 that a question's *type* is the outermost function in the question's functional program, and a question's *size* is the number of functions in its program.

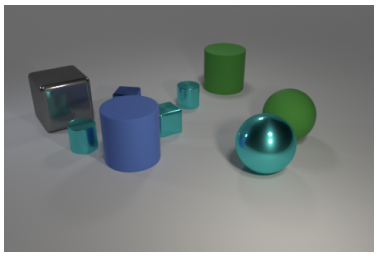


Q: There is a rubber cube in front of the big cylinder in front of the big brown matte thing; what is its size?
A: small
Q-type: query_size
Size: 14

Q: What color is the object that is on the left side of the small rubber thing?
A: gray
Q-type: query_color
Size: 7

Q: There is another cube that is made of the same material as the small brown block; what is its size?
A: large
Q-type: query_size
Size: 9

Q: What number of other matte objects are the same shape as the small rubber object?
A: 1
Q-type: count
Size: 7

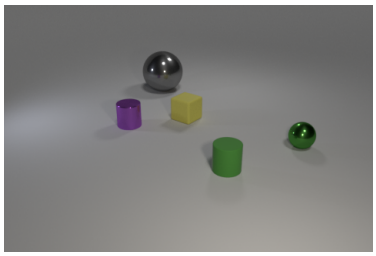


Q: Are there fewer metallic objects that are on the left side of the large cube than cylinders to the left of the cyan shiny block?
A: yes
Q-type: less_than
Size: 16

Q: There is a green rubber thing that is left of the rubber thing that is right of the rubber cylinder behind the gray shiny block; what is its size?
A: large
Q-type: query_size
Size: 17

Q: What is the size of the matte thing that is on the left side of the large cyan object and in front of the small blue thing?
A: large
Q-type: query_size
Size: 14

Q: The green matte thing that is behind the large green thing that is to the right of the big cyan metal object is what shape?
A: cylinder
Q-type: query_shape
Size: 14

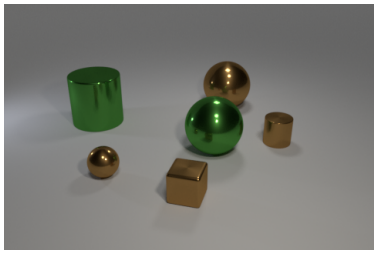


Q: Is the number of tiny metal objects left of the purple metallic cylinder the same as the number of small metal objects to the left of the small metal sphere?
A: no
Q-type: equal_integer
Size: 19

Q: Do the large shiny object and the thing to the left of the big gray object have the same shape?
A: no
Q-type: equal_shape
Size: 13

Q: There is a small thing that is the same color as the tiny shiny ball; what is it made of?
A: rubber
Q-type: query_material
Size: 9

Q: Is there anything else that is the same shape as the tiny yellow matte thing?
A: no
Q-type: exist
Size: 7

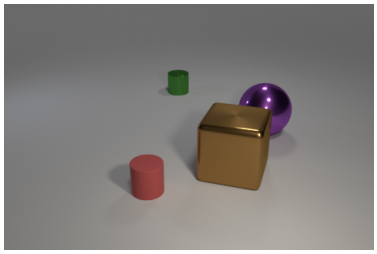


Q: What color is the small shiny cube?
A: brown
Q-type: query_color
Size: 6

Q: There is a tiny shiny sphere left of the cylinder in front of the large cylinder; what color is it?
A: brown
Q-type: query_color
Size: 13

Q: Is there a tiny thing that has the same material as the brown cylinder?
A: yes
Q-type: exist
Size: 7

Q: What is the material of the tiny object to the right of the brown shiny ball behind the tiny shiny cylinder?
A: metal
Q-type: query_material
Size: 14

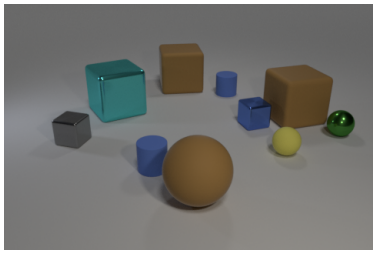


Q: What number of cylinders are purple metal objects or purple matte things?
A: 0
Q-type: count
Size: 9

Q: Does the large purple shiny object have the same shape as the tiny object that is behind the matte thing?
A: no
Q-type: equal_shape
Size: 14

Q: There is a big purple shiny object; what shape is it?
A: sphere
Q-type: query_shape
Size: 6

Q: There is a object that is both on the left side of the brown metal block and in front of the large purple shiny ball; how big is it?
A: small
Q-type: query_size
Size: 16

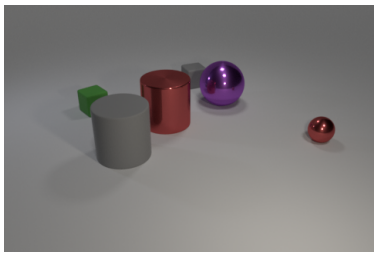


Q: There is a large brown block in front of the tiny rubber cylinder that is behind the cyan block; are there any big cyan metallic cubes that are to the left of it?
A: yes
Q-type: exist
Size: 20

Q: What number of cylinders have the same color as the metal ball?
A: 0
Q-type: count
Size: 7

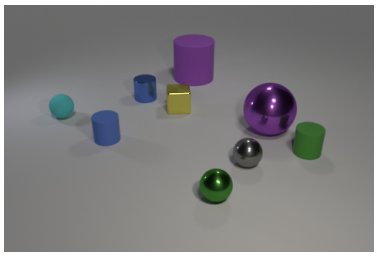
Q: There is a big shiny object; are there any blue shiny cubes behind it?
A: no
Q-type: exist
Size: 9

Q: The cyan block that is the same material as the tiny gray thing is what size?
A: large
Q-type: query_size
Size: 9



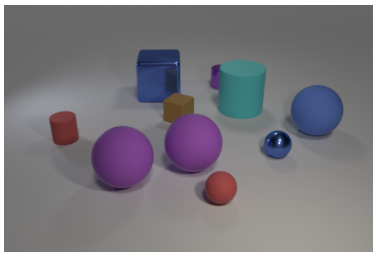
Q: How big is the gray rubber object that is behind the big shiny thing behind the big metallic thing that is on the left side of the purple ball?
A: small
Q-type: query_size
Size: 17

Q: There is a purple ball that is the same size as the red cylinder; what material is it?
A: metal
Q-type: query_material
Size: 9



Q: There is a tiny rubber thing that is the same color as the metal cylinder; what shape is it?
A: cylinder
Q-type: query_shape
Size: 9

Q: What is the shape of the tiny green thing that is made of the same material as the large cylinder?
A: cylinder
Q-type: query_shape
Size: 9



Q: There is a small ball that is made of the same material as the large block; what color is it?
A: blue
Q-type: query_color
Size: 9

Q: Is the size of the red rubber sphere the same as the purple metal thing?
A: yes
Q-type: equal_size
Size: 12

Q: Is there another green rubber cube that has the same size as the green matte cube?
A: no
Q-type: exist
Size: 10

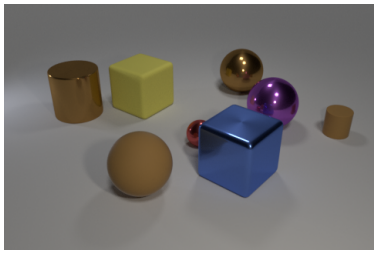
Q: Is the large matte thing the same shape as the big red object?
A: yes
Q-type: equal_shape
Size: 11

Q: Do the blue metallic object and the green metal thing have the same shape?
A: no
Q-type: equal_shape
Size: 11

Q: The big matte thing is what color?
A: purple
Q-type: query_color
Size: 5

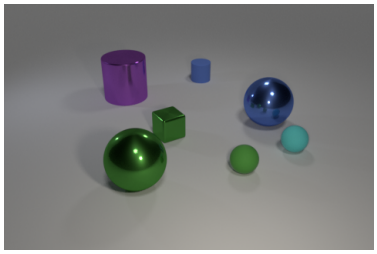
Q: What is the material of the purple cylinder?
A: metal
Q-type: query_material
Size: 5

Q: There is a blue ball that is the same size as the brown thing; what material is it?
A: metal
Q-type: query_material
Size: 8



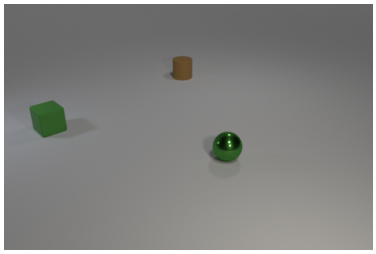
Q: How many small spheres are the same color as the big rubber cube?
A: 0
Q-type: count
Size: 9

Q: Is the tiny ball made of the same material as the large purple ball?
A: yes
Q-type: equal_material
Size: 12



Q: How many small green things are behind the green rubber sphere in front of the blue thing that is in front of the large purple metal cylinder?
A: 1
Q-type: count
Size: 18

Q: The cylinder that is the same size as the blue metallic sphere is what color?
A: purple
Q-type: query_color
Size: 9



Q: What is the color of the matte thing that is left of the thing behind the tiny green thing behind the tiny shiny sphere?
A: green
Q-type: query_color
Size: 15

Q: There is a thing in front of the tiny block; is its color the same as the matte object in front of the tiny brown rubber thing?
A: yes
Q-type: equal_color
Size: 17

Q: There is a large cube that is right of the red sphere; what number of large yellow things are on the right side of it?
A: 0
Q-type: count
Size: 12

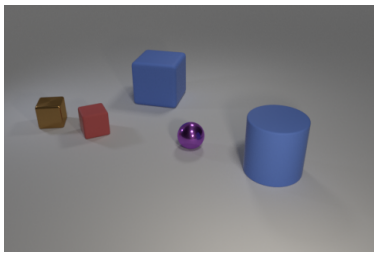
Q: Do the matte cylinder and the purple shiny object have the same size?
A: no
Q-type: equal_size
Size: 11

Q: What is the size of the green metal object right of the large ball that is on the left side of the big blue metal sphere?
A: small
Q-type: query_size
Size: 15

Q: There is a metallic ball that is the same color as the tiny cylinder; what size is it?
A: large
Q-type: query_size
Size: 9

Q: The green thing behind the small green thing on the right side of the brown matte object is what shape?
A: cube
Q-type: query_shape
Size: 12

Q: Is there a green matte cube that has the same size as the shiny thing?
A: yes
Q-type: exist
Size: 8



Q: Are there more brown shiny objects behind the large rubber cylinder than gray blocks?
A: yes
Q-type: greater_than
Size: 14

Q: What color is the matte object to the right of the large block?
A: blue
Q-type: query_color
Size: 8

Q: Do the blue cube and the cylinder have the same size?
A: yes
Q-type: equal_size
Size: 10

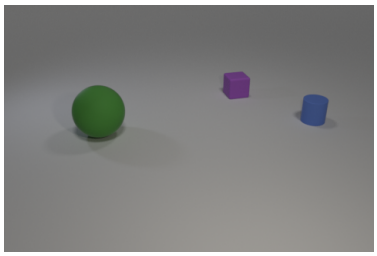
Q: The ball has what size?
A: small
Q-type: query_size
Size: 4

Q: There is a tiny brown rubber thing; is its shape the same as the thing that is in front of the small matte cylinder?
A: no
Q-type: equal_shape
Size: 15

Q: What number of yellow rubber things are the same size as the gray thing?
A: 0
Q-type: count
Size: 7

Q: What number of tiny brown rubber objects are behind the rubber object that is on the right side of the cylinder on the right side of the big cyan cylinder?
A: 0
Q-type: count
Size: 16

Q: Are there an equal number of brown matte objects that are right of the tiny brown object and gray rubber cylinders that are to the left of the large cyan shiny cylinder?
A: yes
Q-type: equal_integer
Size: 20



Q: Are there any other things that are the same shape as the large green thing?
A: no
Q-type: exist
Size: 6

Q: The matte thing that is both in front of the purple cube and to the left of the blue rubber cylinder is what color?
A: green
Q-type: query_color
Size: 15

Q: Are there fewer small rubber cylinders in front of the green ball than purple cylinders?
A: no
Q-type: less_than
Size: 14

Q: What number of other objects are there of the same shape as the large rubber object?
A: 0
Q-type: count
Size: 6

Q: Are there any rubber things that have the same size as the yellow metallic cylinder?
A: no
Q-type: exist
Size: 8

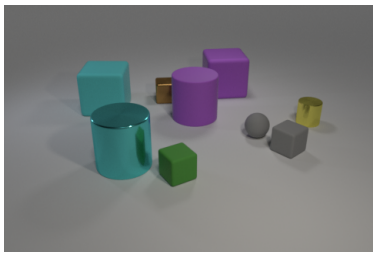
Q: What is the size of the brown sphere?
A: small
Q-type: query_size
Size: 5

Q: What number of cubes are the same color as the rubber ball?
A: 1
Q-type: count
Size: 7

Q: What shape is the tiny metal thing behind the large block in front of the tiny brown block?
A: cube
Q-type: query_shape
Size: 14

Q: What is the shape of the shiny thing that is behind the small blue rubber object and to the right of the tiny brown thing?
A: cube
Q-type: query_shape
Size: 15

Q: Is the number of purple matte cylinders behind the large purple thing less than the number of tiny rubber objects in front of the big cylinder?
A: yes
Q-type: less_than
Size: 18



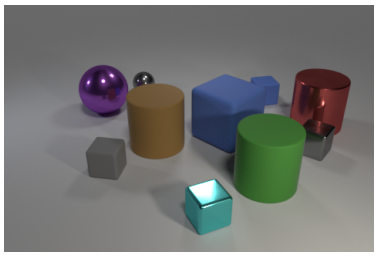
Q: Is the big cyan metal thing the same shape as the brown thing?
A: no
Q-type: equal_shape
Size: 11

Q: Is the size of the cyan cube the same as the metal cylinder that is behind the cyan cylinder?
A: no
Q-type: equal_size
Size: 15

Q: What is the shape of the blue rubber object in front of the brown object to the right of the big metallic block?
A: cylinder
Q-type: query_shape
Size: 13

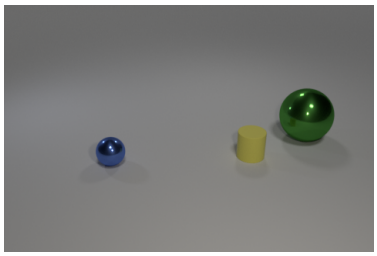
Q: Does the large red object have the same shape as the large blue thing?
A: no
Q-type: equal_shape
Size: 11

Q: There is a large cylinder that is the same color as the tiny cylinder; what is it made of?
A: rubber
Q-type: query_material
Size: 9



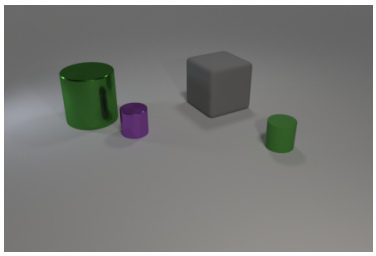
Q: There is a red shiny thing right of the purple metal sphere; what is its shape?
A: cylinder
Q-type: query_shape
Size: 10

Q: What number of purple shiny things are there?
A: 1
Q-type: count
Size: 4



Q: Does the small ball have the same color as the small cylinder in front of the big sphere?
A: no
Q-type: equal_color
Size: 15

Q: How many other things are the same size as the green sphere?
A: 0
Q-type: count
Size: 6



Q: There is a object that is behind the big green metal cylinder; what is its material?
A: rubber
Q-type: query_material
Size: 9

Q: How big is the gray thing?
A: large
Q-type: query_size
Size: 4

Q: Are the red object and the cyan cube made of the same material?
A: yes
Q-type: equal_material
Size: 10

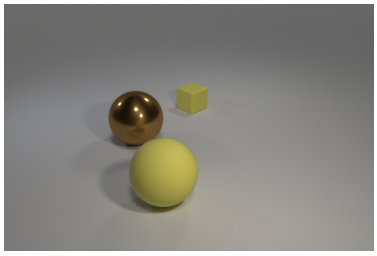
Q: Are there more metallic objects that are right of the large red shiny cylinder than gray matte objects?
A: no
Q-type: greater_than
Size: 14

Q: How many blocks are yellow rubber things or large green shiny things?
A: 0
Q-type: count
Size: 10

Q: There is a shiny object in front of the yellow cylinder; is it the same shape as the yellow thing?
A: no
Q-type: equal_shape
Size: 13

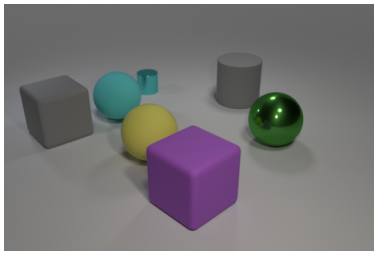
Q: Do the green object behind the tiny green matte cylinder and the small purple object have the same material?
A: yes
Q-type: equal_material
Size: 16

Q: What number of rubber blocks are there?
A: 1
Q-type: count
Size: 4



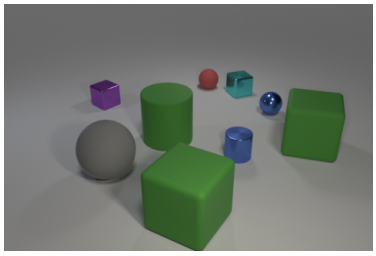
Q: How big is the yellow thing behind the brown shiny thing?
A: small
Q-type: query_size
Size: 8

Q: Is the size of the brown metal ball the same as the yellow thing that is behind the yellow rubber sphere?
A: no
Q-type: equal_size
Size: 16



Q: How many things are either green things or matte cubes behind the green ball?
A: 2
Q-type: count
Size: 11

Q: There is a cyan object that is to the right of the cyan rubber sphere; is its size the same as the gray rubber cylinder?
A: no
Q-type: equal_size
Size: 16



Q: What color is the rubber ball in front of the metal cube to the left of the matte cube left of the blue metallic sphere?
A: gray
Q-type: query_color
Size: 18

Q: What shape is the cyan shiny thing that is the same size as the red matte object?
A: cube
Q-type: query_shape
Size: 9

Q: Are there fewer small yellow things to the left of the large yellow matte ball than large brown objects?
A: yes
Q-type: less_than
Size: 15

Q: What material is the other thing that is the same shape as the brown thing?
A: rubber
Q-type: query_material
Size: 6

Q: There is a sphere to the right of the large yellow ball; what material is it?
A: metal
Q-type: query_material
Size: 9

Q: Are there the same number of tiny cylinders that are behind the cyan metal object and purple blocks right of the gray cube?
A: no
Q-type: equal_integer
Size: 17

Q: There is a green cylinder on the right side of the big gray ball; does it have the same size as the ball that is behind the metal ball?
A: no
Q-type: equal_size
Size: 19

Q: What is the size of the green block behind the big gray matte sphere?
A: large
Q-type: query_size
Size: 11

Appendix B

Inferring and Exceuting Programs for Visual Reasoning: Supplemental Material

B.1 Implementation Details

Code to reproduce our experiments is available online[1]. We also detail some key implementation details here.

B.1.1 Program Generator

In all experiments our program generator is an LSTM sequence-to-sequence model [214]. It comprises two learned recurrent neural networks: the *encoder* receives the natural-language question as a sequence of words, and summarizes the question as a fixed-length vector; the *decoder* receives this fixed-length vector as input and produces the predicted program as a sequence of functions. The encoder and decoder do not share weights.

[1] `https://github.com/facebookresearch/clevr-iep`

The encoder converts the discrete words of the input question to vectors of dimension 300 using a learned word embedding layer; the resulting sequence of vectors is then processed with a two-layer LSTM using 256 hidden units per layer. The hidden state of the second LSTM layer at the final timestep is used as the input to the decoder network.

At each timestep the decoder network receives both the function from the previous timestep (or a special `<START>` token at the first timestep) and the output from the encoder network. The function is converted to a 300-dimensional vector with a learned embedding layer and concatenated with the decoder output; the resulting sequence of vectors is processed by a two-layer LSTM with 256 hidden units per layer. At each timestep the hidden state of the second LSTM layer is used to compute a distribution over all possible functions using a linear projection.

During supervised training of the program generator, we use Adam [108] with a learning rate of 5×10^{-4} and a batch size of 64; we train for a maximum of 32,000 iterations, employing early stopping based on validation set accuracy.

B.1.2 Execution Engine

The execution engine uses a Neural Module Network [4] to compile a custom neural network architecture based on the predicted program from the program generator. The input image is first resized to 224×224 pixels, then passed through a convolutional network to extract image features; the architecture of this network is shown in Table B.1.

The predicted program takes the form of a syntax tree; the leaves of the tree are `Scene` functions which receive visual input from the convolutional network. For ground-truth programs, the root of the tree is a function corresponding to one of the question types from the CLEVR dataset [96], such as `count` or `query_shape`. For predicted programs the root of the program tree could in principle be any function, but in practice we find that trained models tend only to predict as roots those function types that appear as roots of ground-truth programs.

Each function in the predicted program is associated with a *module* which receives

Layer	Output size
Input image	$3 \times 224 \times 224$
ResNet-101 [75] conv4_6	$1024 \times 14 \times 14$
Conv(3×3, $1024 \to 128$)	$128 \times 14 \times 14$
ReLU	$128 \times 14 \times 14$
Conv(3×3, $128 \to 128$)	$128 \times 14 \times 14$
ReLU	$128 \times 14 \times 14$

Table B.1: Network architecture for the convolutional network used in our execution engine. The ResNet-101 model is pretrained on ImageNet [193] and remains fixed while the execution engine is trained. The output from this network is passed to modules representing `Scene` nodes in the program.

either one or two inputs; this association gives rise to a custom neural network architecture corresponding to each program. Previous implementations of Neural Module networks [4, 5] used different architectures for each module type, customizing the module architecture to the function the module was to perform. In contrast we use a generic design for our modules: each module is a small residual block [75]; the exact architectures used for our unary and binary modules are shown in Tables B.2 and B.3 respectively.

In initial experiments we used Batch Normalization [83] after each convolution in the modules, but we found that this prevented the model from converging. Since each image in a minibatch may have a different program, our implementation of the execution engine iterates over each program in the minibatch one by one; as a result each module is only run with a batch size of one during training, leading to poor convergence when modules contain Batch Normalization.

The output from the final module is passed to a classifier which predicts a distribution over answers; the exact architecture of the classifier is shown in Table B.4.

When training the execution engine alone (using either ground-truth programs or predicted programs from a fixed program generator), we train using Adam [108] with a learning rate of 1×10^{-4} and a batch size of 64; we train for a maximum of 200,000 iterations and employ early stopping based on validation set accuracy.

Index	Layer	Output size
(1)	Previous module output	$128 \times 14 \times 14$
(2)	Conv(3×3, $128 \to 128$)	$128 \times 14 \times 14$
(3)	ReLU	$128 \times 14 \times 14$
(4)	Conv(3×3, $128 \to 128$)	$128 \times 14 \times 14$
(5)	Residual: Add (1) and (4)	$128 \times 14 \times 14$
(6)	ReLU	$128 \times 14 \times 14$

Table B.2: Architecture for unary modules used in the execution engine. These modules receive the output from one other module, except for the special `Scene` module which instead receives input from the convolutional network (Table B.1).

Index	Layer	Output size
(1)	Previous module output	$128 \times 14 \times 14$
(2)	Previous module output	$128 \times 14 \times 14$
(3)	Concatenate (1) and (2)	$256 \times 14 \times 14$
(4)	Conv(1×1, $256 \to 128$)	$128 \times 14 \times 14$
(5)	ReLU	$128 \times 14 \times 14$
(6)	Conv(3×3, $128 \to 128$)	$128 \times 14 \times 14$
(7)	ReLU	$128 \times 14 \times 14$
(8)	Conv(3×3, $128 \to 128$)	$128 \times 14 \times 14$
(9)	Residual: Add (5) and (8)	$128 \times 14 \times 14$
(10)	ReLU	$128 \times 14 \times 14$

Table B.3: Architecture for binary modules in the execution engine. These modules receive the output from two other modules. The binary modules in our system are `intersect`, `union`, `equal_size`, `equal_color`, `equal_material`, `equal_shape`, `equal_integer`, `less_than`, and `greater_than`.

Layer	Output size
Final module output	$128 \times 14 \times 14$
Conv(1×1, $128 \to 512$)	$512 \times 14 \times 14$
ReLU	$512 \times 14 \times 14$
MaxPool(2×2, stride 2)	$512 \times 7 \times 7$
FullyConnected($512 \cdot 7 \cdot 7 \to 1024$)	1024
ReLU	1024
FullyConnected($1024 \to \lvert\mathcal{A}\rvert$)	$\lvert\mathcal{A}\rvert$

Table B.4: Network architecture for the classifier used in our execution engine. The classifier receives the output from the final module and predicts a distribution over answers $\mathcal{A}$.

B.1.3 Joint Training

When jointly training the program generator and execution engine, we train using Adam with a learning rate of 5×10^{-5} and a batch size of 64; we train for a maximum of 100,000 iterations, again employing early stopping based on validation set accuracy.

We use a moving average baseline to reduce the variance of gradients estimated using REINFORCE; in particular our baseline is an exponentially decaying moving average of past rewards, with a decay factor of 0.99.

B.1.4 Baselines

We reimplement the baselines used in [96]:

LSTM. Our LSTM baseline receives the input question as a sequence of words, converts the words to 300-dimensional vectors using a learned word embedding layer, and processes the resulting sequence with a two-layer LSTM with 512 hidden units per layer. The LSTM hidden state from the second layer at the final timestep is passed to an MLP with two hidden layers of 1024 units each, with ReLU nonlinearities after each layer.

CNN+LSTM. Like the LSTM baseline, the CNN+LSTM model encodes the question using learned 300-dimensional word embeddings followed by a two-layer LSTM with 512 hidden units per layer. The image is encoded using the same CNN architecture as the execution engine, shown in Table B.1. The encoded question and (flattened) image features are concatenated and passed to a two-layer MLP with two hidden layers of 1024 units each, with ReLU nonlinearities after each layer.

CNN+LSTM+SA. The question and image are encoded in exactly the same manner as the CNN+LSTM baseline. However rather than concatenating these representations, they are fed to two consecutive Stacked Attention layers [241] with a hidden dimension of 512 units; this results in a 512-dimensional vector which is fed to a linear layer to predict answer scores.

This matches the CNN+LSTM+SA model as originally described by Yang et al. [241]; this also matches the CNN+LSTM+SA model used in [96].

Question:	*The brown object that is the same shape as the green shiny thing is what size?*
Fragments:	`(_what _thing)`
Question:	*What material is the big purple cylinder?*
Fragments:	`(material purple);(material big);(material (and purple big))`
Question:	*How big is the cylinder that is in front of the green metal object left of the tiny shiny thing that is in front of the big red metal ball?*
Fragments:	`(_what _thing)`
Question:	*Are there any metallic cubes that are on the right side of the brown shiny thing that is behind the small metallic sphere to the right of the big cyan matte thing?*
Fragments:	`(is brown);(is cubes);(is (and brown cubes))`
Question:	*Is the number of cyan things in front of the purple matte cube greater than the number of metal cylinders left of the small metal sphere?*
Fragments:	`(is cylinder);(is cube);(is (and cylinder cube))`
Question:	*Are there more small blue spheres than tiny green things?*
Fragments:	`(is blue);(is sphere);(is (and blue sphere))`
Question:	*Are there more big green things than large purple shiny cubes?*
Fragments:	`(is cube);(is purple);(is (and cube purple))`
Question:	*What number of things are large yellow metallic balls or metallic things that are in front of the gray metallic sphere?*
Fragments:	`(number gray);(number ball);(number (and gray ball))`
Question:	*The tiny cube has what color?*
Fragments:	`(_what _thing)`
Question:	*There is a small matte cylinder; is it the same color as the tiny shiny cube that is behind the large red metallic ball?*
Fragments:	`(_what _thing)`

Table B.5: Examples of random questions from the CLEVR training set, parsed using the code by Andreas et al. [5] for parsing questions from the VQA dataset [7]. Each parse gives a set of *layout fragments* separated by semicolons; in [5] these fragments are combined to produce *candidate layouts* for the module network. When the parser fails, it produces the default fallback fragment `(_what _thing)`.

CNN+LSTM+SA+MLP. Identical to CNN+LSTM+ SA; however the output

of the final stacked attention module is fed to a two-layer MLP with two hidden layers of 1024 units each, with ReLU nonlinearities after each layer.

Since all other other models (LSTM, CNN+LSTM, and ours) terminate in an MLP to predict the final answer distribution, the CNN+LSTM+SA+MLP gives a more fair comparison with the other methods.

Surprisingly, the minor architectural change of replacing the linear transform with an MLP significantly improves performance on the CLEVR dataset: CNN+LSTM+SA achieves an overall accuracy of 69.8, while CNN+LSTM+SA+MLP achieves 73.2. Much of this gain comes from improved performance on comparison questions; for example on shape comparison questions CNN+LSTM+SA achieves an accuracy of 50.9 and CNN+LSTM+SA+MLP achieves 69.7.

Training. All baselines are trained using Adam with a learning rate of 5×10^{-4} with a batch size of 64 for a maximum of 360,000 iterations, employing early stopping based on validation set accuracy.

B.2 Neural Module Network parses

The closest method to our own is that of Andreas et al. [5]. Their dynamic neural module networks first perform a dependency parse of the sentence; heuristics are then used to generate a set of *layout fragments* from the dependency parse. These fragments are heuristically combined, giving a set of *candidate layouts*; the final network layout is selected from these candidates through a learned reranking step.

Unfortunately we found that the parser used in [5] for VQA questions did not perform well on the longer questions in CLEVR. In Table B.5 we show random questions from the CLEVR training set together with the layout fragments computed using the parser from [5]. For many questions the parser fails, falling back to the fragment `(_what _thing)`; when this happens then the resulting module network will not respect the structure of the question at all. For questions where the parser does not fall back to the default layout, the resulting layout fragments often fail to capture key elements from the question; for example, after parsing the question *What material is the big purple cylinder?*, none of the resulting fragments mention the *cylinder*.

Appendix C

Image Generation from Scene Graphs: Supplemental Material

C.1 Network Architecture

Here we describe the exact network architectures for all components of our model.

C.1.1 Graph Convolution Layer

As described in Section 7.3 of the main text, we process the input scene graph with a *graph convolution network* composed of several *graph convolution layers*.

A graph convolution layer accepts as input a vector of dimension D_{in} for each node and edge in the graph, and computes new vectors of dimension D_{out} for each node and edge. A single graph convolution layer can be applied to graphs of any size of shape due to *weight sharing*. A single graph convolution layer proceeds in two stages.

First, along relationship of the scene graph we apply three functions g_s, g_p, and g_o; these functions take as input the vectors v_s, v_r, and v_o for the starting node, edge, and ending node of the relationship and produce new vectors for the two nodes and the edge. The new vector for the edge $v'_r = g_p(v_s, v_r, v_o)$ has dimension D_{out}, and is

Index	Inputs	Operation	Output shape
(1)	-	Subject vector v_s	D_{in}
(2)	-	Relationship vector v_r	D_{in}
(3)	-	Object vector v_o	D_{in}
(4)	(1), (2), (3)	Concatenate	$3D_{in}$
(5)	(4)	Linear($3D_{in} \to H$)	H
(6)	(5)	ReLU	H
(7)	(6)	Linear($H \to 2H + D_{out}$)	$2H + D_{out}$
(8)	(7)	ReLU	$2H + D_{out}$
(9)	(8)	Split into 3 chunks	H, D_{out}, H
(10)	(9)	1st chunk $\tilde{v}_s$	H
(11)	(9)	2nd chunk v'_r	D_{out}
(12)	(9)	3rd chunk $\tilde{v}_o$	H

Table C.1: Network architecture for the first network g used in graph convolution; this single network implements the three functions g_s, g_p, and g_o from the main text.

used as the output vector from the graph convolution layer for the edge. The new vectors for the starting and ending nodes $\tilde{v}_s = g_s(v_s, v_r, v_o)$ and $\tilde{v}_o = g_o(v_s, v_r, v_o)$ are *candidate vectors* of dimension H. In practice the three functions g_s, g_p, and g_o and implemented with a single multilayer perceptron (MLP) whose architecture is shown in Table C.1.

As a second stage of processing, for each object in the scene graph we collect all of its candidate vectors and process them with a symmeitric pooling function h which converts the set of candidate vectors into a a single vector of dimension D_{out}. Concretely, for object o_i in the scene graph G, let $V_i^s = \{g_s(v_i, v_r, v_j) : (o_i, r, o_j) \in G\}$ be the set of candidate vectors for o_i from relationships where o_i appears as the subject, and let $V_i^o = \{g_o(v_j, v_r, v_i) : (o_j, r, o_i) \in G\}$ be the set of candidate vectors for o_i from relationships where o_i appears as the object of the relationship. The pooling function h takes as input the two sets of vectors V_i^s and V_i^o, averages them, and feeds the result to an MLP to compute the output vector v'_i for object o_i from the graph convolution layer. The exact architecture of the network we use for h is shown in Table C.2.

A graph convolution layer has three hyperparameters defining its size: *input dimension* D_{in}, *hidden dimension* H, and *output dimension* D_{out}. We can therefore specify a graph convolution layer with the notation gconv($D_{in} \to H \to D_{out}$).

Index	Inputs	Operation	Output Shape
(1)	-	Subject candidate set V_i^s	$\lvert V_i^s \rvert \times H$
(2)	-	Object candidate set V_i^o	$\lvert V_i^o \rvert \times H$
(3)	(1), (2)	Set union	$(\lvert V_i^s \rvert + \lvert V_i^o \rvert) \times H$
(4)	(3)	Mean over axis 0	H
(5)	(4)	Linear($H \to H$)	H
(6)	(5)	ReLU	H
(7)	(6)	Linear($H \to D_{out}$)	D_{out}
(8)	(7)	ReLU	D_{out}

Table C.2: Network architecture for the second network h used in graph convolution; this network implements a symmetric pooling function to convert the set of all candidate vectors for an object into a single output vector.

Index	Inputs	Operation	Output Shape
(1)	-	Graph objects	O
(2)	-	Graph relationships	R
(3)	(1)	Object Embedding	$O \times 128$
(4)	(2)	Relationship embedding	$R \times 128$
(5)	(1), (2)	gconv($128 \to 512 \to 128$)	$O \times 128, R \times 128$
(6)	(5)	gconv($128 \to 512 \to 128$)	$O \times 128$, $R \times 128$
(7)	(6)	gconv($128 \to 512 \to 128$)	$O \times 128$, $R \times 128$
(8)	(7)	gconv($128 \to 512 \to 128$)	$O \times 128$, $R \times 128$
(9)	(8)	gconv($128 \to 512 \to 128$)	$O \times 128$, $R \times 128$

Table C.3: Architecture of the graph convolution network used to process input scene graphs. The input scene graph has O objects and R relationships. Due to weight sharing in graph convolutions, the same network can process graphs of any size or topology. The notation gconv($D_{in} \to H \to D_{out}$) is graph convolution with input dimension D_{in}, hidden dimension H, and output dimension D_{out}.

C.1.2 Graph Convolution Network

The input scene graph is processed by a *graph convolution network*, the exact architecture of which is shown in Table C.3. Our network first embeds the objects and relationships of the graph with embedding layers to produce vectors of dimension $D_{in} = 128$; we then use five layers of graph convolution with $D_{in} = D_{out} = 128$ and $H = 512$.

Index	Inputs	Operation	Output Shape
(1)	-	Object embedding vector	128
(2)	(1)	Linear($128 \to 512$)	512
(3)	(2)	ReLU	512
(4)	(3)	Linear($512 \to 4$)	4

Table C.4: Architecture of the box regression network.

C.1.3 Box Regression Network

We predict bounding boxes for images using a *box regression network*. The input to the box regression network are the final embedding vectors for objects produced by the graph convolution network. The output from the box regression network is a predicted bounding box for the object, parameterized as (x_0, y_0, x_1, y_1) where x_0, x_1 are the left and right coordinates of the box and y_0, y_1 are the top and bottom coordinates of the box; all box coordinates are normalized to be in the range $[0, 1]$. The architecture of the box regression network is shown in Table C.4.

C.1.4 Mask Regression Network

We predict segmentation masks for images using a *mask regression network*. The input to the mask regression network are the final embedding vectors for objects from the graph convolution network, and the output from the mask regresion network is a $M \times M$ segmentation mask with all elements in the range $(0, 1)$. The mask regression network is composed of a sequence of upsampling and convolution layers, terminating in a sigmoid nonlinearity; its exact architecture is shown in Table C.5.

C.1.5 Scene Layout

The final embedding vectors for objects from the graph convolution network are combined with the predicted bounding boxes and segmentation masks for objects to give a *scene layout*. The conversion from vectors, masks, and boxes to scene layouts does not have any learnable parameters.

The scene layout has shape $D \times H \times W$ where $D = 128$ is the dimension of

Index	Inputs	Operation	Output Shape
(1)	-	Object embedding vector	128
(2)	(1)	Reshape	$128 \times 1 \times 1$
(3)	(2)	Upsample	$128 \times 2 \times 2$
(4)	(3)	Batch Normalization	$128 \times 2 \times 2$
(5)	(4)	Conv$(3 \times 3, 128 \to 128)$	$128 \times 2 \times 2$
(6)	(5)	ReLU	$128 \times 2 \times 2$
(7)	(6)	Upsample	$128 \times 4 \times 4$
(8)	(7)	Batch Normalization	$128 \times 4 \times 4$
(9)	(8)	Conv$(3 \times 3, 128 \to 128)$	$128 \times 4 \times 4$
(10)	(9)	ReLU	$128 \times 4 \times 4$
(11)	(10)	Upsample	$128 \times 8 \times 8$
(12)	(11)	Batch Normalization	$128 \times 8 \times 8$
(13)	(12)	Conv$(3 \times 3, 128 \to 128)$	$128 \times 8 \times 8$
(14)	(13)	ReLU	$128 \times 8 \times 8$
(15)	(14)	Upsample	$128 \times 16 \times 16$
(16)	(15)	Batch Normalization	$128 \times 16 \times 16$
(17)	(16)	Conv$(3 \times 3, 128 \to 128)$	$128 \times 16 \times 16$
(18)	(17)	ReLU	$128 \times 16 \times 16$
(19)	(18)	conv$(1 \times 1, 128 \to 1)$	$1 \times 16 \times 16$
(20)	(19)	sigmoid	$1 \times 16 \times 16$

Table C.5: Architecture of the mask regression network. For 3D tensors we use $C \times H \times W$ layout, where C is the number of channels in the feature map and H and W are the height and width of the feature map. The notation Conv$(K \times K, C_{in} \to C_{out})$ is a convolution with $K \times K$ kernels, C_{in} input channels and C_{out} output channels; all convolutions are stride 1 with zero padding so that their input and output have the same spatial size. Upsample is a 2×2 nearest-neighbor upsampling.

embededing vectors for objects from the graph convolution network and $H \times W = 64 \times 64$ is the output resolution at which images will be generated.

C.1.6 Cascaded Refinement Network

The scene layout is converted to an image using a *Cascaded Refinement Network* (CRN) consisting of a number of *Cascaded Refinement Modules* (CRMs).

Each CRM recieves as input the scene layout of shape $D \times H \times W = 128 \times 64 \times 64$ and the previous feature map, and outputs a new feature map twice the spatial size of the input feature map. Internally each CRM upsamples the input feature map by a factor of 2, and downsamples the layout using average pooling the match the size of the upsampled feature map; the two are concatenated and processed with two convolution layers. A CRM taking input of shape $C_{in} \times H_{in} \times W_{out}$ and producing an output of shape $C_{out} \times H_{out} \times W_{out}$ (with $H_{out} = 2H_{in}$ and $W_{out} = 2W_{in}$ is denoted as CRM($H_{in} \times W_{in},\ C_{in} \rightarrow C_{out}$). The exact architecture of our CRMs is shown in Table C.6.

Our Cascaded Refinement Network consists of five Cascaded Refinement Modules. The input to the first module is Gaussian noise of shape $32 \times 2 \times 2$ and the output from the final module is processed with two final convolution layers to produce the output image. The architecture of the CRN is shown in Table C.7.

C.1.7 Batch Normalization in the Generator

Most implementations of batch normalization operate in two modes. In *train* mode, minibatches are normalized using the empirical mean and variance of features; in *eval* mode a running mean of feature means and variances are used to normalize minibatches instead. We found that training models in *train* mode and running them in *eval* mode at test-time led to significant image artifacts. To overcome this limitation while still benefitting from the optimization benefits that batch normalization provides, we train our models for 100K iterations using batch normalization in *train* mode, then continue training for an additional 900K iterations with batch normalization in *eval* mode.

Index	Inputs	Operation	Output Shape
(1)	-	Scene Layout	$D \times H \times W$
(2)	-	Input features	$C_{in} \times H_{in} \times W_{in}$
(3)	(1)	Average Pooling	$D \times H_{out} \times W_{out}$
(4)	(2)	Upsample	$C_{in} \times H_{out} \times W_{out}$
(5)	(3), (4)	Concatenation	$(D + C_{in}) \times H_{out} \times W_{out}$
(6)	(5)	Conv$(3 \times 3, D + C_{in} \to C_{out})$	$C_{out} \times H_{out} \times W_{out}$
(7)	(6)	Batch Normalization	$C_{out} \times H_{out} \times W_{out}$
(8)	(7)	LeakyReLU	$C_{out} \times H_{out} \times W_{out}$
(9)	(8)	Conv$(3 \times 3, C_{out} \to C_{out})$	$C_{out} \times H_{out} \times W_{out}$
(10)	(9)	Batch Normalization	$C_{out} \times H_{out} \times W_{out}$
(11)	(10)	LeakyReLU	$C_{out} \times H_{out} \times W_{out}$

Table C.6: Architecture of a Cascaded Refinement Module CRM($H_{in} \times W_{in}$, $C_{in} \to C_{out}$). The module accepts as input the scene layout, and an input feature map of shape $C_{in} \times H_{in} \times W_{in}$ and produces as output a feature map of shape $C_{out} \times H_{out} \times W_{out}$ where $H_{out} = 2H_{in}$ and $W_{out} = 2W_{in}$. For LeakyReLU nonlinearites we use negative slope 0.2.

Index	Inputs	Operation	Output Shape
(1)	-	Scene Layout	$128 \times 64 \times 64$
(2)	-	Gaussian Noise	$32 \times 2 \times 2$
(3)	(1), (2)	CRN$(2 \times 2,\ 32 \to 1024)$	$1024 \times 4 \times 4$
(4)	(1), (3)	CRN$(4 \times 4,\ 1024 \to 512)$	$512 \times 8 \times 8$
(5)	(1), (4)	CRN$(8 \times 4,\ 512 \to 256)$	$256 \times 16 \times 16$
(6)	(1), (5)	CRN$(16 \times 16,\ 256 \to 128)$	$128 \times 32 \times 32$
(7)	(1), (6)	CRN$(32 \times 32,\ 128 \to 64)$	$64 \times 64 \times 64$
(8)	(7)	Conv$(3 \times 3, 64 \to 64)$	$64 \times 64 \times 64$
(9)	(8)	LeakyReLU	$64 \times 64 \times 64$
(10)	(9)	Conv$(1 \times 1, 64 \to 3)$	$3 \times 64 \times 64$

Table C.7: Architecture of our Cascaded Refinement Network. CRM is a Cascaded Refinement Module, shown in Table C.6. LeakyReLU uses a negative slope of 0.2.

Index	Inputs	Operation	Output Shape
(1)	-	Object crop	$3 \times 32 \times 32$
(2)	(1)	Conv$(4 \times 4, 3 \to 64, s2)$	$64 \times 16 \times 16$
(3)	(2)	Batch Normalization	$64 \times 16 \times 16$
(4)	(3)	LeakyReLU	$64 \times 16 \times 16$
(5)	(4)	Conv$(4 \times 4, 64 \to 128, s2)$	$128 \times 8 \times 8$
(6)	(5)	Batch Normalization	$128 \times 32 \times 32$
(7)	(6)	LeakyReLU	$128 \times 32 \times 32$
(8)	(7)	Conv$(4 \times 4, 128 \to 256, s2)$	$256 \times 4 \times 4$
(9)	(8)	Global Average Pooling	256
(10)	(9)	Linear$(256 \to 1024)$	1024
(11)	(10)	Linear$(1024 \to 1)$	1
(12)	(10)	Linear$(1024 \to \lvert\mathcal{C}\rvert)$	$\lvert\mathcal{C}\rvert$

Table C.8: Architecture of our object discriminator D_{obj}. The input to the object discriminator is a 32×32 crop of an object in either a generated or real image. The object discriminator outputs both a score for real / fake (11) and a classification score over the object categories $\mathcal{C}$ (12). In this model all convolution layers have stride 2 and no zero padding. LeakyReLU uses a negative slope of 0.2.

Since discriminators are not used at test-time, batch normalization in the discriminators is always used in *train* mode.

C.1.8 Object Discriminator

Our object discriminator D_{obj} inputs image pixels corresponding to objects in real or generated images; objects are cropped using their bounding boxes to a spatial size of 32×32 using differentiable bilinear interpolation. The object discriminator serves two roles: it classifies objects as real or fake, and also uses an *auxiliary classifier* which attempts to classify each object. The exact architecture of our object discriminator is shown in Table C.8.

C.1.9 Image Discriminator

Our image discriminator D_{img} inputs a real or fake image, and classifies an overlapping grid of 8×8 image patches from its input image as real or fake. The exact architecture of our image discriminator is shown in Table C.9.

Index	Inputs	Operation	Output Shape
(1)	-	Image	$3 \times 64 \times 64$
(2)	(1)	Conv$(4 \times 4, 3 \to 64, s2)$	$64 \times 32 \times 32$
(3)	(2)	Batch Normalization	$64 \times 32 \times 32$
(4)	(3)	LeakyReLU	$64 \times 32 \times 32$
(5)	(4)	Conv$(4 \times 4, 64 \to 128, s2)$	$128 \times 16 \times 16$
(6)	(5)	Batch Normalization	$128 \times 16 \times 16$
(7)	(6)	LeakyReLU	$128 \times 16 \times 16$
(8)	(7)	Conv$(4 \times 4, 128 \to 256, s2)$	$256 \times 8 \times 8$
(9)	(8)	Conv$(1 \times 1, 256 \to 1)$	$1 \times 8 \times 8$

Table C.9: Architecture of our image discriminator D_{img}. The input to the image discriminator is either a real or fake image, and it classifies an overlapping 8×8 grid of patches in the input image as either real or fake. All but the final convolution have a stride of 2, and all convolutions use no padding. LeakyReLU uses a negative slope of 0.2.

C.1.10 Higher Image Resolutions

We performed preliminary experiments with a version of our model that produces 128×128 images rather than 64×64 images. For these models we compute the scene layout at 128×128 rather than at 64×64; we also add an extra Cascaded Refinement Module to our Cascaded Refinement Network; we add one additional convolutional layer to both D_{obj} and D_{img}, and for these models D_{obj} receives a 64×64 crop of objects rather than a 32×32 crop. During trainging we reduce the batch size from 32 to 24.

The images in Figure 7.6 from the main text were generated from a version of our model trained to produce 128×128 images from Visual Genome.

C.2 Image Loss Functions

In Figure C.1 we show additional qualitative results from our model trained on COCO, comparing the results from different ablated versions of our model.

Omitting the discriminators from the model (L1 only) tends to produce images that are overly smoothed. Without the object discriminator (No D_{obj}) objects tend to be less recognizable, and without the image discriminator (No D_{img}) the generated

images tend to appear less realistic overall, with low-level artifacts. Our model trained to use ground-truth layouts rather than predicting its own layouts (GT Layout) tends to produce higher-quality images, but requires both bounding-box and segmentation mask annotations at test-time.

The bottom row of Figure C.1 shows a typical failure, where all models struggle to synthesize a realistic image from a complex scene graph for an indoor scene.

C.3 User Study

As discussed in Section 4.5 of the main paper, we perform two user studies on Amazon Mechanical Turk to compare the perceptual quality of images generated from our method with those generated using StackGAN.

In the first user study, we show users an image generated from a COCO caption using StackGAN, and an image generated using our method from a scene graph built from the COCO object annotations corresponding to the caption. We ask users to select the image that better matches the caption. In each trial of this user study the order of our image and the image from StackGAN are randomized.

In the second user study, we again show users images generated using both methods, and we ask users to select the COCO objects that are visible in the image. In this experiment, if a single image contains multiple instances of the same object category then we only ask about its presence once. In each Mechanical Turk HIT users see an equal number of results from StackGAN and our method, and the order in which they are presented is randomized.

For both studies we use 1024 images from each method generated from COCO val annotations. All images are seen by three workers, and we report all results using majority opinions.

StackGAN produces 256×256 images, but our method produces 64×64 images. To prevent differing resolutions from affecting worker opinion, we downsample StackGAN results to 64×64 with bicubic interpolation before presenting them to users.

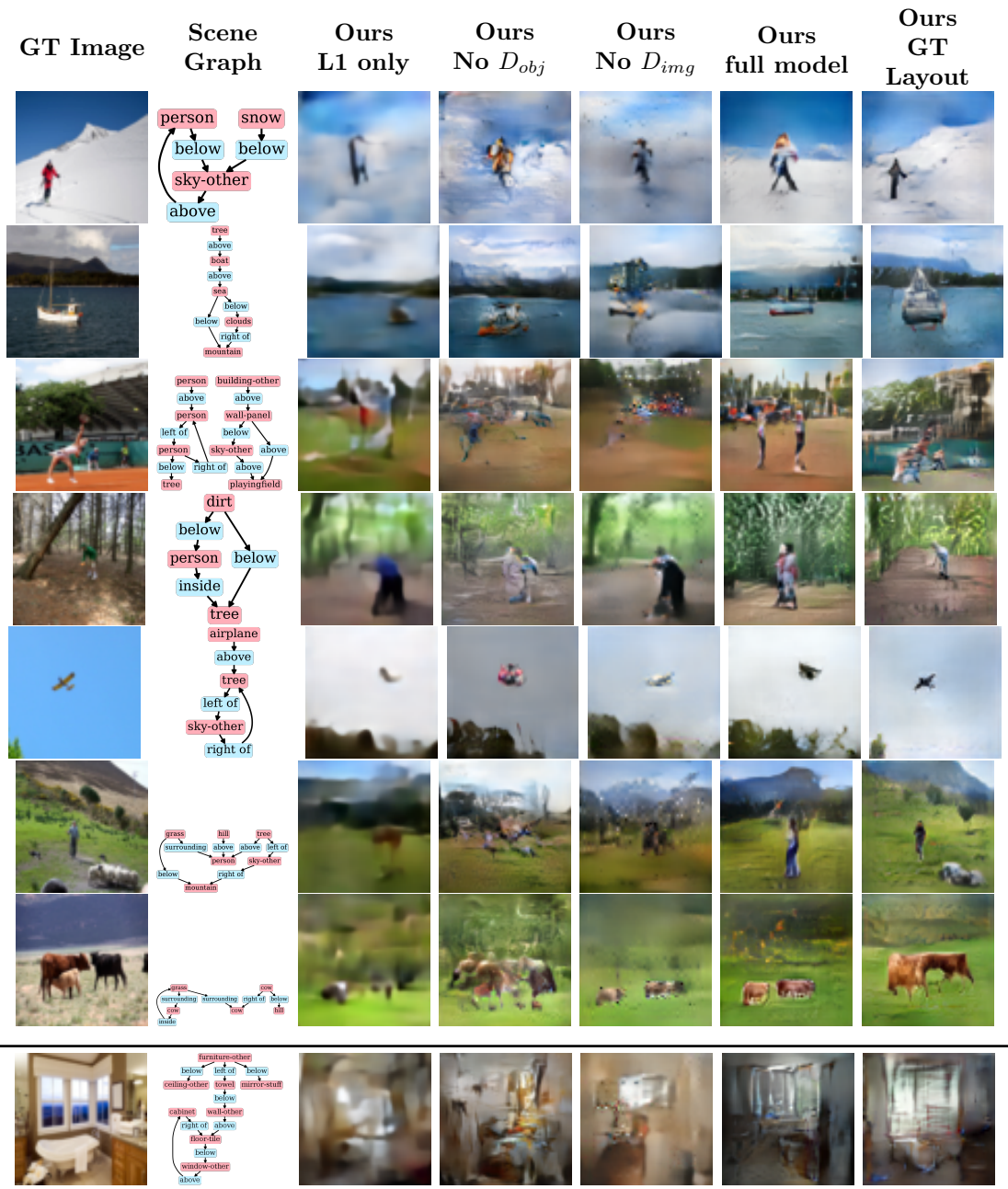


Figure C.1: Images generated from our model and ablations on COCO. We show the original image, synthetic scene graph generated from its COCO annotations, and results from several versions of our model. Our model with no discriminators (L1 only) tends to be overly smooth; omitting the object discriminator (No D_{obj}) causes objects to be less recognizable; omitting the image discriminator (No D_{img}) leads to low-level image artifacts. Using ground-truth rather than predicted layouts (GT Layout) tends to give higher quality images. The bottom row shows a typical failure case, where all versions of our model struggle with complex scene graphs for indoor scenes. Graphs best viewed with magnification.

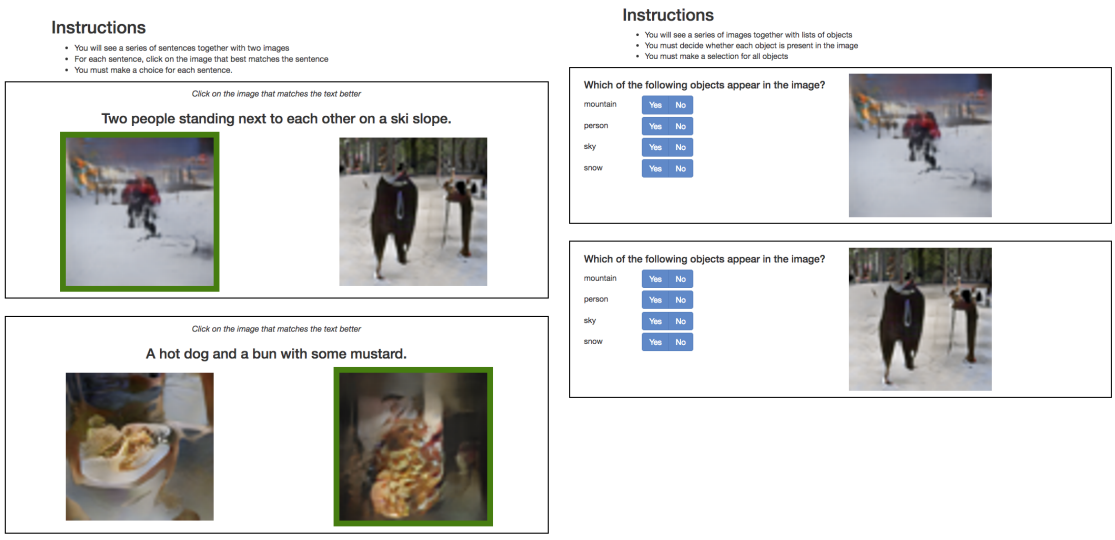


Figure C.2: Screenshots of the user interfaces for our user studies on Amazon Mechanical Turk. **Left:** User interface for the user study from Figure 7.7 of the main text. We show users an image generated by StackGAN from a COCO caption, and an image generated with our method from a scene graph built from the COCO object annotations corresponding to the caption. We ask users to select the image that best matches the caption. **Right:** User interface for the user study from Figure 7.8 of the main text. We show again show users images generated using StackGAN and our method, and we ask users which COCO objects are present in each image.

Bibliography

[1] A. Agrawal, D. Batra, and D. Parikh. "Analyzing the Behavior of Visual Question Answering Models". *EMNLP*. 2016.

[2] B. Alexe, T. Deselaers, and V. Ferrari. "Measuring the objectness of image windows". *TPAMI* (2012).

[3] P. Anderson, B. Fernando, M. Johnson, and S. Gould. "Spice: Semantic propositional image caption evaluation". *ECCV*. 2016.

[4] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. "Neural module networks". *CVPR*. 2016.

[5] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. "Learning to compose neural networks for question answering". *NAACL*. 2016.

[6] B. Andres, T. Beier, and J. H. Kappes. "OpenGM: A C++ library for discrete graphical models". *arXiv preprint arXiv:1206.0111* (2012).

[7] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Zitnick, and D. Parikh. "VQA: Visual question answering". *ICCV*. 2015.

[8] M. Balog, A. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. "DeepCoder: Learning to write programs". *ICLR*. 2017.

[9] K. Barnard, P. Duygulu, D. Forsyth, N. De Freitas, D. M. Blei, and M. I. Jordan. "Matching words and pictures". *JMLR* (2003).

[10] Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". *TPAMI* (2014).

[11] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. "A neural probabilistic language model". *JMLR* (2003).

[12] T. L. Berg, A. C. Berg, and J. Shih. "Automatic attribute discovery and characterization from noisy web data". *ECCV*. 2010.

[13] A. Bergamo, L. Torresani, and A. Fitzgibbon. "PiCoDes: Learning a Compact Code for Novel-Category Recognition". *NIPS*. 2011.

[14] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam, 2016.

[15] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. "Spectral networks and locally connected networks on graphs". *ICLR*. 2014.

[16] H. Caesar, J. Uijlings, and V. Ferrari. "COCO-Stuff: Thing and Stuff Classes in Context". *arXiv preprint arXiv:1612.03716* (2016).

[17] J. Cai, R. Shin, and D. Song. "Making Neural Programming Architectures Generalize via Recursion". *ICLR*. 2017.

[18] Y. Cao, C. Wang, Z. Li, L. Zhang, and L. Zhang. "Spatial-Bag-of-Features". *CVPR*. 2010.

[19] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals. "Listen, attend and spell". *ICASSP*. 2016.

[20] A. Chang, W. Monroe, M. Savva, C. Potts, and C. D. Manning. "Text to 3d scene generation with rich lexical grounding". *ACL*. 2015.

[21] A. X. Chang and C. D. Manning. "TokensRegex: Defining cascaded regular expressions over tokens". *Tech. Rep. CSTR 2014-02* (2014).

[22] A. X. Chang, M. Savva, and C. D. Manning. "Learning Spatial Knowledge for Text to 3D Scene Generation". *EMNLP*. 2014.

[23] A. X. Chang, M. Savva, and C. D. Manning. "Learning Spatial Knowledge for Text to 3D Scene Generation." *EMNLP*. 2014.

[24] J. Chen, P. Kuznetsova, D. Warren, and Y. Choi. "Deja Image-Captions: A Corpus of Expressive Image Descriptions in Repetition". *NAACL*. 2015.

[25] Q. Chen and V. Koltun. "Photographic image synthesis with cascaded refinement networks". *ICCV*. 2017.

[26] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick. "Microsoft COCO captions: Data collection and evaluation server". *arXiv preprint arXiv:1504.00325* (2015).

[27] X. Chen and C Lawrence Zitnick. "Mind's eye: A recurrent visual representation for image caption generation". *CVPR*. 2015.

[28] X. Chen, A. Shrivastava, and A. Gupta. "NEIL: Extracting visual knowledge from web data". *ICCV*. 2013.

[29] K. Cho, A. Courville, and Y. Bengio. "Describing multimedia content using attention-based encoder-decoder networks". *IEEE Transactions on Multimedia* (2015).

[30] M. J. Choi, J. J. Lim, A. Torralba, and A. S. Willsky. "Exploiting Hierarchical Context on a Large Database of Object Categories". *CVPR*. 2010.

[31] W. Choi, Y.-W. Chao, C. Pantofaru, and S. Savarese. "Understanding Indoor Scenes Using 3D Geometric Phrases". *CVPR*. 2013.

[32] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. "Total Recall: Automatic Query Expansion with a Generative Feature Model for Object Retrieval". *ICCV*. 2007.

[33] R. Collobert, K. Kavukcuoglu, and C. Farabet. "Torch7: A matlab-like environment for machine learning". *NIPS BigLearn Workshop*. 2011.

[34] A. Das, S. Kottur, K. Gupta, A. Singh, D. Yadav, J. Moura, D. Parikh, and D. Batra. "Visual Dialog". *CVPR*. 2017.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". *CVPR*. 2009.

[36] M. Denkowski and A. Lavie. "Meteor universal: Language specific translation evaluation for any target language". *EACL Workshop on Statistical Machine Translation*. 2014.

[37] C. Desai, D. Ramanan, and C. C. Fowlkes. "Discriminative models for multi-class object layout". *IJCV* (2011).

[38] J. Devlin, S. Gupta, R. Girshick, M. Mitchell, and C. L. Zitnick. "Exploring nearest neighbor approaches for image captioning". *arXiv preprint arXiv:1505.04467* (2015).

[39] S. Divvala, A. Farhadi, and C. Guestrin. "Learning Everything about Anything: Webly-Supervised Visual Concept Learning". *CVPR*. 2014.

[40] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. "Long-term recurrent convolutional networks for visual recognition and description". *CVPR*. 2015.

[41] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. "Convolutional networks on graphs for learning molecular fingerprints". *NIPS*. 2015.

[42] S. El Hihi and Y. Bengio. "Hierarchical recurrent neural networks for long-term dependencies". *NIPS*. 1995.

[43] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. "Scalable object detection using deep neural networks". *CVPR*. 2014.

[44] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. "The PASCAL visual object classes (VOC) challenge". *IJCV* (2010).

[45] H. Fang, S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, et al. "From captions to visual concepts and back". *CVPR*. 2015.

[46] A. Farhadi, I. Endres, and D. Hoiem. "Attribute-centric recognition for cross-category generalization". *CVPR*. 2010.

[47] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. "Describing objects by their attributes". *CVPR*. 2009.

[48] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. "Every picture tells a story: Generating sentences from images". *ECCV*. 2010.

[49] L. Fei-Fei, R. Fergus, and P. Perona. "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories". *Computer Vision and Image Understanding*. 2004.

[50] V. Ferrari and A. Zisserman. "Learning visual attributes". *NIPS*. 2007.

[51] M. Fisher, M. Savva, and P. Hanrahan. "Characterizing structural relationships in scenes using graph kernels". *SIGGRAPH*. 2011.

[52] D. F. Fouhey and C. L. Zitnick. "Predicting Object Dynamics in Scenes". *CVPR*. 2014.

[53] P. Frasconi, M. Gori, and A. Sperduti. "A general framework for adaptive processing of data structures". *IEEE Transactions on Neural Networks* (1998).

[54] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach. "Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding". *EMNLP*. 2016.

[55] H. Gao, J. Mao, J. Zhou, Z. Huang, L. Wang, and W. Xu. "Are you talking to a machine? Dataset and methods for multilingual image question answering". *NIPS*. 2015.

[56] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. "Compact bilinear pooling". *CVPR*. 2016.

[57] L. A. Gatys, A. S. Ecker, and M. Bethge. "Image style transfer using convolutional neural networks". *CVPR*. 2016.

[58] J. Gauthier. "Conditional generative adversarial nets for convolutional face generation". *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester* (2014).

[59] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". *CVPR*. 2012.

[60] D. Geman, S. Geman, N. Hallonquist, and L. Younes. "Visual Turing test for computer vision systems". *PNAS* (2015).

[61] R. Girshick. "Fast R-CNN". *ICCV*. 2015.

[62] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". *CVPR*. 2014.

[63] C. Goller and A. Kuchler. "Learning task-dependent distributed representations by backpropagation through structure". *IEEE International Conference on Neural Networks*. 1996.

[64] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial nets". *NIPS*. 2014.

[65] M. Gori, G. Monfardini, and F. Scarselli. "A new model for learning in graph domains". *IEEE International Joint Conference on Neural Networks*. 2005.

[66] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh. "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering". *CVPR*. 2017.

[67] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwinska, S. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. Badia, K. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. "Hybrid computing using a neural network with dynamic external memory". *Nature* (2016).

[68] A. Graves. "Generating sequences with recurrent neural networks". *arXiv preprint arXiv:1308.0850* (2013).

[69] A. Graves, G. Wayne, and I. Danihelka. "Neural turing machines". *arXiv preprint arXiv:1410.5401* (2014).

[70] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. "DRAW: A recurrent neural network for image generation". *ICML*. 2015.

[71] A. Grover and J. Leskovec. "node2vec: Scalable feature learning for networks". *SIGKDD*. 2016.

[72] A. Gupta and L. S. Davis. "Beyond nouns: Exploiting prepositions and comparative adjectives for learning visual classifiers". *ECCV*. 2008.

[73] A. Gupta, J. Johnson, A. Alahi, and L. Fei-Fei. "Characterizing and improving stability in neural style transfer". *ICCV*. 2017.

[74] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. "Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks". *CVPR*. 2018.

[75] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". *CVPR*. 2016.

[76] K. He, X. Zhang, S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". *ICCV*. 2015.

[77] K. He, X. Zhang, S. Ren, and J. Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition" (2015).

[78] M. Henaff, J. Bruna, and Y. LeCun. "Deep convolutional networks on graph-structured data". *arXiv preprint arXiv:1506.05163* (2015).

[79] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural computation* (1997).

[80] M. Hodosh, P. Young, and J. Hockenmaier. "Framing image description as a ranking task: Data, models and evaluation metrics". *JAIR* (2013).

[81] R. Hu, M. Rohrbach, J. Andreas, T. Darrell, and K. Saenko. "Modeling Relationships in Referential Expressions with Compositional Modular Networks". *CVPR*. 2017.

[82] D. A. Hudson and C. D. Manning. "Compositional attention networks for machine reasoning". *ICLR*. 2018.

[83] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". *ICML*. 2015.

[84] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-image translation with conditional adversarial networks". *CVPR*. 2017.

[85] A. Jabri, A. Joulin, and L. van der Maaten. "Revisiting visual question answering baselines". *ECCV*. 2016.

[86] M. Jaderberg, K. Simonyan, A. Zisserman, et al. "Spatial transformer networks". *NIPS*. 2015.

[87] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena. "Structural-RNN: Deep learning on spatio-temporal graphs". *CVPR*. 2016.

[88] H. Jegou, M. Douze, and C. Schmid. "Product Quantization for Nearest Neighbor Search". *TPAMI* (2011).

[89] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. "Aggregating Local Image Descriptors into Compact Codes". *TPAMI* (2012).

[90] Y. Jia, M. Salzmann, and T. Darrell. "Learning cross-modality similarity for multinomial data". *ICCV*. 2011.

[91] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional architecture for fast feature embedding". *ACM international conference on Multimedia*. 2014.

[92] C. Jiang, Y. Zhu, S. Qi, S. Huang, J. Lin, X. Guo, L.-F. Yu, D. Terzopoulos, and S.-C. Zhu. "Configurable, Photorealistic Image Rendering and Ground Truth Synthesis by Sampling Stochastic Grammars Representing Indoor Scenes". *IJCV* (2018).

[93] J. Johnson, A. Alahi, and L. Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". *ECCV*. 2016.

[94] J. Johnson, L. Ballan, and L. Fei-Fei. "Love Thy Neighbors: Image Annotation by Exploiting Image Metadata". *ICCV*. 2015.

[95] J. Johnson, A. Gupta, and L. Fei-Fei. "Image generation from scene graphs". *CVPR*. 2018.

[96] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. "CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning". *CVPR*. 2017.

[97] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. "Inferring and executing programs for visual reasoning". *ICCV*. 2017.

[98] J. Johnson, A. Karpathy, and L. Fei-Fei. "Densecap: Fully convolutional localization networks for dense captioning". *CVPR*. 2016.

[99] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei. "Image retrieval using scene graphs". *CVPR*. 2015.

[100] A. Joulin and T. Mikolov. "Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets". *NIPS*. 2015.

[101] D. Jurafsky and J. H. Martin. *Speech & language processing*. Pearson Education India, 2000.

[102] K. Kafle and C. Kanan. "Visual Question Answering: Datasets, Algorithms, and Future Challenges". *arXiv preprint arXiv:1610.01465*. 2016.

[103] Ł. Kaiser and I. Sutskever. "Neural GPUs learn algorithms". *ICLR*. 2016.

[104] A. Karpathy and L. Fei-Fei. "Deep visual-semantic alignments for generating image descriptions". *CVPR*. 2015.

[105] A. Karpathy, J. Johnson, and L. Fei-Fei. "Visualizing and understanding recurrent networks". *ICLR Workshop*. 2016.

[106] A. Karpathy, A. Joulin, and L. Fei-Fei. "Deep Fragment Embeddings for Bidirectional Image Sentence Mapping". *NIPS*. 2014.

[107] S. Kazemzadeh, V. Ordonez, M. Matten, and T. Berg. "ReferItGame: Referring to Objects in Photographs of Natural Scenes". *EMNLP*. 2014.

[108] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". *ICLR*. 2015.

[109] D. P. Kingma and M. Welling. "Auto-encoding variational bayes". *ICLR*. 2014.

[110] T. N. Kipf and M. Welling. "Semi-supervised classification with graph convolutional networks". *ICLR*. 2017.

[111] R. Kiros, R. Salakhutdinov, and R. S. Zemel. "Unifying visual-semantic embeddings with multimodal neural language models". *NIPS Deep Learning Workshop*. 2014.

[112] D. Klein and C. D. Manning. "Accurate unlexicalized parsing". *ACL*. 2003.

[113] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. "A clockwork rnn". *ICML*. 2014.

[114] P. Krähenbühl and V. Koltun. "Geodesic Object Proposals". *ECCV*. 2014.

[115] J. Krause, J. Johnson, R. Krishna, and L. Fei-Fei. "A Hierarchical Approach for Generating Descriptive Image Paragraphs". *CVPR*. 2017.

[116] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations". *IJCV* (2017).

[117] J. Krishnamurthy and T. Kollar. "Jointly Learning to Parse and Perceive: Connecting Natural Language to the Physical World." *TACL* (2013).

[118] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". *NIPS*. 2012.

[119] G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg. "Baby talk: Understanding and generating image descriptions". *CVPR*. 2011.

[120] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. "Picture: A probabilistic programming language for scene perception". *CVPR*. 2015.

[121] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. "Deep convolutional inverse graphics network". *NIPS*. 2015.

[122] K. Kurach, M. Andrychowicz, and I. Sutskever. "Neural random-access machines". *ICLR*. 2016.

[123] P. Kuznetsova, V. Ordonez, A. C. Berg, T. L. Berg, and Y. Choi. "Generalizing Image Captions for Image-Text Parallel Corpus." *ACL*. 2013.

[124] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. 2001.

[125] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. "Building machines that learn and think like people". *Behavioral and Brain Sciences* (2016).

[126] C Lampert, H. Nickisch, and S. Harmeling. "Attribute-based classification for zero-shot learning of object categories". *TPAMI* (2013).

[127] C. H. Lampert. "Detecting objects in large image collections and videos by efficient subimage retrieval". *ICCV*. 2009.

[128] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* (1998).

[129] H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky. "Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task". *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*. 2011.

[130] H. J. Levesque, E. Davis, and L. Morgenstern. "The Winograd schema challenge." *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*. 2011.

[131] J. Li, M.-T. Luong, and D. Jurafsky. "A hierarchical neural autoencoder for paragraphs and documents". *ACL*. 2015.

[132] L.-J. Li, R. Socher, and L. Fei-Fei. "Towards Total Scene Understanding: Classification, Annotation and Segmentation in an Automatic Framework". *CVPR*. 2009.

[133] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. "Gated graph sequence neural networks". *ICLR*. 2015.

[134] C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao. "Neural symbolic machines: Learning semantic parsers on freebase with weak supervision". *ACL* (2017).

[135] P. Liang, M. I. Jordan, and D. Klein. "Learning dependency-based compositional semantics". *ACL*. 2011.

[136] D. Lin, S. Fidler, C. Kong, and R. Urtasun. "Visual Semantic Search: Retrieving Videos via Complex Textual Queries". *CVPR*. 2014.

[137] R. Lin, S. Liu, M. Yang, M. Li, M. Zhou, and S. Li. "Hierarchical recurrent neural network for document modeling". *EMNLP*. 2015.

[138] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. "Microsoft COCO: Common objects in context". *ECCV*. 2014.

[139] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, and K. Murphy. "Improved Image Captioning via Policy Gradient optimization of SPIDEr". *ICCV*. 2017.

[140] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". *CVPR*. 2015.

[141] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". *IJCV* (2004).

[142] C. Lu, R. Krishna, M. Bernstein, and L. Fei-Fei. "Visual relationship detection with language priors". *ECCV*. 2016.

[143] J. Lu, J. Yang, D. Batra, and D. Parikh. "Hierarchical Question-Image Co-Attention for Visual Question Answering". *NIPS*. 2016.

[144] L. Ma, Z. Lu, and H. Li. "Learning to Answer Questions From Image Using Convolutional Neural Network". *AAAI*. 2016.

[145] A. L. Maas, A. Y. Hannun, and A. Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". *ICML*. 2013.

[146] M. Malinowski and M. Fritz. "A Multi-World Approach to Question Answering about Real-World Scenes based on Uncertain Input". *NIPS*. 2014.

[147] M. Malinowski and M. Fritz. "Towards a Visual Turing Challenge". *NIPS Workshop on Learning Semantics*. 2014.

[148] M. Malinowski, M. Rohrbach, and M. Fritz. "Ask your neurons: A neural-based approach to answering questions about images". *ICCV*. 2015.

[149] A. Mallya and S. Lazebnik. "Learning Models for Actions and Person-Object Interactions with Transfer to Question Answering". *ECCV*. 2016.

[150] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. "The Stanford CoreNLP natural language processing toolkit". *ACL (System Demonstrations)*. 2014.

[151] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille. "Deep captioning with multimodal recurrent neural networks (m-rnn)". *ICLR*. 2015.

[152] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. "Building a large annotated corpus of English: The Penn Treebank". *Computational linguistics* (1993).

[153] T. Mensink, E. Gavves, and C. G. Snoek. "COSTA: Co-Occurrence Statistics for Zero-Shot Classification". *CVPR*. 2014.

[154] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space". *arXiv preprint arXiv:1301.3781* (2013).

[155] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. "Recurrent neural network based language model." *INTERSPEECH*. 2010.

[156] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. "Distributed representations of words and phrases and their compositionality". *NIPS*. 2013.

[157] M. Mirza and S. Osindero. "Conditional generative adversarial nets". *arXiv preprint arXiv:1411.1784* (2014).

[158] M. Mitchell, X. Han, J. Dodge, A. Mensch, A. Goyal, A. Berg, K. Yamaguchi, T. Berg, K. Stratos, and H. Daumé III. "Midge: Generating image descriptions from computer vision detections". *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. 2012.

[159] R. Montague. "The proper treatment of quantification in ordinary English". *Approaches to natural language*. Springer, 1973, pp. 221–242.

[160] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. "The Role of Context for Object Detection and Semantic Segmentation in the Wild". *CVPR*. 2014.

[161] A. Neelakantan, Q. V. Le, and I. Sutskever. "Neural programmer: Inducing latent programs with gradient descent". *ICLR*. 2016.

[162] A. Newell and J. Deng. "Pixels to Graphs by Associative Embedding". *NIPS*. 2017.

[163] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. "DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference." *VLDS*. 2012.

[164] NVIDIA. *Programming guide*. 2010.

[165] A. Odena, C. Olah, and J. Shlens. "Conditional Image Synthesis with Auxiliary Classifier GANs". *ICML*. 2017.

[166] A. Oliva and A. Torralba. "Modeling the shape of the scene: A holistic representation of the spatial envelope". *IJCV* (2001).

[167] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu, O. Vinyals, and A. Graves. "Conditional Image Generation with PixelCNN Decoders". *NIPS*. 2016.

[168] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. "Pixel recurrent neural networks". *ICML*. 2016.

[169] M. Oquab. *stnbhwd*. `https://github.com/qassemoquab/stnbhwd`. 2015.

[170] V. Ordonez, X. Han, P. Kuznetsova, G. Kulkarni, M. Mitchell, K. Yamaguchi, K. Stratos, A. Goyal, J. Dodge, A. Mensch, et al. "Large Scale Retrieval and Generation of Image Descriptions". *IJCV* (2015).

[171] R. Osfield, D. Burns, et al. "Open scene graph". *Library–OSG. http://www.openscenegraph.or* (2004).

[172] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. "BLEU: a method for automatic evaluation of machine translation". *ACL*. 2002.

[173] D. Parikh and K. Grauman. "Relative attributes". *ICCV*. 2011.

[174] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. "Film: Visual reasoning with a general conditioning layer". *AAAI*. 2018.

[175] B. Perozzi, R. Al-Rfou, and S. Skiena. "Deepwalk: Online learning of social representations". *SIGKDD*. 2014.

[176] F. Perronnin, Y. Liu, J. Sanchez, and H. Poirier. "Large-scale image retrieval with compressed Fisher vectors". *CVPR*. 2010.

[177] O. Pfungst. *Clever Hans (The horse of Mr. von Osten): A contribution to experimental animal and human psychology*. New York: Henry Holt, 1911.

[178] J. Platt. "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods". *Advances in large margin classifiers*. 1999.

[179] B. A. Plummer, L. Wang, C. M. Cervantes, J. C. Caicedo, J. Hockenmaier, and S. Lazebnik. "Flickr30k Entities: Collecting Region-to-Phrase Correspondences for Richer Image-to-Sentence Models". *ICCV*. 2015.

[180] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation". *CVPR* (2017).

[181] D. Qin, C. Wengert, and L. Van Gool. "Query Adaptive Similarity for Large Scale Object Retrieval". *CVPR*. 2013.

[182] A. Radford, L. Metz, and S. Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". *ICLR*. 2016.

[183] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning. "A multi-pass sieve for coreference resolution". *EMNLP*. 2010.

[184] A. Ray, G. Christie, M. Bansal, D. Batra, and D. Parikh. "Question Relevance in VQA: Identifying Non-Visual And False-Premise Questions". *EMNLP*. 2016.

[185] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". *CVPR*. 2016.

[186] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. "Learning What and Where to Draw". *NIPS*. 2016.

[187] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. "Generative Adversarial Text-to-Image Synthesis". *ICML*. 2016.

[188] S. Reed and N. De Freitas. "Neural programmer-interpreters". *ICLR*. 2016.

[189] S. E. Reed, A. van den Oord, N. Kalchbrenner, S. Gómez, Z. Wang, D. Belov, and N. de Freitas. "Parallel Multiscale Autoregressive Density Estimation". *ICML*. 2017.

[190] M. Ren, R. Kiros, and R. Zemel. "Exploring models and data for image question answering". *NIPS*. 2015.

[191] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards real-time object detection with region proposal networks". *NIPS*. 2015.

[192] A. Rohrbach, M. Rohrbach, W. Qiu, A. Friedrich, M. Pinkal, and B. Schiele. "Coherent multi-sentence video description with variable level of detail". *German conference on pattern recognition*. 2014.

[193] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. "Imagenet large scale visual recognition challenge". *IJCV* (2015).

[194] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. "LabelMe: a database and web-based tool for image annotation". *IJCV* (2008).

[195] M. A. Sadeghi and A. Farhadi. "Recognition using visual phrases". *CVPR*. 2011.

[196] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved techniques for training gans". *NIPS*. 2016.

[197] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. "A simple neural network module for relational reasoning". *NIPS*. 2017.

[198] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The graph neural network model". *IEEE Transactions on Neural Networks* (2009).

[199] S. Schuster, R. Krishna, A. Chang, L. Fei-Fei, and C. D. Manning. "Generating semantically precise scene graphs from textual descriptions for improved image retrieval". *EMNLP Vision and Language Workshop*. 2015.

[200] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". *ICLR*. 2014.

[201] K. Shih, S. Singh, and D. Hoiem. "Where to look: Focus regions for visual question answering". *CVPR*. 2016.

[202] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". *ICLR*. 2015.

[203] R. Socher and L. Fei-Fei. "Connecting modalities: Semi-supervised segmentation and annotation of images using unaligned text corpora". *CVPR*. 2010.

[204] R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng. "Grounded compositional semantics for finding and describing images with sentences". *TACL* (2014).

[205] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. "Parsing natural scenes and natural language with recursive neural networks". *ICML*. 2011.

[206] A. Sperduti and A. Starita. "Supervised neural networks for the classification of structures". *IEEE Transactions on Neural Networks* (1997).

[207] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." *JMLR* (2014).

[208] H. Stewenius and S. H.G. J. Pilet. "Size Matters: Exhaustive Geometric Verification for Image Retrieval". *ECCV*. 2012.

[209] B. Sturm. "A Simple Method to Determine if a Music Information Retrieval System is a Horse". *IEEE Transactions on Multimedia* (2014).

[210] B. Sturm. *Horse taxonomy and taxidermy*. `http://c4dm.eecs.qmul.ac.uk/horse2016/`. 2016.

[211] H. Su, J. Deng, and L. Fei-Fei. "Crowdsourcing annotations for visual object detection". *AAAI Workshops*. 2012.

[212] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. "End-To-End Memory Networks". *NIPS*. 2015.

[213] I. Sutskever, J. Martens, and G. E. Hinton. "Generating text with recurrent neural networks". *ICML*. 2011.

[214] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to sequence learning with neural networks". *NIPS*. 2014.

[215] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". *CVPR*. 2015.

[216] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. "Scalable, high-quality object detection". *arXiv preprint arXiv:1412.1441* (2014).

[217] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. "Line: Large-scale information network embedding". *WWW*. 2015.

[218] M. Tapaswi, Y. Zhu, R. Stiefelhagen, A. Torralba, R. Urtasun, and S. Fidler. "MovieQA: Understanding Stories in Movies through Question-Answering". *CVPR*. 2016.

[219] L. Theis, A. v. d. Oord, and M. Bethge. "A note on the evaluation of generative models". *ICLR*. 2016.

[220] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. "YFCC100M: the new data in multimedia research". *CACM* (2016).

[221] L. Torresani, M. Szummer, and A. Fitzgibbon. "Efficient Object Category Recognition using Classemes". *ECCV*. 2010.

[222] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. "Selective search for object recognition". *IJCV* (2013).

[223] R. Vedantam, C Lawrence Zitnick, and D. Parikh. "CIDEr: Consensus-based image description evaluation". *CVPR*. 2015.

[224] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. "Show and tell: A neural image caption generator". *CVPR*. 2015.

[225] M. Wang, Y. Tang, J. Wang, and J. Deng. "Premise Selection for Theorem Proving by Deep Graph Embedding". *NIPS*. 2017.

[226] P. J. Werbos. "Generalization of backpropagation with application to a recurrent gas market model". *Neural Networks* (1988).

[227] J. Weston, A. Bordes, S. Chopra, A. Rush, B van Merriënboer, A. Joulin, and T. Mikolov. "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks". *ICLR*. 2016.

[228] J. Weston, S. Chopra, and A. Bordes. "Memory Networks". *ICLR*. 2015.

[229] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". *Machine Learning* (1992).

[230] T. Winograd. *Understanding Natural Language*. Academic Press, 1972.

[231] J. Wu, J. B. Tenenbaum, and P. Kohli. "Neural Scene De-rendering". *CVPR*. 2017.

[232] Q. Wu, D. Teney, P. Wang, C. Shen, A. Dick, and A. van den Hengel. "Visual Question Answering: A Survey of Methods and Datasets". *Computer Vision and Image Understanding*. 2016.

[233] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. "Sun database: Large-scale scene recognition from abbey to zoo". *CVPR*. 2010.

[234] J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu. "A theory of generative convnet". *ICML*. 2016.

[235] C. Xiong, S. Merity, and R. Socher. "Dynamic memory networks for visual and textual question answering". *ICML*. 2016.

[236] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei. "Scene graph generation by iterative message passing". *CVPR*. 2017.

[237] H. Xu and K. Saenko. "Ask, Attend, and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering". *ECCV*. 2016.

[238] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. "Show, attend and tell: Neural image caption generation with visual attention". *ICML*. 2015.

[239] J. Yang, B. Price, S. Cohen, and M.-H. Yang. "Context Driven Scene Parsing with Attention to Rare Classes". *CVPR*. 2014.

[240] M. Y. Yang, W. Liao, H. Ackermann, and B. Rosenhahn. "On Support Relations and Semantic Scene Graphs". *ISPRS Journal of Photogrammetry and Remote Sensing* (2017).

[241] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. "Stacked attention networks for image question answering". *CVPR*. 2016.

[242] B. Yao, A. Khosla, and L. Fei-Fei. "Classifying actions and measuring action similarity by modeling the mutual context of objects and human poses". *ICML*. 2011.

[243] L. Yao, A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville. "Describing videos by exploiting temporal structure". *ICCV*. 2015.

[244] Z. Yao, X. Yang, and S. Zhu. "Introduction to a large scale general purpose groundtruth dataset: methodology, annotation tool, and benchmarks". *EMMCVPR*. 2007.

[245] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo. "Image captioning with semantic attention". *CVPR*. 2016.

[246] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier. "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions". *TACL* (2014).

[247] H. Yu, J. Wang, Z. Huang, Y. Yang, and W. Xu. "Video paragraph captioning using hierarchical recurrent neural networks". *CVPR*. 2016.

[248] L. Yu, E. Park, A. Berg, and T. Berg. "Visual Madlibs: Fill in the blank Image Generation and Question Answering". *ICCV*. 2015.

[249] W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus. "Learning simple algorithms from examples". *ICML*. 2016.

[250] W. Zaremba and I. Sutskever. "Learning to execute". *arXiv preprint arXiv:1410.4615* (2014).

[251] W. Zaremba and I. Sutskever. "Reinforcement learning neural Turing machines". *arXiv preprint arXiv:1505.00521* (2015).

[252] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Neural Networks". *ECCV*. 2014.

[253] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks". *ICCV*. 2017.

[254] P. Zhang, Y. Goyal, D. Summers-Stay, D. Batra, and D. Parikh. "Yin and Yang: Balancing and answering binary visual questions". *CVPR*. 2016.

[255] X. Zhang, Z. Li, L. Zhang, W.-Y. Ma, and H.-Y. Shum. "Efficient indexing for large scale visual search". *ICCV*. 2009.

[256] Y. Zhao and S.-C. Zhu. "Scene Parsing by Integrating Function, Geometry and Appearance Models". *CVPR*. 2013.

[257] B. Zhou, Y. Tian, S. Sukhbataar, A. Szlam, and R. Fergus. "Simple Baseline for Visual Question Answering". *arXiv:1512.02167*. 2015.

[258] J. Zhu, R. Kaplan, J. Johnson, and L. Fei-Fei. "HiDDeN: Hiding Data with Deep Networks". *ECCV*. 2018.

[259] Y. Zhu, O. Groth, M. Bernstein, and L. Fei-Fei. "Visual7W: Grounded Question Answering in Images". *CVPR*. 2016.

[260] C. L. Zitnick and P. Dollár. "Edge Boxes: Locating Object Proposals from Edges". *ECCV*. 2014.

[261] C. L. Zitnick and D. Parikh. "Bringing semantics into focus using visual abstraction". *CVPR*. 2013.

[262] C. L. Zitnick, D. Parikh, and L. Vanderwende. "Learning the visual interpretation of sentences". *ICCV*. 2013.