Interpretable Counting for Visual Question Answering

Alexander Trott
atrott@salesforce.com

Caiming Xiong
cxiong@salesforce.com

Richard Socher
rsocher@salesforce.com

Abstract

Questions that require counting a variety of objects in images remain a major challenge in visual question answering (VQA). The most common approaches to VQA involve either classifying answers based on fixed length representations of both the image and question or summing fractional counts estimated from each section of the image. In contrast, we treat counting as a sequential decision process and force our model to make discrete choices of what to count. Specifically, the model sequentially selects from detected objects and learns interactions between objects that influence subsequent selections. A distinction of our approach is its intuitive and interpretable output, as discrete counts are automatically grounded in the image. Furthermore, our method outperforms the state of the art architecture for VQA on multiple metrics that evaluate counting.

1 Introduction

Visual question answering (VQA) is an important benchmark to test for context-specific reasoning over complex images. While the field has seen substantial progress, counting-based questions have seen the least improvement [Chattopadhyay et al., 2017]. Intuitively, counting should involve finding the number of distinct scene elements or objects that meet some criteria, see Fig. 1 for an example. Our intuition about counting seems at odds with the effects of attention (a near-ubiquitous mechanism in VQA) where a weighted average obscures any notion of distinct elements. As such, we are motivated to re-think the typical approach to counting in VQA and propose a method that embraces the discrete nature of the task.

We introduce the Interpretable Reinforcement Learning Counter (IRLC), which treats counting as a sequential decision process. We treat learning to count as learning to enumerate the relevant objects in the scene. As a result, IRLC not only returns a count but also the objects supporting its answer. We supervise only the final count and train the decision process using reinforcement learning (RL).

IRLC achieves a higher accuracy and lower error than the current state of the art model for VQA when both are trained on counting questions. Furthermore, we compare the grounded counts of our model to the attentional focus of the state of the art baseline to demonstrate the interpretability gained through our approach.

2 Model

We experiment with three models. These models use identical strategies to encode the image and the question but differ in terms of how those encodings are used to produce a count.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Figure 1: IRLC takes as input a counting question and image. Detected objects are added to the returned count through a sequential decision process. The above example illustrates actual model behavior after training.

2.1 Object Detection

Our approach is inspired by the strategy of Anderson et al. [2017] and Teney et al. [2017]. Their model, which represents current state of the art in VQA, infers objects as the input to the question-answering system. This inference is performed using the Faster R-CNN architecture [Ren et al., 2015]. The Faster R-CNN proposes a set of regions corresponding to objects in the image. It encodes the image as a set of bounding boxes $\{b_1, ..., b_N\}$, $b_i \in \mathbb{R}^4$ and complementary set of object encodings $\{v_1, ..., v_N\}$, $v_i \in \mathbb{R}^{2048}$, corresponding to the locations and feature representations of each of the N detected objects, respectively.

Rather than train our own vision module from scratch, we make use of the publicly available object proposals learned in Anderson et al. [2017]. These provide rich, object-centric representations for each image in our dataset. These representations are fixed when learning to count and are shared across each of the QA models we experiment with.

2.2 Counting

When answering a question, each architecture begins the QA process by encoding the question and comparing it against each detected object via a scoring function. We define q as the final hidden state of an LSTM [Hochreiter and Schmidhuber, 1997] after processing the question and compute a score vector for each object:

$$h^t = \text{LSTM}\left(x^t, h^{t-1}\right) \quad q = h^T \tag{1}$$

$$s_i = f^S\left([q, v_i]\right) \tag{2}$$

Here, x_t denotes the word embedding of the question token at position t and $s_i \in \mathbb{R}^n$ denotes the score vector of object i. Following Anderson et al. [2017], we implement the scoring function $f^S : \mathbb{R}^m \to \mathbb{R}^n$ as a layer of Gated Tanh Units (GTU) [van den Oord et al., 2016].

SoftCount. As a baseline approach, we train a model to count directly from the outputs s of the scoring function. We allow each object to contribute a value between 0 and 1. The total count C is the sum of these fractional, object-specific count values. We train this model by minimizing the Huber loss associated with the counting error $|C - C^{\text{GT}}|$. Counts are rounded at test time.

Attention Baseline (UpDown). As a second baseline, we re-implement the QA architecture introduced in Anderson et al. [2017], which the authors refer to as UpDown. We focus on this architecture because (i) it represents the current state of the art for VQA 2.0, (ii) it is designed for the type of visual representation we employ, and (iii) it exemplifies the common, attention-based approach to VQA. This architecture converts the score vectors to attention weights, which are used to compute an attention-weighted sum of the object encodings. The resulting fixed-length representation of the image and the question encoding are used to estimate the count through additional GTU layers.

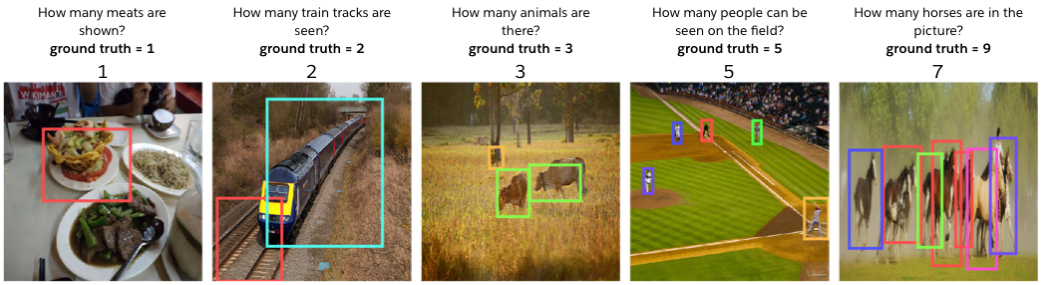

Figure 2: Grounded counts produced by IRLC. Counts are formed from selections of detected objects. Each image displays the objects that IRLC chose to count.

Interpretable RL Counter (IRLC). The goal of our model is to count by enumerating the subset of objects that the question refers to. To implement this as a sequential decision process, we project the object scores $s \in \mathbb{R}^{N\text{x}n}$ to a vector of logits $\kappa \in \mathbb{R}^N$, representing how likely each object is to be counted, where N is the number of detected objects:

$$\kappa = Ws + b \tag{3}$$

And we model how each action affects subsequent choices through a matrix of interaction terms $\rho \in \mathbb{R}^{N\text{x}N}$ that are used to update the logits κ. The value ρ_{ij} represents how selecting object i will change κ_j. We calculate this interaction from the question, the object encodings' dot product, and the coordinates (which also includes simple overlap metrics):

$$\rho_{ij} = f^{\rho}\left(\left[Wq, \hat{v}_i^{\mathrm{T}}\hat{v}_j, b_i, b_j, \mathrm{IoU}_{ij}, \mathrm{O}_{ij}, \mathrm{O}_{ji}\right]\right) \tag{4}$$

where $f^{\rho} : x \in \mathbb{R}^m \Rightarrow \mathbb{R}$ is a 2-layer MLP with ReLU activations.

For each step t of the counting sequence we greedily select the action with the highest value (interpreted as either selecting the next object to count or terminating), and update κ accordingly:

$$a^t = \mathrm{argmax}_i \left[\kappa^t, \zeta\right] \tag{5}$$

$$\kappa^{t+1} = \kappa^t + \rho(a^t, \cdot) \tag{6}$$

where ζ is a learnable scalar representing the logit value of the terminal action, and κ^0 is the result of Equation 3. The action a^t is expressed as the index of the selected object. $\rho(a^t, \cdot)$ denotes the row of ρ indexed by a^t. Each object is only allowed to be counted once. We define the count C as the timestep when the terminal action was selected $t : a^t = N + 1$.

This approach bears some similarity to Non-Maximal Suppression (NMS), a staple technique in object detection to suppress redundant proposals. However, our approach is far less rigid and allows the question to determine how similar and/or overlapping objects interact.

Training IRLC. Because of the discrete decision process, training requires that we use techniques from Reinforcement Learning. Given our formulation, a natural choice is to apply REINFORCE [Williams, 1992]. To do so, we calculate a distribution over action probabilities p^t from κ^t and generate a count by iteratively sampling actions from the distribution:

$$p^t = \mathrm{softmax}\left(\left[\kappa^t, \zeta\right]\right) \qquad a^t \sim p^t \tag{7}$$

$$\kappa^{t+1} = \kappa^t + \rho(a^t, \cdot) \tag{8}$$

We calculate the reward using Self-Critical Sequence training [Rennie et al., 2017, Anderson et al., 2017, Paulus et al., 2017], a variation of policy gradient. We define $E = |C - C^{\mathrm{GT}}|$ to be the count error and use $R = E^{\mathrm{greedy}} - E$, where E^{greedy} is the baseline count error obtained by greedy action selection. We train by minimizing the counting loss:

$$\tilde{L}_C = -R \sum_t \log p^t\left(a^t\right) \tag{9}$$

Additional details are located in the Appendix (Sec. B).

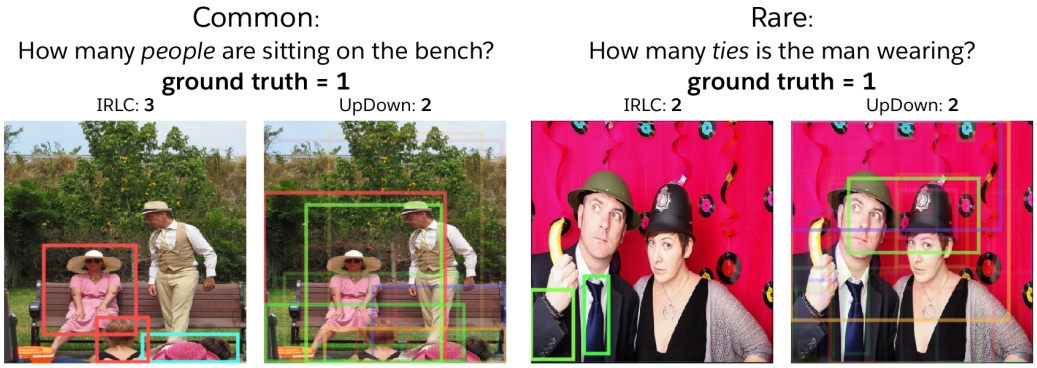

Figure 3: Examples of failure cases with common and rare subjects ("people" and "ties," respectively). Each example shows the output of IRLC, where boxes correspond to counted objects, and the output of UpDown, where boxes are shaded according to their attention weights.

3 Experiments

3.1 Counting Performance

We experiment with models trained on a new dataset, which we call HowMany-QA (Sec. C.1), built from the counting questions of VQA 2.0 [Agrawal et al., 2015] and Visual Genome [Krishna et al., 2016]. For consistency with past work, we evaluate using the standard VQA test metric of accuracy. To capture the typical scale of counting error, we include root-mean-squared-error (RMSE).

	Accuracy	RMSE
SoftCount	50.2	**2.37**
UpDown	51.5	2.69
IRLC	**56.9**	**2.39**

Table 1: Test set performance

IRLC achieves the highest overall accuracy and (with SoftCount) the lowest RMSE on the HowMany-QA test set (Table 1). The performances of the two baseline models suggest that accuracy and RMSE are not redundant; therefore, we emphasize that IRLC outperforms the state of the art model for both metrics.

3.2 Qualitative Analysis

The design of IRLC is inspired by the ideal of interpretable VQA [Chandrasekaran et al., 2017]. One hallmark of interpretability is the ability to predict failure modes. We argue that this is made more approachable by requiring IRLC to identify the objects in the scene that it chooses to count.

Figure 3 illustrates two failure cases that exemplify observed trends in IRLC. In particular, IRLC has little trouble counting people (they are the most common subject) but encounters difficulty with referring phrases (in this case, "sitting on the bench"). When asked to count ties (a rare subject), IRLC includes on the wrong type of object (a sleeve). These failures are obvious by virtue of the grounded counts, which point out exactly which objects IRLC counted. In comparison, the attention focus of UpDown (representing the closest analogy to a grounded output) does not identify any pattern. From the attention weights, it is unclear which scene elements form the basis of the returned count. Further visualizations are provided in the Appendix (Sec. A).

4 Conclusion

We present an interpretable approach to counting in visual question answering, based on learning to enumerate objects in a scene. By using RL, we are able to train our model to make binary decisions about whether a detected object contributes to the final count. We experiment with two additional baselines and control for variations due to visual representations and for the mechanism of visual-linguistic comparison. The counts produced by our model are not only more accurate than the baselines' but are also visually grounded in the detected objects, thereby improving both performance and interpretability.

References

A. Agrawal, J. Lu, S. Antol, M. Mitchell, C. L. Zitnick, D. Parikh, and D. Batra. VQA: Visual Question Answering. *International Journal of Computer Vision*, 2015.

P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang. Bottom-Up and Top-Down Attention for Image Captioning and VQA. *arXiv*, 2017.

A. Chandrasekaran, D. Yadav, P. Chattopadhyay, V. Prabhu, and D. Parikh. It Takes Two to Tango: Towards Theory of AI's Mind. *arXiv*, 2017.

P. Chattopadhyay, R. Vedantam, R. R. Selvaraju, D. Batra, and D. Parikh. Counting Everyday Objects in Everyday Scenes. In *CVPR*, 2017.

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. In *CVPR*, 2017.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv*, 2014.

R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. S. Bernstein, and F.-F. Li. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *International Journal of Computer Vision*, 2016.

T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context. In *ECCV*, 2014.

Y. Luo, C.-c. Chiu, N. Jaitly, and I. Sutskever. Learning Online Alignments with Continuous Rewards Policy Gradient. In *ICASSP*, 2017.

V. Minh, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *ICML*, 2016.

R. Paulus, C. Xiong, and R. Socher. A Deep Reinforced Model for Abstractive Summarization. *arXiv*, 2017.

J. Pennington, R. Socher, and C. Manning. Glove: Global Vectors for Word Representation. *EMNLP*, 2014.

S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, 2015.

S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel. Self-critical Sequence Training for Image Captioning. In *CVPR*, 2017.

D. Teney, P. Anderson, X. He, and A. van den Hengel. Tips and Tricks for Visual Question Answering: Learnings from the 2017 Challenge. In *CVPR*, 2017.

A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional Image Generation with PixelCNN Decoders. In *NIPS*, 2016.

R. J. Williams. Simple statistical gradient-following methods for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

R. J. Williams and J. Peng. Function Optimization using Connectionist Reinforcement Learning Algorithms. *Connection Science*, 3(3):241–268, 1991.

A Examples

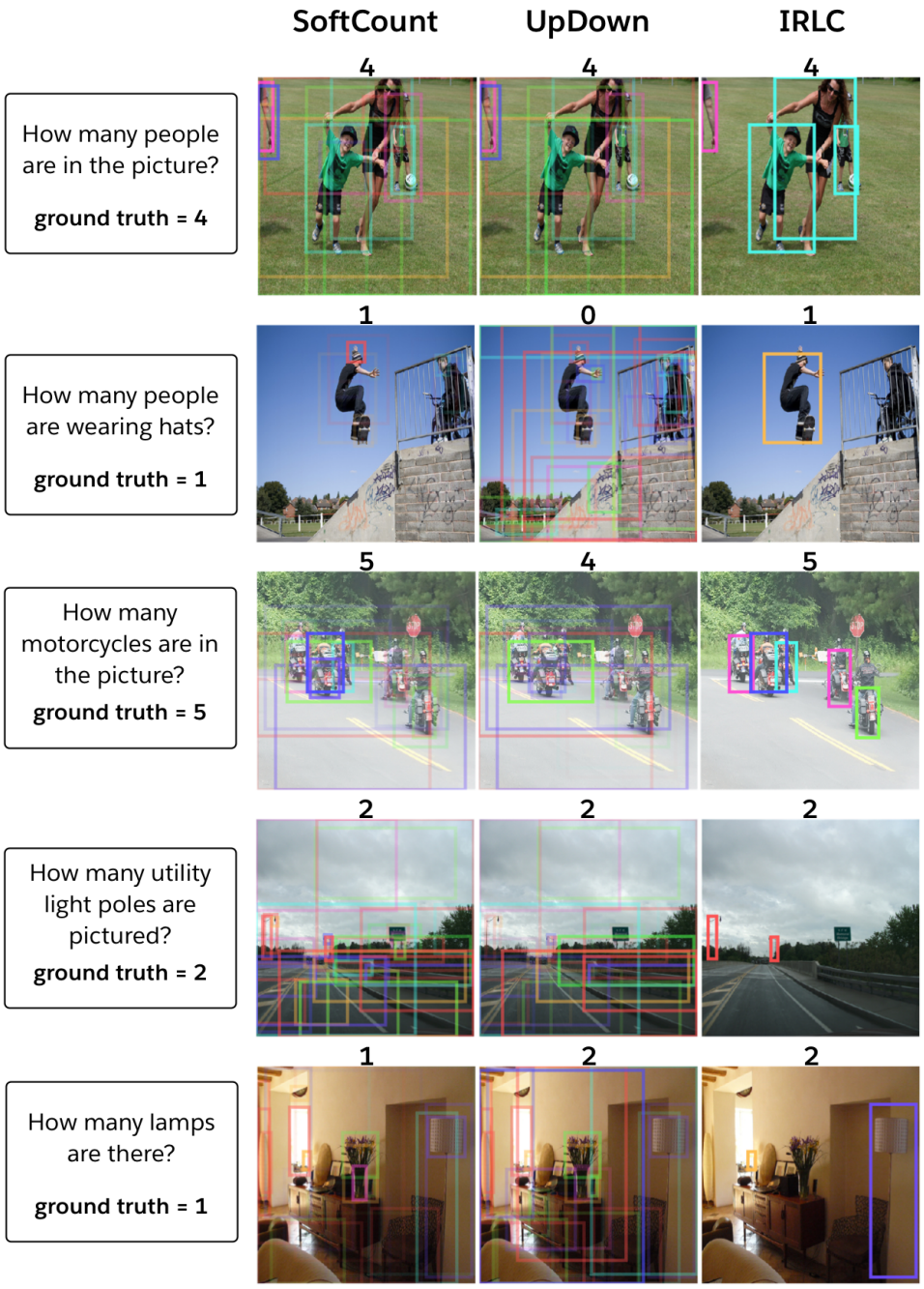

Figure 4: Example outputs produced by each model. For SoftCount, objects are shaded according to the fractional count of each (0=transparent; 1=opaque). For UpDown, we similarly shade the objects but use the attention focus to determine opacity. For IRLC, we plot only the boxes from objects that were selected as part of the count.

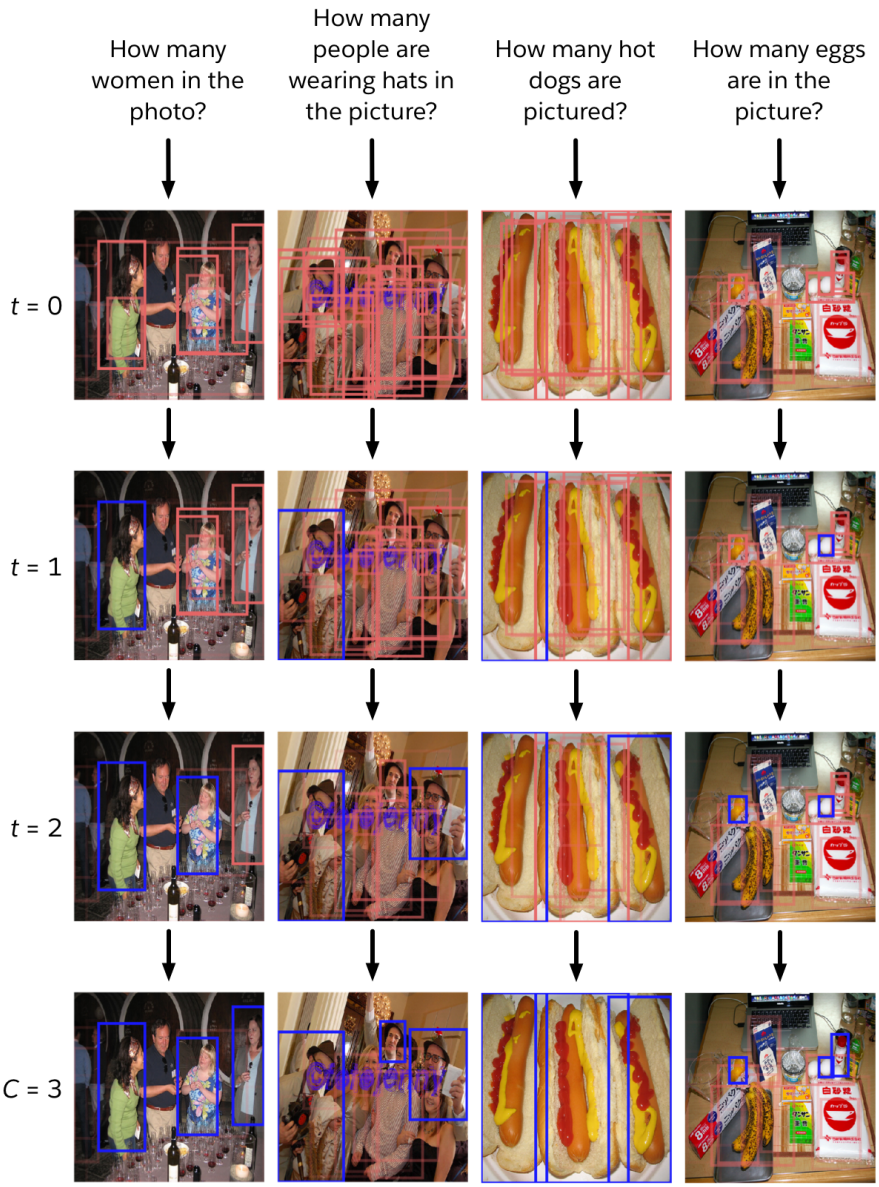

Figure 5: Sequential counting of IRLC. At each timestep, we illustrate the unchosen boxes in pink, and shade each box according to κ^t (corresponding to the probability that the box would be selected at that time step; see main text). We also show the already-selected boxes in blue. For each of the questions, the counting sequence terminates at $t = 3$, meaning that the returned count C is 3. For each of these questions, that is the correct answer. The example on the far right is a 'correct failure,' a case where the correct answer is returned but the counted objects are not related to the question. These kinds of subtle failures are revealed with the grounded counts.

B Training and implementation details

Each of the considered counting models makes use of the same basic architecture for encoding the question and comparing it with each of the detected objects. For each model, we initialized the word embeddings from GloVe [Pennington et al., 2014] and encoded the question with an LSTM of hidden size 1024. The only differences in the model-specific implementations concern the hidden size of the scoring function f^S and joint-training with caption grounding (see below). We determined these specifics from the optimal settings observed during initial experiments. We use a hidden size of 512

for SoftCount and UpDown and a hidden size of 2048 for IRLC. We observed that the former two models were more prone to overfitting, whereas IRLC benefited from the increased capacity.

We include joint training of caption grounding for SoftCount and IRLC. The goal of caption grounding is, given a set of objects and a caption, to identify the object that the caption describes. We build this auxiliary data from the region captions available as part of the Visual Genome dataset. We use an LSTM to encode the caption and compare it to each of the objects:

$$h_i^t = \text{LSTM}\left(x_i^t, h_i^{t-1}\right) \tag{10}$$

$$\alpha_{ij} = W f^S\left(\left[h_i^T, v_j\right]\right) + b \tag{11}$$

$$p_{ij} = \text{softmax}\left(\alpha_i\right)_j \tag{12}$$

where $h \in \mathbb{R}^{1024}$, $x_i^t \in \mathbb{R}^{300}$ is the embedding for the token at timestep t of the caption indexed by i, T is the caption length, and $f^S : \mathbb{R}^m \to \mathbb{R}^n$ is the scoring function (Sec. 2.2). $\alpha_{ij} \in \mathbb{R}$ is the score of the pair formed by caption i and object j. We only score the subset of objects that have a caption and train p_{ij} to be largest when $i = j$. (Note: counting and caption-grounding make use of the same scoring function but encode the question/caption with separate LSTMs.) With IRLC, we also tie the weight vector W that projects the score vectors during counting and caption grounding. During training, we randomly select four of the images in the batch of examples to use for caption grounding (rather than the full 32 images that make up a batch). Caption grounding with detected boxes also requires that we assign predicted detections with ground truth boxes. To do so, we select assignments greedily using an IoU threshold of 0.5. We weight the loss associated with caption grounding by 0.1 relative to the counting loss. We did not perform any joint training with UpDown, since the two objectives appeared to interfere with one another during initial experiments.

When training on counting, we optimize using Adam [Kingma and Ba, 2014]. For SoftCount and UpDown, we use a learning rate of $3\text{x}10^{-4}$ and decay the learning rate by 0.8 when the training accuracy plateaus. For IRLC, we use a learning rate of $5\text{x}10^{-4}$ and decay the learning rate by 0.99999 every iteration. For all models, we regularize using dropout and apply early stopping based on the Development set accuracy (see below).

When training IRLC, we include two auxiliary objectives to aid learning. For each sampled sequence (Eq. 7), we measure the total negative policy entropy H across the observed time steps. We also measure the average interaction strength at each time step (Eq. 8) and collect the total:

$$\tilde{P}_{\text{H}} = -\sum_t H\left(p^t\right) \qquad \tilde{P}_{\text{I}} = \sum_{i \in \{a^0 \ldots a^t\}} \frac{1}{N} \sum_j L_1\left(\rho_{ij}\right) \tag{13}$$

where L_1 is the Huber loss. Including the entropy objective is a common strategy when using policy gradient [Williams and Peng, 1991, Minh et al., 2016, Luo et al., 2017] and is used to improve exploration. The interaction penalty is motivated by the *a priori* expectation that interactions should be sparse. From our observations, both terms significantly influence the strategy the model ultimately learns to use. During training, we minimize a weighted sum of the three losses, normalized by the number of sampled decision steps ($C + 1$).

$$L = \frac{\tilde{L}_C + \lambda_H \tilde{P}_H + \lambda_I \tilde{P}_I}{C + 1} \tag{14}$$

When training IRLC, we apply the sampling procedure 5 times per question and average the losses. We weight the penalty terms using $\lambda_H = 0.002$ and $\lambda_I = 0.01$.

C Datasets

Within the field of VQA, the majority of progress has been aimed at the VQA dataset [Agrawal et al., 2015] and, more recently, VQA 2.0 [Goyal et al., 2017], which expands the total number of questions in the dataset and attempts to reduce bias by balancing answers to repeated questions. VQA 2.0 consists of 1.1M questions pertaining to the 205K images from COCO [Lin et al., 2014]. The examples are divided according to the official COCO splits.

In addition to VQA 2.0, we incorporate the Visual Genome (VG) dataset [Krishna et al., 2016]. Visual Genome consists of 108K images, roughly half of which are part of COCO. VG includes its

Figure 6: Examples of question-answer pairs that are excluded from HowMany-QA. This selection exemplifies the common types of "number" questions that do not require counting and therefore distract from our objective: (from left to right) time, general number-based answers, ballparking, and reading numbers from images. Importantly, the standard VQA evaluation metrics do not distinguish these from counting questions; instead, performance is reported for "number" questions as a whole.

own visual question answering dataset. We include examples from that dataset when they pertain to an image in the VQA 2.0 training set. We also point out that the pre-trained vision module used to extract each image's feature representation was trained from the object and attribute annotations from VG.

C.1 HowMany-QA

In order to evaluate counting specifically, we define a subset of the QA pairs, which we refer to as HowMany-QA. Our inclusion criteria were designed to filter QA pairs where the question asks for a count, as opposed to simply an answer in the form of a number (Fig 6). For the first condition, we require that the question contains one of the following phrases: "how many", "number of", "amount of", or "count of". We also reject a question if it contains the phrase "number of the", since this phrase frequently refers to a printed number rather than a count (i.e. "what is the number of the bus?"). Lastly, we require that the ground-truth answer is a number between 0 to 20 (inclusive). The original VQA 2.0 train set includes roughly 444K QA pairs, of which 57,606 are labeled as having a "number" answer. Focusing on counting questions results in a still very large dataset with 47,542 pairs showing the importance of this subtask. We will make the filtering scripts available so future research can compare on this same dataset.

Due to our filter and focus on counting questions, we cannot make use of the official test data since its annotations are not available. Hence, we divide the validation data into separate development and test sets. More specifically, we apply the above criteria to the official validation data and select 5,000 of the resulting QA pairs to serve as the test data. The remaining 17,714 QA pairs are used as the development set.

As mentioned above, the HowMany-QA training data is augmented with available QA pairs from Visual Genome, which are selected using the same criteria. Table 2 provides a breakdown of the size and composition of HowMany-QA.

Split	QA Pairs	Images
Train	83,642	31,932
from VQA 2.0	47,542	31,932
from VG	36,100	0
Dev.	17,714	13,119
Test	5,000	2,483

Table 2: Size breakdown of HowMany-QA. The contributions of VQA 2.0 and Visual Genome are provided for the train split. (Neither development or test included Visual Genome data.)

C.2 Evaluation metrics

Accuracy. The VQA dataset includes annotations from ten human reviewers per question. The accuracy of a given answer a depends on how many of the provided answers it agrees with. It is

scored as correct if at least 3 humans answers agree:

$$\texttt{Acc}\,(a) = \texttt{min}\left[\frac{\text{\#humans that said } a}{3},\ \ 1\right] * 100\% \tag{15}$$

Each answer's accuracy is averaged over each 10-choose-9 set of human answers. As described in the main text, we only consider examples where the consensus answer was in the range of 0-20. We use all ten labels to calculate accuracy, regardless of whether individual labels deviate from this range. We report the average accuracy over the test set questions.

RMSE. This metric simply quantifies the typical deviation between the model count and the ground truth. Across a set of N questions, this metric is defined as

$$\texttt{RMSE} = \sqrt{\frac{1}{N}\sum_i (\hat{C}_i - C_i)^2} \tag{16}$$

Here, $\hat{C}_i$ and C_i are the predicted and ground truth counts, respectively, for question i. RMSE is a measurement of error, so lower is better.