

Papers I Read

Learning to Count Objects in Natural Images for Visual Question Answering

2018 • [Count Based VQA](#) • [ICLR 2018](#) • [AI](#) • [CV](#) • [ICLR](#) • [NLP](#) • [VQA](#) • [SOTA](#)

06 May 2018

Introduction

- Most of the visual question-answering (VQA) models perform poorly on the task of counting objects in an image. The main reasons are:
 - Most VQA models use a soft attention mechanism to perform a weighted sum over the spatial features to obtain a single feature vector. These aggregated features helps in most category of questions but seems to hurt for counting based questions.
 - For the counting questions, we do not have a ground truth segmentation of where the objects to be counted are present on the image. This limits the scope of supervision.
- Additionally, we need to ensure that any modification in the architecture, to enhance the performance on the counting questions, should not degrade the performance on other classes of questions.
- The paper proposes to overcome these challenges by using the attention maps (and not the aggregated feature vectors) as input to a separate **count** module.
- [Link to the paper](#)

Notes

The basic idea is quite intuitive: when we perform weighted averaging based on different attention maps, we end up averaging the features corresponding to the different instances of an object. This makes the feature vectors indistinguishable from the scenario where we had just one instance of the object in the image.

Even multiple glimpses (multiple attention steps) can not resolve this problem as the weights given to one feature vector would not depend on the other feature vectors (that are attended to). Hard attention could be more useful than soft-attention but there is not much empirical evidence in support of this hypothesis.

The proposed **count** module is a separate pipeline that can be integrated with most of the existing attention based VQA models without affecting the performance on non-count based questions.

The inputs to the **count** module are the attention maps and the object proposals (coming from some pre-trained model like the RCNN model) and the output is a count-feature vector which is used to answer the count based question.

The top level idea is the following - given the object proposals and the attention maps, create a graph where nodes are objects (object proposals) and edges capture how similar two object proposals are (how much do they overlap). The graph is transformed (by removing and scaling edges) so that the count of the object can be obtained easily.

To explain their methodology, the paper simplifies the setting by making two assumptions:

- The first assumption is that the attention weights are either 1 (when the object is present in the proposal) or 0 (when the object is absent from the proposal).
- The second assumption is that any two object proposals either overlap completely (in which case, they are corresponding to the exact same object and hence receive the exact same weights) or the two proposals have zero overlap (in which case, they

must be corresponding to completely different objects).

These simplifying assumptions are made only for the sake of exposition and do not limit the capabilities of the **count** module.

Given the assumptions, the task of the count module is to handle the exact duplicates to prevent double-counting of objects.

As the first step, the attention weights (\mathbf{a}) are used to generate an attention matrix (\mathbf{A}) by performing an outer product between \mathbf{a} and \mathbf{a}^T . This corresponds to the step of creating a graph from the input.

\mathbf{A} corresponds to the adjacency matrix of that graph. The attention weight for the i^{th} proposal corresponds to the i^{th} node in the graph and the edge between the nodes i and j has the weight $\mathbf{a}_i * \mathbf{a}_j$.

Also note that the graph is a weighted directed graph and the subgraph of vertices satisfying the condition $\mathbf{a}_i = 1$ is a complete directed graph with self-loops. Given such a graph, the number of vertices, $V = \sqrt{E}$ where E could be computed by summing over the adjacency matrix. This implies that if the proposals are distinct, then the count can be obtained trivially by performing a sum over the adjacency matrix.

The objective is now to eliminate the edges such that the underlying objects are the vertices of a complete subgraph. This requires removing two type of duplicate edges - intra-object edges and inter-object edges.

Intra-object edges can be removed by computing a distance matrix, \mathbf{D} , defined as $1 - \text{IoU}$, where IoU matrix corresponds to the Intersection-over-Union matrix. A modified adjacency matrix \mathbf{A}' is obtained by performing the element-wise product between $f_1(\mathbf{A})$ and $f_2(\mathbf{D})$ where f_1 and f_2 are piece-wise linear functions that are learnt via backpropagation.

The inter-object edges are removed in the following manner:

- Count the number of proposals that correspond of each instance of an object and then scale down the edges corresponding to the different instances by that number.
- This creates the effect of reducing the weights of multiple proposals equivalent to a single proposal.
- The number of proposals corresponding to an object is not available as an annotation in the training pipeline and is estimated based on the similarity between the different proposals (measured via the attention weights \mathbf{a} , adjacency matrix \mathbf{A} and distance matrix \mathbf{D}).
- The matrix corresponding to the similarity between proposals ($\mathbf{sim}_{i,j}$) is transformed into a vector corresponding to the scaling factor of each node (\mathbf{s}_i)

\mathbf{s} can be converted into a matrix (by doing outer-product with itself) so as to scale both the incoming and the outgoing edges. The self edges (which were removed while computing \mathbf{A}' are added back (after scaling with \mathbf{s}) to obtain a new transformed matrix \mathbf{C} .

The transformed matrix \mathbf{C} is a complete graph with self-loops where the nodes corresponds to all the relevant object instances and not to object proposals. The actual count can be obtained from \mathbf{C} by performing a sum over all its values as described earlier. The original count problem was a regression problem but it is transformed into a classification problem to avoid scale issues. The network produces a \mathbf{k} -hot \mathbf{n} -dimensional vector called \mathbf{o} where \mathbf{n} is the number of object proposals that were feed into the module (and hence the upper limit on upto how large a number could the module count). In the ideal setting, \mathbf{k} should be one, as the network would produce an integer value but in practice, the network produces a real number so \mathbf{k} can be upto 2. If \mathbf{c} is an exact integer, the output is a 1-hot vector with the value in index corresponding to \mathbf{c} set to 1. If \mathbf{c} is a real number, the output is a linear interpolation between two one-hot vectors (the one-hot vectors correspond to the two integers between which \mathbf{c} lies).

count module supports computing the confidence of a prediction by defining two variables $p_{\mathbf{a}}$ and $p_{\mathbf{D}}$ which

compute the average distance of $f_6(\mathbf{a})$ and $f_7(\mathbf{D})$ from 0.5.

The final output \mathbf{o}' is defined as $f_8(p_{\mathbf{a}} + p_{\mathbf{D}}) \cdot \mathbf{o}$

All the different f functions are piece wise linear functions and are learnt via backpropagation.

Experiments

The authors created a new category of count-based questions by filtering the number-type questions to remove questions like “What is the time right now”. These questions do have a neumerical answer but do not fall under the purview of count based questions and hence are not targeted by the **count** model.

The authors augmented a state of the art [VQA model](#) with their **count** module and show substantial gains over the count-type questions for the [VQA-v2 dataset](#). This augmentation does not drastically impact the performance on non-count questions.

The overall idea is quite crisp and intuitive and the paper is easy to follow. It would be even better if there were some more abalation studies. For example, why are the piece-wise linear functions assumed to have 16 linear components? Would a smaller or larger number be better?

Related Posts

[Hints for Computer System Design](#) 07 Jan 2022


[Synthesized Policies for Transfer and Adaptation across Tasks and Environments](#) 29 Mar 2021

[Deep Neural Networks for YouTube Recommendations](#) 22 Mar 2021



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Be the first to comment.