Bottom-Up and Top-Down Attention for Image Captioning

20210784 Ayhan Suleymanzade
20190885 Ali Ahmed
Team 14

Abstract

Top-down visual attention mechanisms have been used extensively in image captioning to enable deeper image understanding through fine-grained analysis and even multiple steps of reasoning. We propose a novel bottom-up attention mechanism that enables attention to be calculated at the level of objects and other salient image regions. This is the natural basis for attention to be considered. Within our approach, the bottom-up mechanism proposes image regions, each with an associated feature vector.

We augment the base model in the paper with an Auto-Reconstructor Network, an Adaptive Attention mechanism via a Visual Sentinel, and our novel approach of salient attention via salience map features. We show that our proposed ARNet architecture and Visual Sentinel models achieve comparable good results to the authors' implementation, and even lead to improvements in some cases. The code for our project is available at: https://github.com/MisakiTaro0414/CS492I-Project

1. Introduction

Image captioning is a very interesting task combining image and language understanding as a research of both computer vision and natural language processing. The goal of our project is to find the most optimal way to pre-process the input image, make its content presentable and generate a sequence of words by making connections between visual and textual parts. This research has gained much popularity in modern captioning frameworks, outlined in [5]. In our visual system, we can make visual and textual part connections by focusing volitionally by top-down signals (top-down attention) and automatically by bottom-up signals corresponded from novel, salient image regions (bottom-up attention). However, in early approaches, the connection is made by considering only top-down attention by taking output of one or more layers of a convolutional neural net (CNN). In this approach, the input regions correspond to a uniform grid of equally sized and shaped neural receptive fields – irrespective of the content of the image. Thus, this approach does not give consideration to the determination of the specific image regions for attention by lacking granularity. By considering the bottom-up mechanism as well, our model proposes a set of salient image regions, with each region represented by a pooled convolutional feature vector. As an outcome of this model, we are able to produce a more accurate description of the input image.

2. Method

First, given a set of image features $V = v_1, ..., v_k$, such that each $v_i \in \mathbb{R}^{\mathbb{D}}$ encodes a salient region of the image and it can be defined as the spatial output of our bottom-up attention model, defined in 2.1. Second, the captioning model consisting of 2 LSTM layers, uses a 'soft' top-down attention mechanism to learn the weight of each feature obtained from bottom-up attention mechanism and then generates the caption using sequential architecture.

2.1. Bottom-Up Attention using Faster-RCNN

In our network, we have used spatial regions in terms of bounding boxes. Faster-RCNN [6] was used as an encoder network to extract bottom-up features from the images. The main reason for choosing Faster-RCNN is that it shares convolution features with the down-stream detection network, the region proposal step is nearly cost-free. This enables a unified, deep-learning-based object detection system to run at near real-time frame rates. The learned RPN(Region Proposal Network) also improves region proposal quality and thus the overall object detection accuracy.

The output of Faster-RCNN is a set of bounding boxes for each image. As shown in 1, a convolutional network extracts feature maps from an input image, and passes it as an input to the Region Proposal Network. The RPN then generates bounding boxes. Features from both RPN and Faster-RCNN are shared and RoI pooled to generate classified bounding boxes as bottom-up attention features. The generated bounding boxes are subjected to last mean pooled convolutional layer such that the dimension D of each bottom-up feature is 2048. We have used total of 36

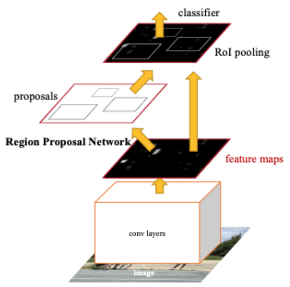


Figure 1. Faster-RCNN architecture

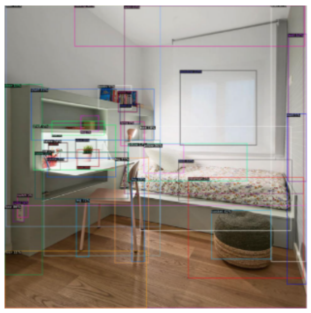


Figure 2. Bottom-up features obtained during inference

bounding boxes in our setting to generate captions.

2.2. Caption Generation

Our captioning model consists of top-down attention mechanism, together with attention on the bottom-up features. It therefore enables the caption generation with partially generated sentences. Within our model, we employ two LSTM layers to generate captions. For the base models, the memory cells obtained from LSTM do not have any importance in caption generation. However, we will excessively use these cells in the amenities brought to the model. The captioning model is described in Figure 3.

2.2.1 Top-Down Attention LSTM

The first layer of LSTM layers is called Top-Down Attention LSTM. The input to the attention LSTM at each time step consists of the previous output of the language LSTM (second LSTM layer), mean-pooled bottom-up features $\tilde{v} = \frac{1}{k}\sum_i v_i$ and an embedding of the previous generated word. This input vector is, therefore, represented as:

$$x_t^1 = [\mathbf{h}_{t-1}^2, \tilde{v}, W_e\Pi_t]$$

The embedding matrix W_e is a learnable parameter in our model. Π_t is one-hot encoding of the input word using vocabulary mapping. For a time step t we generate an attention weight as follows:

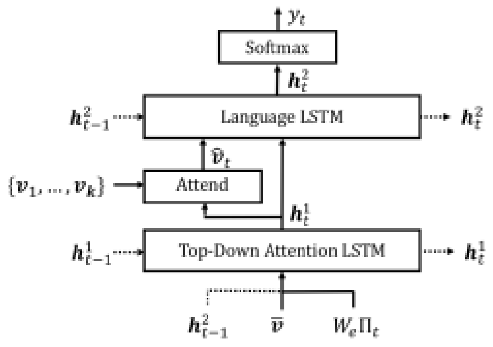


Figure 3. Captioning model architecture.

$$a_{i,t} = w_a^T tanh(W_{va}v_i + W_{ha}h_t^1)$$
$$\alpha_t = softmax(a_t)$$

All the input features are then used to calculate a convex combination, resulting in a single attended image feature $\hat{v}$ for our Language LSTM model:

$$\hat{v}_t = \sum_{i=1}^{K} \alpha_{i,t} v_i$$

2.2.2 Language LSTM

The concatenation of attended image features and the output of the attention LSTM, given by

$$x_t^2 = [\hat{v}_t, h_t^1]$$

are used by the language LSTM to generate conditional distributions over the words in vocabulary, calculated as

$$p(y_t|y_{1:t-1}) = softmax(W_p h_t^2 + b_p)$$

where W_p and b_p are the learnable parameters and biases. To calculate the distribution over the complete output sequence the product of conditional distributions is used:

$$p(y_{1:T}) = \prod_{t=1}^{T} p(y_t|y_{1:t-1})$$

2.3. Objective

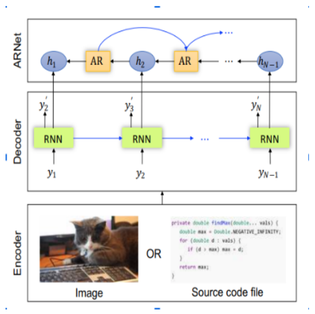
We train our model on a cross-entropy loss based on ground truth sequences of the form $y_{1:T}^*$ and a captioning model with parameters θ. The resulting loss function is a log-likelihood function over a conditional distribution p.

$$L_{XE}(\theta) = -\sum_{t=1}^{T} log(p_\theta(y_t^*|y_{1:t-1}^*))$$

3. Improvement Approaches

3.1. Auto-Reconstructor Network (ARNet)

Even though our proposed encoder-decoder model achieves remarkable performances, we detect some issues



```
private double findMax(double... vals) {
    double max = Double.NEGATIVE_INFINITY;
    for (double d : vals) {
        if (d > max) max = d;
    }
    return max;
}
```

Figure 4. Auto-Reconstructor Network (ARNet)

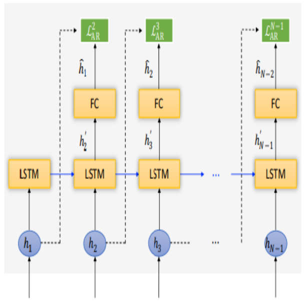


Figure 5. Auto-Reconstructor Network (ARNet): Architecture

which decrease the performance. First of all, our decoder network mostly relies on the input instead of the previous hidden state; basically, our transition operator can be described as input-dependent. More clearly, to generate the word y_t at timestep t, we rely on the hidden state h_{t-1} which has not fully exploited the latent relationship with its previous one h_{t2}. Secondly, more importantly, the disrepancy in training and inference due to teacher forcing can degrade the results as well. During the training phase, we take the ground-truth word y_t as input of the decoder model to predict the next word, enabling us to keep track of the generation output. However, the ground-truth word y_t is not available during the inference phase. This problem is called exposure bias. To solve these issues, we include the novel architecture introduced by [7]. This architecture introduces ARNet coupling, which is a model meant to strengthen the connection between neighboring hidden states by reconstructing the 'present with the past'. ARNet helps in regularizing the transition dynamics of the decoder model, mitigating the discrepancies in sequence prediction.

3.1.1 Architecture

In the ARNet architecture (shown in 5), the LSTM unit combined with fully connected layer is used to reconstruct the past hidden state h_{t-1} from the current one h_t. The reconstruction can be written as:

$$c_t' = f_t' * c_{t-1}' + i_t' * g_t'$$
$$h_t' = o_t' * tanh(c_t')$$

Here, i_t', f_t', o_t', c_t', and h_t' represent the input gate, forget gate, output gate, memory cell, and hidden state of the LSTM cell, respectively. Another fully connected layer is used to map the generated h_t' with h_{t-1}.

The output of the fully connected layer, $\hat{h}_{t-1}$, is then the reconstructed previous hidden state. The error in state generation can just be defined as the Euclidean distance between the true and the generated previous hidden state.

$$L^t_{AR} = ||h_{t-1} - \hat{h}_{t-1}||^2_2$$

Since we try to minimize this error, we encourage the current hidden state to embed maximum information from the previous one.

3.1.2 Training Procedure

In our model, we have employed ARNet only for the hidden states obtained from the first LSTM layer (Top-Down LSTM layer). Instead of using ARNet on the pretrained encoder-decoder model, we opted for training the both models together from scratch and add the loss of ARNet as a regularizing loss to our total loss function, defined as cross entropy loss. Thus the total loss in our model can be given as follows:

$$L = L_{XE} + \lambda \sum L^t_{AR}$$

Here, λ is a hyperparameter meant for balancing the encode-decoder architecture and the contributions from ARNet. We set it to be 0.005 considering the values in original paper implementation of ARNet.

3.2. Adaptive Attention via a Visual Sentinel

We implement a visual sentinel presented in [1] by ourselves, in order to introduce adaptive attention into our model. In our attention model, model attends to the image at every time step, irrespective of which word is going to be emitted next. However, not all words in the caption have corresponding visual signals. Visual Sentinel is an additional attention technique in which another learnable features will be used while generating "non-visual" words (e.g. "the", "of", and "on") instead of visual features. At each time step, the visual sentinel is computed from the previous hidden state and generated word. The adaptive encode-decoder framework can automatically decide when to rely on visual signals and when to rely on the language model. By changing the structure of attention mechanism, we add an additional visual sentinel vector to our model, as shown in Figure 6

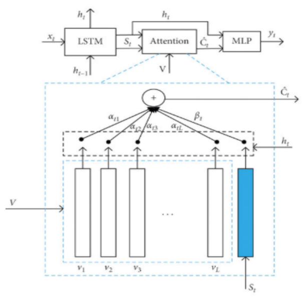


Figure 6. The Visual Sentinel Framework: Introduced by [1]

3.2.1 Adaptive Attention Model

With the help of a concept introduced in [1], we extend our model to adaptive attention via a Visual Sentinel.

This enables our model to store both long-term and short-term information about the visual and linguistic information. Our model can then can then fall back on this new component that it learns from this to choose to not attend to the image. The gate which decides whether or not to attend to the image is called the 'sentinel gate' and depends on the input to decoder model and the previous hidden state. To generate the sentinel gate, we have used the first layer LSTM (Top-Down LSTM) hidden and cell states. Thus, to obtain the visual sentinel vector s_t, we extend our decoder model following equations:

$$g_t = \sigma(W_x \mathbf{x}_t + W_h \mathbf{h}_t - 1) \; s_t = g_t * tanh(m_t)$$

Here, W_x and W_h are learnable weight parameters, x_t is the input to the LSTM at time step t, and g_t is the gate applied on the memory cell m_t. To generate the sentinel gate, we have used the first layer LSTM (Top-Down LSTM) hidden and cell states.

Now, after applying the fully connected layers to bottom-up features and obtained sentinel vector, these vectors are concatenated and are given as input to attention model. Thus, the attention weights are composed of 37 features instead of 36 features. By combining attention weighted sentinel with attention weighted bottom-up features, we finally obtain the context vector as follows:

$$\hat{c}_t = \beta_t s_t + (1 - \beta_t) c_t$$

where β_t is the attention weight for sentinel vector and produces values in the range [0,1].

3.3. Saliency Map Attention

Saliency maps process images to differentiate visual features in images. For example, coloured images are converted to black-and-white images in order to analyse the strongest colours present in them. Inspired by the fact that saliency maps can also be a good source of attention model, we enhanced the encoder part and attention mechanism in our model by considering the saliency maps.

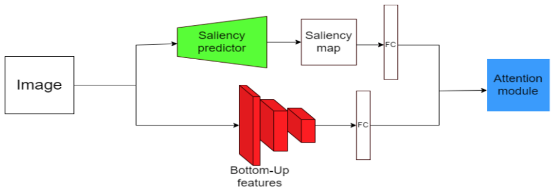


Figure 7. Attention to saliency. Credits to [4]

The architecture is described in Figure 7. To obtain the Saliency maps, we have used the existing pretrained Saliency predictor proposed in link in our MSCOCO dataset. Next, fully connected layers are applied to both the bottom-up features and the obtained sentinel vector to obtain the the feature sizes of 2048. Then, the vectors are concatenated and are passed as an input to the attention model. The obtained attention weights will play a balancing role between the attention given to the saliency map features and the attention given to bottom up features. Similar to Adaption Attention network, we will obtain the context vector by combining attention-weighted saliency map features with attention-weighted bottom-up features.

4. Implementation Details

4.1. Dataset

In order to train and evaluate our captioning model, we employ the MSCOCO 2014 dataset [3]. We use 'Karpathy' splits [2] to divide the data into training and validation sets, since it has been used by previous works in the literature, as well. This split consists of 113,287 images - with 5 captions each - for training and 5000 images each for validation and testing.

4.2. Bottom-Up Features

We obtain pre-trained bottom-up features for the MSCOCO dataset from GitHub. These contain 36 features per image, which can be recreated using the source code provided by the owner of the repository. These features are then stored as an HDF5 file, along with PKL files to map images to their index.

4.3. Training

Here are the some specifications regarding our experiment:

- Epochs: 50
- Initial Learning Rate: 0.01
- Optimizer: Adamax

- Dropout: 0.5
- Batch Size: 100
- Early Stopping: Yes (after 20 unsuccessful epochs)
- Feature Size: 2048
- Attention Size = Decoder Size = Embedding Size: 1024

We have also employed learning rate scheduler to decrease learning rate to 0.75 times its original value after 5 unsuccessful epochs.
In addition, we set the hyperparameter λ to be 0.005 in Auto-Reconstructive Network. Using NVIDIA RTX 3090 , it took more than 30 hours to train each experiment.

4.4. Evaluation

In caption generation, we have used beam search algorithm which selects multiple tokens for a position in a given sequence based on conditional probability. Basically, our model iteratively considers the set of the k best sentences up to time t as candidates to generate sentences of size t+1, and keeps only the resulting best k of them. The beam size k is set to be 5 in our model. We employ beam search decoding, only the captions from the decoded beam are sampled in our model. We use the same metrics used in the paper, since they are the standard for the Image-captioning literature. We score the generated captions along 7 categories, namely: BLEU-1, BLEU-2, BLEU-3, BLEU-4, METEOR, ROUGE-L, and CIDEr. In order to check the epoch performance, we used BLEU-4 score only.

5. Experimental Results

We train and evaluate our model on just one large dataset: MSCOCO2014, same as the authors do. However, it should be noted that we also generate and use the saliency maps for the entire dataset, which took us more than 2 days. We train the model on several changes in our architecture, which allows us to conduct an ablation study as well. The quantitative results are summarized in a table, and qualitative results are summarized as generated captions and attention maps.

5.1. Quantitative Results

Table 1 summarizes our results with each of the contributions upon the baseline model.

5.2. Qualitative Results

Figure 8 and Figure 9 summarize our qualitative results.

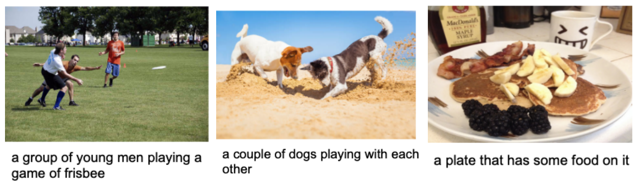

a group of young men playing a game of frisbee
a couple of dogs playing with each other
a plate that has some food on it

Figure 8. Generated Captions for 3 test images

6. Analysis

We managed to obtain significant results as compared to the original implementation, mainly due to our implementation of ARNet and Adaptive Attention. Although the original paper results are better in most of the evaluation metrics, we would like to mainly compare our results with the replicated paper. By mitigating exposure bias issue with regularized transition dynamics, ARNet can be considered the most successful novelty in our project. As a further work, we believe that utilizing the ARNet model for the hidden state of both the LSTM layers would produce even better results. In addition, simplified model using 1 LSTM layer did not produce very low scores, even producing second best METEOR score. Considering that training time has lowered more than 2 times using 1 layer of LSTM, we can utilize this model in case of limited resources. We were expecting better results by combination of Adaptive attention with ARNet structure. We speculate that the architecture needs to be modified further in order to utilize the combination to the maximum effectiveness. Due to long experiment times, we could not do hyperparameter tuning. As further work, we believe that by hyperparameter tuning, the replicated paper and our modifications on them would outperform the original paper.

7. Project Contributions

There is no original implementation of the paper implemented in PyTorch since it is implemented in Caffee. The contribution ideas that we implemented in the project can be divided into two parts. For ARNet and Visual Sentinel, there are existing papers that describe how to implement the ideas. The coding for those models, however, is completely done by ourselves. In addition, Saliency map architecture is a completely novel architecture that we proposed, and the code is completely written by ourselves. To obtain the saliency maps, we have used an existing implementation using this link. The code for inference using Faster R-CNN, visualization of attended bottom up features, and simplified LSTM model is written by ourselves as well. We have used detectron2 library to run a pretrained Faster R-CNN model to obtain the bounding boxes (bottom-up features) in inference. For reference, we found out this implementation very

Table 1. **Quantitative Results**: Comparison across 7 categories: Authors' Implementation vs Ours (+Improvements). **Bold** for best results, <u>Underscore</u> for second-best results

Model	Bleu-1	Bleu-2	Bleu-3	Bleu-4	METEOR	ROUGE-L	CIDEr
Original Implementation	**77.2**	-	-	**36.2**	<u>27.0</u>	**56.4**	**113.5**
Our Replication	75.4	59.5	45.9	35.4	**27.2**	56.0	<u>112.1</u>
ARNet	<u>76.1</u>	**60.5**	**46.9**	<u>36.1</u>	26.8	<u>56.2</u>	111.7
Adaptive Attention	75.8	<u>59.9</u>	46.3	35.8	**27.2**	<u>56.2</u>	112.0
1 Layer LSTM	75.5	59.4	45.8	35.4	<u>27.0</u>	55.8	111.5
ARNet + Adaptive Attention	75.7	<u>59.9</u>	<u>46.4</u>	35.8	26.9	56.0	110.8
Saliency Map Attention	75.8	59.7	46.1	35.5	<u>27.0</u>	56.1	111.9

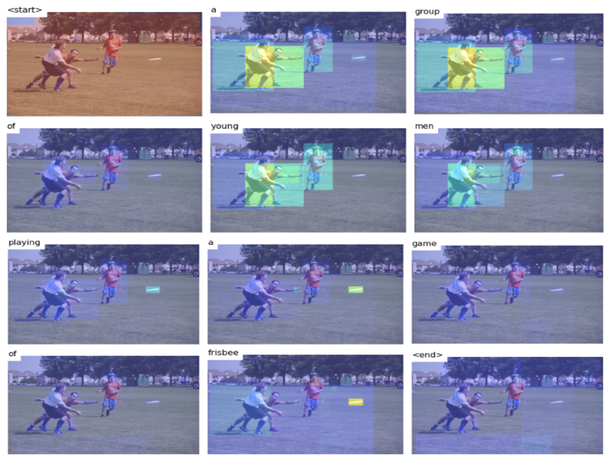


Figure 9. Attention Boxes generated by our model

useful.

8. Individual Contributions

Table 2 summarizes the contribution of each team member in our project implementation.

Table 2. Individual Contributions.

Name	Contributions
Ayhan Suleymanzade	Reproduced results (Baseline model implementation) Implemented all of improvement approaches (ARNet, Adaptive Attention, Saliency Map Attention, Simplified Model) Implemented inference by combining Faster-RCNN with the originial model Added novel visualization which shows the attentions on the bounding boxes generated Presented project and edited the report
Ali Ahmed	Obtained the saliency map images Solved GPU related issues Wrote the report using LaTeX

References

[1] Devi Parikh Richard Socher Jiasen Lu, Caiming Xiong. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. *CVPR*.

[2] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions, 2014.

[3] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[4] Giuseppe Serra Rita Cucchiara Marcella Cornia, Lorenzo Baraldi. Paying more attention to saliency: Image captioning with saliency and context attention.

[5] Lorenzo Baraldi Silvia Cascianelli Giuseppe Fiameni Rita Cucchiara Matteo Stefanini, Marcella Cornia. From show to tell: A survey on deep learning-based image captioning.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[7] Wenhao Jiang Jian Yao Wei Liu Xinpeng Chen, Lin Ma. Regularizing rnns for caption generation by reconstructing the past with the present. *CVPR*.