Visual Question Answering: Encoder Decoder Variants

Natcha Simsirim, Shiyan Yin
{nsimsiri, shiyanyin}@cs.umass.edu

Abstract

This report presents a baseline for visual question answering task. Given an image and a question in natural language, our model produces one-word answers according to the content of the image. We implemented 3 different architectures, Vanilla, Fusion, Attention, with a total of 32 models with different parameters and structures to compare the accuracy of the labels. We used the dataset Balanced Real Images VQA 2.0 from visualqa.org, and our best model, the attention model, achieves 41% accuracy on a small training dataset with 5000 questions. We also compared our result with a benchmark created by other researchers in their paper, and their accuracy is around 55% which is slightly higher than us. We also have more thoughts to improve the current models by tuning more parameters and training more data in the future.

1. Introduction

The task of Visual Question Answering (VQA) is as follows: given an image and a question that pertain contextual information on the image, have a machine output the answer to the question. This implies that a visual-semantic model must be built to correlate natural language information to visual information. To be more concrete, the model that is able to perform such a task must learn how to jointly reason over the two representation of data from a different domain. For instance, for the question is the cat black?, the model must be able to understand what the question is asking for (a yes or no answer), the entity of interest (cat) and question and the question interesting with respect to the object (color). As a result, a combined representation must be formed that can help us reason over the two domains. While there are many variants of VQA, for instance, the answers may be structurally different in that it may be a single word or a sentence, or even semantically different, in that the answer generated may have information in the dataset learned and is knowledge-based, the VQA task were dealing with is a classification on the known answer vocabulary set. Meaning, we will predict a single token answer, such that the answer has previously shown up in the training data. We will discuss this in more details in the later sections.

Deep learning has shown to produce many states of the art results over the past years, especially in machine learning tasks in the domain of image processing, and natural language processing. One of the challenges VQA faces is the need to for cross-domain understanding, and the features pertaining to these cross-domain challenges may be difficult to handcraft. Thus, deep representation became a field of interest especially in machine learning tasks that deal with data from multiple domains. Neural networks have shown great results in automatically learning these deep representations (hence the deep).

The way we approached VQA in this paper, is we treated the deep learning aspect as an encoder-decoder model, that learns to encode image and textual information, and decode the representation to perform a classification task where the predicted classes are the limited set of answer vocabulary the data set we have used has provided for us. We will discuss the dataset in more details in section x. Encoder-decoder models have shown to be state of the art technique in machine translation and image captioning [12]. In machine translation, we have a sentence S from one language, and we want to translate this to a sentence T in another language by learning a model and maximize $p(T|S,)$. Similarly, for our task, we seek to learn a model and maximize $p(S|I,)$ where S is the sentence we would like to generate and is the image. In the translation case, the state of the art was achievable by representing the probability distribution via a Recurrent Neural Network (RNN)that provides a robust way to encode an arbitrary length sequential information into a fixed length vector representation. Furthermore, the RNN could also be used to decode the representation and produce back a sentence or a representation vector that could be used for classification. To take this step a bit further, for VQA, we will leverage the encoder-decoder architecture used in the state of the art neural translation technique described previously. The challenge is finding a way to embedding inputted images, i.e representing visual semantics. Convolution Neural Networks (CNN) has shown to be a very promising deep learning method that learns image features and is able to represent them into a fixed length vector. Examples are the Oxford VGGNet[8]

or the GoogleNet [9]. We can use a trained CNN and connect the network to an RNN decoder to generate sentences. The in-depth details of the architecture will be described in the later sections.

2. Related Works

In this section, we will describe the recent state of the art papers in Visual Question Answering (VQA). As previously stated in the introduction, VQA has many variants, in both free form answering or sequence generation, or classification with both knowledge-based resources, and without knowledge-based resources.

2.1. Free-Form VQA

A recent paper Ask Me Anything: Free-form Visual Question Answering Based on Knowledge from External Sources [11] attempts to build a deep learning model with external knowledge sources in trying to perform the following variant of the VQA task: Given an image and an open-ended question, generate an answer in the form of sentence that have plausible soundness. Unlike the task were doing where we are mostly dealing with questions such as what, or where - questions that have direct answers obtainable from looking at the image, task is to answer the question of why. For example, instead of the what is the color of the umbrella?, they only want to answer why is the umbrella open?. This implies the model must have learned some common knowledge that may not be apparent in the training data. The general idea for their technique is to learn a visual-semantic embedding between an image and a caption. This means a vector that contains both visual and natural language information. Then use an encoder-decoder LSTM to learn the previously stated visual-semantic embedding between the images and captions, and then is trained on the open-ended relationship between the questions, images, and external knowledge base. By learning the relationship between captions and the image that it describes, the network not only learns the language model but also its meaning with respect to the image. The external knowledge base they used is DBPedia [**?**] which extracts information from Wikipedia. The data used from DBPedia is an encoding vector using the Doc2Vec algorithm. This knowledge-based encoding allows more free-form answers, as it is able to map known word encoding to other words that may not be in the VQA-dataset or captioning dataset. Finally, the image features were obtained through a CNN. The dataset used was the MS-COCO dataset [4] (same as ours), as well as caption from the MS-COCO Image Captioning challenge. The question annotation dataset is from VQA Dataset [5] and with this dataset, they achieved the result of 55.96%.

2.2. Neural Module Network

Recent trends in performing many cross-domain tasks such as machine translation, image captioning and even other VQA tasks tend to focus on building end-to-end deep networks. This paper, Neural Module Network [**?**], proposes to solve VQA by aggregating decompositional neural network, or the so-called neural module network, where they had smaller, atomic network learns a specific question task, or operations, and during test time, the structure of the sentence is broken into multiple operations and fed into the relevant network that will aggregate the output of these smaller networks into a final single answer. Image features are obtained through an image module network (CNN), an attention module is used to learn where to place weights on regions of the image, and sentence structure. Furthermore, the standard dependency parser is used to decompose the grammar structure of the sentence to individual language modules.

2.3. Stacked Attention Networks for Image Question Answering

Attention mechanism has shown to improve the performance of many encoder-decoder models in tasks such as machine translation, and image captions. What attention is, is a neural network layer(s) that learns to give higher weights on certain parts of input than others. In Stacked Attention Networks for Image Question Answering [**?**], the network in doing VQA consists of the image model, question model, and attention model. The model first extracts the final max-pooling layer of the VGG-16, which contains spatial information and fed through a fully connected layer that transforms the image feature to be the same as word embedding features. The language model they tried is an LSTM and a CNN. The word embeddings are fed into the LSTM at each time unit, while the unigram, bigram, and trigram convolution filters are used on the sentence input for the model. Max-pooling is used to dilute the CNN output to a single vector representing the language model. Finally, the stacked attention network, which is their final component of the model is to create an attention map which is built from combining the image feature vector and question vector (from either the LSTM and CNN). Then a weight is imposed on this single representation such that each portion of the weight corresponds to each portion of the image, and question. Thus, during training, the network will learn the feature of how to put more weights on different parts of the sentence and image. The model was evaluated on DAQUAR-all, DAQUARreduced, COCO-QA[5] and VQA.

2.4. Show, Ask, Attend, and Answer: A Strong Baseline For Visual Question Answering

This paper proposes a very similar VQA solution to that of [13]. In [2], VQA is also a classification task. The model

consists of image features extraction using ResNet152, and an LSTM as a language model wherein each time step, the word embedding is fed. Then the image feature and final hidden state are concatenated and gone through a series of deconvolution layers, followed by a softmax. These series of deconvolution layers and represents how they did the attention mechanism. The output of the attention softmax becomes the weight for the image region. What we have now is a weighted image feature based on information on the question. This weighted image feature is then concatenated to the hidden output of the LSTM, and is put through multiple fully connected layers, which goes through the final softmax for which the answer is predicted. On VQA2.0 Dataset (the one were using), this network achieves 59.67% on the validation set. The paper reports that the l2 normalization, dropout, and the number of FC layers, and use of attention greatly impacted the performance of the model.

3. Technical Approach

Our paper is tackling the VQA problem as a classification problem where given an image I_i and a question Q_i where the question and image mapping is represented by the lowercase i, we predict a single answer a_k from $A = a_0, ..a_K$ in which this set is used in both the training set and validation set. We perform classification from the following maximum likelihood equation:

$$ans = \underset{k}{\operatorname{argmax}}(P(a_k|I_i, Q_i))$$

3.1. Vanilla Encoder-Decoder

As previously stated, our contribution in this paper is in the experimentation of using the encoder-decoder neural network architecture that has previously shown state of the art for tasks such as machine translation and image captioning [10]. The way our encoder-decoder variant works for VQA is we first pass an image through a pre-trained CNN (In particular, we used ResNet152) and retrieve the previous to last layer which contains high level semantic related features which are of size 1 x 1 x 2048. We then feed this feature into a 2 dimensional fully connected layer that encodes this image feature in a feature that will layer on be compatible in shape with the embedded language features. We then begin to create an encoding for the language feature, where a sentence is inputted to an embedding layer, which essentially turns each token into a one-hot vector representing that word. The same hidden unit number is used in both the fully-connected layer that encodes the image feature and the embedding layer. Finally, we model the previously mentioned maximum likelihood on the answers by training a Recurrent Neural Network (RNN) where a many-to-one prediction occurs. We use an RNN because it is a neural network model that is capable of handling variable length sequences such as sentences.

$$ans = \underset{k}{\operatorname{argmax}}(P(a_k|I_i, q_0, q_1, ..q_n))$$

The way we did this is on the first time-step we feed the encoded image feature, and the subsequent time-step, each one-hot word vector is fed into the RNN. In cohesion to the many-to-one training, the final hidden state of the RNN, is fed into a fully connected layer of hidden units the same size as the answer vocabulary size, and a softmax function which outputs the probability distribution on the most likely answer. The final layers after the RNN are essentially our neural network classifier.

The intuitive idea is that the hidden state of the RNN stores the visual-semantic embedding since the first time-step uses the image feature as input.

In our experiment, we use three variants of the RNN [7]. The default RNN, Gated-Recurrent Unit (GRU) and (Long-Short Term Memory) LSTM unit. An RNN is simply an affine transformation on the weights vectors of hidden layer and output layer, where at each time step, the input is combined with the previously hidden state and uses the weight vectors of the hidden layer to store the transformation features, and output the new hidden state which will be used for the next time step. The hidden state will also be fed into the affine transformation on the output weight vectors for the final output of the cell at each time step. Here the hidden state keeps a probability distribution of the prefix sequence on the task the network is trying to model. For many-to-one learning, backpropagation through time is used to where gradients are propagated through the hidden states the update weights in the RNN cell. One issue from this is that the gradient may vanish for long sequences, and the model may not the features from the earlier words. This problem is crucial for us since the first few words indicate the question type - such as What is, or How many, which is an extremely important feature (We have trained this network on the first few hidden units representing the prefix of the sentence by mistake and have gotten approximately 15-20% accuracy on our validation set) . To alleviate the vanishing gradient issue for backpropagation in time, we will use both the Long-Short Term Memory (LSTM) unit and Gated Recurrent Unit (GRU). The gist of LSTM is it adds an additional memory cell within the RNN layer that is able to memorize the input up to this timestep. This mechanism helps the LSTM keep the backpropagation gradient steep and lessens the effect of the vanishing gradient [2]. The GRU, on the other hand, is a variant of LSTM but with an update gate and a reset gate which tells the network how much of the past information needs to be passed along [1].

3.2. Fusion Encoder-Decoder

One issue that we may foresee in the first encoder-decoder model is that since the image feature is fed into

the LSTM in the first time-step, and we know that information is passed along through the hidden unit of the RNN, a lot of information on the image may be lost along the way. In the fusion encoder-decoder model, instead of feeding the image features in the first time-step and treating it with equal weight as the word embedding we will instead concatenate this image feature with the final hidden unit, and feeding this into our classifier. Furthermore, we also extend our classifier to have multiple fully-connected layers with an additional dropout and ReLU units. We do this to treat the image feature vector equally as the question embedding and the complexity added to the classifier allows a higher model capacity in learning the visual-semantic embedding between the concatenated image and word features.

3.3. Encoder-Decoder with Attention

Our final contribution is to apply attention to a joint version of the vanilla and fusion encoder-decoder model. This model is split into 3 main layers - the language model layer, attention layer, and classifier layer. In the language model layer, we simply use the vanilla encoder-decoder. For the attention layer, we leverage the idea from [3] to lay out weights on the images - to, intuitively, have the network learn which regions of the image it should focus on during questions. This implies not using the encoded image features, but use the raw image feature from ResNet152 to perform the fusing in the attention layer (recall that for our previous models, our image feature is the output of an FC layer with the raw ResNet image feature as input in order to make the shape compatible).

For the attention layer, we approximate the a spatial distribution of $\sum_{l=1}^{N} A_{(x,y)} = 1$ where A is a weight that is estimated through a two convolution layer as suggested by [3]. The x,y here are the spatial location of the image we would like the network to "attend" to. Finally, we find the weighted sum of each image features "stripes" $O_x = \sum_y A_{(x,y)} * I_y$ for the image feature I and perform the "fusing" or concatenation of embed and image attention map features, which will be fed to the classifier.

Finally, our loss for all models is the following:

$$Loss = \frac{1}{K} \sum_{k=1}^{K} - log(P(a_k | I, Q))$$

4. Experimentation & Evaluation

Our experiment consists of comparing the three previously mentioned models: the vanilla encoder-decoder (also dubbed as naive), fused encoder-decoder and attention encoder-decoder on the the hyper-parameters including $hidden\_units = \{256, 512\}$, $embedding\_size = \{256, 512\}$ and $learning\_rate = \{0.001, 0.01\}$. The model is trained on 5,000 questions, image, answer tuple, and validated on 300 previously unseen tuples. All the validation accuracy are averaged for each model with different recurrent units and parameters for each architecture. Thus we train a total of 32 models with results shown in figure 3. Following the evaluation guidelines from [3] and for the sake of feasibility, we compare each hyper-parameter while keeping a default set of hyper-parameters fixed.

4.1. Experiment Set-Up

We ran all our experiments on a server-desktop with a quad-core 3.1 GHz Intel I7 processor on 16GB of RAM, and a GeForce GTX 1080 graphics card that has 6 GB of RAM. Each model approximately took 10-15 minutes to train. We trained a total of 32 models implemented in PyTorch 0.4 [5] with Python 3.5. We made sure that each tensor in our training model is loaded to our GPU by using PyTorch's $.cuda()$ and $.to(device)$ tensor functions where a device is an object that figures out whether or not you have a GPU that can be utilized. By doing so allows us to train the model by up to a factor of 10x.

4.2. Preprocessing

Extract 5000 one-word-answer questions and save these items as JSON files. See the structure of the item in the figure 2.

Use NLTK to tokenize questions a list of words, and add special tags on the questions like: $\langle start \rangle + question + \langle end \rangle$. At the same time, we create the vocabulary table for all questions and add special tags $\langle pad \rangle$ and $\langle unk \rangle$ in the table. We did a similar process on the answers, but we only added special tags $\langle unk \rangle$ in the vocabulary table for answers.

Once we got these vocabulary tables, we encoded the questions and answers using the index of the words. Moreover, we got all the corresponding question_length, question_id, image_id of questions, and all the unique image names, and we save these data as JSON and h5 files.

We created a function to process image data. Since there are many questions point to the same images, so we created an image data dictionary to save all the unique image data. We resize each image to 256 * 256 and normalized the image.

4.3. Training

We trained our models on an image, question and answer tuple of batch size 100, for 30 epochs, using the Adam optimizer with variable learning rate that will be our hyperparameter. We will refer to PyTorch's neural network library as nn. Each model would use a $nn.RNN$ variant such as $nn.LSTM$ and $nn.GRU$ which all take in a batch of encoded questions and a corresponding $lengths$ list object with the corresponding question lengths into as

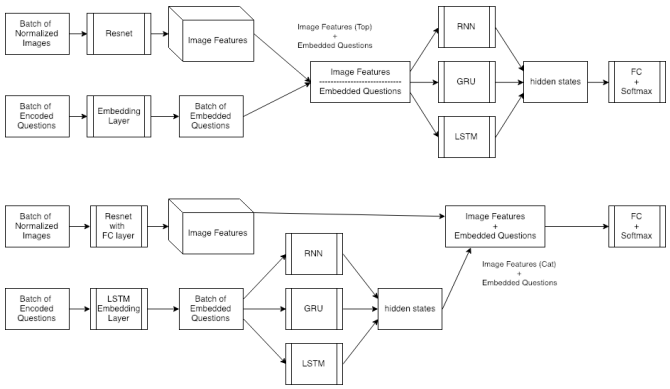

Figure 1. Architecture of Vanilla Encoder-Decoder and Fusion Encoder-Decoder

```
{
  "ques_id": "question_id",
  "img_path": "image_path",
  "question": "question",
  "question_type": "question_type",
  "ans": "ans"
}
```

Figure 2. The data type of the item we extract from raw data

a $PackedSequence$. This special PyTorch data-structure would allow the $nn.RNN$ and its sub-classes to output a non-zero hidden unit corresponding to the hidden unit of the last index of each variable length questions. This detail is important to us since if we simply take the final hidden output, gradients may have vanished for smaller length questions as it will trail along zeros that are the difference between the maximum question length and its length.

4.4. Results & Discussions

Figure 3 shows the final validation accuracy of the 32 models that were trained on the previously mentioned architectures and hyper-parameters. We see that the attention model and the GRU variant of the vanilla encoder-decoder model performs the best. We must also note here that all the attention models use LSTM as their recurrent unit, and that the difference between $attention\_*$ and $attention\_cn\_*$ as shown in the figure is the cn variant uses the LSMT memory cell as output to the attention mechanism while the other, and better-performing variant uses the hidden units. The reason we used a cell-state in the first place (as it is uncommonly used because the cell-state only encodes selective features from the past, which may lose some crucial signals), is because it is common in LSTM layer stacking.

While our result beats our own baseline, we will compare our result to a comparable state of the art can be found in [3]'s paper where they trained their model on 6,000 training data and achieved up to 55% validation accuracy. Although they used the VQA-1.0 version of the dataset, this is the closest state of the art setting a paper has that we can compare to as other papers do not report lower training size results. One speculation for as to why their model may perform better is that the VQA-1.0 dataset has more non-uniformity in terms of answers distribution. The skewed nature of the dataset may induce a lower number of answer vocabulary making their classification task slightly easier to train on. Another speculation is their paper uses a 14 x 14 x 2048 convolution output instead of 1 x 1 x 2048 image feature. This implies their image feature contains spatial information while ours did not, and as a result, the attention model will benefit more from their method of image feature extraction.

In this paper, we are also interested to see how each architecture is ranked and how each recurrent unit affects the validation accuracy on each of our model. Figure 5 compares the loss of the best performing model in its architectural category. We see that despite the attention model's lack of change in the loss it is able to get an improvement over the two models in terms of validation accuracy. This may imply that the naive and fusion encoder-decoder models are over-fitting, leading us to believe that one way to improve these models is to add regularization layers such as dropout, l2-normalization, or even batch normalization within the embedding layers inside the network.

Albeit confusing, the top-right validation accuracy represents the average accuracy for the models of its respective category. For example, for figure 5, on average, the attention model performs the best out of all the model, followed by the vanilla encoder-decoder and the fusion network. An interesting remark that can be made here is the fusion network was supposed to be an improvement on the vanilla encoder-decoder in that we speculated a boost in performance can be made via treating image features as "equal" as the language model. However, it is not the case. To take advantage of the "fusion", our results show that attention stacking from the attention model would be required. An observation to make from Figure 4 is that the LSTM, on average performs the best, while followed by the GRU and

Model	Accuracy	Model	Accuracy
attention_0.001_256_256	39.00	fusion_rnn_0.001_256_256	35.67
attention_0.001_512_512	**40.67**	fusion_rnn_0.001_512_512	33.33
attention_0.01_256_256	**41.00**	fusion_rnn_0.01_256_256	36.67
attention_0.01_512_512	**40.00**	fusion_rnn_0.01_512_512	14.00
attention_cn_0.001_256_256	36.00	naive_gru_0.001_128_256	**41.67**
attention_cn_0.001_512_512	37.67	naive_gru_0.001_128_512	**40.33**
attention_cn_0.01_256_256	39.33	naive_gru_0.01_128_256	36.67
attention_cn_0.01_512_512	39.67	naive_gru_0.01_128_512	37.33
fusion_gru_0.001_256_256	37.00	naive_lstm_0.001_128_256	37.33
fusion_gru_0.001_512_512	**40.67**	naive_lstm_0.001_128_512	39.67
fusion_gru_0.01_256_256	39.00	naive_lstm_0.01_128_256	38.00
fusion_gru_0.01_512_512	37.67	naive_lstm_0.01_128_512	39.00
fusion_lstm_0.001_256_256	37.00	naive_rnn_0.001_128_256	39.67
fusion_lstm_0.001_512_512	39.33	naive_rnn_0.001_128_512	39.00
fusion_lstm_0.01_256_256	39.33	naive_rnn_0.01_128_256	35.33
fusion_lstm_0.01_512_512	38.67	naive_rnn_0.01_128_512	31.67

Figure 3. Table of Validation Accuracy for all Models

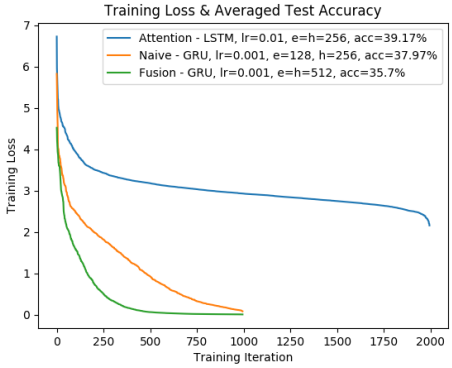

Figure 4. Training Loss of the best Model for each Architecture with Averaged Accuracy

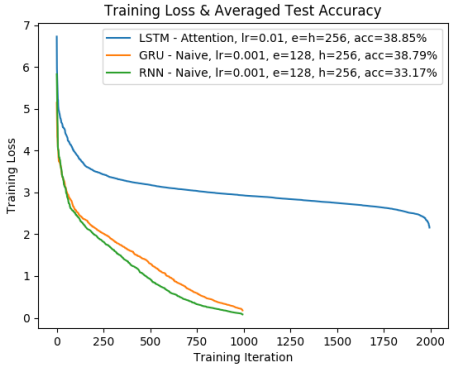

Figure 5. Training Loss of the best Model methods for each RNN methods with Averaged Accuracy

RNN. This results is also reflected in other state of the art papers such as [6].

5. Conclusion

We presented multiple encoder-decoder neural networks that are able to perform within the range of the state of the art deep models on similar training size and task. We have also shown that the encoder-decoder network that is very popular in translation task - whether it is from image to sentence, or sentence-to-sentence, have the capability to embed rationale implied by the question and images in finding an appropriate answer. Furthermore, we have also learned that stacked-attention can also be used to improve the naive encoder-decoder model on its LSTM-cell variant, but we also believe using a better feature extraction method will give way to improvement. Our code can be found here: https://github.com/CS670-VQA/vqa.

References

[1] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[2] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[3] V. Kazemi and A. Elqursh. Show, ask, attend, and answer: A strong baseline for visual question answering. *CoRR*, abs/1704.03162, 2017.

[4] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[5] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[6] H. Salehinejad, J. Baarbe, S. Sankar, J. Barfett, E. Colak, and S. Valaee. Recent advances in recurrent neural networks. *CoRR*, abs/1801.01078, 2018.

[7] A. Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[10] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014.

[11] Q. Wu, P. Wang, C. Shen, A. van den Hengel, and A. R. Dick. Ask me anything: Free-form visual question answering based on knowledge from external sources. *CoRR*, abs/1511.06973, 2015.

[12] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[13] Z. Yang, X. He, J. Gao, L. Deng, and A. J. Smola. Stacked attention networks for image question answering. *CoRR*, abs/1511.02274, 2015.