Visual Question Answering: May I Have Your Attention Please?

Praateek Mahajan
pmahaja2@jhu.edu

Iskandar Atakhodjaev
iatakho1@jhu.edu

Vibhu Jawa
vjawa1@jhu.edu

Abstract

VQA is a new dataset containing open-ended questions about images. These questions require an understanding of vision, language and commonsense knowledge to answer. We experiment with various methods and reproduce them in PyTorch. We try to analyse the different models presented and also our own model which uses attention. In totality we experiment with 7 different models, in a constrained environment of training upto only 10 epochs. We explain the models we tried, and the choices we made while creating our own model. We additionaly release an extremely lightweight yet modular code which can be used to perform more experiments with no time setting up the framework.

1 Introduction

With the emergence of deep networks several problems such as image classification or speech recognition have been trivialized. At the same time, the emerging of many successful deep structures in the field of natural language processing has lead to impressive performance in traditional Question Answering techniques.

However in the past few years there has been an increase in research in Visual Question Answering. The problem can be described as providing an algorithm with an image which is the subject and a question which needs to be answered based on the image.

Prior work has been referred in Section 3.

2 Dataset

VisualQA.org (Goyal et al., 2017) provides us with an enormous dataset comprisiong of nearly $260,000$ images and about 1 million questions with roughly 3 answers per question. We have atleast 3 questions per image. The questions are both open ended and true-false quesitons, such as

- Is there any person in the picture? (Binary)
- Which animal is that? (Open-ended)

Also apart from out in the wild images from MS-COCOA dataset (Goyal et al., 2017) also provide with abstract images, which are mostly like clip art drawings.

We use their dataset for training our model, but with the such a huge dataset comes the problem of computation and storage, which for our project has been the major bottleneck.

2.1 Image preprocessing

For the $260,000$ images we extracted features using a pretrained ResNet-152. Since the resnet is trained on ImageNet it outputs 1000 logits, and we instead decided to use the features extracted second to the last layer. The dimension of these features were 2048 x 14x 14. All these extracted features are saved in a hdf5 file for quick retrieval within disk. We let the features maintain a 3D state, since it will be easier to downsample later, but these features could later be used in Attention modeling.

Inflating our roughly $448 * 448$ images to $2048 * 14 * 14$ resulted in a enormous growth in the dataset, to about `461GB`.

2.2 Question/Answer preprocessing

We pre-process both question and answer before feeding it to the network. For questions we tokenize the sentence into words and map each word to a word id (our vocabulary size is $13,390$). Our questions had a median length of 6 words and maximum length of 22 words. To tackle the padding, we extend PyTorch's dataloader to account for our specific training data, such that every question is padded to the maximum question length as per batch size. Extending PyTorch's DataLoader allows us to :

- **Prefetch the data to the gpu** while one batch is being computed.

- **Pad as per batch and not maximum length** this saves us on average of 16 time steps computation in the LSTM model on average.

For answer we choose only the top 2000 answers which can be used to answer the questions in our training set. Therefore any random model will give an accuracy of only 0.05%, a model would have learned if it performs better than random. Most of our models, in Sec 3 achieve atleast 30% accuracy. After the preprocessing we create a json object for every question answer pair, so that for every question we have the top-k answers by the human annotators, and the corresponding image. While training we sample one of the answers from the top-k answers, so that our model learns to predict not only the most common answer, rather something that even humans often predict. For example, for a question like "Which vehicle is that?", if 6 humans marked it as "Bicycle" and 4 marked it as "Bike", we would want our model to learn that both the answers are acceptable.

When our model predicts any of the answers which atleast three annotators predicted we mark the answer as correct and treat that as the official metric for accuracy.

3 Models

In all the following models, our last layer has 2000 neurons (for each class). We apply log softmax to our last layer, and use Negative Log Likelihood as our loss function.

3.1 Just Using Questions To Answer

We embed our questions using different techniques (embed 13900 words into 300 features) and feed it to an Bidirectional - LSTM (with 1024 hidden) to extract a single feature vector (h_n) of the question. We take h_n and pass it to a multi-layer perceptron followed by last layer described above.

We explore with both glove embedding and a random initialization of embeddings, so as to learn our own embedding. We learned two things from this experiment:

- Learning the embedding performed better even though it was more computationally expensive (30% vs 25% on first 3 epochs). We would attribute this to the fact that Glove embedding which has been trained on Tweets, and the word-word co-occurrence of twitter dataset is relatively why different than our questions.
- The extra computation was worth it as out bottleneck is in data loading not GPU computations.
- We learn that given a complete sentence, it is useful to extract information from both ends of the sentence, rather than just from the start. Therefore it is better to use a bidirectional LSTM than a unidirectional one.

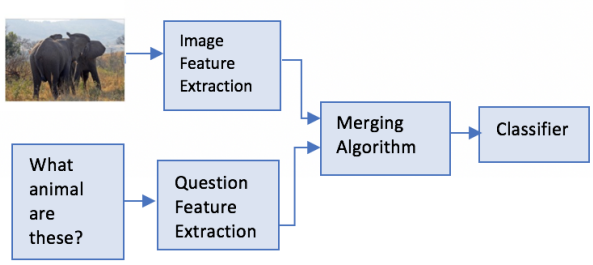

Figure 1: Model using both question and image

3.2 Fusion of Image and Text features

See Figure 1 We extract a feature (He et al., 2015) and down-sample the features from $2048 * 14 * 14$ to 2048. And use the model defined in Section 3.1 to extract text features. However while merging these two features we experiment with the different techniques of merging including 1) stacking 2) adding 3) taking a dot product.

These experiments revealed that taking a dot product gave us the best results in the initial iterations followed by addition, and stacking gave us worst results.

We expect that this happened because the gradient flow will be the most in dot product , while the gradient flow is the same in stacking and addition we believe the linear layer used to down sample provides more learning space to features.

The best accuracy we obtained was 40.9 % using the dot product in the final layer.

3.3 CNN Model to extract text features

We also tried to use CNN to extract features,for each question we first padded them to a maximum length and then used convolution with different strides, stride = 1 for extracting uniqgram features, stride = 2 for extracting bi-gram and stride = 3 for tri-gram. We then do a relu over all of them followed by a max pool layer and then concatenate all of them. This gives us the question embedding

which is then merged to image features using the techniques defined in the above section. This gave us an accuracy of 40.2 percent.

As we have the whole sentence available to use to do prediction, using a convolution model does indeed make sense. They are especially useful if we want to train the model with less parameters as CNN helps us vastly reduce the number of parameters and are heavily parallelizable.

3.4 Our Attention Model

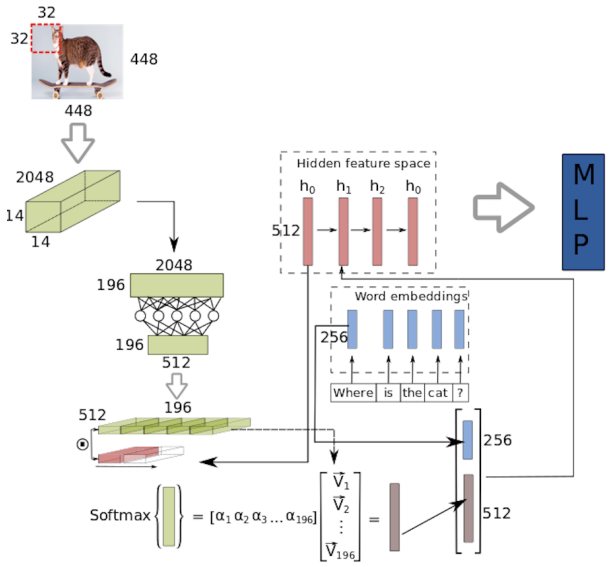

Figure 2: Our Attention Model

Inspired from the results on vanilla models, we try to make a smarter model. This model we developed straight from our intuition using the concept of attention. The intuition from high level is that, for a question "Where is the **man** sitting?", we would like our model to focus more on the patch in the image where the man is. Using that image information, we can encode better hidden states of our RNN.

In this model we extract our image features using ResNet, explained in Section 3.2. We get a feature vector of $2048 * 14 * 14 = 2048 * 196$, for our $448 * 448$ image. Each of $32 * 32$ patch in our image is represented by a 2048 dimensional vector.

We do a max pooling on our $2048 * 196$ image vector to get a 196 dimensional vector. This max-pooling just ensures that for every $32 * 32$ patch, it selects the most important feature among the 2048 channels. We pass these 196 dimensional vector to a fully connected layer to get a vector of 512 dimensions. This 512 dimensional vector acts our intial state h_0 for the GRU. We believe this sets as defining a prior for our GRU.

We take another FC layer to downsize our $2048 * 196$ dimensional image to $512 * 196$ (called img), this is used to attend our hidden states. We use an embedding size of 256 for our words.

For the purpose of the algorithm, we can explain it iteratively :

```
for i = 1 to question_length:
  context = h_{i-1} · img^{512*196}
  α = softmax(context)
  features = Σ_k^196 α_i * img[i]
  vec = features to a FC(512 to 256)
  input = embedding(word[i]) · vec
  out, h_i = rnn(input, h_{i-1})
```

The last h_n is passed to a multilayer perceptron explained in the beginnging of the section, to extract one of the 2000 labels. This model however only gives us 37% accuracy. We learn from this experiment that, while we were giving the RNN an input of a dot prdocut between the word man and the previous hidden state, this doesn't really help us solve the purpose of attending the image where man really is. In fact we could have possibly added the image using the input rather than the previous hidden state. Here we also explored how stacking the attended image compares with taking a dot product, and we realise that dot product results in better results, possibly due to better gradients.

3.5 Stacked Attention Model

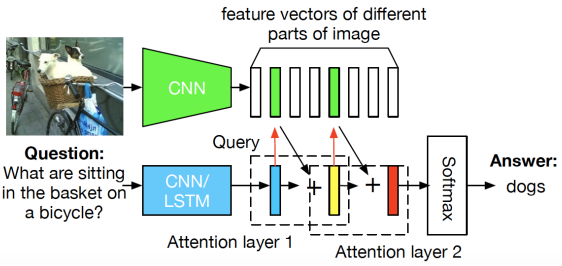

Figure 3: Stacked Attention

We implemented SAN which predicts the answer via multi-step reasoning , presented by (Yang et al., 2016). See figure 3.

In SAN, we take the image feature matrix, $m * d$ (where m is the number of regions and d is the dimensionality of each feature) and calculate the

Table 1: Results for question type

Question Type	Top-1	Top-3	Top-5	Top-7
Number	0.313	0.639	0.774	0.828
Yes/No	0.640	1	1	1
Others	0.324	0.494	0.561	0.600
All	0.452	0.716	0.764	0.789

Table 2: Results for Image Sources

Image Source	Top-1	Top-3	Top-5	Top-7
Abstract	0.535	0.834	0.875	0.892
Mscocoa	0.436	0.694	0.743	0.769
Total	0.452	0.716	0.764	0.789

Table 3: Accuracies of all the models

Model	**Acc**
Only LSTM (glove)	25%
Only LSTM (our embedding)	30%
Fusion (LSTM + CNN)	41%
Fusion (CNN + CNN)	41%
Our Attention Model	37%
Stacked Attention	45%

attention for all the image regions using using the question embedding which we obtained from LSTM. The attention is calculated using the below equations:

$$h_a^k = tanh(W_{I,A}^k v_I \bigoplus (W_{Q,A}^k * u^{k-1} + b_A^k)) \quad (1)$$

$$p_I = softmax(W_P^k h_a^k + b_P^k) \quad (2)$$

We initialize the intital query, (u_0) to be the question embedding obtained from the bi-directional LSTM. We then use the aggregated image feature vector to form a new query vector.

$$\tilde{v_I^k} = \Sigma_i p_i^k * v_i \quad (3)$$

$$u_k = \tilde{v_I^k} + u_{k-1} \quad (4)$$

We then repeat the above operations twice and feed the last layer into a soft max to obtain the prediction.

We obtained the best results using this models which are presented in Table 1 and 2.

The accuracy curve is presented in Figure. 4

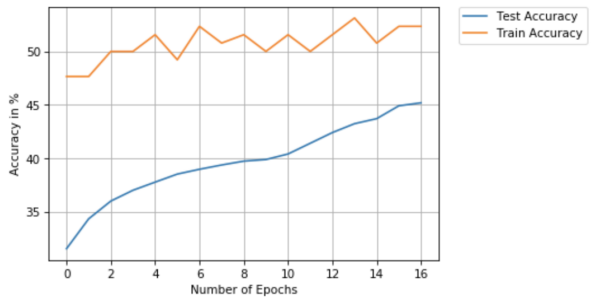

Figure 4: Accuracy Plot

4 Results

We explored 5 major models, and two models with different fusion strategies. Our dataset originally about 150 GB was expanded to nearly 461 GB after extracting features using ResNet. Fetching these features from disk became a bottleneck for us. Even having access to multiple GPUs (thanks to MARCC) our progress/experimentation was slowed down by high I/O cost.

We experimented with the number of workers to fetch the data and concluded that the higher the better as long as we don't run out of CPU memory. We used a batch size of 32 consistently as suggested here (Masters and Luschi, 2018).

We can see that in Table. 2 that our model performs better for Abstract images, which are PNG images drawn using clipart. We would expect this to be the case, because these image have no noise in them compared to the MSCOCOA image, which were captured in the while.

Also we want to keep the code open-source, since our modular code allows people to just plugin different models, embedding as per there choise and run the train set. It can be found at `http://github.com/praateekmahajan/vqa-2018`.

5 Discussion

VQA problem can be solved in many ways and approaches. In this project we implemented several models starting from trivial LSTM model without any image participation and ending up using Stacked Attention Models that is considered one of the state of the art models. Taking into account limited time given for the project, we think we achieved relatively good results. On average one epoch took us about 1.5 hours. We ran all our models only for 10 epochs, which took roughly 15 hours. This prevented us from comparing the per-

formance of different optimizer, hidden layer sizes within the network.

Second, it would be very interesting to analyze "theoretical" performance of VQA models by capturing the accuracies from Object Detection/Classification models and sentence classification models. In other words, we believe that there is a accuracy limit that could be achieved in VQA models and this limit exists due to the separate errors in Image and Language classification models.

Third, during this project we also found another interesting model called Hierarchical Co-Attention model (Lu et al., 2016). The main idea of this method is to join visual attention of the image and question attention to boost the performance of previous approaches. Specifically this method represents the questions in hierarchical structure: a) word level b) phrase level c) sentence level. Each level contributes to feature encodings that are getting more specific from lower level to upper level. Co-attended features from each level are combined with image encodings for final answer prediction.

Also adding a touch of generative models to this VQA, has evolved in the near past. Specifically iVQA (Liu et al., 2018). They use an Auto Encoder to encode both image and question in a latent state, and then decode using an answer key, to generate more such questions. Using iVQA one can imagine, train something like a GAN to make more robust classifiers and generators.

As a summary, this project was both challenging and extremely interesting for our team for several reasons. One of the reason is that we learned such things as data preprocessing of large real world datasets (about 0.5TB). Another useful thing we learned is PyTorch extension modules that enabled us to write custom DataLoader class

References

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Feng Liu, Tao Xiang, Timothy M. Hospedales, Wankou Yang, and Changyin Sun. 2018. Inverse visual question answering: A new benchmark and VQA diagnosis tool. *CoRR*, abs/1803.06936.

Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. 2016. Hierarchical question-image co-attention for visual question answering. *CoRR*, abs/1606.00061.

D. Masters and C. Luschi. 2018. Revisiting Small Batch Training for Deep Neural Networks. *ArXiv e-prints*.

Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. 2016. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–29.