Visual Question-Answering

Natcha Simsiri
nsimsiri@umass.edu

Shiyan Yin
shiyanyin@cs.umass.edu

1 Problem Formulation

The process of building machine learning models to perform high level human tasks will give us insight to tackling harder machine learning tasks. Visual Question-Answering (VQA) is one such task and being able to have a machine perform this feat will give us great insight into how machines can represent and reason with visual-semantic space. The task of VQA is as follows: given an image, and a question that pertains contextual information on the image, have a machine output the answer to the question. The task of VQA can become very complex depending on the dataset (such as free-form QA), we tackle the subset of the problem: single answer VQA task. We treat the problem as a text and image classification problem. We will use the encoder-decoder model, where the "encoder" is a generic term for a multimodal representation of the image and text, and the "decoder" is the module that performs a classification. We will note that. In our problem setup, we will have a finite and same set of question and answer vocabulary across all of our data splits.

We took inspiration in an encoder-decoder model from the state of the art technique in machine translation and image captioning (Xu et al., 2015). 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 $p(T|S, \theta)$. Similarly, for our task, we seek to learn a model θ and maximize $p(S|I, \theta)$ where S is the sentence we would like to generate and is the image. We leverage this framework in performing VQA. The general approach we have is to build an 'encoder' that will learn visual-semantic representations and use this as our feature for a classification task. We work with three models in this paper, a baseline "naive" encoder-decoder model, an image and text "fusion" encoder-decoder and an attention based encoder-decoder.

2 What you proposed vs. what you accomplished

Here is a list of tasks we have set out to accomplished. The ones lined out are accomplished tasks. We were able to accomplish all report required tasks, however, the final unperformed task was much more ambitious than we sought out for. If we had more time or perform an extension to this paper, we would look into visualizing the soft attentions as it provides further analysis for the visuals-semantic embeddings learned.

- ~~Collect and preprocess dataset~~
- ~~Build and train the encoder-decoder/encoder-classification baseline on various RNN types on collected dataset and examine its performance~~
- ~~Build and train an insightful but quick improvement to the baseline (fuse) and a fancy model (attention model)~~
- ~~Make the attention model perform better than baseline model~~
- ~~Perform error analysis to figure out what kinds of examples our approach struggles with~~
- *Ambitious Goal: Perform error analysis by visualizing attention weights. We did not complete this task as we ran out of time. The entire experiment pipeline and debugging took more time than we thought we could invest our time in*

3 Related work

The works we explore in Visual Question-Answering (VQA) will mostly be of the deep-learning root. The task itself has multiple variants, for instance, a few papers implemented models solely using the VQA dataset such as DAQUAR, COCO-QA and VQA. However, some other papers used external resources to build a knowledge base and will discuss this first.

VQA is not only limited to the single-token classification tasks (what we are doing), but richer generative approaches that uses external databases to predict open-ended questions. In "Ask Me Anything: Free-form Visual Question Answering Based on Knowledge from External Sources" (Wu et al., 2015) attempts to build a deep learning models 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. 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. In this work, the encoder-decoder LSTM was trained to learn the previously stated visual-semantic embedding between the images and captions, and then is trained on the open-ended questions, images, and external knowledge base, DBpedia which uses information from Wikipedia (Auer et al., 2007). Another recent trend 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 (Andreas et al., 2015) uses decompositional neural networks to learn atomic representation on specific question tasks, or operations, and allows construction of free-form results from the building the question specific network on the fly. One drawback is the computational resource to build these networks and train them on a small scale setting, as their core network themselves learn a two fold representation - the atomic construction of neural networks and having the network learn VQA itself.

While various works tackle VQA from different creative standpoints, recent work leverage state of the art deep learning results to try to tackle the same dataset we are working on. The first works to use both CNNs and RNNs to perform VQA was surfaced by Gao et al (Gao et al., 2015) and Malinowski et al (Malinowski et al., 2015) where both uses Recurrent Neural Networks (RNN) to encode question and use the final hidden representation as input to a classifier. In Gao et al, the classifier itself is a decoder RNN and the task is not end to end - they separately train the encoder model while using using the test-time output embedding as input to the classifier. Another work done by Ren et al. (Ren et al., 2015) exclusively tackles a similar task to us - a classification of singular answer VQA task. They use the an image feature from the VGG19 network, downsize through a linear layer and feed the embedding into an LSTM followed by the sequence of questions to perform a many-to-one prediction. Similarly, (Ma et al., 2015) also performs the same process except instead of using LSTM to create a text embedding, they use a CNN to do so.

In recent years, many works have produced a strong baseline for VQA and especially the newly class-balanced VQA2.0. The work of Stacked Attention Networks for Image Question Answering (Yang et al., 2015) produces the state of the art baseline for VQA2.0. They use multiple attention on the CNN (VGG-16 features) to obtain finer-grain visual information for better image embedding. Then, the quesiton is encoded multiple folds on a CNN to obtain various n-grams of the question which is then embedded and fed to an LSTM network. At each time step soft attention is computed over image regions and a resulting context vector is produced and given to the next timestep. This model differs to ours in that we do not perform time-step level attention, but use the attention at the end for each timestep.

Inspired by the usage of stacked attention, a recent contender for the state-of-the-art is the work from (Kazemi and Elqursh, 2017), where they use both an LSTM to encode the question and a CNN to encode the image, and perform an attention based concatenation operation to obtain an answer-ready embedding to which they feed through an linear classifier. The attention based concatenation operation is essentially a concatenation of the a ResNet152 image feature on an upscaled LSTM hidden units, which is then passed through multiple CNN layers to achieve an attention map - where the LSTM hidden states are reused on this attention map to obtain the final net-

work embedding.

Finally, as concatenation of the LSTM hidden units and image feature technique seems to give promising results with the previous papers, in the work done by (Fukui et al., 2016), the two stated embeddings are concatenated using a multimodal pooling operation. The operation can be thought of as an outer product on image feature (which can be a tensor) and a bilinearly-upscaled LSTM hidden unit. The spatial attention built using this technique allows a the model to perform on par with the previous two papers at the high computation cost. We do follow the general architecture of multimodal concatenation (we call it fusion), but we do not use anything resembling an outer-product on tensors.

4 Dataset

We will use the existing VQA 2.0 dataset from the official website https://visualqa.org/download.html. The text data we will use is the question and annotation for each image. We will not annotate the text because all the data is clearly labeled. VQA is an activate challenge and their dataset is based on the coco-dataset, so the dataset is reliable and easy to get. Furthermore, this dataset also seems like a good benchmark for us as the other state of the art also uses this dataset as their benchmark. The VQA 2.0 also guarantees that the classes are balanced. This dataset contains 82,783 training images, 40,504 validation images and 81,434 testing images. On the NLP side, VQA 2.0 comes with 4,437,570 training and 2,143,540 answers and 443,757 training, 214,354 validation and 447,793 testing questions, for each question there are up to 10 answers.

Multiple papers have stated that training this entire dataset may take up over a day, and for feasibility purposes of debugging our model, and accomplishing this report, we only work with the validation dataset, but refine our own train-test split. We picked approximately 5,000 questions to be our dataset from the VQA2.0 validation split, and had approximately 90-10 split for the train and validation set.

As shown in Figure 3, we see that the majority of the question has some form singular answer - such as a count, boolean value, or an object identification. In our code, we run some tokenization and filtration schemes to guarantee that our only


```
{
    "answer_type": "other",
    "answers": [
        {
            "answer": "spectating",
            "answer_confidence": "yes",
            "answer_id": 1
        },
        {
            "answer": "watching",
            "answer_confidence": "yes",
            "answer_id": 2
        },
        ….
    ],
    "image_id": 262148,
    "multiple_choice_answer": "watching",
    "question_id": 262148001,
    "question_type": "what are the"
}
```


Figure 1: Example annotation data-point. An annotation has a 1-1 mapping to a question json (not shown) that is queried by the "question_id" field, and the corresponding image queried by the "image_id". The dataset is also partitioned into question types with different prefixes as noted in "question_type" field - it is useful to note that we only work a limited question type form. The question corresponding to this set of answer or annotation is "**What are the people in the background doing?**"

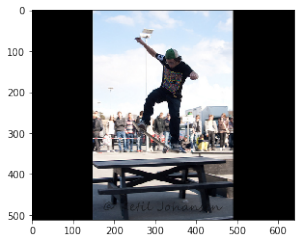


Figure 2: The corresponding image to Figure 1 with the question "**What are the people in the background doing?**"

filter only produces singular answer and all words are lower-cased with punctuation separated.

4.1 Data Preprocessing

We use NLTK to tokenize questions to 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. As a result we have an encoded question using the question vocabulary and an encoded answer using the answers vocabulary. Once again, we filter out answers to only annotations with 1 answer token.

We implemented a $torch.utils.data.DataSet$ subclass to allow simple shuffled, minibatch iteration of our dataset. Since we only use a subset of the dataset, we also would like to maintain the

	0	1		0	1
0	how many	20462	0	are the	5264
1	is the	17265	1	is there a	4679
2	what	15897	2	what type of	4040
3	what color is the	14061	3	where is the	3716
4	what is the	11353	4	is it	3566
5	none of the above	8550	5	what are the	3282
6	is this	7841	6	does the	3183
7	is this a	7492	7	is	3169
8	what is	6328	8	is there	3120
9	what kind of	5840	9	what color are the	3118

Figure 3: Top 20 most question types in our splits and their counts. This statistics is for 20K questions.

class balance across the splits. When building the DataSet class, we made sure the class distribution are the same across train/test splits. 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 224 by 224 and normalized the image.

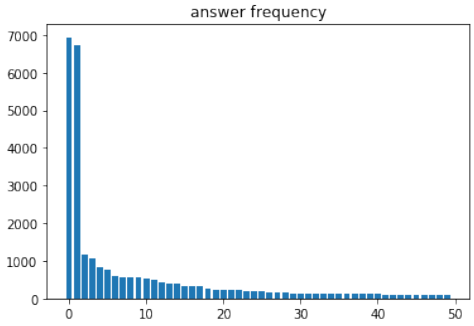


Figure 4: Distribution of labeled data on our DataSet sampler. The Y-axis indicate the counts for each class and the X-axi are each classes (answers). The far left tall bars indicate yes/no questions, followed by numbers, colors and objects. Our dataset splits train/test so that if the train distribution is as of shown, the test distribution will be the same (but lower frequency on all classes.

5 Baselines

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:

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

5.1 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 (Vinyals et al., 2014). The way our encoder-decoder variant works for VQA is we first pass an image through a pre-trained CNN (Convolution Neural Network) and retrieve the previous to last layer which contains high level semantic related features which are of size 1 x 1 x 2048. CNNS have shown to be a very promising deep learning method that learns image features and is able to represent them into a fixed length vector. Works on these deep CNNs that are widely used are Oxford VGGNet(Simonyan and Zisserman, 2014) or the GoogleNet (Szegedy et al., 2015). We first freeze the pretrained-CNN layers as we do not want to fine-tune the CNN, and from here, given and image, we will be able to obtain the image features. We then feed these feature into a 2 dimensional fully connected layer that that is used to learn a representation for mapping from the image feature to the RNN size 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

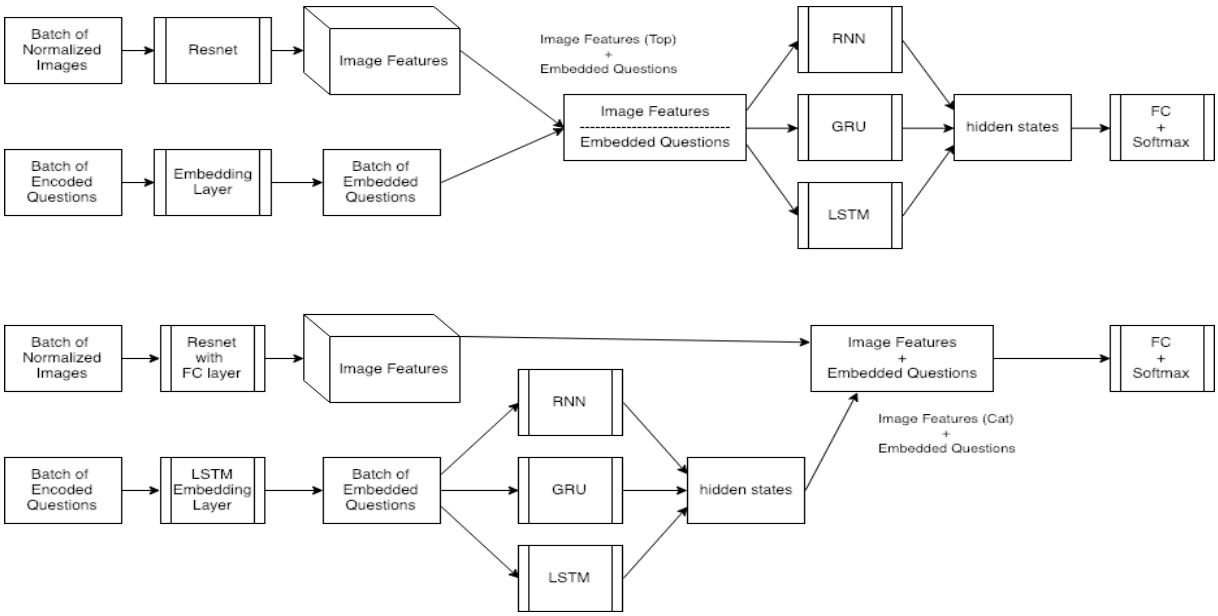


Figure 5: Architecture of Vanilla Encoder-Decoder (baseline) and Fusion Encoder-Decoder

Model	Accuracy	Model	Accuracy
Attention_128_512_256	0.276	Fusion_rnn_0.001_256_256	0.356
Attention_256_512_256	0.316	Fusion_rnn_0.001_512_512	0.333
		Fusion_rnn_0.01_256_256	0.327
		Fusion_rnn_0.01_512_512	0.245
Attention_128_512_512	0.386	Naive_gru_0.001_256_256	0.376
Attention_128_512_512	0.387	Naive_gru_0.001_512_512	0.366
Attention_256_1024_512	0.405	Naive_gru_0.01_256_256	0.367
Attention_256_1024_512	0.395	Naive_gru_0.01_512_512	0.356
Fusion_gru_0.001_128_128	0.373	Naive_lstm_0.001_256_256	0.386
Fusion_gru_0.001_256_256	0.406	Naive_lstm_0.001_512_512	0.364
Fusion_gru_0.01_128_128	0.365	Naive_lstm_0.01_256_256	0.370
Fusion_gru_0.01_256_256	0.385	Naive_lstm_0.01_512_512	0.354
Fusion_lstm_0.001_256_256	0.370	Naive_rnn_0.001_256_256	0.305
Fusion_lstm_0.001_512_512	0.393	Naive_rnn_0.001_512_512	0.309
Fusion_lstm_0.01_256_256	0.383	Naive_rnn_0.01_256_256	0.245
Fusion_lstm_0.01_512_512	0.386	Naive_rnn_0.01_512_512	0.256

Figure 6: Validation accuracy of all our models trained on various hyper-parameters

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. In general, the encoder-decoder naming scheme is unlike the conventional sequence to sequence "encoder-decoder", but in our case, the "decoder" is simply a fully-connected layer followed by a softmax with the input given from the encoder.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 (Sherstinsky, 2018). 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

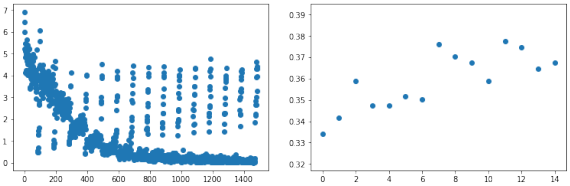
used to where gradients are propagated through the hidden states the update weights in the RNN cell. One issue from this is that the gradient may vanish for long sequences, and the model may not the features from the earlier words. This problem is crucial for us since the first few words indicate the 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 learn how much 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 (Hochreiter and Schmidhuber, 1997). The GRU, on the other hand, is a variant of LSTM but with an update gate and a reset gate which similar to the LSTM, tells the network how much of the past information needs to be passed along (Chung et al., 2014).

5.2 Baseline Evaluation

We trained our base line on 4,000 image, question and answer pairs, validated our trained model on 1,000 pairs and tested the best model on 600 pairs. We experimented with three hyperparameters: the learning rate, number of hidden units, number of embeddings. We trained all combinations of the following hyperparameter numbers, $rnn\_type = \{'LSTM', 'GRU', 'RNN'\}$, $embed\_size = \{256, 512\}$, $hidden\_size = \{256, 512\}$, $learning\_rate = \{0.01, 0.001\}$. We have found that the parameter set $rnn\_type = 'LSTM'$, $embed\_size = \{256\}$, $hidden\_size = \{256\}$, $learning\_rate = \{0.001\}$, works best for us with a validation accuracy score of 0.386. Table 6 displays all the models' We chose this model because not only is it the most intuitive model to perform a variable length input classification task, it is also widely used in various research work as a baseline.

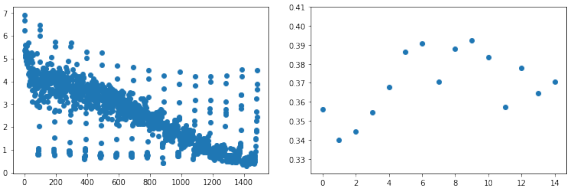
We trained for approximately 15 epochs with a batch-size of 50 using the Adam-optimizer with a cross-entropy loss function. Each epoch takes approximately 3-4 minutes to train, totalling to approximately 1 hour to train a model. In our training code, we also log the loss and batch accuracy, and by doing so helps us debug the model. One caveat we came across was, as previously stated, to find which hidden state works best. Since each batch has variable length sentences, we took advantage of PyTorch's built-in $nn.PackedSequence$ to not only help us mask the correct hidden unit to pass to the classification layer, but also mask allow a cleaner gradient flow (without having to pass gradients through our padding).

Additionally, we also tried some different hyperparameters for fun. We used $embed\_size = \{300\}$, $hidden\_size = \{1024\}$, $learning\_rate = \{0.001\}$ over GRU and LSTM. Below are the results we got in the experiments including both the training loss over all the training samples and the test accuracy over each epoch. We think both of them achieve good accuracy with these hyperparameters.



(a) Training loss (b) Test accuracy

Figure 7: Naive GRU, Best accuracy 37% at epoch 11



(a) Training loss (b) Test accuracy

Figure 8: Naive LSTM, Best accuracy 39% at epoch 9

6 Approach

We built upon the baseline encoder-decoder model to build a fuse encode-decoder and a variant of that in which uses attention.

6.1 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.

6.2 Fusion Encoder-Decoder with Attention

Our final contribution is to apply attention mechanism to a joint version of the vanilla and fusion encoder-decoder model. In our baseline model (CNN + LSTM), we combined the image feature with the question feature, so the model actually observes the whole image for each word in every time step in the decoding process. The same problem in the fusion model, as it observes the whole image in the classification layer. Based on the results and the intuition, this is not a wise way to achieve the goal. In a sentence, the information behind a word usually represents only some part of the image rather than the whole image. For example, if we have an image of a dog at the center of the image and the question is whats the color of this dog, then when the model focuses on the word dog, we want the model can focus the area where the dog is, rather than the feature from the whole image. Thus with the baseline model, it cant focus on the most informative area in the image, which brings us the motivation of the attention model.

This attention 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 embedding function to embed the questions. For the attention layer, we leverage the idea from (Kazemi and Elqursh, 2017) 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).

In our VQA project, we will take advantage of the output from the last CNN layer, the final feature maps. For example, if the dimension of the feature map is H x H x C where H means the height and width of each feature and C represents the number of channels. This means we extracted C low-level features from the image and it contains the spatial information because of the theory of CNN. At the same time, we use the language model to get the features of the question. To combine these 2 features, we apply another convolutional layer on the feature maps to transform the number of channels to the attention_feature_size - a hyperparameter we set for our attention layer. Similarly, we apply another linear layer to transform the dimension of the question features to the attention_feature_size. Next, we tile the question features and add these 2 features directly followed by a Relu activation function to get the combination. Then, we apply a convolutional layer over the combination to get our attention content - glimpses (Kazemi and Elqursh, 2017). We use glimpses to transform the feature combination to multiple channels for computing the attention weights. This is a novel way to strengthen the performance of attention.

We apply some reshape operations on the attention content and use a softmax to get its weights over channels. Next, we multiply the weights with the original feature maps and take the weighted sum on it. Now, we finally get the attention-based image features. The rest works are straightforward. We concatenate the attention-based image features and the question features form the language model, and feed the combination into our classifier which contains several linear layers to transform the dimension size to the number of answers for final classification.

We successfully implemented the attention model in our attention.py file, as well as the baseline model in the naive.py file. We used functions from Pytorch majorly to implement them. From the result, attention performs better than the naive model, which satisfy our expectation. We found an existing VQA project implemented by Pytorch (https://github.com/Cadene/vqa.pytorch) and we referenced it to learn how the attention model works.

6.3 Approach evaluation

Our experiment set-up for both the fusion encoder-decoder model and the attention model is the same as previously stated in the baseline evaluation (Section 5.2. We used PyTorch (Paszke et al., 2017) and ran our models on a desktop GPU machine with a 1080TI and gypsum's titanx clusters with the argument "srun –partition titanx-short –gres=gpu:1 –mem=8000 ./cluster_scripts/run_jupyter.sh" which will start a jupyter server where all the experiment code. The only difference is for the attention model, we have an extra $attention\_size$ hyper-parameter indicating the attention weights sizes. The attention also has different hyperparameter ranges with $embed\_size = \{128, 256\}$ and $hidden\_size = \{256, 1024\}$. We ran both the attention model and the fusion model on a 4000-1000-600 split (same as baseline) and we have seen the attention model outperforming other models in in terms of validation accuracy. Due to the long training time on the attention model (approximately 6-7 minutes per epoch), we did not have the time and resources to perform rigorous grid search on the hyper-parameters and we did a greedy search method where if a certain change in hyperparameter is giving bad result, we do not explore further. For instance, in Table 6, by lowering attention size to 256, we saw a drastic model underperformance on the validation set. In the end, the attention and fusion model was able to out-perform the baseline as shown in the Figure 9.

Model	Accuracy
Baseline	0.362
Fusion	0.382
Attention	0.399

Figure 9: Our final result on running all models with the best hyper-parameters from Figure 6.

7 Error analysis

What kinds of inputs do your baselines fail at? What about your approach? Are there any semantic or syntactic commonalities between these difficult examples? We would like to see a manual error analysis (e.g., annotate around 100 failed examples for various properties).

Figure 10 shows approximately 50 sentences for both the best baseline model and attention model, giving us a total of 100 error cases. We will first note that the majority of the answer types in this case are either a number, color or a yes or no question. It also is imperative in which we state that while we see very few errors on the yes/no type of questions, it accounts for approximately 40% of the total error given in the 600 question test-set.

7.1 Case of attention learning better image-text representation

While we see that both the baseline and attention model performs poorly in many simple questions, such as color or counting questions - which contributes to more than half of this error analysis table (Figure 10), there are some subtle answers that the attention model makes that resembles some learning of a visual-semantic representation. For instance in the question "Where is the cat sitting on", "What is the knife in?" and "What is the pizza in", we see both answer incorrectly, but the attention model is able to capture answers from objects in the image. In the first picture about the cat, the attention model predicts cat itself, while there is no beach spotted for the base model. In the second case, "What is the knife in?", there is a picture of a book, but no banana - however the attention model is able to predict the correct object depicted in the image despite not being the correct answer. We believe the attention model is able to learn an underlying task in theme in this dataset in that the dataset is mostly about identifying what is in the picture.

7.2 Case of color recognition

We also believe, despite the the attention model's (and the baseline's) inability to predict the correct color, manual observation made on the kinds of color predicted by the attention model shows the attention model learns better color mapping, for instance, regardless of what the ground-truth's color is, the baseline model seems to be predicting the same colors - white predominantly, while the attention model seems to be predicting more varying colors. One explanation is after manually looking at the pictures, there exists areas of the image where the attention's model prediction holds - but it is not what the question is asking about.

7.3 Reflection and Future Work

Despite the incorrect predictions, one general point we see here is the majority of the time, both model learns to predict the correct types of answer.

Questions	L	Base	Att	Questions	L	Base	Att
where is the cat sitting ?	table	beach	cat	what colors are this bus ?	green	white	people
is the water clear ?	yes	no	no	what is this bottle ?	wine	cow	wine
if he hits the ball where should he run ?	yes	yes	baseball	how many giraffes are in this photo ?	2	1	2
how many people are here ?	7	2	2	how many giraffes in the horizon ?	2	1	2
how many bananas are in the photo ?	7	1	1	how many skis ?	2	1	2
what color is his shirt ?	grey	red	grey	what color is tall tower that is in the background ?	brown	white	white
what color is this man wearing ?	grey	red	grey				
what color shirt is the person to the farthest right wearing ?	white	brown	white	what is the main color in this picture ?	brown	white	black
what are the color of the shirts in the image ?	white	baseball	white	what color does the building look ?	brown	white	red
what color is the first train car ?	white	red	yellow	is this a man or woman ?	woman	man	female
what is the man wearing on his head ?	hat	skateboarding	hat	what are the umbrellas hanging from ?	trees	brocoli	brocoli
what color is the dog ?	black	white	white	what is missing on this toilet ?	nothing	hat	toilet
what color are the words on the sign ?	black	left	red	what is the knife in ? (there is a book)	nothing	banana	book
what kind of saddle does the horse have on ?	black	none	white	what color is the watch face ?	blue	white	brown
what number of these teddy bears are brown ?	3	brown	15	what color is her hair ?	blue	white	white
how many people are boarding the bus ?	3	2	2	what color is the sky ?	blue	white	red
how many women are in the photo ?	3	2	2	what is the boy doing ? (there is sitting)	skating	sitting	skating
how many women are there ?	3	2	2	where is the motorcycle parked ?	street	ground	nothing
how many shadows of people are there ?	3	2	2	what is the pizza in ? (no banana, there is fork)	plate	Banana	fork
				what is the woman doing with the cell phone ?	Talking	Walking	Nothing

Figure 10: Manual observation on question, answer and predicted output pairs

For instance, if the question is "what color", the model will predict a color, if the model asks "how many", the model will predict a number. Thus at a high level - both model learns a representation of some form of clustering of the questions' prefix to the answer. What we think would be a good improvement to the task of visual-question answering is if we can apply multi-task learning on the question post-fix given a specific task queried on the question's prefix (imagine having an N-way FC layer on each types of questions with the post-fix being passed through the entire network, and routing through to the correct FC head for the correct type of question).

8 Contributions of group members

The two members in our team collaborated in equal efforts.

- member 1 - Natcha Simsiri: Did data collection, data visualization and pre-processing, built the experiment framework and collaborated in building each models. Ran experiments on Gypsum clusters and performed error analysis.

- member 2 - Shiyan Yin: Researched, experimented and built the attention model. Collaborated in working on the baseline, fusion network and experiment process. Did initial model unit-testing on Desktop machine.

9 Conclusion

We presented multiple encoder-decoder neural networks that are able to perform similar to reported accuracy on a similar baseline (Ren et al., 2015). Not only that, but our contribution with attention was able to achieve a higher score than the baseline and gave insightful information on VQA as a whole. We learned a lot about working with the PyTorch library, and especially their RNN classes which were harder to use than we thought. We would look into a more complex time-state level attention models as mentioned in (Yang et al., 2015) or Multi-Task learning models based on question prefixes.

References

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2015). Deep compositional question answering with neural module networks. *CoRR*, abs/1511.02799.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg. Springer-Verlag.

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

Fukui, A., Park, D. H., Yang, D., Rohrbach, A., Darrell, T., and Rohrbach, M. (2016). Multimodal compact bilinear pooling for visual question answering and visual grounding. *CoRR*, abs/1606.01847.

Gao, H., Mao, J., Zhou, J., Huang, Z., Wang, L., and Xu, W. (2015). Are you talking to a machine? dataset and methods for multilingual image question answering. *CoRR*, abs/1505.05612.

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

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

Ma, L., Lu, Z., and Li, H. (2015). Learning to answer questions from image using convolutional neural network. *CoRR*, abs/1506.00333.

Malinowski, M., Rohrbach, M., and Fritz, M. (2015). Ask your neurons: A neural-based approach to answering questions about images. *CoRR*, abs/1505.01121.

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

Ren, M., Kiros, R., and Zemel, R. S. (2015). Image question answering: A visual semantic embedding model and a new dataset. *CoRR*, abs/1505.02074.

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

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

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

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

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

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

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