Implementing Show And Tell: A Neural Image Caption Generator

Patrick Williams
patrickwilliams@wustl.edu

Abstract

Image Captioning is a very important and unique task in Computer Vision that mixes image vision with natural language processing. Show And Tell: A Neural Image Caption Generator is able to combine these unique tasks by using convolutional neural networks trained for object classification to create a feature vector of images and a recurrent neural networks to learn from captions attached to images in order to learn how to write a caption for an image. At the time of the paper, Show And Tell introduced a new concept in image captioning using neural networks and received state of the art results. This paper implements the neural network used in Show And Tell and analyzes the results.

1 Introduction

There are many different tasks in the field of computer vision. One of the harder ones is the ability to take an image and form a description for that image. The ability to caption an image with software is an incredibly useful tasks and can have different applications. It can be used to narrate to a blind person what is in the image or describe to a child what an image contains. At a higher level, being able to automatically caption images would have very useful uses. For example, if an image captioning network were to be trained to learn the names of historical figures and events it could be used to spread knowledge not known by the average person looking at an image.

The ability to caption an image is not easy though. Having a network understand what is really happening inside of an image and the significance of an image cannot be easily modeled with a program. First you need to be able to identify exactly what is happening in the image, which is a very difficult computer vision task in itself that receives a lot of research. After you have identified what is happening in an image, you then need to be able to describe it. There is a gap there between the knowledge of what is in an image and being able to express semantic knowledge in natural language.

The neural network, Show and Tell, attempts to address this problem by leveraging different neural network architectures to join the problem of extracting features from an image and transferring that information to natural language. At the time Show and Tell was written, the popularity of deep learning was starting to rise. It had been less than three years since the publication of AlexNet [8], which signifies the start of convolutional neural networks being successfully applied to images for object classification. Since then, convolutional neural networks have taken off with a broad range of different techniques that can be used to apply them to improve different vision tasks by allowing the network to find the best solution itself through back-propagation.

Show and Tell uses these object classification neural networks in order to create a feature vector that represents the image. By removing the last softmax layer used to assign probabilities to classes from an object classification convolutional neural network, you can take the last fully-connected layer as an image feature layer. By then using recurrent neural network architectures, that have the ability to remember by feeding the last timestep output into the current timestep and using gates to decide what to remember, you can concatenate that image feature vector with one hot-word encoded vectors of vocabulary in order to transfer the knowledge of what the image contains in the image feature vector into words. The recurrent neural network accomplishes this by outputting the next predicted word in the sequence and back-propagating the loss from a one hot-word encoded ground truth vector that comes from a dataset with images that have descriptions. The end goal is to produce the best sequence of words S for the sentence where the probability of a sentence is...

$$p = p(S_1|Image)p(S_2|Image; S_1)p(S_3|Image; S_1, S_2)... \\ p(S_t|Image; S_1...S_{t-1}) \quad (1)$$

where S_t is the "END" character signifying the end of a sentence.

2 Background & Related Work

The idea of image captioning is a growing idea within the field of computer vision. There has not been a ton of work done to produce image captions for photos using computer vision techniques, most likely because it is a very difficult problem and there has not been many good methods for

this topic in the past. With the growth of neural networks which can be custom made for different image tasks and self-learn through back-propogation on datasets, more is possible in the field of computer vision. Most previous work related to this topic has been done with making captions to describe video data. These often are made up of complex graphs of logic systems which are then converted to text through rule-based systems [2]. There has been some work into image captioning. Farhadi et al. use object classifications of triplets and convert it to text using templates [3]. Li et al. uses object detection with similarities between detected objects to form a sentence through their relationships [4].

More similar to the work of image captioning with neural networks, there has been work done to concatenate image vectors and words together in the same vector space, but not for the purpose of image descriptions [5]. Kiros et al. does a similar process by training a feed-forward neural network to predict the next word given an image and previous words [6]. The difference being this paper uses recurrent neural networks which are better for sequencing data. Mao et al. used a recurrent neural network for the same process [7], but Mao used a more basic recurrent neural network architecture and they do not feed the image feature vector directly to the recurrent neural network.

3 Method

3.1 Convolutional Neural Network

The first part of the image captioning neural network is the convolutional neural network. The convolutional neural network used for image captioning actually is not trained originally for image captioning. To implement the convolutional neural network, you want to train a convolutional neural network for object classification on a diverse image classification dataset. The most common one used would be ImageNet since there are 1,000 different categories. Training a neural network for object classification works well for our problem because we can remove the final softmax layer in the neural network and take the final fully-connected layer as a feature vector to represent the image. When we describe an image, we are looking for different features in the image to be able to caption that image. Object classification in the image works in a similar way where we need to be able to extract all of the important features within an image in order to classify the image in different ways. Due to the similar nature of describing an image problem with object classification and image captioning, a neural network trained for object classification on images works well to create a good feature vector to describe the features of an image.

3.2 Word Embedding

We want to be able to take our feature vector from our convolutional neural network and a partial caption of all the words so far and concatenate them together to put into a recurrent neural network to predict the next word in the sequence. Before we can do so however, we need to move the words into the same vector space as the image feature. To do so, we want to take our DxCharacters hot-word encoded partial caption (where D = vocabulary size) and matrix multiply it by a FxD matrix. Therefore, we have the matrix multiplication of (F x D) x (D x Characters) = (F x Characters) matrix. Our FxD matrix that moves the partial caption into the feature vector space will be learned during training our model architecture through back-propogation. We then concatenate our Fx1 image feature vector by our FxCharacters partial caption together to input a Fx(Characters+1) into our recurrent neural network where (Characters+1) is the timestep.

3.3 Recurrent Neural Network

After you have concatenated the image feature vector with the partial caption word embedded vector, you can now feed the input into your recurrent neural network architectures. Recurrent neural networks are a type of neural network that are good for handling sequencing data. In Long-Term-Short-Term-Memory (LSTM) networks, they contain two memory vectors m_t and c_t. m_t is a vector representing what the recurrent neural network remembers from previous timesteps. Timesteps determine how many times the recurrent neural network feeds into itself and it is a dynamic variable that can be changed without altering the network architecture. Timesteps are common to all recurrent neural network architectures. c_t is a vector with numbers between -1 and 1 and is multiplied by the new output to update m_t so that it can decide what to remember. LSTM contains input gates, forget gates, and output gates. In GRU's, there are no cell states. GRU's contain a hidden state (memory input to next GRU in sequence) to transfer information and gates. The gates are a reset gate to decide how much information to forget and an update gate that decides what information to add. The final timestep in the recurrent neural network sequence will output the new word prediction for the next word to come in the caption. At recurrent neural network output, you should output a vector of vocabulary size and put the final output through a softmax layer.

Both LSTM's and GRU's are popular recurrent neural network architectures that are used and perform well. While the LSTM was used in the original paper, I decided to use GRU's in my implementation due to them being quicker to train.

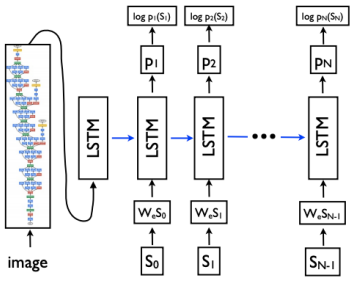

Figure 1: Show And Tell Architecture

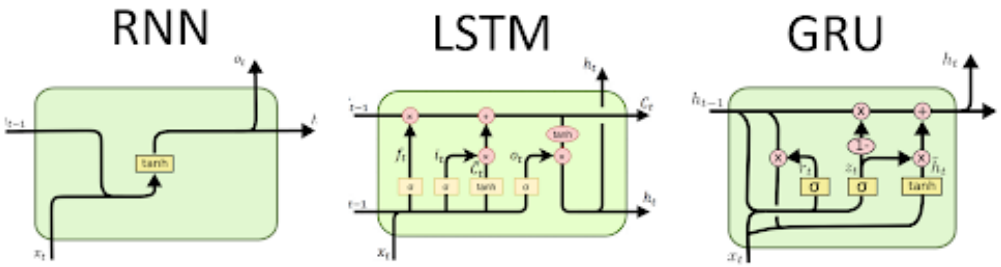

Figure 2: Different Recurrent Neural Network Architectures

3.4 Training Procedure

For training, you want to train on each sentence and use categorical cross entropy with the ground truth being a one word hot encoded vector for each correct word in the training caption sequence. It is possible to train a recurrent neural network on an entire sentence at a time by returning the whole sequence or just a single word by returning only the final timestep output. For memory purposes I trained the network on a single word at a time so I did not produce a massive tensor. It is also possible to train in batches by concatenating sentences together in the batch size domain. Training in batches is beneficial for training because it allows you to follow a more direct path to a local minima of the neural network architecture whereas there will be more noise in whether you back-propagate into the correct direction with a smaller batch size. While training sentences in batches would be beneficial, I also elected not to pursue this route as I was close to the memory limits on my GPU.

3.5 Testing Procedure

For testing, you do not know the true word so a slightly different procedure must be taken. After inputting the image feature vector and START word, you want to predict the next word that comes into the sequence until you reach a STOP word. However, since each choice for the best predicted word is a local maximization given the partial caption thus far, there is no guarantee choosing the best predicted word at every step will give you the best sentence. Instead, you want to keep the top-N partial captions as you continue to expand until you reach STOP words. N can be any number greater than or equal to one. The larger N is, the more confident you can be that you chose the best possible sentence but the longer it takes to create a sentence. For my testing, I chose N = 10.

4 Experimental Results

For my experimental results, I trained my network on a flickr30k dataset containing images with captions. For my convolutional neural network, I loaded in a VGG-19 [9] network pre-trained on ImageNet. I removed the softmax layer of this network and took the final fully-connected layer to be my feature vector from this network. The feature vector was 1000 long. I then joined this network with my embedded partial caption to predict the next word of the sequence and trained my network architecture on the first 10,000 images of the flickr30k dataset. I built my network architecture using Keras.

For original testing, I trained my network in Google Colaboratory on the first 200 images of the flickr30K dataset. My vocabulary was 606 words. This model was able to learn some features of pictures and make a sentence, but the description was often vague, uninteresting, and sometimes wrong.

After building confidence in my training procedure, I switched my training to a GPU on Google Cloud. After setting up my environment, I planned to train on all 30,000 flickr30k images, but after hours of downloading the zip file to a Google Cloud bucket I was unable to successfully unzip the 8.5GB file with unzip or 7z x on the vm instance. Instead, I successfully loaded in the first 10,000 training images to train. Originally, the vocabulary of my captions were 7,800. Many of these words are hardly used and cannot be learnt well by the model, so I removed any words not seen more than ten times in my training dataset. After this procedure, the vocabulary of my dataset shrunk down to 1,225 words before adding the START and STOP words to signify the beginning and end of sentences. After some testing, I settled on using an Adam optimizer and a learning rate = 0.0001. For training, I would shuffle the data images and sentences by shuffling an indexing array after every epochs.

The training process was very interesting to watch. At first, the neural network would spit out sequences of random words. The network then learned "a" was a very common word, so it would spit out the word "a" at every word to try and reduce its cost. The network then learned that the work "in" was also very common, so it would spit out a sequence of a's and in's. Every once and a while you would see sentences of new words from the network. This was a sign that the network was starting to learn some features from the images. Eventually, the sentences started to look like real sentences.

For some reason, however, the training appeared to hit a wall where it would no longer improve. The network trained for 10 hours but did not see any real improvements after the first 4 hours. I believe that there are a few different reasons that could explain this. The primary reason I think is that in my training procedure if a word was not part of the models vocabulary, I skipped over that word. I believe that this strategy led to my model not being able to fully understand sentence grammar. I saw a trend throughout training where sentences would end with "some.". The word "some" did not make sense to be the last words of these sentences and is usually followed by a noun. Since I removed words that were not seen more than 10 times, unique nouns would be skipped over and my model would think that the word some could grammatically be followed by a period. While removing these words from the vocabulary was necessary to stay within the memory constraints for training my model so I would not make a tensor too large, it seemed to severely limit my model. A better strategy was needed to deal with this problem. Something that could have helped my model was if I had more training data. That would have both helped my model learn all the different words and features better as well as allow my models vocabulary to grow so I did not have to remove as many words. Another training step that could have helped was to put my sentences in batches. At one point I started to print out the sentence from a single image during training and saw the meaning behind the output sentence changing far too quickly. The differences mean my model was not doing a good job heading towards a local minima and instead bouncing around. I tried to severely decrease the learning rate but it did not prove to help much. Putting sentences into batches could have helped to go towards a local minima. Overall, my model appeared to learn some features. The captions on the test images appear to describe some actual details of the scenes and look like real sentences. Much is left to be desired however as the training procedures could have used some improvements.

5 Conclusion

details of the scenes and look like real sentences. Much is left to be desired however as the training procedures could have used some improvements.

Training a neural network for image captioning is a hard task. Even after putting the image feature vector and sentences in the same domain for recurrent neural network input, the network takes hours to train on even a small vocabulary size of 1,227 words and 10,000 training images. To make a better image captioning neural network, more training images are required so that you can expand the vocabulary. However, the implementation in this paper shows the potential for neural networks to train to describe images. This implementation showed that neural networks have the capability to learn image features from an object classification neural network and describe it with a vocabulary using recurrent neural networks.

One choice in my implementation that I made was to throw away words that were not included in the vocabulary. I think an area where this paper could be improved is to create a word that stands for "OTHER" that is sequenced into the network for words not in the vocabulary. I believe that including this word will help the image captioning network to learn sentence structure when words are not included in the vocabulary since the network will be able to learn that something is supposed to be in that spot. This could prove to be a bad method however, as when testing it is difficult to decide how to handle the word "OTHER" being a possible word. It may still prove beneficial to use the "OTHER" word in training and when testing decide not to include the partial caption in your top-N list whenever you expand to the "OTHER" word. Another

Figure 3: Output from Google Colab Network On Test Set After Training

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_6 (InputLayer)            (None, None)         0
__________________________________________________________________________________________________
input_5 (InputLayer)            (None, None, 1000)   0
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, None, 1000)   1227000     input_6[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, None, 1000)   0           input_5[0][0]
                                                                 embedding_3[0][0]
__________________________________________________________________________________________________
gru_3 (GRU)                     (None, 1227)         8201268     concatenate_3[0][0]
__________________________________________________________________________________________________
softmax_3 (Softmax)             (None, 1227)         0           gru_3[0][0]
==================================================================================================
Total params: 9,428,268
Trainable params: 9,428,268
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 4: Summary of My Keras Model for Image Captioning

```
batch_loss = 5.694066
the people is in a are people on a people on his hand people people sitting on some people on
batch_loss = 5.776931
two young boy is sitting on a girl wearing blue shirt in front.
batch_loss = 5.746498
are sitting are in some are are standing on in front are sitting are sitting are are are standing on
batch_loss = 5.742562
the man in people standing outside people are in a people people on the camera are with a people people
batch_loss = 5.815365

the side is in front are sitting on a are sitting on some.
batch_loss = 5.755904
the side is sitting on the man and a girl wearing blue shirt in his hand.
batch_loss = 5.789825
a girl wearing blue shirt standing outside on a girl wearing blue shirt in front.
batch_loss = 5.831965
two men wearing blue shirt standing outside on a crowd.
batch_loss = 5.773825
the top is sitting on some.
```

Figure 5: Output of Example Sentences During Training

Figure 6: Final Results on Images from Test Set

solution may be to have the ”OTHER” word as part of the input vocabulary but not include it in the network output vocabulary. Due to my network already being 6 hours in training, I decided not to make this adjustment but think it would be an interesting area for future pursuit.

Another area for potential improvement could be to allow the convolutional neural network to be trainable after the recurrent neural network architecture is trained. It is possible that the output of the model could be improved with some fine-tuning to the convolutional neural network architecture with its new task of image captioning.

Overall, image captioning is a difficult task and an important area for research within computer vision. The ability to take an image input and describe it is very useful to test what can be learned with modern machine learning techniques about images and creating text. Describing an image has many applications such as being able to tell someone who is blind what is in an image. With the advancement of neural network architectures, computer hardwares ability to build complex neural networks, and larger datasets it is possible to leverage these technologies to make image captioning algorithms better than previous capable. Show and Tell, A Neural Image Caption Generator shows that neural networks can be used to improve the capabilities we have to caption an image with an algorithm and opens up an area for future research in computer vision.

Acknowledgments

Used pre-trained VGG19 model found on github user machrisaa and used flickr30k dataset for training.

References

[1] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and Tell: A Neural Image Caption Generator. In arXiv:1411.455v2, 2015.

[2] B. Z. Yao, X. Yang, L. Lin, M. W. Lee, and S.-C. Zhu. I2t: Image parsing to text description. Proceedings of the IEEE, 98(8), 2010.

[3] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. Every picture tells a story: Generating sentences from images. In ECCV, 2010.

[4] S. Li, G. Kulkarni, T. L. Berg, A. C. Berg, and Y. Choi. Composing simple image descriptions using web-scale n-grams. In Conference on Computational Natural Language Learning, 2011.

[5] R. Socher, A. Karpathy, Q. V. Le, C. Manning, and A. Y. Ng. Grounded compositional semantics for finding and describing images with sentences. In ACL, 2014.

[6] R. Kiros and R. Z. R. Salakhutdinov. Multimodal neural language models. In NIPS Deep Learning Workshop, 2013.

[7] J. Mao, W. Xu, Y. Yang, J. Wang, and A. Yuille. Explain images with multimodal recurrent neural networks. In arXiv:1410.1090, 2014.

[8] A. Krizhevsky, I. Sutskever, and G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Network. 2012.

[9] K. Simonyan and A Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In arXiv:1409.1556v6, 2015.