Image Captioning Bot

PROJECT SYNOPSIS

OF PROJECT-6th Sem. (CSR-355)

BACHELOR OF ENGINEERING
Computer Science

SUBMITTED BY

AMAN BAHUGUNA **DEEPAK YADAV**

GYAN RANJAN KUMAR

Feb 2021

SUPERVISOR
Er. GAGANJOT KAUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, CHANDIGARH UNIVERSITY, GHARUAN (P.B.) – 140413

Table of Contents:

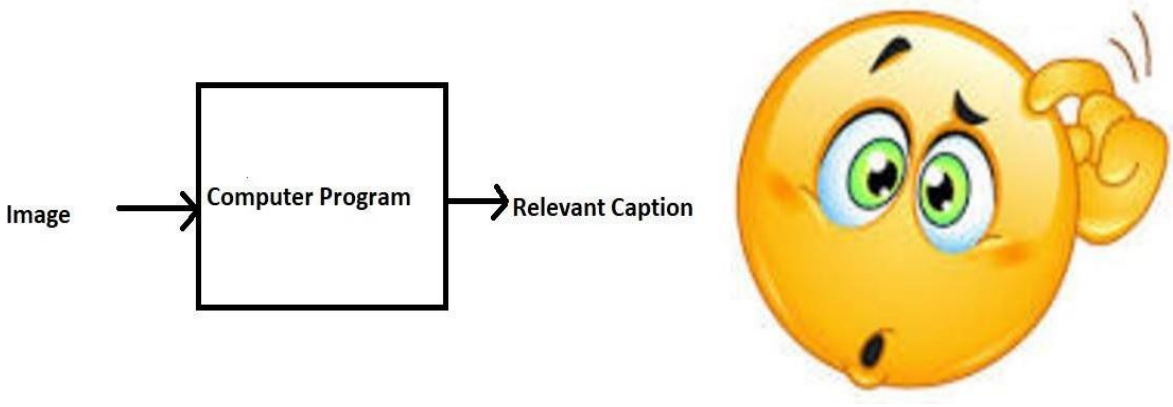
1.1 Introduction

What do you see in the below picture?

Well some of you might say "**A white dog in a grassy area**", some may say "**White dog with brown spots**" and yet some others might say "**A dog on grass and some pink flowers**".

Definitely all of these captions are relevant for this image and there may be some others also. But the point I want to make is; it's so easy for us, as human beings, to just have a glance at a picture and describe it in an appropriate language. Even a 5 year old could do this with utmost ease.

But, can you write a computer program that takes an image as input and produces a relevant caption as output?



Just prior to the recent development of Deep Neural Networks this problem was inconceivable even by the most advanced researchers in Computer Vision. But with the advent of Deep Learning this problem can be solved very easily if we have the required dataset.

This problem was well researched by **Andrej Karapathy** in his PhD thesis at Stanford [1], who is also now the **Director of AI at Tesla.** The purpose of this blog post is to explain (in as simple words as possible) that how Deep Learning can be used to solve this problem of generating a caption for a given image, hence the name **Image Captioning.**

To get a better feel of this problem, I strongly recommend to use this state-of-the-art system created by Microsoft called as **Caption Bot**. Just go to this link and try uploading any picture you want; this system will generate a caption for it.

1.2. Motivation

We must first understand how important this problem is to real world scenarios. Let's see few applications where a solution to this problem can be very useful.

- Self-driving cars — automatic driving is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self-driving system.
- Aid to the blind — we can create a product for the blind which will guide them travelling on the roads without the support of anyone else. We can do this by first converting the scene into text and then the text to voice. Both are now famous applications of Deep Learning.
- CCTV cameras are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.
- Automatic Captioning can help, make Google Image Search as good as Google Search, as then every image could be first converted into a caption and then search can be performed based on the caption.

1.3. Prerequisites

Basic Deep Learning concepts like Multi-layered Perceptrons, Convolution Neural Networks, Recurrent Neural Networks, Transfer Learning, Gradient Descent, Backpropagation, Overfitting, Probability, Text Processing, Python syntax and data structures, Keras library, etc.

- Library: Keras, Tensor flow, NumPy, Pandas, etc.
- Tools: AWS/ Heroku, Flask.
- Platform: Jupyter Notebook or Google Co-lab

2.1. Data Collection

There are many open source datasets available for this problem, like Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc.

We can used flickr8k dataset that is available on kaggle. You can run download_data.sh shell script. Before that put kaggle.json which is kaggle api key in same directory.

I have used the Flickr 8k dataset which will have provided by the University of Illinois at Urbana-Champaign. Also training a model with large number of images may not be feasible on a system which is not a very high end PC/Laptop.

This dataset contains 8000 images each with 5 captions (as we have already seen in the Introduction section that an image can have multiple captions, all being relevant simultaneously).

These images are bifurcated as follows:

- Training Set — 6000 images
- Dev Set — 1000 images
- Test Set — 1000 images

2.2. Understanding the data

If you have downloaded the data from the link that I have provided, then, along with images, you will also get some text files related to the images. One of the files is “Flickr8k.token.txt” which contains the name of each image along with its 5 captions. We can read this file as follows

```
# Below is the path for the file "Flickr8k.token.txt" on your disk
filename = "/dataset/TextFiles/Flickr8k.token.txt"
file = open(filename, 'r')
doc = file.read()
```

The text file looks as follows:

```
101654506_8eb26cfb60.jpg#0      A brown and white dog is running through the snow .
101654506_8eb26cfb60.jpg#1      A dog is running in the snow
101654506_8eb26cfb60.jpg#2      A dog running through snow .
101654506_8eb26cfb60.jpg#3      a white and brown dog is running through a snow covered field .
101654506_8eb26cfb60.jpg#4      The white and brown dog is running over the surface of the snow

1000268201_693b08cb0e.jpg#0     A child in a pink dress is climbing up a set of stairs in an ent
1000268201_693b08cb0e.jpg#1     A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2     A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3     A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4     A little girl in a pink dress going into a wooden cabin .
```

From Kaggle Datasets

Thus every line contains the <image name>#i <caption>, where $0 \leq i \leq 4$ i.e. the name of the image, caption number (0 to 4) and the actual caption.

Now, we create a dictionary named "descriptions" which contains the name of the image (without the .jpg extension) as keys and a list of the 5 captions for the corresponding image as values.

2.3. Data Cleaning

When we deal with text, we generally perform some basic cleaning like lower-casing all the words (otherwise "hello" and "Hello" will be regarded as two separate words), removing special tokens (like '%', '$', '#', etc.), eliminating words which contain numbers (like 'hey199', etc.).

Create a vocabulary of all the unique words present across all the 8000*5 (i.e. 40000) image captions (**corpus**) in the data set : This means we have 8763 unique words across all the 40000 image captions. We write all these captions along with their image names in a new file namely, "*descriptions.txt"* and save it on the disk.

However, if we think about it, many of these words will occur very few times, say 1, 2 or 3 times. Since we are creating a predictive model, we would not like to have

all the words present in our vocabulary but the words which are more likely to occur or which are common. This helps the model become more **robust to outliers** and make less mistakes.

Hence we consider only those words which **occur at least 10 times** in the entire corpus. The code for this is below:
So now we have only 1651 unique words in our vocabulary. However, we will append 0's (zero padding explained later) and thus total words = 1651+1 = **1652** (one index for the 0) [2]

3.1. Dividation Work in Team:

We are three member in a team:
- AMAN BAHUGUNA (18BCS2441)
- DEEPAK YADAV (18BCS2446)
- GYAN RANJAN KUMAR (18BCS2431)

We have divided our work into three parts, in which Data Collection, Cleaning, and Feature Engineering part will be handled by Aman Bahuguna.

Deepak Yadav – Applying Deep Learning Algorithm on the Model, Text mining and Model Architecture.

Gyan Ranjan Kumar – Accuracy, Evolution and Model Deployment in Heroku by usingPython Framework (Flask etc.)

3.2. Data Preparation using Generator Function

This is one of the most important steps in this case study. Here we will understand how to prepare the data in a manner which will be convenient to be given as input to the deep learning model [5].

Hereafter, I will try to explain the remaining steps by taking a sample example as follows:

Consider we have 3 images and their 3 corresponding captions as follows:

(Train image 1) Caption -> The black cat sat on grass

(Train image 2) Caption -> The white cat is walking on road

(Test image) Caption -> The black cat is walking on grass

Now, let's say we use the first two images and their captions to train the model and the third image to test our model.

Now the questions that will be answered are: how do we frame this as a supervised learning problem?, what does the data matrix look like? how many data points do we have?, etc.

First we need to convert both the images to their corresponding 2048 length feature vector as discussed above. Let "Image_1" and "Image_2" be the feature vectors of the first two images respectively

Secondly, let's build the vocabulary for the first two (train) captions by adding the two tokens "startseq" and "endseq" in both of them: (Assume we have already performed the basic cleaning steps)

Caption_1 -> "startseq the black cat sat on grass endseq"

Caption_2 -> "startseq the white cat is walking on road endseq"

vocab = {black, cat, endseq, grass, is, on, road, sat, startseq, the, walking, white}

Let's give an index to each word in the vocabulary:
black -1, cat -2, endseq -3, grass -4, is -5, on -6, road -7, sat -8, startseq -9, the -10, walking -11, white -12 Now let's try to frame it as a supervised learning problem where we have a set of data points D = {Xi, Yi}, where Xi is the feature vector of data point 'i' and Yi is the corresponding target variable.

Let's take the first image vector Image_1 and its corresponding caption "startseq the black cat sat on grass endseq". Recall that, Image vector is the input and the caption is what we need to predict. But the way we predict the caption is as follows:

For the first time, we provide the image vector and the first word as input and try to predict the second word, i.e.:

Input = Image_1 + 'startseq'; Output = 'the'

Then we provide image vector and the first two words as input and try to predict the third word, i.e.:

Input = Image_1 + 'startseq the'; Output = 'cat'

And so on…

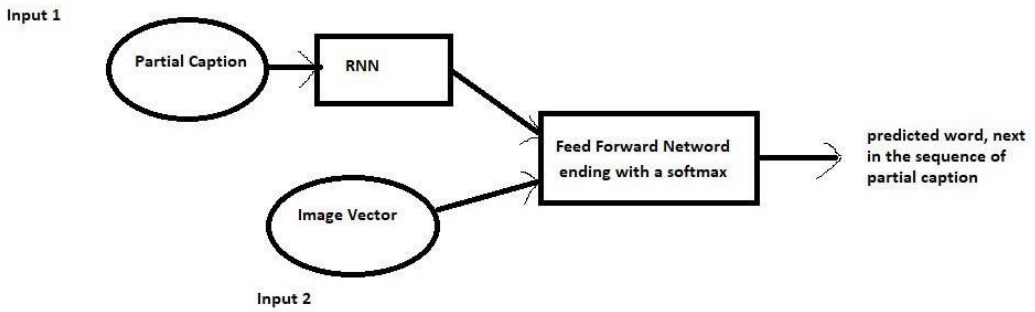
3.4. Word Embeddings

As already stated above, we will map the every word (index) to a 200-long vector and for this purpose, we will use a pre-trained GLOVE Model:

4.1. Model Architecture

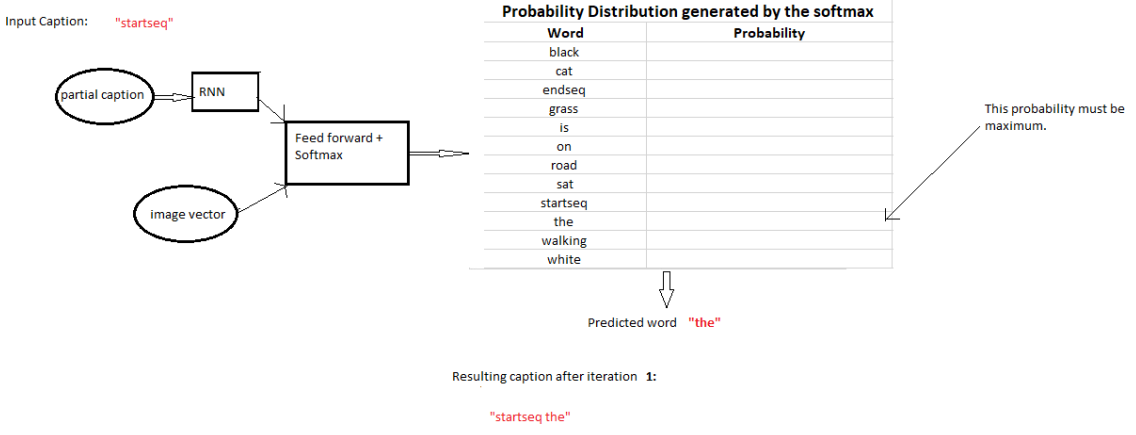
Since the input consists of two parts, an image vector and a partial caption, we

cannot use the Sequential API provided by the Keras library. For this reason, we use the Functional API which allows us to create Merge Models [1].
First, let's look at the brief architecture which contains the high level sub-modules:



4.1. Inference

So till now, we have seen how to prepare the data and build the model. In the final step of this series, we will understand how do we test (infer) our model by passing in new images, i.e. how can we generate a caption for a new test image.[2]
Recall that in the example where we saw how to prepare the data, we used only first two images and their captions. Now let's use the third image and try to understand how we would like the caption to be generated[4].
The third image vector and caption were as follows:



4.2. Evaluation

To understand how good the model is, let's try to generate captions on images from the test dataset (i.e. the images which the model did not see during the training).

Output — 1


```
Greedy: motorcyclist is riding an orange motorcycle
```

Important Point:

We must understand that the images used for testing must be semantically related to those used for training the model. For example, if we train our model on the images of cats, dogs, etc. we must not test it on images of air planes, waterfalls, etc. This is an example where the distribution of the train and test sets will be very different and in such cases no Machine Learning model in the world will give good performance.

5. Conclusion and Future work

5.1. BLEU Score

BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more reference translations. A perfect match

results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.

The primary programming task for a BLEU implementer is to compare n-grams of the candidate with the n-grams of the reference translation and count the number of matches. These matches are position-independent. The more the matches, the better the candidate translation is. Unfortunately, MT systems can over generate "reasonable" words, resulting in improbable, but high-precision, translations. Intuitively the problem is clear: a reference word should be considered exhausted after a matching candidate word is identified. We formalize this intuition as the modified unigram precision [2].

We first compute the n-gram matches sentence by sentence. Next, we add the clipped n-gram counts for all the candidate sentences and divide by the number of Candidate n-grams in the test corpus to compute a modified precision score, pn, forthe entire test corpus.

The BLEU metric ranges from 0 to 1. Few translations will attain a score of 1 unless they are identical to a reference translation. For this reason, even a human translator will not necessarily score 1.

NLTK provides the sentence_bleu() function for evaluating a candidate sentence against one or more reference sentences. NLTK also provides a function called corpus_bleu() for calculating the BLEU score for multiple sentences such as a paragraph or a document.

5.2. Future Work

Note that due to the stochastic nature of the models, the captions generated by you (if you try to replicate the code) may not be exactly similar to those generated in my case. Of course this is just a first-cut solution and a lot of modifications can be made to improve this solution like:[3]

- Using a larger dataset.
- Changing the model architecture, e.g. include an attention module.
- Doing more hyper parameter tuning (learning rate, batch size, number of layers, number of units, dropout rate, batch normalization etc.).
- Use the cross validation set to understand over fitting.
- Using Beam Search instead of Greedy Search during Inference.
- Using BLEU Score to evaluate and measure the performance of the model.

6. References

1. Chunjie Guo.Weiwei Yang . Show, attend and tell: Image captioning based on deep reinforcement learning" in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2048–2057, Boston, MA, USA, June 2015

2. Zhou L, Xu C, Koch P, et al. Image Caption Generation with Text-Conditional Semantic Attention{J}. 2016.

3. Xu K, Ba J, Kiros R, et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention{J}. Computer Science, 2015:2048—2057

4. Andrej Karpathy Li Fei-Fei Deep Visual-Semantic Alignments for Generating Image Descriptions

5. A. Gupta and P. Mannem. From image annotation to image description. In Neural information processing. Springer, 2012.

Signature_____