Development of Deep Learning Image Caption Generator

Osman Mamun

April 21, 2021

Contents

1 Introduction

It is relatively easy for human to be able to describe any environments they are in, e.g., when shown a photo, any human can describe very easily what's going on in the picture. This unique ability is fundamental to our existence which we use everyday, both knowingly and unknowingly, to almost every situation we encounter. Making machine enable to such ability has been an overarching goal for artificial intelligence community for a long time.

Even though there is a great progress that has been made in various computer vision tasks, such as object recognition, image classification etc., it is still a challenging task to train a computer understand an image and form a coherent sequence of word to describe the image. It is an unique challenge to solve as it unifies two different fields of machine learning, i.e., computer vision natural language processing. Image caption generator will facilitate a lot of tasks, such as automatic captioning of items on a ecommerce website, helping blind people safely navigate the world, enabling self driving car to communicate with the passengers about an oncoming complicated situation to signal manual intervention etc.

2 Data Acquisition and Cleaning

For this work, we acquire two publicly available dataset, namely MS-COCO (45K images) and Flicker30K (30K images) dataset. They were downloaded using the tensorflow data download API. In the Figure 1, a code snippet is shown to collect data with the tensorflow API. More details about acquiring and concatenating the data can be found in this IPython notebook.

3 Image Feature Extraction

Instead of training a CNN from the scratch, pretrained image model is used to train the model. In this work, VGG16 and InceptionV3 pretrained model are used to extract the image features. The last layer of both the pretrained layer is discarded and two fully connected layers are trained on the dataset to fine-tune the model for this particular task. In Figure 2 and 3, we show the architecture of the both model.

```
[4]
def download_data(name, url_dict):
    root = '/Users/mamu867/PNNL_Mac/Springboard/image_caption_generator/data/raw/' + name
    #file_loc = {}
    for dname, url in url_dict.items():
        if not os.path.exists(os.path.join(root, f'{dname}.zip')):
            zip = tf.keras.utils.get_file(f'{dname}.zip',
                                          cache_subdir=root,
                                          origin = url,
                                          extract = True)
            os.remove(zip)
        else:
            print(f'{dname} already exists!')
    return

[5]
coco_url_dict = {'captions_train_val': 'http://images.cocodataset.org/annotations/
annotations_trainval2014.zip',
                 'train': 'http://images.cocodataset.org/zips/train2014.zip',
                 #'val': 'http://images.cocodataset.org/zips/val2014.zip',
                 #'test': 'http://images.cocodataset.org/zips/test2014.zip'
                }

#coco = ImageDataset(name='COCO')
#coco.download_data(coco_url_dict)
download_data('COCO', coco_url_dict)
```

Figure 1: Data collection for MS-COCO and Flicker30K dataset.

Figure 2: Architecture of the VGG16 model.

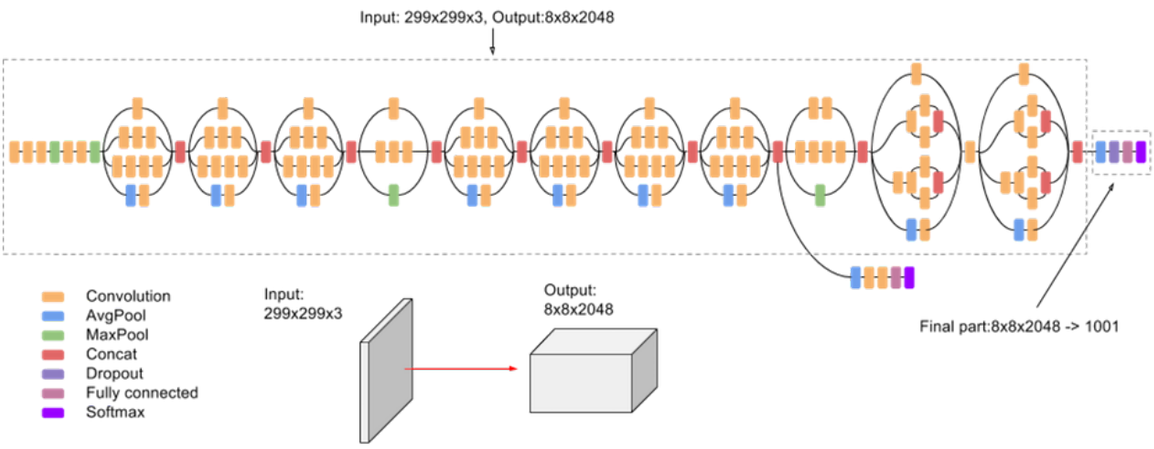

Figure 3: Architecture of the Inception V3 model.

4 Word Tokenizer

Next, we used keras word tokenizer to tokenize the words in a sentence and to pad it to the maximum length of the tokenizer for this corpus. Also, <start>

and <end> token is added in the beginning and end of the sentence for sequence generation process. In Figure 4, we show a code snippet that was used to tokenize a particular corpus. For detail description of the image feature extraction and word tokenization, we refer interested reader to this IPython notebook.

```python
import tensorflow as tf

class WordVectorizer:
    def fit(self, texts=None, top_k=5000):
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
        oov_token="<unk>", filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~ ')
        self.tokenizer.fit_on_texts(texts)
        self.tokenizer.word_index['<pad>'] = 0
        self.tokenizer.index_word[0] = '<pad>'
    def predict(self, texts=None):
        train_seqs = self.tokenizer.texts_to_sequences(texts)
        cap_vector = tf.keras.preprocessing.sequence.pad_sequences(train_seqs,
        padding='post')
        return cap_vector
```

Figure 4: Tokenization of the caption text.

5 RNN Decoder

For RNN decoder with attention mechanism, we used Bahdanau attention mechanism as described in the original paper by Bahdanau et al. In Figure 5, an illustration of the Bahdanau attention mechanism is shown. Also, in Figure 6, the decoder architecture used in this study is shown. We used a word embedding layer, a gated recurrent unit (GRU), and two fully conneted layer to predict the caption sequence. In the begining of the training, the decoder is fed with the image features along with the <start> token, and the model is sequentially trained to predict the next word until the <end> token is reached. In the attention, the alignment score of the previous hidden states with the new hidden states are computed and then softmaxed to identify which part of the sequence is important to find the next word in the sequence.

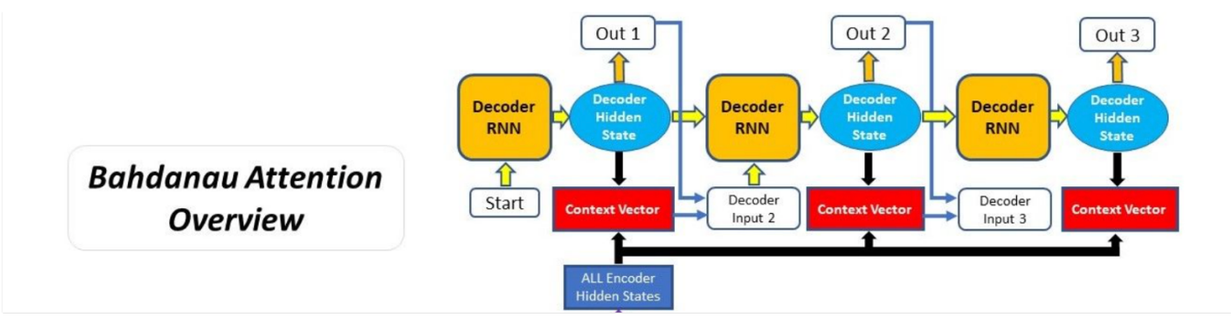

Figure 5: Bahdanau attention mechanism.

```
Model: "rnn__decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        multiple                  2560256
_________________________________________________________________
gru (GRU)                    multiple                  1575936
_________________________________________________________________
dense_1 (Dense)              multiple                  262656
_________________________________________________________________
dense_2 (Dense)              multiple                  5130513
_________________________________________________________________
bahdanau_attention (Bahdanau multiple                  394753
=================================================================
Total params: 9,924,114
Trainable params: 9,924,114
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: RNN decoder to generate caption sequence from image features.

6 Results Discussion

6.1 Flicker30K dataset with VGG16 pretrained model

We trained the Flicker30K dataset with image features derived with the VGG16 pretrained model for 20 epochs. In Figure 7, we can see that the loss is still going down and additional training can be done to improve the model performance. In terms of time requirements for training, the 1st epoch took about 10 minutes and the subsequent epochs took about 80 seconds on average with a GPU processor available in Google colab. In terms of training time, it's fairly reasonable time, and can be easily trained for additional 20 epochs. It is also worth noting here that the model is trained with $12,000$ images. In Figure 8, an image with the real caption "<start> a man in a white boat is rowing through fog <end>" is shown where the model predicted caption is "<start> a man in dark boat <end>". It seems like our model is pretty good at understanding the major components of an image; however, it was not particularly good at describing the fine detail in the image, e.g., fog in this particular case. Overall, the performance is pretty satisfactory even though the model can reliably generate caption only for images that are from the same dataset, i.e., from the same distribution. For the holdout test set of $1,000$ images, the model's BLEU score was computed to be 0.65.

6.2 Flicker30K dataset with Inception V3 pretrained model

Next, we train the Flicker30K dataset with the inception v3 pretrained model generated image features. We also trained the model for 20 epochs and the run

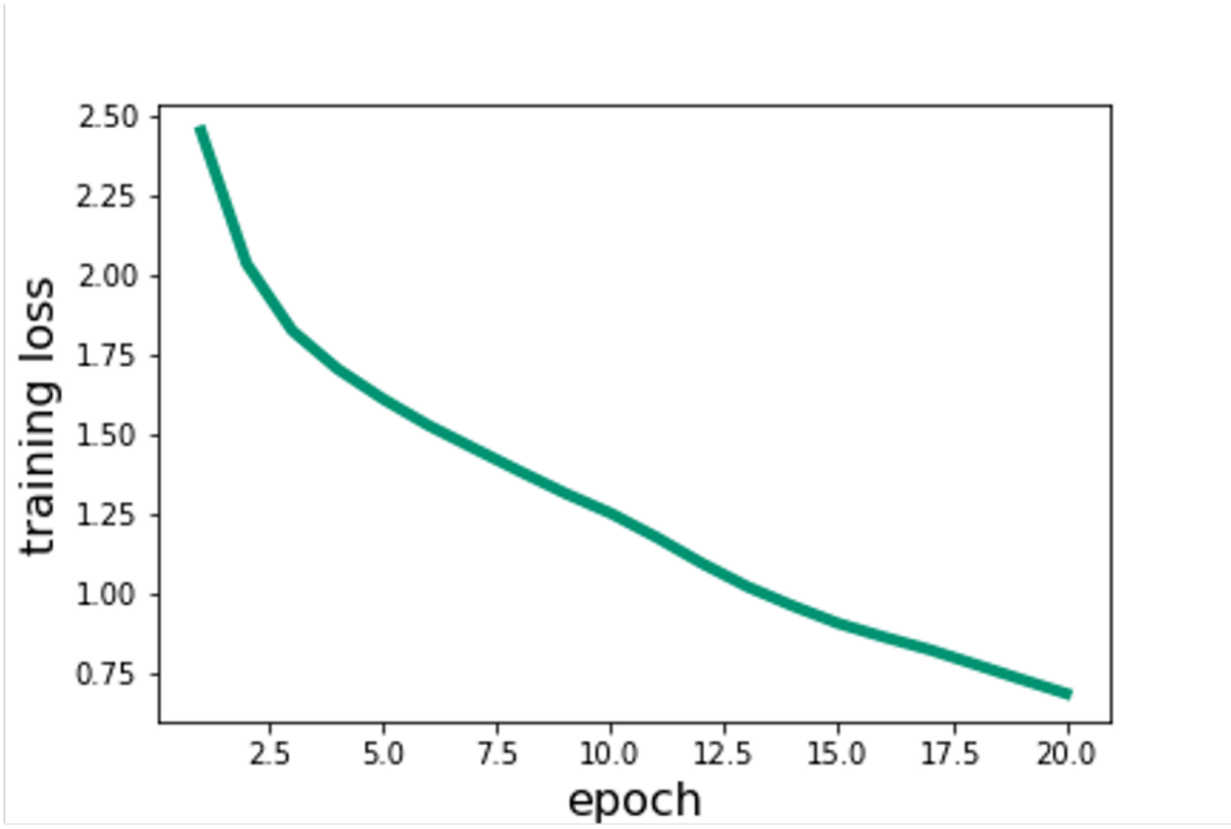

Figure 7: Learning curve for Flicker30K data with the VGG16 model extracted features.

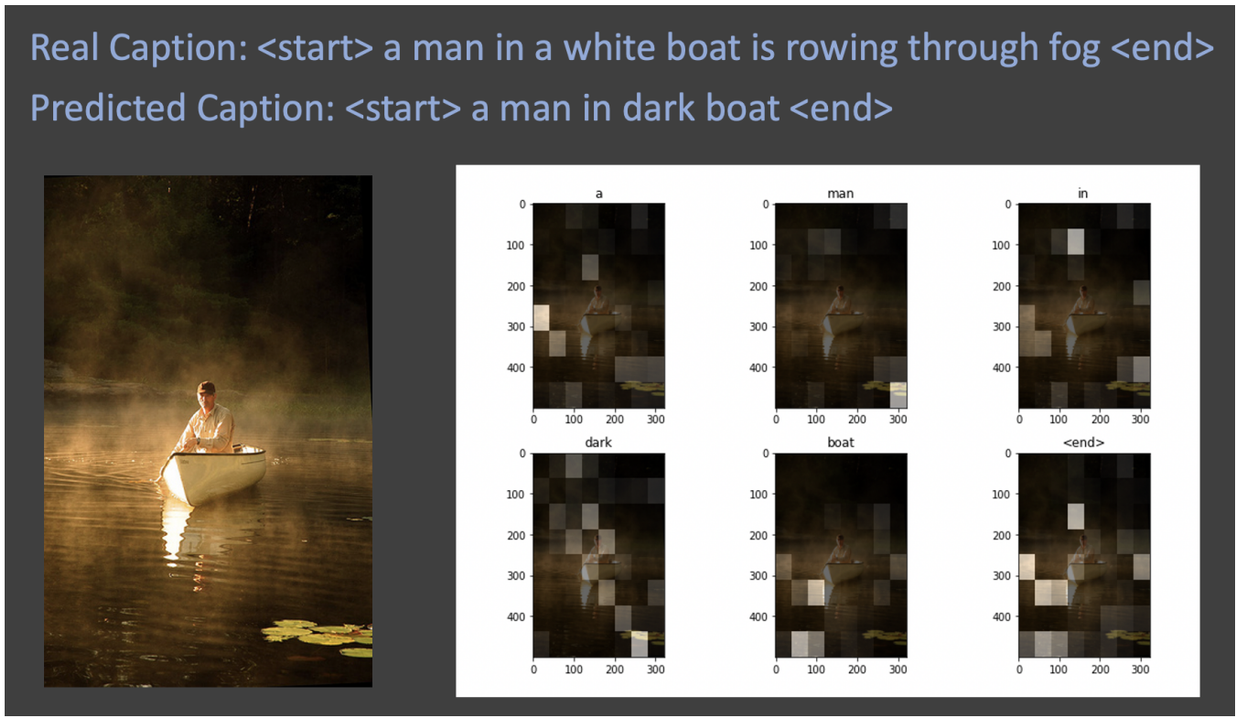

Figure 8: A sample image with real and predicted caption (with vgg16 image model) along with the attention plot visualizing the part of the image responsible for the prediction.

time of this model is roughly similar to the runtime of the previous model. In Figure 9, the learning curve shows that we can still train the model to improve the performance even further. In Figure 10 11, we show this model predicted caption and the corresponding real caption and the images. For the 1st image, the model is capable of extracting some major components from the image to describe it reasonably well but it fails to describe the fine detail present in the image. In

the 2nd image, the model produced a different version of the caption; however, considering the fact that it was able to detect that a boy is waterskiing with his hand shows the model's impressive capacity to be able to understand the context of the image.

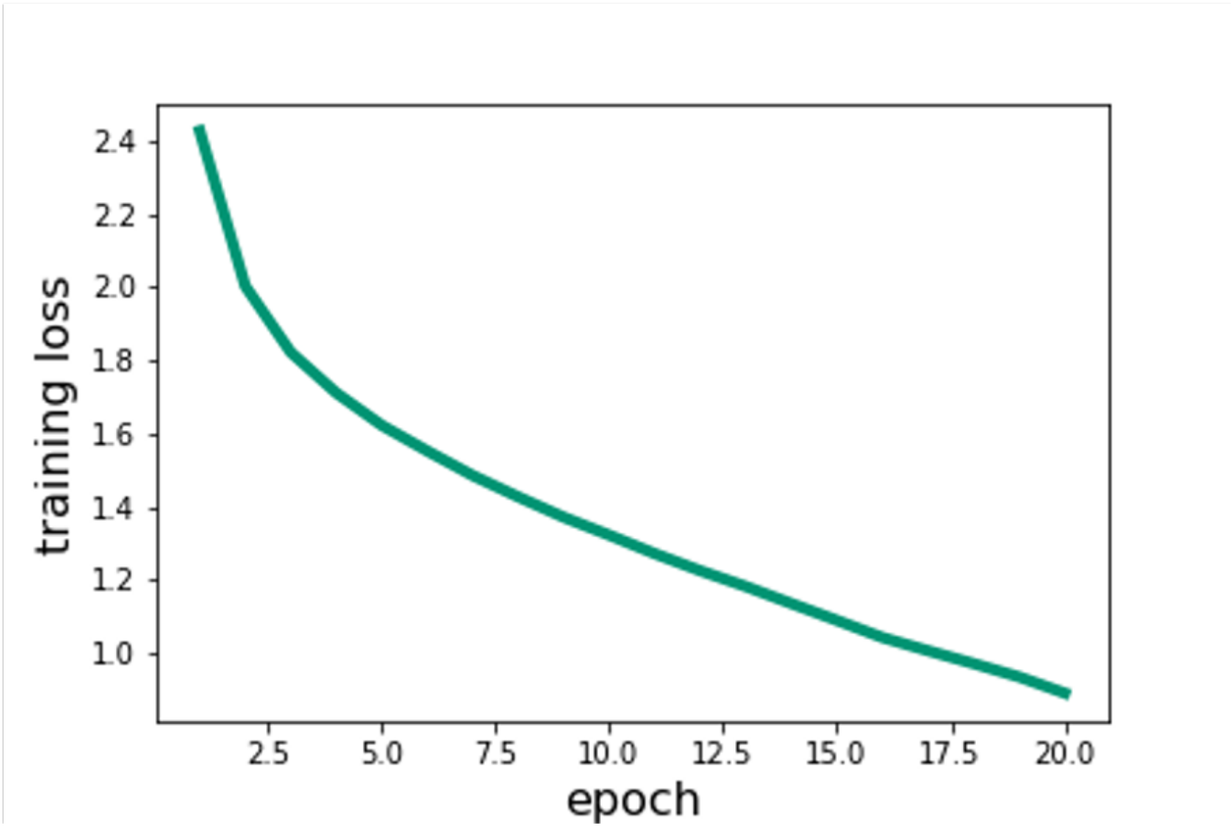

Figure 9: Learning curve for Flicker30K data with the inception v3 model extracted features.

Figure 10: A sample image with real and predicted caption (with inception v3 image model) along with the attention plot visualizing the part of the image responsible for the prediction.

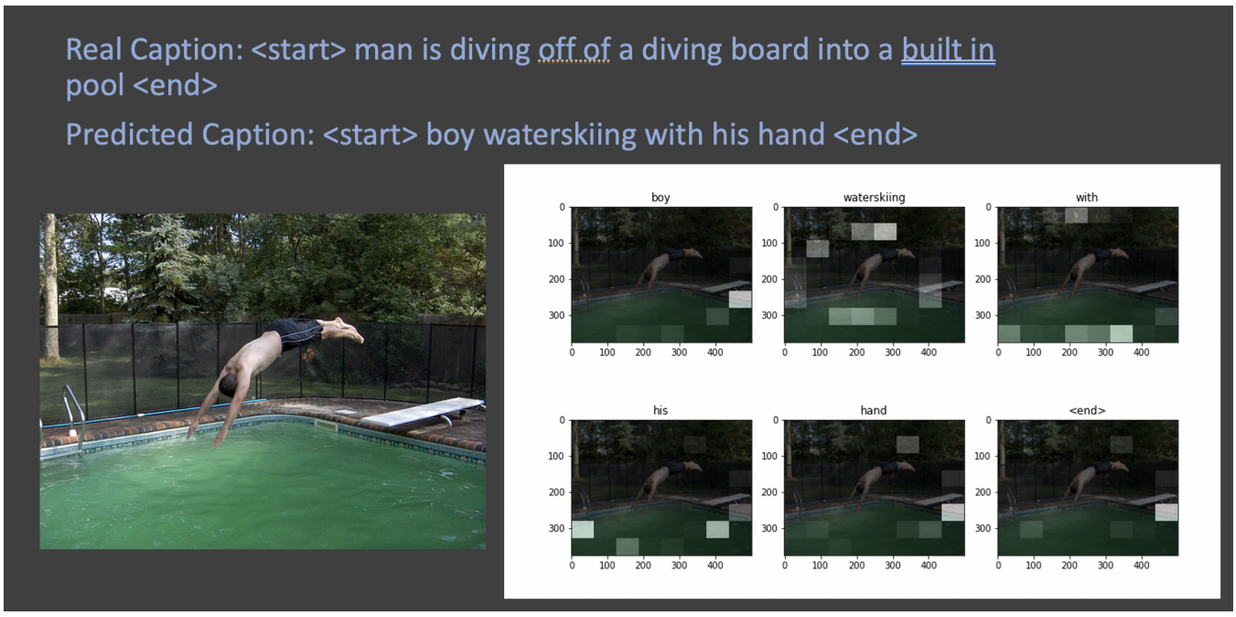

Figure 11: A sample image with real and predicted caption (with inception v3 image model) along with the attention plot visualizing the part of the image responsible for the prediction.

7 Conclusion

In this work, we demonstrate an image caption generator model from the publicly available dataset. Even though the model is pretty rudimentary, but it is still can perform reasonably well to generate caption for images drawn from the same data distribution. Below are some ideas for future work:

1. Using different pre-trained image model.
2. Using pre-trained language model for word embedding.
3. Use a larger dataset.
4. Build a web-app for customer to interact with the final model.