Image Caption Generator

Jasmeetsingh Khalsa
Cpts 534 Neural Network Design and Application
Final Report
Spring 2022
11743550

1 Introduction

We know how easy it is for our minds to tell what a given image is about, but can a computer tell what the image is representing? Computer vision researchers worked on this a lot and they considered this impossible till now! With the help of deep learning techniques, availability of huge datasets and computer power, we can build models that can generate captions for an image.

In this project we will see a Image Caption Generator which is a challenging Neural Network problem where a textual description must be generated for a given photograph. It requires both methods from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order. The primary aim would be to build a Network model that would generate captions for an input image which explains the context of the image. To build the model, we will use basic Neural Networks such as Convolutional Neural Networks and LSTMs (a type of Recurrent Neural Network) along with a technique called attention mechanism which recently played an important role in computer vision and is recently widely used in image caption generation tasks.

2 Literature Review

Recently the development of image description system has drawn increasing attention and it has become one of the most important topics in Computer Vision. Early image description generation methods aggregate image information using static object class libraries in the image and modeled using statistical language models. In paper 'Generating image descriptions using dependency relational patterns', Aker and Gaizauskas [1] use a dependency model to summarize multiple web documents containing information related to image locations and propose a method for automatically tagging geotagged images. In paper 'Corpus-Guided Sentence Generation of Natural Images' Yang et al. [2] proposed a language model trained from the English Gigaword corpus to obtain the estimation of motion in the image and the probability of collocated nouns, scenes, and prepositions and used these estimates as parameters of the hidden Markov model. Here, the image description was obtained by predicting the most likely nouns, verbs, scenes, and prepositions that make up the sentence.

Relation to our work: The methods described in the above papers are brainstorming and have their own characteristics, but all have the common disadvantage that none of them make any intuitive feature observations on objects or actions in the image, nor do they give an end-to-end mature general model to solve this problem. This is what we try to achieve through our work in this project. We will propose a model that will make feature observations on objects and will extract useful information for prediction.

The initial step of the project is to create a baseline model using a simple combination of CNN along with a RNN [3]. This technique will successfully generate captions for the input image. This is a classic application of Neural networks. However, the problem with a 'classic' image captioning model is that when the model is trying to generate the next word of the caption, this word is usually describing only a part of the image. It is unable to capture the essence of the entire input image. Using the whole representation of the image h to condition the generation of each word cannot efficiently produce different words for different parts of the image. This is exactly where deep learning technique called Attention mechanism is helpful.

3 Dataset

Out of the vast varieties of datasets available for this problem like Flickr 8k , Flickr 30k, MS COCO, etc. The Flickr8K dataset will be used for the model training of image caption generators. Flickr8k image comes from Yahoo's photo album site Flickr, which contains 8,000 photos, 6000 image training, 1000 image verification, and 1000 image testing.

This dataset is readily available and can be directly downloaded from the internet. The downloading process takes some time due to the large size(1GB) of the dataset. 8091 images are stored inside the Flicker_8k_Dataset folder and the text files with captions of images are stored in the Flickr_8k_text folder. In short the flicker dataset consists of:

1. Flicker_8k_Dataset: Folder with 8091 images.

2. Flickr_8k_text: Folder with Text files with captions of images

4 Baseline

A basic image captioning system would encode the image, using a pre-trained Convolutional Neural Network(ENCODER) that would produce a hidden state h. Then, it would decode this hidden state by using a LSTM(DECODER) and generate recursively each word of the caption.

To build a baseline image caption generator model we would simply merge CNN with LSTM. Therefore:

Image Caption Generator Model(CNN-RNN model) = CNN + LSTM.

- CNN- To extract features from the image. A pre-trained model called Xception is used for this.
- LSTM- To generate a description from the extracted information of the image.

First, we write functions to load the datasets for model training. Here 3 functions are primarily written, 1. load_photos(fname), 2. load_clean_descriptions(fname, image), 3. load_features(photos).

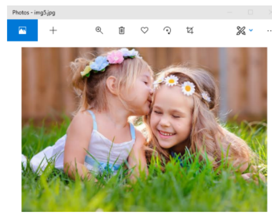
Then we tokenize the vocabulary to a numerical representation for machines to understand. This is done by mapping each word of vocabulary with a separate unique index value.

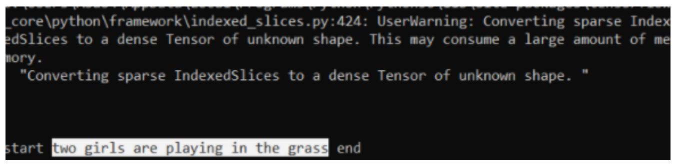
Next step is to define a CNN-RNN model. Here, we used Keras model in order to define the structure of the model. This includes:

1. A feature extractor with a dense layer, which will extract the feature from the images of size 2048 and will decrease the dimensions to 256 nodes.
2. A sequence processor followed by the LSTM layer, the textual input is handled by this embedded layer.
3. Decoder which will merge the output of the above two layers and process the dense layer to make the final prediction.

Now, we train the Image Caption Generation Model. Here, we generate the input and output sequences to train our model with 6000 training images. We create a function named model.fit_generator() to fit the batches to the model.

Finally comes testing, Here the task is to test the model accuracy by inputting test image data. Here, we see that the proposed baseline successfully generates captions for the test image.





5 Main Approach

The code for the main approach is available in Github. I tried to keep the Colab notebook as descriptive as possible so that anyone can understand the purpose of each cell.

5.1 Concept of Attention Mechanism

In Attention Mechanism, the image is first divided into n parts, and we compute with a CNN representations of each part h1,. . . , hn. When the RNN is generating a new word, the attention mechanism is focusing on the relevant part of the image, so the decoder only uses specific parts of the image.

There are two types of Attention Mechanism:

1. Global Attention: Attention is placed on all source positions.
2. Local Attention: Attention is placed only on a few source positions.

Global attention takes into consideration all encoder hidden states to derive the context vector (c(t)). In order to calculate c(t), we compute a(t) which is a variable length alignment vector. The alignment vector is derived by computing a similarity measure between h_t and $\bar{h}_s$ where h_t is the source hidden state while $\bar{h}_s$ is the target hidden state. Similar states in encoder and decoder are actually referring to the same meaning.

We use a score function which is a content-based function using which we calculate the similarity between the hidden states of the target and the source.

$$score(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s \\ h_t^T W_a \bar{h}_s \\ v_a^T tanh(W_a[h_t; \bar{h}_s]) \end{cases}$$

As Global attention focus on all source side words for all target words, it is computationally very expensive and is impractical when translating for long sentences. To overcome this deficiency **Local attention** chooses to focus only on a small subset of the hidden states of the encoder per target word.

$$f_{\text{ATT}} = V_{\text{attn}}^T * tanh(U_{\text{attn}} * h_j + W_{\text{attn}} * s_t)$$

$Where,$

$$V_{\mathrm{attn}}^T \in R^d, U_{\mathrm{attn}} \in R^{\mathrm{d*d}}, W_{\mathrm{attn}} \in R^{\mathrm{d*d}}, s_{\mathrm{t-1}} \in R^d, h_j \in R^d$$

Now, general Score is given as:

$$e_{\mathrm{jt}} = f_{\mathrm{ATT}}(s_{\mathrm{t-1}}, h_j)$$

Where, e_{jt} means at every t^{th} timestep of ENCODER, how important j^{th} is the pixel location in the input image. $s_{\mathrm{t-1}}$ is previous state of DECODER. h_j is state of ENCODER. f_{ATT} is a simple Feed Forward Neural Network, which is a linear transformation of input $(U_{\mathrm{attn}} * h_j + W_{\mathrm{attn}} * s_t)$ and then a non-linearity(tanh) on top of that, and then again one more transformation(V_{attn}^T). This is a scalar quantity.

$$f_{\mathrm{ATT}} = V_{\mathrm{attn}}^T * tanh(U_{\mathrm{attn}} * h_j + W_{\mathrm{attn}} * s_t)$$

$Where,$

$$V_{\mathrm{attn}}^T \in R^d, U_{\mathrm{attn}} \in R^{\mathrm{d*d}}, W_{\mathrm{attn}} \in R^{\mathrm{d*d}}, s_{\mathrm{t-1}} \in R^d, h_j \in R^d$$

To get the Probability distribution we do Softmax:

$$\alpha_{\mathrm{jt}} = Softmax(e_{\mathrm{jt}})$$

$$i.e.\ \alpha_{\mathrm{jt}} = \frac{e^{\mathrm{e_{jt}}}}{\sum_{k=1}^{T_x} e^{\mathrm{e_{kt}}}},\quad such\ that \sum_{j=1}^{T_x} \alpha_{\mathrm{jt}} = 1\ and\ \alpha_{\mathrm{ij}} \geq 0$$

Now, when we know the input, we need to feed Weighted sum combination of input to Decoder.

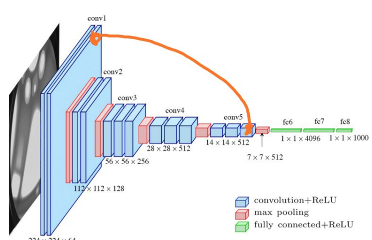
$$C_t = \sum_{j=1}^{T} \alpha_{\mathrm{jt}} h_j \quad such\ that \sum_{j=1}^{T} \alpha_{\mathrm{jt}} = 1\ \alpha_{\mathrm{ij}} \geq 0$$

where, C_t is a context vector, i.e. the Weighted sum of input.

$$S_t = RNN(S_{\mathrm{t-1}}, [e(\hat{y}_{\mathrm{t-1}}), C_t])$$

where, $S_{\mathrm{t-1}}$ is previous state of Decoder. $e(\hat{y}_{\mathrm{t-1}})$ is previous predicted word. C_t is the context vector, i.e. the Weighted sum of input.

Usually for images, we use representations from one of the fully connected layers. But consider a image where a man is throwing a frisbee. When we say the word 'man' that means we need to focus only on man in the image, and when we say 'throwing' then we have to focus on his hand in the image. Similarly, when we say 'frisbee' we have to focus only on the frisbee in the image. This means 'man', 'throwing' and 'frisbee' comes from different pixels in image. Now consider a VGG-16 (pre-trained model used for main approach) representation of man throwing a frisbee below,



Looking at the figure, we can see that the convolution layers do not exactly contain any information in it. However, every location of convolution layers corresponds to some location of image. For example, Consider the output of 5th convolution layer of VGGNet is a 14*14*512 size feature map. This 5th layer has 14*14 pixel locations which corresponds to certain portion in image, that means we have in total 196 such pixel locations. And finally, we can treat these 196 locations (each having 512 dimensional representation). The model will then learn an attention over these locations (Which in turn corresponds to actual locations in images).

Description of code:

In code, we start with creating the utility functions. The purpose of creating the utility functions is to: 1. Load the files. 2. To clean data i.e removing punctuations, single characters, numerical values from text. Next we create dataframe from the raw data and perform some basic exploratory data analysis. After this we performed preprocessing of images and captions. In preprocessing, we are preprocessing the captions (adding '<start>' and '<end>' tags to every caption), so that the ML model understands the starting and ending of each caption.

Next step is to define the pre-trained image model (VGG-16). The VGG16 model was pre-trained on the ImageNet data-set for classifying images. The VGG model contains a convolutional part and a fully connected part which is used for the image classification. Then, we prepared the images and created the image dataset i.e. reshape every image to 224*224*3 shape before feeding it to VGG16 model. For captions, we performed tokenizations where we tokenize the captions and created vocabulary of words present in our data corpus. Then we created vector notations for each word in our vocabulary. For words not appearing in the vocabulary, we gave <unk> notation. After getting sequences to the words in captions, the sequences are of varied length. So, we need pad sequences to pad with the maximum length of the captions. To perform a train-test split, we split the dataset (images and captions) into 80:20 ratio i.e.[train:test]

Implementing Attention Mechanism: The entire step-by-step process of applying Attention according to Minh-Thang Luong's paper: "Effective Approaches to Attention-based Neural Machine Translation":

Global Attention:

1. Producing the Encoder Hidden States: Encoder produces hidden states of each element in the input sequence.
2. Decoder RNN: The previous decoder hidden state and decoder output is passed through the Decoder RNN to generate a new hidden state for that time step.
3. Calculating Alignment Scores: Using the new decoder hidden state and the encoder hidden states, alignment scores are calculated.
4. Softmaxing the Alignment Scores: the alignment scores for each encoder hidden state are combined and represented in a single vector and subsequently softmaxed.
5. Calculating the Context Vector: the encoder hidden states and their respective alignment scores are multiplied to form the context vector.
6. Producing the Final Output: The context vector is concatenated with the decoder hidden state generated in step 2 as passed through a fully connected layer to produce a new output.
7. The process (Steps 2-6) repeats itself for each time step for the decoder until an token is produced or output is past the specified maximum length.

Local Attention:

1. Producing the Encoder Hidden States: Encoder produces hidden states of each element in the input sequence.
2. Calculating Alignment Scores: Between the previous decoder hidden state and each of the encoder's hidden states are calculated.
3. Softmaxing the Alignment Scores: The alignment scores for each encoder hidden state are combined and represented in a single vector and subsequently softmaxed
4. Calculating the Context Vector: The encoder hidden states and their respective alignment scores are multiplied to form the context vector.
5. Decoding the output: The context vector is concatenated with the previous decoder output and fed into the Decoder RNN for that time step along with the previous decoder hidden state to produce a new output.

6. The process (Steps 2-5) repeats itself for each time step for the decoder until an token is produced or output is past the specified maximum length.

Finally comes, Selecting Optimizer, defining Loss Function and setting checkpoints. In training, The Encoder output, the hidden state(initialized to 0) and the Decoder input(which is the <start> token) are passed to the Decoder. The Decoder returns predictions and the Decoder hidden state. The Decoder hidden state is then passed back into the model and the predictions are used to calculate the loss. While training, we use the 'Teacher Forcing Technique' to decide the next input of the Decoder. Teacher Forcing is a technique where the target word is passed as the next input to the Decoder. This technique helps to learn the correct sequence or correct statistical properties from the sequence, quickly. Final step is to calculate the gradient and apply it to the optimizer and backpropogate.

Testing step is similar to training step, it is just that we do not update the gradients, and provide the predicted output as decoder input to next RNN cell at next time steps. Test step is required to find out whether the model built is overfitting or not.

6 Evaluation Metric

The evaluation of Captioning Model: The evaluation function is similar to the training loop, except we don't use Teacher forcing here. The input to Decoder at each time step is its previous predictions, along with the hidden state and the Encoder output.
The following are the two methods to evaluate the captions:

1. Greedy Approach
2. Beam Search

Greedy Approach: Is also called as Maximum Likelihood Estimation (MLE) i.e. we select that word which is most likely according to the model for the given input. And sometimes this method is also called as Greedy Search, as we greedily select the word with maximum probability.
Beam Search: Here we take top k predictions, feed them again in the model and then sort them using the probabilities returned by the model. So, the list will always contain the top k predictions. In the end, we take the one with the highest probability and go through it till we encounter $< end >$ or reach the maximum caption length.

To evaluate the results obtained by the main approach, we will use BLEU Score. BLEU measure will help to evaluate the result of the test set generated captions. BLEU is simply taking the fraction of n-grams in the predicted sentence that appears in the ground-truth.

BLEU is a well-acknowledged metric to measure the similarly of one hypothesis sentence to multiple reference sentences. Given a single hypothesis sentence and multiple reference sentences, it returns value between 0 and 1. The metric close to 1 means that the two are very similar.

7 Results and Analysis

In conclusion, in this project we have successfully seen that baseline model works perfectly fine and it generates proper captions without applying the attention mechanism. After completing the implementation of main approach, we saw that the model generates captions with improved accuracy and scores.

So in all, we can say that the native first-cut model in main approach, without any rigorous hyper-parameter tuning does a decent job in generating captions for images. 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 Neural Network model in the world will give give good performance.

In our main approach, when we compare the results in code for Greedy search and Beam search. We see that Beam Search generated better results than Greedy Search. This might be because beam search makes two improvements over greedy search:

1. With Greedy search, we took just the single best word at each position. In contrast, Beam search expands this and takes the best "N" words.
2. With Greedy search, we consider each position in isolation. Once we had identified the best word for that position, we did not examine what came before it (i.e. in the previous position), or after it. In contrast, Beam search picks the "N" best sequences so far and considers the probabilities of the combination of all of the preceding words along with the word in the current position.

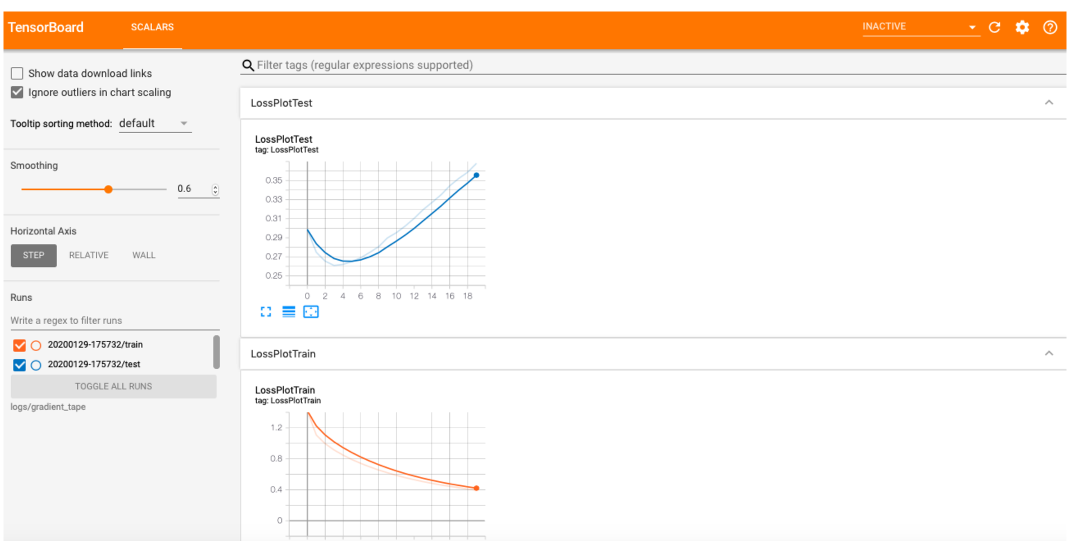
Performance of baseline compared to main approach:

The baseline is short and simple to setup and shows a reasonable chance of providing decent results. The implementation of baseline shows us how hard the problem of image caption generation is. The accuracy scores of BLEU scores shows this difficulty.

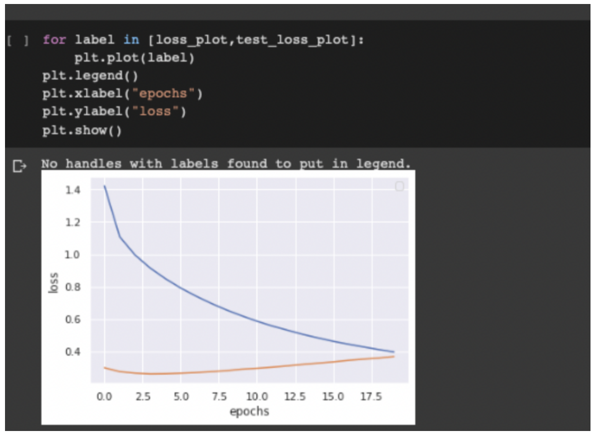
Looking at the BLEU scores of our Baseline and Main Approach, we can say that in average the baseline's BLEU score is 35% comparing this to the average BLEU scores of beam search for all k predictions in main approach i.e. 71.86%. This significant difference in scores of both baseline and main approach shows the power of learning certain features of an image then training a model on those features instead of training a model on a random part of an image. This proves the overall aim of this project.

8 Error Analysis

When training our model in main approach, we used tensorboard logs to visualize changes in our model and training. Tensorboard logs is a tool for providing the measurements and visualizations needed during the machine learning workflow. It enables tracking experiment metrics like loss and accuracy, visualizing the model graph, projecting embeddings to a lower dimensional space.



We plotted the train and test losses to check for overfitting. The following graph shows overfitting. This might be due to less training data. Training the model with larger datasets like MS-COCO or Flickr30 could help us solve this issue.



9 Future Work

There could be many future possible applications for this applications. Image Caption Generators could be used for recommendations in editing applications, usage in virtual assistants, for image indexing, for visually impaired persons, for social media, and several other natural language processing applications.
This is just a particular first-cut solution to the Image Caption Generator and a lot of future modifications can be made to improve this solution. Improvements like:

- Using a larger dataset to improve the accuracy and performance
- Changing the model architecture. Such as adding BatchNormalization Layer, Dropouts, etc.)
- Doing more hyper parameter tuning (learning rate, batch size, number of layers, number of units, dropout rate, batch normalization etc.)
- Keep researching on this topic and optimizing the solution even further.
- API'fy this model using web frameworks like FLASK or DJANGO and deploy it in AWS.

10 Link to Code

Both Baseline and the Main Approach is available on my Github page.

Link: https://github.com/Foster1466/CPTS_534-Neural-Network-Design-and-Application

As datasets are too large to download, I have provided links of dataset in both notebooks. Just for simplicity, I am also providing the links here:

https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip

https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip

References

1. A. Aker and R. Gaizauskas, “Generating image descriptions using dependency relational patterns,” in Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, vol. 49, no. 9, pp. 1250–1258, Uppsala, Sweden, July 2010.
2. Y. Yang, C. L. Teo, H. Daume, and Y. Aloimonos, “Corpus-guided sentence generation of natural images,” in Proceeding of the Conference on Empirical Methods in Natural Language Processing, pp. 444–454, Edinburgh, UK, July 2011.
3. Global Attention : https://arxiv.org/pdf/1508.04025.pdf
4. Local Attention : https://arxiv.org/pdf/1502.03044.pdf
5. BLEU score: https://machinelearningmastery.com/calculate-bleu-score-for-text-python/
6. AppliedAiCourse : https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/4150/attention-models-in-deep-learning/8/module-8-neural-networks-computer-vision-and-deep-learning
7. Neural Machine Translation(Research Paper) : https://arxiv.org/pdf/1409.0473.pdf
8. Image Captioning with Visual attention: https://www.tensorflow.org/tutorials/text/imagecaptioning
9. Blogs : https://towardsdatascience.com/intuitive-understanding-of-attention-mechanism-in-deep-learning-6c9482aecf4f
10. Deep Learning Lectures by Prof. Mitesh M Khapra(IIT Madras)
11. Intuitive Understanding of attention mechanism in deep learning: https://towardsdatascience.com/intuitive-understanding-of-attention-mechanism-in-deep-learning-6c9482aecf4f
12. CS231n: Video by Andrej Karpathy on Image Captioning : https://www.youtube.com/watch?v=NfnWJUyUJYUlist=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC