Image Captioning Project Report

194.077 Applied Deep Learning

Nadia Verdha (11739391)

January 13, 2024

1 Introduction

Nowadays, Computer Vision and Natural Language Processing have taken a huge step forward due to major advancements in the field of Deep Learning and large volumes of data available. Different types of tasks like image classification, object detection, caption generator, sentiment analysis etc., which before were thought impossible, can now be solved by smart models and computers. Having always been fascinated by these two branches, I decided to concentrate for this project on the topic of Image Captioning, which actually lies at the intersection of Computer Vision and Natural Language Processing.

2 Related Work

Show, Attend and Tell by Kelvin Xu et al. [1] which was published back on 2015 served as a good introduction to this topic. It suggests a CNN-RNN(Convolutional Neural Network - Recurrent Neural Network) model for generating image captions for Flickr30k and COCO datasets. Another interesting paper I read was VLP by Zhou Luowei et al. [2] which proposes a unified decoder-encoder model that uses a model pre-trained on another dataset, fine tunes it, and then uses it to generate captions for COCO and Flickr30k datasets.
The first paper, together with this Pytorch tutorial [3] helped me thoroughly during this project since it was my first time working with RNN and attention based models.

3 Problem

As previously mentioned image captioning has been a fundamental and important task in the Deep Learning domain. It is an interesting artificial intelligence problem where a descriptive sentence is generated for a given image. It requires the dual techniques from computer vision to understand the contents of the image and natural language processing to turn the understanding of the image into words that make sense. Image captioning can be used in various applications such as recommendations in image editing applications, in social media and it can bring huge benefits for visually impaired people. Recently, state-of-the-art deep learning implementations have achieved valuable results on solutions for this problem.

[1] https://arxiv.org/pdf/1502.03044.pdf
[2] https://arxiv.org/pdf/1909.11059v3.pdf
[3] https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning

4 Solution

Image captioning is a complex task which requires high-dimensional data and powerful models to be solved. Most image captioning models use an encoder-decoder architecture that allows user to train the model in an end-to-end fashion. Therefore, I also built a similar model for my project.

4.1 Dataset

The most used datasets in the image captioning world include the COCO, Flickr8k and Flickr30k datasets. Since neural networks are famous for very high computation requirements I decided to work with Flickr30k dataset for this project, which contains around 31000 images collected from Flickr, together with 5 reference sentences provided by human annotators for each of these images, and is some kind of a sweet spot between the above mentioned datasets' based on size.

4.2 Model

As mentioned above, my model was composed of an encoder and decoder network with attention. For the encoder part I used an already pre-trained image feature extraction model(CNN) such as ResNet50. The already pre-trained models usually generalize well and can be effectively applied for different tasks. However, the option of fine-tuning it to the dataset was also implemented. The encoder, therefore, is able to generate feature vectors that describe the input images. On the other side, as the decoder a Long Short-Term Memory was used. LSTM produces a caption by generating one word at a time conditioned on a context vector, the previous hidden state and the previously generated words. Due to the fact that RNNs cannot hold too much knowledge and sometimes captions can be too long, using an architecture composed of an encoder-decoder only is not always optimal. Therefore, introducing an attention mechanism between these two networks can considerably enhance the performance of the solution. The attention mechanism takes the outputted image feature vectors from CNN along with a hidden state and assigns a weight to each image pixel. These weights are concatenated with an input word for that time step and are passed to the LSTM. This approach allows the decoder to focus on the most important parts of the image while generating an output sequence. The figure 1 below shows the structure of the model implemented:

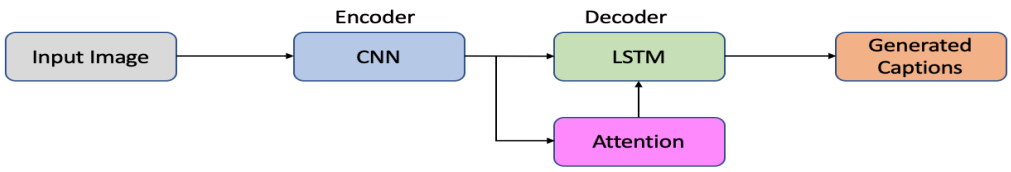

Figure 1: Encoder- Decoder with attention

4.3 Data Preprocessing

Before fitting the data into the implemented models, data preprocessing had to be performed. Images were resized to 256x256 for uniformity and they were also standardized and normalized by mean and standard deviation of the ImageNet images' RGB channels. Captions on the other size were both the input and the target of the Decoder as each word was used to generate the next one. However, in order to generate the very first word we need a zeroth word such as a <sos> token (start of sentence). Moreover, the decoder also needs to learn to predict the last word and that is why we also need another token such as <eos> (end of sentence). This is important because the decoder needs to know when to stop decoding during inference. Since the captions are varying in their lengths, <pad> tokens were also introduced for uniformity. Furthermore, a vocabulary, which is an index mapping for each word in corpus, was created. This is necessary because Pytorch library needs words encoded as indices to look up embeddings for them.

4.4 Training

As mentioned above, training was achieved in and end-to-end fashion. Since the goal is to generate a sequence of words CrossEntropyLoss was chosen as loss function. For evaluating the models the so-called BLEU metric was used which is actually a standard in the image captioning generator architectures. For finding the best model, a pocket algorithm was in place, where every time there was an improvement in BLEU-4 metric a new best model was saved. However, on the other side, if the BLEU-4 metric did not experience an improvement for more than 6 epochs, early stopping was triggered. Worth mentioning is also the fact that Teacher Forcing was used during model training and validation as a very common technique when dealing with RNNs. This is done in a way that at each decode-step the ground truth (target caption) was supplied to the decoder and not the last generated word. In a way this speeds up the traning process and leads,therefore, to faster convergence. However, on the other side the model might end up relying too heavily on teacher forcing and might not perform that well then during inference.

4.5 Inference

A linear layer is used to transform the decoder's output into a score for each word in the vocabulary. The greedy approach would be to choose the word with the highest score and use it to predict the next word. However, this is not optimal, because the rest of the sequence depends on the first word. If that word is not the best, then we end up with a sub-optimal solution. Therefore, for inference Beam Search was implemented. Beam Search is very useful in language modeling sequence because it really finds the most optimal sequence. At the first decode step, k (k=5) top candidates are considered. Then, k second words are generated for the first k words. Next, the top k [first word, second word] combinations are chosen based on additive scores. The same procedure goes on at each decode step. After k sequence terminates, meaning <eos> token is generated, the sequence with the best overall score is chosen.

4.6 Experiments

Since it was my first time working on this kind of project, I implemented various models and learned along the way from my mistakes.
The very first model I implemented was a model with similar parameters as in the paper and it was also not fine-tuned. However, somehow my results were different. A reason for that could be that I used ResNet50 as an Encoder while the paper uses VGGnet. In order to somehow improve the results, I decided to fine-tune the used encoder. As seen in the table below this led to a slight improvement of the results. What was interesting during the training process of model II was that the model improved itself for the first few epochs and then it stagnated. Despite the fact that early stopping was triggered, I decided to continue training the model for a few other epochs with some changed parameters:

- Decreased regularization parameter alpha_c (descreased the strength of regularization of the model)
- Decreased lr_decay_factor which might lead to the model converging more slowly

Even though BLEU-4 of model III seemed to increase during training, this was not the case when evaluating the model on test set.
Soon after fitting model III, I realized that something could be slightly wrong with my implementation. In the beginning I had not sorted captions that were inputted to the Decoder based on their length, and I realized that this might really be important. Sorting captions allows them to be aligned with each-other and leads to the model focusing on important parts and not on the pad tokens.Therefore, I implemented this change in model IV and as a result, there was an increase in the BLEU-4 metric. Last but not least, I decided to train the model again and this time also fine-tune the encoder.Its results are represented in the last row of table below.
The table below summarizes my experiments and displays the performance of the models on Test set:

Models	BLEU-1	BLEU-2	BLEU-3	BLEU-4
M I (w/o fine-tuning Encoder)	54.46	34.82	21.12	12.29
M II (w/ fine-tuning Encoder)	55.78	35.71	22.13	13.82
M III (M II,trained w/ changed params for 3 more epochs)	55.93	34.82	21.12	13.71
M IV (sorted captions)	64.86	41.61	25.43	15.71
M V (sorted captions fine-tuning encoder)	65.27	42.49	26.39	16.43

As seen above, my best model achieved BLEU-4 score of 16.43 which is actually lower than the metric achieved by the implementations of the above mentioned papers, as also shown in the table below:

Implementation	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Show, Attend and Tell	66.7	43.4	28.8	19.1
VLP	-	-	-	31.1
My Best Model	65.27	42.49	26.39	16.43

4.7 Demo Application

Last but not least, the final part of the asignment revolved around building a demo application to show the results achieved in the course of this project. I decided to build a small website application using a very user-friendly library, Streamlit. The demo application works in a way that the user can simply choose and upload a picture taken from the test folder and once he/she has done that, the image together with its generated caption will be shown on the screen. This procedure can go on for as long as the user prefers to do so. The figure 2 below display the small demo application implemented:

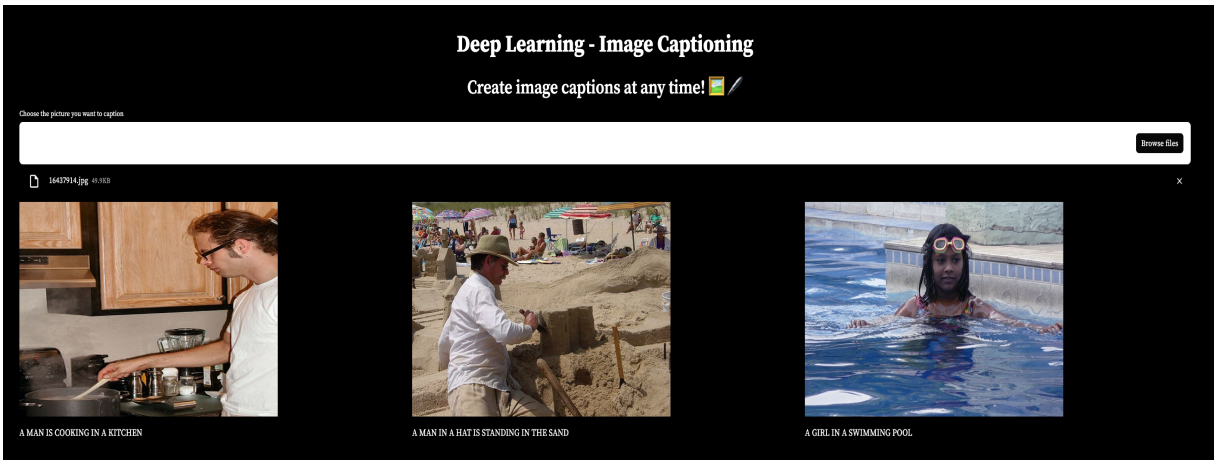

Figure 2: Demo Application

5 Insights and Take-aways

This whole project was a big learning experience for me because it was my first time dealing with RRNs and attention-based models and I definitely learned a lot. One of the main take-aways was that preprocessing the data took a lot more time as expected. Model building on itself was such a difficult and long process which in all honesty would not have been possible without various implementations and tutorials already available online. As in in-depths of the project itself, the actual sorting of the captions before feeding them to the RNN really gave my models the boost they needed, and had I done it in the very beginning, I would for sure have had more time to perform other experiments. Moreover, as there is considerable cristism for BLEU metric as it does not really correlate well with human judgment, I think an opportunity would also be to use METEOR scores for validating and evaluating the models.
All in all, I really enjoyed working on this project and now I am even more interested in getting involved with other deep learning tasks!