



IE 498JS SP2020 Final Project

Show and Tell: A Neural Image Caption Generator

Abid Hossain
Hyunjoon Rhee
Jack Yutong Li
Sejeong Yoo

Abstract

The problem of accurately describing an image combines methods and models popular in both computer vision and natural language processing applications. In this analysis, we implemented a well-known Neural Image Captioning model (Vinyals et. al 2015) on the COCO dataset. The model is trained on Blue Waters with different tuning parameters. Our best model achieved a BLEU-4 score of 42.6, and generated captions that are reasonably accurate and intuitive.

Introduction

Recent advances in deep learning, along with the increase in computational power, has dramatically reshaped the landscape of computer vision and natural language processing problems. Image Captioning is a problem that lies within the intersection of computer vision and natural language processing. Given an image, the model needs to generate an accurate, sensible, and grammatically correct caption for the image. This ability to generate coherent descriptions has wide applications, ranging from image search engines to assisting the visually impaired. Combining both advances in computer vision and machine translation, Vinyals et. al (2015) proposed a generative model based on a deep recurrent architecture to generate image descriptions. They implemented a deep convolutional neural network as an encoder to generate vectorized image features. These features are then served as input into the deep recurrent network, where the model is trained to maximize the likelihood of the target description.

In this study, we implement the proposed model (Vinyals et. al, 2015) to the COCO data set to explore its functionality. The model is trained under different tuning parameters on Blue Waters. Our best model achieved a Bleu score of XX and generated sensible textual description for images in the test set.

This report is conducted in the following order. We give an overview of the model architecture, along with the considered tuning parameters in Section 2. Section 3 describes the preprocessing of the COCO data set, including the image and their corresponding captions. The evaluation criterion and results are presented in Section 4, and the summary of this analysis is given in Section 5.

Model

Most previous solutions tried to stitch together solutions of subproblems: encoding and decoding. However, Vinyals et. al (2015) proposed a single joint model. The inspiration comes from recent advances in machine translation, where given a description in source language S we want to generate a description in language T by maximizing $P(T/S)$.

The goal is to directly maximize the probability of correct description given the image by the following equation:

$$\arg \max_{\theta} \sum_{(I^i, S^i) \in D} \log p(S^i | I^i, \theta) \dots\dots\dots(1)$$

Here, (I^i, S^i) : image, caption pair, D : training dataset, θ : trainable parameters of the model.

We can apply the chain rule of independent probability distribution to the joint probability over the caption (S) :

$$\log p(S|I, \theta) = \sum_{t=0}^N \log p(S_t | I, \theta, S_0, S_1, \dots, S_{t-1}) \dots\dots\dots(2)$$

Here, S_t is the t -th word in caption S .

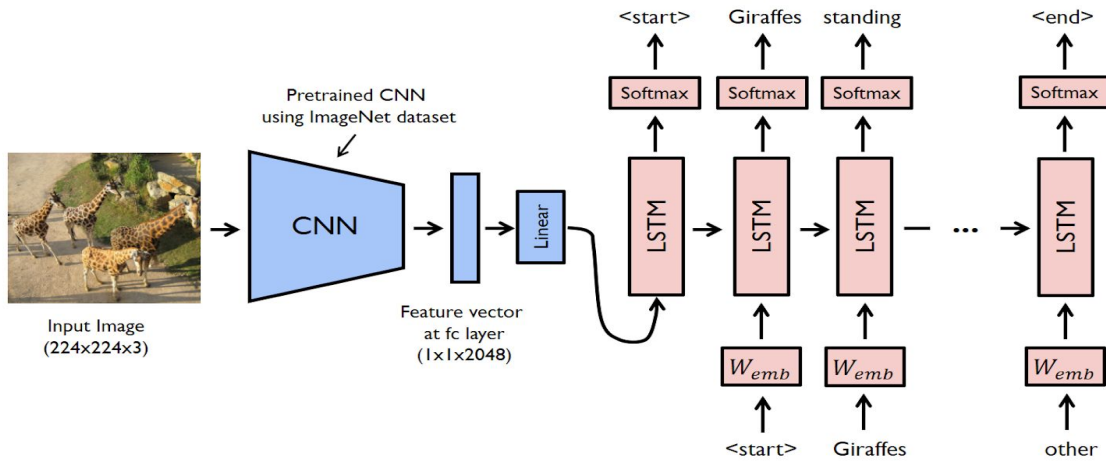
At training time (S, I) is an example training paper and we optimize the sum of log probabilities as described in (2) over the whole training set using stochastic gradient descent.

The paper proposes to replace RNN as an encoder by a deep convolutional neural network (CNN). CNNs are natural image encoders because they can capture spatial relationships.

In our implementation, the Encoder is a resnet-152 model pretrained on the ILSVRC-2012-CLS image classification dataset. They are currently state-of-the-art for object recognition and detection. We use recurrent neural network (RNN) to model $p(S_t | I, \theta, S_0, S_1, \dots, S_{t-1})$, which has a memory. The memory is updated after seeing a new input x_t by a nonlinear function f :

$$h_t = f(x_t, h_{t-1}) \dots\dots\dots(3)$$

For f we use long-short-term-memory (LSTM), which shows state-of-the-art performance on sequence tasks. The following image describes overall flow of our model:



High-level Overview of the Model Architecture proposed in Vinyal et. al 2015

Data Set

We used the MSCOCO dataset which has 82783 training images and 40401 validation images. Each image comes with 5 human captions. Since human captions are of different length we

padded them to make them of the same length and introduced ‘start’ and ‘end’, and ‘unknown’ for less used words in vocabulary.

For data augmentation, we used random crop (224, 224), horizontal flip, and normalization . We also convert all images to RGB format to have 3 channels which is needed for CNN encoder.

Data Preprocessing

Dataset Statistics

Table 1 shows the statistics of the image datasets used in this project. For each image in the dataset, we have 5 captions.

Dataset	Size (MODIFY: Training + Validation Only)		
	Train	Validation	Test
MSCOCO	82,783	40,504	40,775

Table 1: Statistics of the datasets used in this project

How to process images

For the Encoder part, we used ResNet152, we processed images such that they are accustomed to the input of ResNet152, which is (3, 256, 256). Moreover, we converted the pixel values to be in the range [0,1], and then normalized the image by the mean and standard deviation of the ImageNet images' RGB channels, that is mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] for each channel.

PyTorch follows the NCHW convention, where N denotes number of samples in a batch, C denotes the number of channels, H denotes the height, and W denotes the width of an image. Therefore, we followed the convention and preprocessed our input data accordingly.

How to process captions

Captions are used as input and output of the Decoder, because each word is used to generate the next word. We followed the convention of the language model, and added the <start>, and <end> symbol at the start and end of each caption. Moreover, we limited the maximum length of each caption to 50 and added the <pad> symbol to those captions that are shorter than 50.

Training Method

The training process can be divided into two stages: The CNN part which encodes the image, and the LSTM network which generates caption. The loss is calculated after the caption generated for each training pair (I,S).

Encoding:

We use the pre-trained resnet-152 model which has been trained on ImageNet dataset. We fine tune it with adding one more linear layer at the end.

Decoding:

We use the lstm layer in order to train the model. The dropout rate that we have used is 0.5. We receive an output through the model, and after receiving the output, we flatten into batch size. The model has used Adam as the optimizer for both encoding and decoding part of the process.




Analysis and Results

In this section, we present the results of our implemented Neural Image Captioning model on the Flickr30 and COCO dataset.

Hyperparameters:

In order to predefine all the hyperparameters before starting the training process, we have chosen to crop the model in order to normalize the image, which was 224, with a batch size of 128. The learning rate for the Adam optimizer was 0.001.

For the decoder, we have chosen to use the embedding size as 512, and same for the hidden layer size as well. We have used different hidden layers and embedding size for training, and the results were different with these different hyperparameters. One of the hyperparameters that we have compared was the number of layers of the model. The layers we experimented with are 2, 3, 5, and 7. Another hyperparameter that we varied was the epochs. We compared the difference in the loss and the results through training the model through different numbers of epochs.

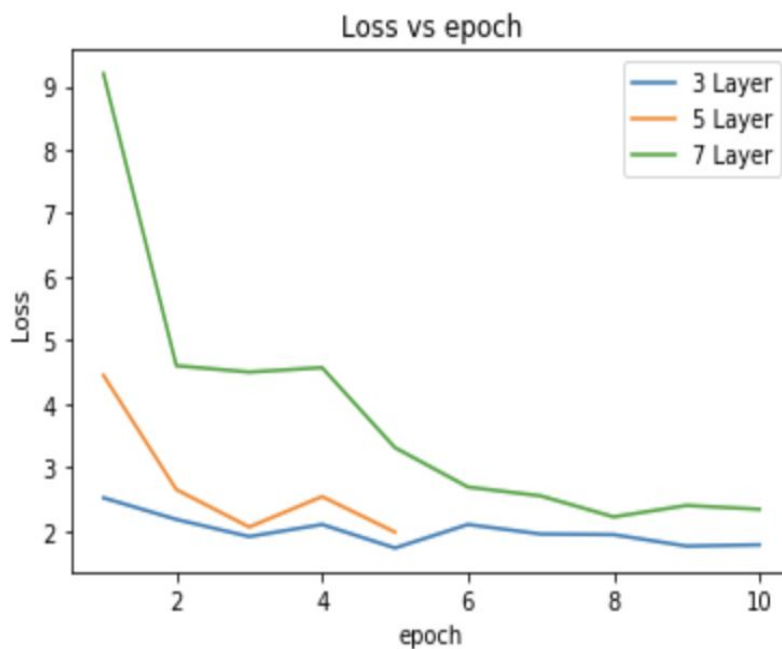
Correct caption	Somewhat related caption	Completely unrelated caption
		

<p>2 layers, 5 epoch: “<<start>> a group of young children playing soccer on a field . <<end>>”</p> <p>5 layers, 5 epoch: “<<start>> a group of people playing soccer on a field . <<end>> ”</p> <p>3 Layers, 10 epoch: “<<start>> a group of young boys playing soccer on a field . <<end>>”</p> <p>7 layers, 10 epoch: “<<start>> a group of people playing soccer in a field .<<end>>”</p>	<p>2 layers, 5 epoch: “<<start>> a group of people standing around a table with a cake . <<end>>”</p> <p>5 layers, 5 epoch: “<<start>> a group of people standing around a table . <<end>> ”</p> <p>3 Layers, 10 epoch: “<<start>> a group of people standing around a luggage carousel . <<end>>”</p> <p>7 layers, 10 epoch: “<<start>> a man is sitting at a table with laptop . <<end>>”</p>	<p>2 layers, 5 epoch: “<<start>> a man is cooking in a building with a crowd of people . <<end>>”</p> <p>5 layers, 5 epoch: “<<start>> a man in a kitchen preparing food in a kitchen . <<end>>”</p> <p>3 Layers, 10 epoch: “<<start>> a group of people standing in front of a fire hydrant . <<end>>”</p> <p>7 layers, 10 epoch: “<<start>> a man standing in a kitchen with a refrigerator . <<end>>”</p>
---	---	--

Table 2: Some example of captions

Training and validation result:

From the plot of loss vs. epoch, it's clear that 3-lstm layer has the least loss.



Model Architecture	7-layer, 10-epoch, hidden size = 512	2-layer, 10-epoch, hidden size = 512	3-layer, 10-epoch, hidden size = 512	5-layer, 5-epoch, hidden size = 256	Theoretical
Bleu-1	0.645	0.699	0.696	0.668	0.642
Blue-2	0.444	0.504	0.498	0.468	0.447
Bleu-3	0.395	0.428	0.423	0.407	0.385
Bleu-4	0.418	0.431	0.426	0.423	0.401

Table 3: Bleu scores on validation dataset according to different model

- Theoretical bleu score is computed by comparing one human caption to other's from the list of 5 captions per image. It is the same for all model architecture.
- the 3-lstm-layer-model performs best.

Dataset	BLEU1 Our code paper	BLEU2 Our code paper	BLEU3 Our code paper	BLEU4 Our code paper
COCO	69.6 NA	49.8 NA	42.3 NA	42.6 27.7

Table 4: Model score compared to paper (3-layer, 10-epoch, hidden size = 512)

Computation Time

Our model was trained and tested on Bluewaters. Specifically, we used the xk-nodes of Bluewaters.

More information on xk node: 8 Bulldozer core/node.

CPU characteristics: 2 CPU/core (16 CPU/node), 156.8 GF peak CPU performance, 51.2 GB/s CPU memory bandwidth.

GPU characteristics: 2688 CUDA cores, 1.31 TF peak GPU performance, 250 GB/s peak GPU memory bandwidth.

1. For 3-lstm layer and 10-epoch, 512 hidden and vocabulary size the training took ~12.6 hour on 4 xk nodes with 16 processes per node. So, total training time is $4 \times 16 \times 12.6$ hrs = 806.4 hours.

2. For 7-lstm layer and 10-epochs, 512 hidden and vocabulary size, the training took ~20 hours on 4 xk nodes with 15 processes per node. Total training time = $4 \times 16 \times 20 = 1280$ hours.

Discussion and Conclusion

In this section, we summarize and discuss the findings of our analysis.

Collecting all the results of the model, we can conclude that we have successfully implemented the model of the paper. After trying multiple layers of the model with different epochs, we could conclude that 2 layers of LSTM model showed the most accuracy in generating captions with high BLEU scores. Especially, with BLEU-4, which the paper has mentioned, we have out-performed what the paper has by almost 1.5 times the score. From the BLEU score that we have calculated we can see the downward trend of the accuracy as the number of layers increases. This may be the resultant of overfitting. 7-layers model was showing a huge overfitting issue. All the other layers that we have tried were successful in generating captions after 5 epochs, but for 7 layers 5 epochs were not sufficient. We had to run 10 epochs to generate captions, but the accuracy was low compared to the other layers. Besides the overfitting issue, layers were able to perform similarly to the expected value, or even performed better than the value we were expecting. Despite the fact that there are some pictures that are generated with lack of similarity, the overall performance of the model and the training showed high accuracy in general.

In conclusion, with numerous trials with different numbers of hyperparameters such as number of layers of LSTM, epochs, and hidden layer, we could show high performance in overall results of calculating BLEU scores. Especially when comparing the value of BLEU-4, which is mentioned in the paper as 27.7, we have been able to score 42.6, which is much higher than the expectation. The computational time was at max 1280 node hours and much less with fewer LSTM layers. With such results, we conclude that we have successfully reached our objective of calculating the BLEU scores.

References

1. Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
2. <https://medium.com/explorations-in-language-and-learning/metrics-for-nlg-evaluation-c89b6a781054>
3. <https://github.com/kelvinxu/arctic-captions>