ECE 285 - Fall 2019 - Final Project - Team SaaS
IMAGE CAPTIONING

Aparna Srinivasan
A53309738
a2sriniv@ucsd.edu

Anirudh Swaminathan
A53316083
aswamina@eng.ucsd.edu

Savitha Srinivasan
A53302260
sasriniv@eng.ucsd.edu

Sidharth Suresh
A53323680
sisuresh@ucsd.edu

Github repository: https://github.com/Anirudh-Swaminathan/ece_285_fa19_project

The trained models can be found here: trained models

Introduction

Image Captioning is the process of generating textual description of an image. It is one of most important problem statements that involves both computer vision and natural language processing. The goal of image captioning involves detecting and recognizing objects, understanding the scene type or location, and learning object properties in order to generate captions that are as descriptive of an image as possible. Further, understanding both the synaptic and semantic aspects of the language is essential to meaningful captions (1).

There are innumerable applications to image captioning in the fields of bio-medicine, commerce, military, education, digital libraries, and web searching and this list not exhaustive. Today, social media platforms can also directly generate descriptions from images that are posted (2). In general, there are two paradigms to image captioning – "top down" and "bottom up". The top-down paradigm starts from a "gist" of an image and converts it into words, while the bottom-up one first comes up with words describing various aspects of an image and then combines them. In this project, the top-down approach proposed by Vinyals et al. (3) is implemented. Further, various experiments are carried to analyse the performance of the image captioning system.

Implementation technique

0.1 Overview

The task of image captioning can be divided into two modules logically – 1) is an image based model which extracts the features and the nuances out of the image, and 2) the language based model which translates the features and objects given by the image based model to a natural sentence. The image based model, which is also known as the encoder is usually a Convolutional Neural Network (CNN), whereas, the language based model, which is usually known as the decoder is usually a Recurrent Neural Network (RNN). 1.

Usually the CNN at the encoder is a pretrained model. This implies that the parameters of the models have already been trained for another task (eg. image classification) and the learned parameters are used for the image captioning task. This is called transfer learning. The last layer of the pretrained CNN model, which would be a softmax layer is removed in order to extract the hidden representation. This hidden representation along with the embedded caption is used as the input to the decoder, which in turn generates the caption. So, in general, only the decoder is trained.

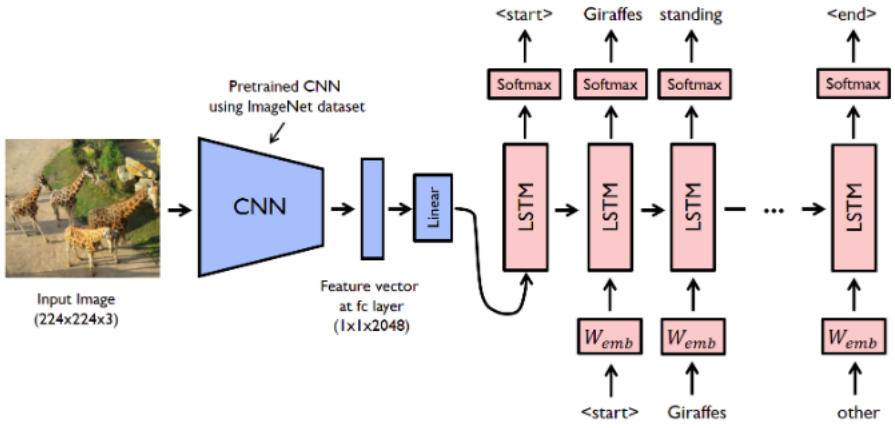

Figure 1: Block Diagram showing an overview of image captioning

0.2 CNN

There are various CNN models that can be used for generating the hidden representations of the image including VGG-16 (4), Alexnet (5) , GoogLeNet (6) etc. The CNN model generally consists of repeating blocks of convolution, activation/non-linear transformation and pooling. These models are capable of learning the local spatial information from images.

0.3 RNN

The RNN is used for learning temporal information and predicting a time series. Especially, the improved version of RNN called the Long Short Term Memory (LSTM)(7) units have the capability to learn patterns in both long and short sequences. Furthermore, LSTMs also overcome the problem of vanishing gradient by using its forget gate.

The LSTM unit has the memory cell c which encodes knowledge at every time step of what inputs have been observed until that step. The three gates namely the input, output and the forget gate are used to check whether c should forget the the current cell value, if it should read its input (input gate i) or whether to output the new cell value (output gate o). The cell updates and gate definitions are given below:

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1}) \tag{1}$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1}) \tag{2}$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1}) \tag{3}$$

$$c_t = f_t \times c_{t-1} + i_t \times h(W_{cx}x_t + W_{cm}m_{t-1}) \tag{4}$$

$$m_t = (o_t \times c_t) \tag{5}$$

$$p_{t+1} = linear(m_t) \tag{6}$$

Here, W are the weights of the respective gates. These multiplicative gates are used to train the LSTM and they also take care of the exploding and vanishing gradient problem. The output m_t from the LSTM is fed into a linear layer to predict the captions.

With respect to image captioning, where the goal is to maximize the probability of the correct description given the image can be formulated as follows:

$$\theta* = argmax_\theta \sum_{I,S} logP(S|I;\theta) \tag{7}$$

where θ are the parameters of the model, I is an image, and S its correct transcription.

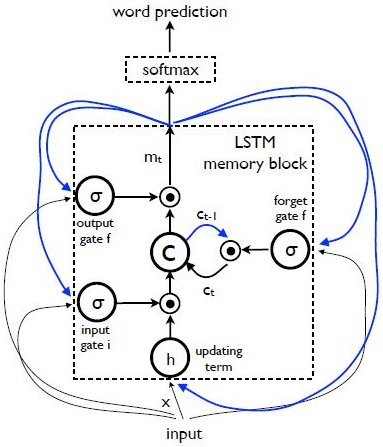

Figure 2: LSTM Memory Block

Since S represents any sentence, its length is unbounded. Thus, it is common to model the joint probability over S0, ..., SN where N is the length of this particular caption as:

$$logP(S|I;\theta) = \sum_{t=1}^{n} log(S_t|I, S_0, S_1...S_{t-1};\theta) \tag{8}$$

For a given training sample pair (S,I), the sum of the log probabilities is optimized using different techniques.

The LSTM then models the joint probability distribution, and the predictions are obtained from a linear layer.

0.4 Dataset

The MSCOCO dataset is used to train and evaluate the model. COCO is a large-scale object detection, segmentation, and captioning dataset, which has the following data:

- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image

The dataset contains 120,000 images in total. The dataset is split into 80,000 and 40,000 for training and validation respectively, and each image has 5 captions. (8)

0.5 Creation of vocabulary

A counter consisting of words and their number of occurrences was created using all the captions in the COCO dataset. In order to remove words that are less likely to occur in the caption, words with frequency of less than 5 were removed. Further, the following strings were added to the counter - the strings “start_vec" denoting the start of a caption, “end_vec" denoting the end of a caption, “unk_vec" denoting any unkown words that may be present in a validation set, but not present in the counter and “pad_vec" for padding the captions to the same length. Finally, a dictionary which contained a unique integer id as a key for each word was created.

0.6 Evaluation Metric

To evaluate the algorithm , the Bilingual Evaluation Understudy algorithm, or BLEU score was used. A set of high-quality human translations are obtained for a given piece of text, and the machine's translation is compared against these human baselines, section by section at an n-gram level. Typically an output score of '1' matches perfectly with the human translations, and a '0' means that the output sentence is completely unrelated to the ground truth. The most representative within Machine Translation and Image Captioning include: BLEU 1-4 (n-gram with n=1-4)(9), CIDEr (10), ROUGE-L (11) and METEOR (12). These approaches are quite similar in that they measure syntactic similarities between two pieces of text, while each evaluation metric is designed to be correlated to some extent with human judgement. The BLEU scorer that was officially provided by the MSCOCO-Dataset team was used for this project directly, with a few modifications. In particular, the code from Python2 was changed to Python3.

Experimental Setup

All experiments were carried out using the default train, validation and test split of the COCO-2014 dataset. The dictionary of words and their unique ids were created from the train set. The encoder-decoder model was implemented for a variety of pretrained encoder architectures namely – 1) VGG-16, 2) GoogLeNet, and 3) Alexnet. The last layer of the pretrained model was replaced with a trainable fully connected layer with 512 units and linear activation. The architecture of the decoder included an Embedding layer, LSTM and a fully connected layer. The number of embeddings was set to the size of the vocabulary, i.e. 8844 and the embedding size was set as 512. The LSTM layer had 512 hidden units and tanh activation. Finally, the fully connected layer had 8844 units and linear activation. The LSTM layer used is the standard LSTM module provided by PyTorch.

The combination of VGG-16 as the encoder and Adam optimizer was chosen as the baseline. The experiments were carried out for other optimizers including – 1) Stochastic Gradient Descent (SGD) without any momentum,2) SGD with momentum 0.9 and 3)SGD with momentum 0.9 and Nesterov acceleration. For all experiments, the cross entropy loss as the cost function and the BLEU score as the performance metric. The models were trained for a maximum of 5 epochs with batch size of 128.

All experiments were performed using Pytorch 1.3.1 and Python 3.7. Further, the package *nntools.py* containing the mechanisms of abstract classes and inheritance and provided in the course, was used for defining the neural network, loss function and the training and evaluation of the dataset. A modified version of it, called *nntools_modified.py* was created, to save losses on a per mini-batch basis, and we also modified it to save the validation losses as well, apart from defining our own CaptionStatsManager that computed the BLEU1-4 scores in addition to the loss.

Results and Discussions

0.7 Sample output from the baseline model

Figure 3 shows a pair of test images and their corresponding captions generated by the baseline model (VGG-16 + LSTM + Adam Optimizer). It is observed that the caption generated for the first image is very similar to the ground truth, while the second one shows a case when the model fails.

0.8 Comparison of Optimizers

Figure 4 - 7, show the cross entropy loss and the BLEU scores at the mini-batch level.

Considering the plots corresponding to the loss it is observed that the loss converges the fastest when Adam optimizer is used. This could be attributed to the adaptive learning rate that is computed at the mini-batch level. The adaptive learning rate is a function of the squared gradient. Consequently, whenever the gradient is larger the learning rate is increased and vice versa. SGD with its constant learning rate would require more epochs for the loss to converge as shown in the figures.

With respect to the variants of SGD, it is observed that learning rate decreases much quicker when SGD is used with momentum than with just pure SGD. This is because momentum helps in accelerating the SGD in the relevant direction and dampen any oscillations. SGD with Nesterov

Figure 3: Generated caption for given images.

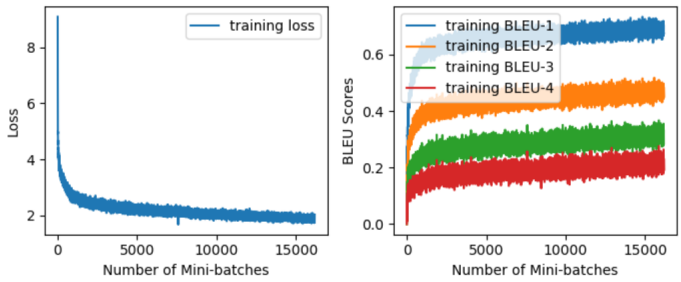

Figure 4: Training loss and BLEU score at the mini-batch level for 5 epochs, when using Adam optimizer.

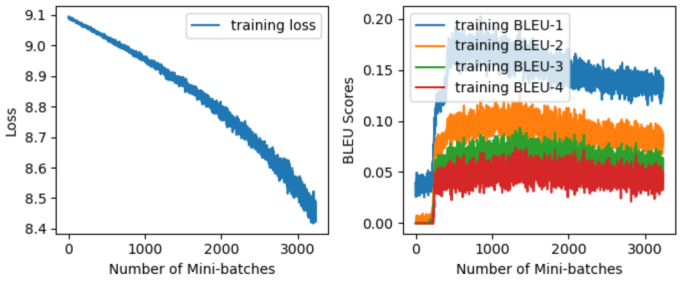

Figure 5: Training loss and BLEU score at the mini-batch level for 1 epoch, when using SGD optimizer.

accelerated gradient provides a slightly better convergence rate than standard momentum due to the "look ahead" calculations of gradients.

Similarly, considering the BLEU scores for the different optimizers it is observed that it is the highest for Adam optimizer and the least for vanilla SGD. Further, it is also observed that in general BLEU-1 > BLEU-2 > BLEU-3 > BLEU-4. This could be attributed that the fact as the number of n-grams considered increases from BLEU-1 to BLEU-4, the similarity would also decrease.

	Epochs	**Loss**	**BLEU - 1**	**BLEU - 2**	**BLEU - 3**	**BLEU - 4**
Adam	5	2.3566	0.6727	0.4455	0.2828	0.1862
SGD	1	8.4492	0.1360	0.0840	0.0594	0.0452
SGD+Momentum	1	4.7064	0.2620	0.1707	0.1152	0.0821
SGD+Momentum+Nesterov	4	3.9406	0.4077	0.2742	0.1777	0.1176

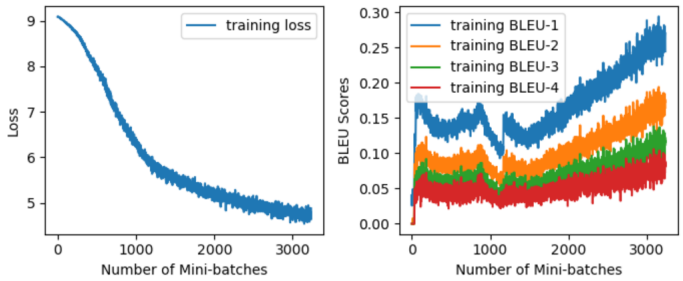

Figure 6: Training loss and BLEU score at the mini-batch level for 1 epoch, when using SGD optimizer with momemtum of 0.9.

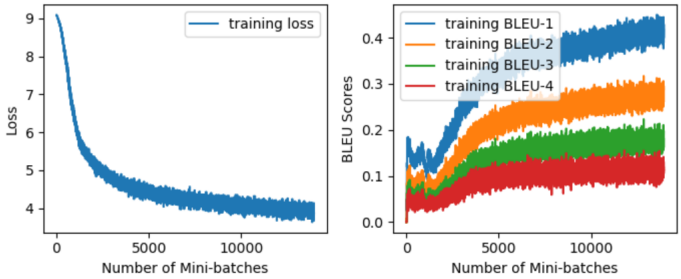

Figure 7: Training loss and BLEU score at the mini-batch level for 4 epochs, when using SGD optimizer with momemntum of 0.9 as Nesterov acceleration.

0.9 Comparison of encoder models

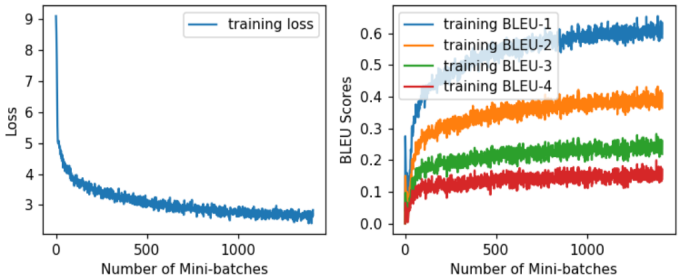

Figure 8: Training loss and BLEU score at the mini-batch level for 1 epoch, when using AlexNet for the encoder, and Adam optimizer.

Figure 4, 8 and 9 shows the training loss and BLEU curves for VGG-16, Alexnet and GoogleNet as encoder models. From the plots, it is observed that AlexNet and VGG16 have reach similar losses and BLEU scores for the same number of mini-batches. GoogleNet seems to perform better than these two architectures, as can be seen from the plot. This might be attributed to the deeper architecture of GoogleNet that can extract rich hidden representations from the input image.

Conclusion

The “Show and Tell: A neural image caption generator" by Vinyals et al. was implemented in Pytorch. In addition to the approach proposed by them, experiments were performed to analyse the impact of different optimizers and pretiained encoders on the performance of the image captioning system. It was observed that by using Adam optimizer, the loss converged much faster than any variants of SGD. This is in corroboration with the theoretical support for the same. Further, GoogleNet was observed to perform better than VGG-16 and Alexnet, which could be attributed to its deeper architecture.

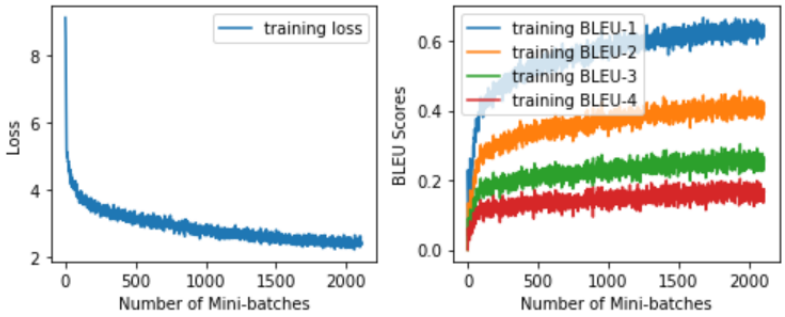

Figure 9: Training loss and BLEU score at the mini-batch level for 1 epoch, when using GoogleNet for the encoder, and Adam optimizer.

References

[1] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," *CoRR*, vol. abs/1810.04020, 2018. [Online]. Available: http://arxiv.org/abs/1810.04020

[2] C. Chunseong Park, B. Kim, and G. Kim, "Attend to you: Personalized image captioning with context sequence memory networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[3] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," 2014.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[8] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[9] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318. [Online]. Available: https://doi.org/10.3115/1073083.1073135

[10] R. Vedantam, C. L. Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation," *CoRR*, vol. abs/1411.5726, 2014. [Online]. Available: http://arxiv.org/abs/1411.5726

[11] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://www.aclweb.org/anthology/W04-1013

[12] A. Lavie and A. Agarwal, "Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments," in *Proceedings of the Second Workshop on Statistical Machine Translation*, ser. StatMT '07. Stroudsburg, PA, USA: Association for Computational Linguistics, 2007, pp. 228–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=1626355.1626389