Neural Networks for Image Captioning

Neural Net Ninjas

Asfiya Baig, Balaji Shankar Balachandran, Sushruth Nagesh

A53312622, A53307326, A53314208

Department of Electrical and Computer Engineering

University of California, San Diego

San Diego, CA 92122

`(asbaig,bbalacha,snagesh)@eng.ucsd.edu`

https://github.com/sushruthn96/Image_Captioning_ML_IP

Acknowledgement - Our team would like to extend utmost gratitude towards Prof. Charles Deledalle for his guidance throughout the course. We would also like to thank Raghav K Subramanian for his mentorship during the course of the project.

Abstract

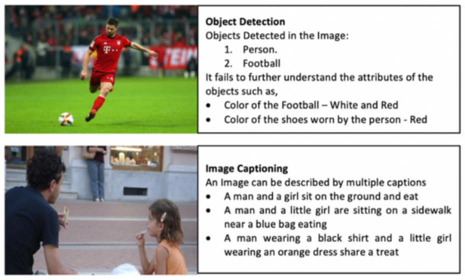
Image captioning using neural networks is used to generate natural sentences describing an image. It uses both Natural Language Processing and Computer Vision to generate the captions. The model used in this project is based on "Show and Tell: A Neural Image Caption Generator" [1]. The model consists of Convolutional Neural Network(CNN) as well as Recurrent Neural Network(RNN). The CNN is used for feature extraction from image and RNN is used for sentence generation. The model is trained in such a way that if input image is given to model it generates captions which nearly describes the image. The model is tested on MSCOCO-2015 test set.

1 Introduction

Humans can construct a concise description of an image in the form of a sentence relatively easily. Such descriptions might identify the most interesting objects, what they are doing, and where this is happening. These descriptions are rich, because they are in sentence form. Such visually descriptive language is potentially a rich source of information about the world, especially the visual world. It will be of great help if machines learn this visually descriptive language based on images. Recently, deep learning methods have achieved state-of the-art results on examples of this problem. It has been demonstrated that deep learning models are able to achieve optimum results in the field of caption generation problems.

1.1 Motivation

Captioning has always been important, but with changing technology, the rise of video popularity, and the growth of the deaf and hard of hearing population, it's become more pertinent than ever before. Today, there's more video uploaded to the web in one month than television has created in the past thirty years. The importance of captioning lies in its ability to make video more accessible in numerous ways. It allows deaf and hard of hearing individuals to watch videos, helps people to focus on and remember the information more easily, and lets people watch it in sound-sensitive environments.



Object Detection
Objects Detected in the Image:
1. Person.
2. Football

It fails to further understand the attributes of the objects such as,
- Color of the Football – White and Red
- Color of the shoes worn by the person - Red

Image Captioning
An Image can be described by multiple captions
- A man and a girl sit on the ground and eat
- A man and a little girl are sitting on a sidewalk near a blue bag eating
- A man wearing a black shirt and a little girl wearing an orange dress share a treat

Figure 1: Object Detection and Image Captioning Tasks.

2 Methodology

Two crucial design choices have to be made to materialize the RNN: what is the exact form of f and how are the images and words fed as inputs x_t. Based on the state-of-the art performance on sequence tasks such as translation, we choose Long-Short Term Memory (LSTM) net for f. This model is outlined in the sub section. For the representation of images, we use a Convolutional Neural Network (CNN).

2.1 Recurrent Neural Networks

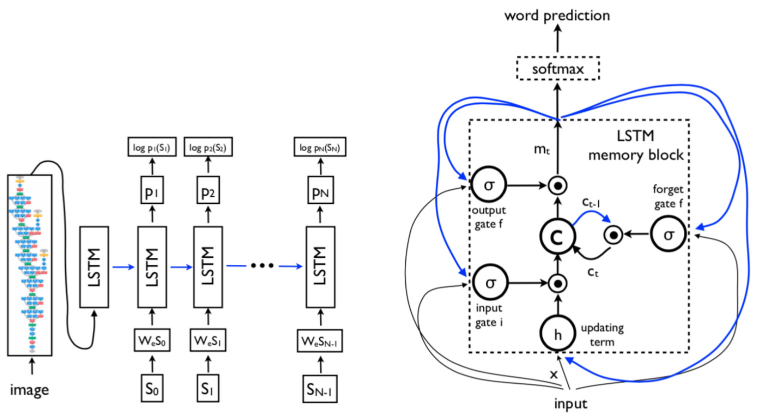
An RNN in general is a neural network designed to handle sequences of variable size. They incorporate internal feedback loops that fed the net's output back into the next along with new input, and have achieved state of the art results on time series prediction, especially in the domain of natural language processing. The RNN's major challenge is the ability to learn long-term dependencies.

2.2 Long Short Term Memory(LSTM)

Long Short Term Memory networks (LSTMs) are special RNNs, capable of learning long-term dependencies. RNNs struggle with remembering information for a very long time and have the problem of vanishing and exploding gradients, which results in complexity during training. LSTMs use structures called gates to regulate the flow of information to memory cells, which encode the inputs observed at every time step till the current step. Gates are usually composed of a sigmoid layer with an output between 0 to 1, with 0 representing "no information through" and 1 representing "let all information through". There are three gates: Input (i), Output (o) and Forget (f) gates used to control if new input can be read, new cell value can be given as output, and whether the cell state can be forgotten/has to be retained. Let represent the sigmoid function and h represent the tanh function, and let $\odot$ represent the element-wise product between two matrices. Also, assume that W_{ij} represents trained parameters as part of the LSTM.

$$
\begin{gathered}
i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1}) \\
f_t = \sigma(W_{ix}x_t + W_{fm}m_{t-1}) \\
\sigma_t = \sigma(W_{ox}x_t + W_{om}m_{t-1}) \\
c_t = f_t \odot c_{t-1} + i_t \odot h(W_{cx}x_t + W_{cm}m_{t-1}) \\
m_t = o_t \odot c_t
\end{gathered}
$$

There are several other variants to LSTMs such as the Gated Recurrent Unit (GRU), Depth Gated RNNs, Clockwork RNNs etc. but these overall they help learn long term dependencies using different approaches.



(a) Encoder-Decoder network (b) LSTMs.

Figure 2: Show and Tell - Building blocks.

2.3 Show and Tell Model

2.3.1 Overview

This paper showed that we get State of the Art (SOTA) results when we directly maximize the probability of the correct description given the Image. Assuming that the Image is represented by I, denotes the parameters of our model and S is the correct transcription, we can find optimal parameters $\acute{\theta}$ for the model such that:-

$$\acute{\theta} = argmax_{\theta} \sum_{(I,S)} log(p(S|I;\theta))$$

To simplify the above expression, we can remove θ for convenience. Assuming that N is the length of this sentence:-

$$\log\left(p(S|I;\theta)\right) = \log\left(p(S|I)\right) = \sum_{t=0}^{i=N} \log\left(p(S_t|I, S_0, S_1, ..., S_{t-1})\right)$$

As you can see, the sum of log probabilities is optimized here over the training set. The probability of every word S_t , given that that were generated before it are $S_{t1}, ..., S_2, S_1$ is summed here. The LSTM model described in Section 2.2 is used here. This model provides output mt that goes through a softmax layer, and gives us p_{t+1} - a probability distribution over all words in the dictionary. The best next word is selected and used as part of the caption.

$$p_{t+1} = Softmax(m_t)$$

The Convolutional neural network used here is the GoogleNet. The LSTM predicts each word of the sentence from the Image Embedding, and it is useful to create a copy of the LSTM the image, for each sentence word. All LSTMs in the above image have same parameters, and output of one LSTM is the input of the next. Therefore, if we assume that I is the input image, and S=($S_0, S_1, ...S_N$) represents the caption where S_0 and S_N are represented by a special start and stop token: CNN Embedding is the initial state of the first LSTM

$$x_{-1} = CNN(I)$$

Use same word Embedding W_e

$$x_t = W_e x S_t,\ t\,\epsilon 1, 2, ..., N-1$$

Probability is given by output of last LSTM

$$p_{t+1} = LSTM(x_t),\ t\,\epsilon 1, 2, ..., N-1$$

We only use the Image I once here, and the word embedding W_e is the same for all t, thus mapping words and the image to the same space. The word embedding vectors here are independent of size of dictionary as opposed to a one hot embedding, where length of the word is the size of the dictionary and these can be jointly trained with the model.

2.3.2 Loss function

Loss function used here is the negative log likelihood of correct word at each step and loss is minimized by using stochastic gradient descent with fixed learning rate, random weight initialization and no momentum.

$$L_{I,S} = -\sum_{t=1}^{N} log p_t(S_t)$$

3 Experimental setting

3.1 Evaluation metrics

We use BLEU score to evaluate the model performance quantitatively. BLEU score works by counting matching n-grams in candidate translation to n-grams in reference text.

3.2 Dataset

The dataset used in this study is the MSCOCO-2015 dataset. It contains the training, validation and test sets. The training and validation sets have 5 independent human generated captions describing each image. The test dataset does not have corresponding ground truth captions. The images were gathered by searching for pairs of 80 object categories and various scene types on Flickr. The goal of the MS COCO image collection process was to gather images containing multiple objects in their natural context. Given the visual complexity of most images in the dataset, they pose an interesting and difficult challenge for image captioning [4].

3.3 Training details

Our baseline model consists of two parts -

i) Encoder

ii)Decoder

For the Encoder we have used the ResNet CNN network. In particular, we are using ResNet-152. We deleted the last fully connected layer of the Resnet 152 network and added a fully connected layer with output size equal to the word embedding size. We also used a batch normalization layer after the last FC layer.

Before feeding the images to the ResNet CNN network, image pre-processing is done. We are resizing the images from MS COCO dataset to size (224,224) since each of the images have different sizes. This is followed by random crops of the images and normalizing the images using the mean and standard deviation of ImageNet Dataset. These parameters can be considered as universal standard parameters as ImageNet dataset is huge and has over 14 million images.

The decoder network consists of an embedding layer connected to an LSTM layer. The baseline model consists of random weights for different features of the words in the dictionary. The words of the dictionary are derived from the training image captions. The words are extracted once for all and stored in a pickle file for further use. Further, we are using 256 as size of embeddings for each word in the dictionary.

The LSTM layer has 512 features in its hidden state. We are not stacking up LSTM layers and using only one. Finally, we use a fully connected layer to predict the probability for each of the words in the vocabulary. We are also using Adam optimizer without any weight decay in our baseline model. Learning rate is fixed at 0.001. Our baseline model can be summarized as <ResNet-152, random word embeddings, LSTM with 512 hidden feature size and Adam Optimizer>.

Our experiments are about varying the above mentioned network architectures, parameters to obtain better accuracy.

3.4 Training Experiments

Training experiments of 4 types were performed –

3.4.1 CNN architecture of the Encoder

3 pre-trained models were used for the training process: ResNet-152, VGG16, and Inception-v3. VGG16 network gives a very good accuracy on ImageNet dataset. But it is computationally expensive due to higher memory and time requirements - an effect of using large kernel sizes. Inception-v3 solves this problem by approximating sparse CNN by normal dense construction. It achieves better accuracy than VGG16 and is also faster than VGG16 on the ImageNet dataset. ResNet-152 uses residual units and has very deep layers. Therefore, it achieves better accuracy compared to the Inception-v3 and VGG16 network architectures.

3.4.2 Types of RNN Layers

3 different intermediate RNN layers were used for the training process - the LSTM layer, GRU layer and multi-layer Elman layer. The Elman multi-layer RNN handles variable-length sequential input using a recurrent, shared hidden state. But it suffers from the vanishing and exploding gradient problem during training. GRU's and LSTM's have the same objective of tracking long-term dependencies effectively while mitigating the vanishing/exploding gradient problems. The LSTM does so with the help of the input, forget, and output gates; the input gate regulates how much of the new cell state to keep, the forget gate regulates how much of the existing memory to forget, and the output gate regulates how much of the cell state should be exposed to the next layers of the network. The GRU layer operates using a reset gate and an update gate. The reset gate sits between the previous activation and the next candidate activation to forget previous state, and the update gate decides how much of the candidate activation to use in updating the cell state.

3.4.3 Hidden Layer Size

Two hidden layer sizes were considered for the RNN layer – 512 and 1024. Increasing the hidden layer feature size increases the complexity of the LSTM unit, thus allowing the unit to learn more complex sequences. We tried different hidden feature sizes with different RNN layers (LSTM, GRU and Elman layers).

3.4.4 Word Embeddings

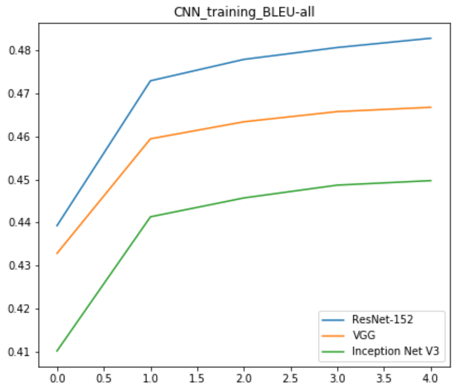
We have experimented with random word embedding initializations and GloVe (Global Vectors) word embedding initialization. GloVe has word embedding vectors trained on 6 billion tokens. We are using 300-dimensional vector representation in GloVe word embeddings to represent each word in the vocabulary.

4 Results & Observations

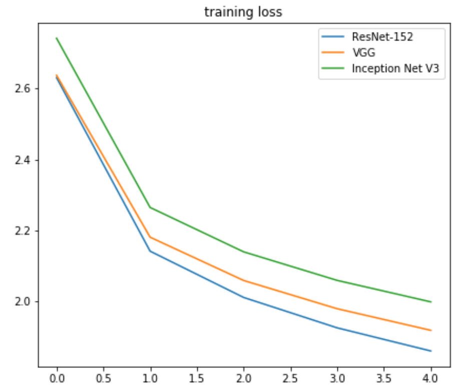
(Note: x-axis in all the graphs below represent number of epochs. y-axis represents BLEU score in accuracy plots and cross entropy loss in loss plots)

4.1 CNN architecture of the Encoder

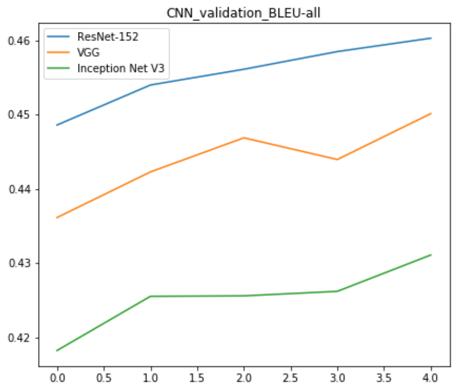
It is evident from Figure 3 that the training and validation losses display a decreasing trend for the three CNN architectures. Highest accuracy (BLEU score) and lowest loss curve is observed for the ResNet-152 CNN based encoder. This can be



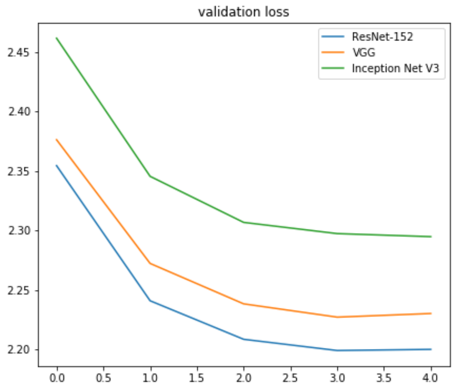
(a) BLEU score for CNN models - training



(b) Training loss for CNN models



(c) BLEU score for CNN models - validation



(d) Validation loss for CNN models

Figure 3: Experiments with different CNN architectures

attributed to the use of 152 layers which makes the network very deep and also due to the residual blocks which have skip connections.

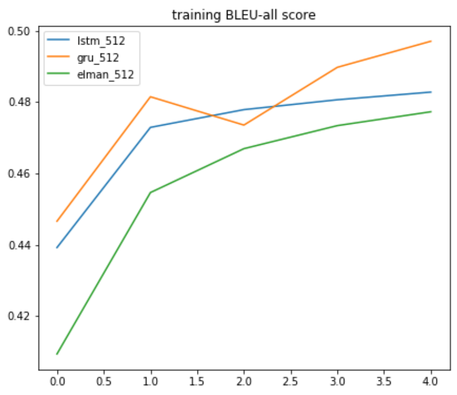
Although Inception-v3 is expected to perform better than VGG16, we observed results that oppose the expectation. This is because the pre-trained models we used were trained on the ImageNet dataset, on which the accuracy of classification is higher when Inception-v3 is used in comparison to VGG16. In our case, we use the MSCOCO-2015 dataset, and therefore the discrepancy observed is justified.

Summarizing, ResNet architecture performs better compared to VGG16, which in turn performs better than Inception-v3. This can be visualized in the loss and BLEU score plots in Figure 3.

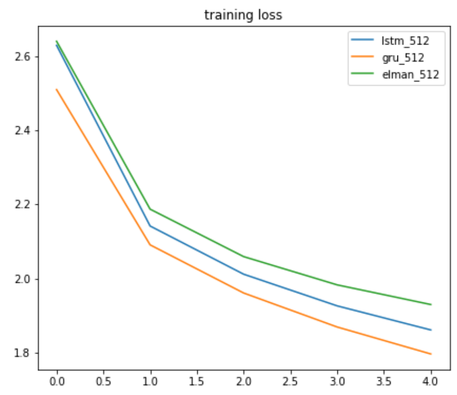
4.2 Types of RNN Layers

We noticed that we got better loss and BLEU score while using the GRU layer compared to the LSTM layer [Figure 4]. This can be because of the maximum caption length used for training. We used a maximum segment length of 20 words which is quite small. Each of the ground truth captions was around 25 words long. This length may be too short for a complex LSTM to show its true power. LSTM is more suited for a larger caption length. On the other hand, Gated Recurrent Units (GRU) may be more suited in this case. GRU has a smaller number of parameters to learn compared to LSTM. So, it may be more suited for this case where the caption lengths are not too long.

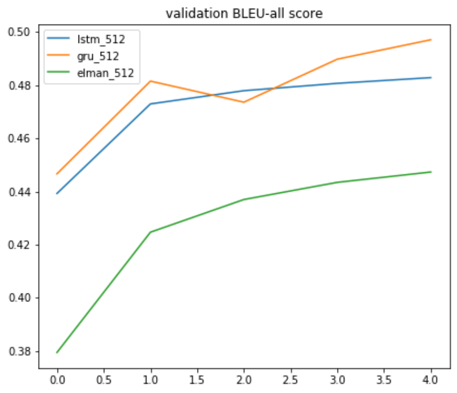
Multilayer Elman layer has the worst performance amongst the three. Elman layers have standard RNN units and it is difficult to train these RNNs to solve problems that require learning long-term temporal dependencies. This is because of vanishing gradient problem.



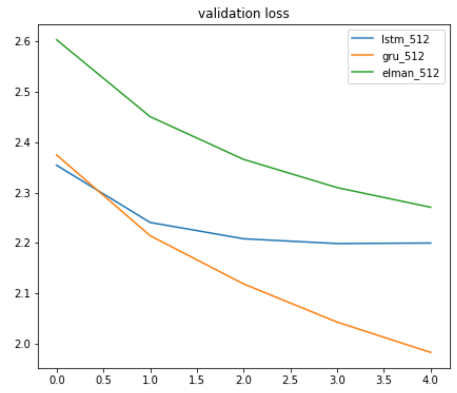
(a) BLEU score-Decoder models - training



(b) Training loss for Decoder models



(c) BLEU score-Decoder models - validation



(d) Validation loss for Decoder models

Figure 4: Experiments with different Decoder models - 512 hidden layer feature size

Summarizing, the GRU based decoder performed better than the LSTM based decoder, which in turn performed better than the Elman based decoder. This can be visualized in the loss and BLEU score plots in Figure 4.

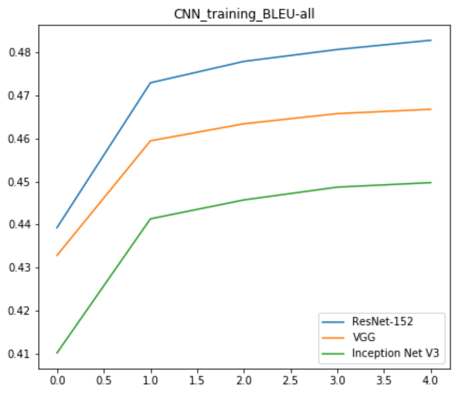
4.3 Hidden Layer Size

By varying the hidden layer feature size, we observed change in performance. We observed LSTM with 1024 hidden layer feature size performed better than LSTM with 512 hidden layer feature size[Figure 4 and Figure 5]. Increasing the hidden layer feature size of the hidden layer increases the network complexity. Doing so causes the LSTM to learn complex sequences and give better results. But this comes with the trade-off of computation time. We noticed that the training time for one epoch increased from 1 hour to 1.5 hours when we changed hidden layer feature size from 512 to 1024. This is due to the increase in the number of learnable parameters when we change to a higher feature size.

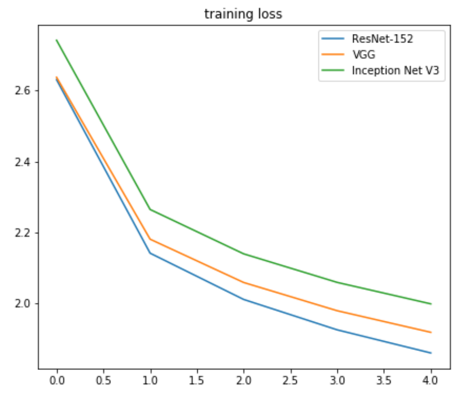
Summarizing, LSTM with 1024 hidden layer feature size gave us better BLEU score and training loss compared to LSTM with 512 hidden layer feature size. This can be visualized in the loss and BLEU score plots in Figure 4 and Figure 5.

4.4 Word Embeddings

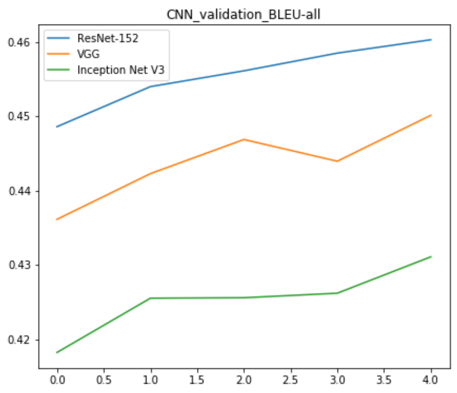
The BLEU score accuracy and the loss obtained in both the cases (random word embedding initializations and GloVe) are almost similar. But, we got slightly better validation loss with GloVe word embeddings. The results obtained with random word embeddings initialization proves the robustness of the model. The weights in random word embeddings adapt pretty well and help to increase the accuracy of the model. This can also be attributed to the low variance used for the random initializations which helps the model learn well.



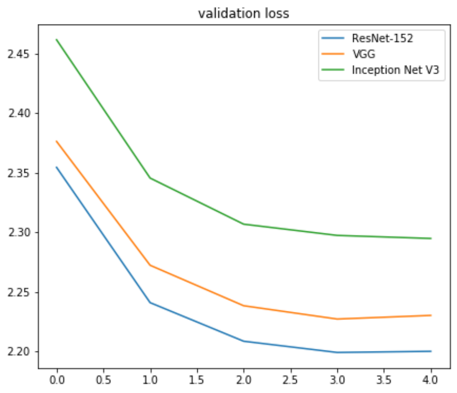
(a) BLEU score- decoder models- training



(b) Training loss- decoder models

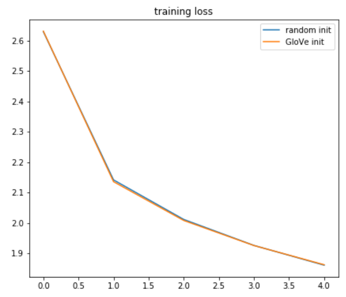


(c) BLEU score- Decoder models- validation

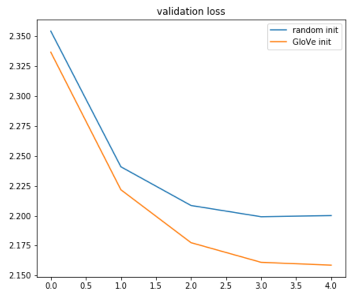


(d) Validation loss- Decoder models

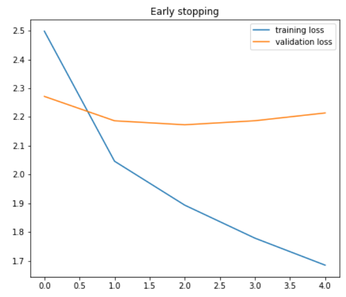
Figure 5: Experiments with 1024 hidden layer feature size



(a) Training Loss- word embeddings



(b) Validation Loss- word embeddings



(c) Early stopping use case

Figure 6: Word embeddings and early stopping experiments

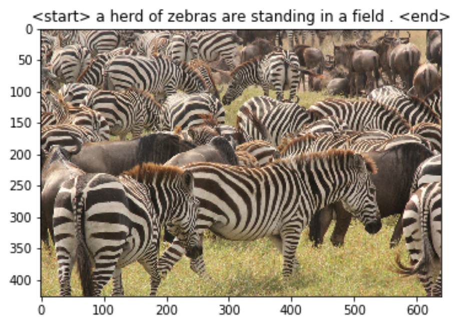
To summarize, there wasn't much difference between GloVe and Random word embeddings [Figure 6 (a), (b)].

4.5 Early Stopping

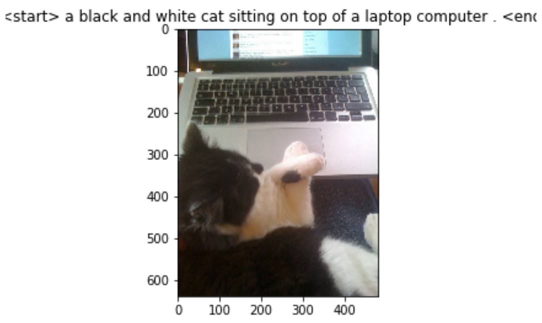
To add on, we can observe that the validation loss for lstm based decoder with 1024 hidden layer feature size starts to increase after 2 epochs even though the training loss is decreasing. This is a classic example of overfitting. Early stopping can help us prevent overfitting and obtain better results. We can possibly stop training after 2 epochs. We can use other techniques like adding l2 regularization or dropouts [Figure 6 (c)].

4.6 Conclusion

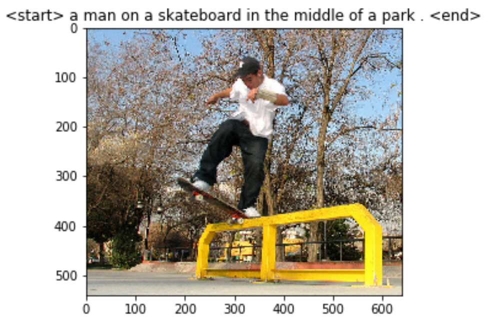
We tested the models on the test data obtained from the MSCOCO-2015 dataset. From Figure 7 we observe that the baseline model performs reasonably well. The captions generated are coherent with the content of the images. But there are some cases of poor captions being generated as can be seen in Figure 8. The captions generated do not match with the content of the images. These results could probably be improved by choosing the best training settings from the results of the above experiments.



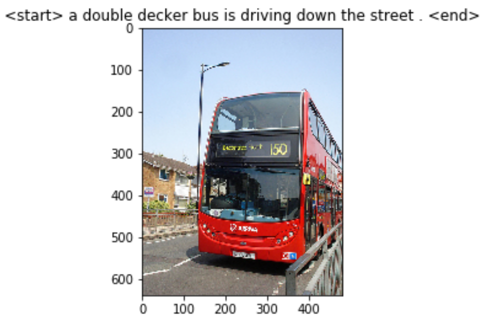
(a)


<start> a black and white cat sitting on top of a laptop computer . <enc

(b)


<start> a man on a skateboard in the middle of a park . <end>

(c)


<start> a double decker bus is driving down the street . <end>

(d)

Figure 7: Good image captioning results

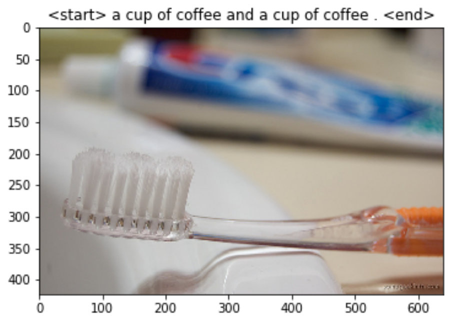
5 Discussion

5.1 Learning

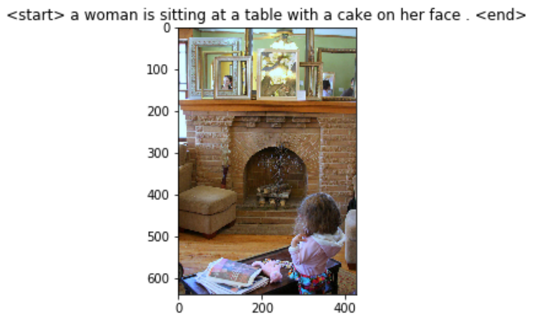
By implementing this project, we learned a lot about RNNs and how to build networks by combining two different types of neural networks (CNNs and RNNs). We also understood the importance of properly structuring our project codebase. We also tried to follow recommended design patterns while building neural network models. Additionally, we have become proficient in using the PyTorch framework for building Deep Neural networks. We are now familiar with a large portion of the PyTorch documentation. We learnt the procedure to carry out experiments to improve the accuracy of our model. We now have an intuitive understanding of which hyper-parameters to tune to obtain better results.Moreover, we learnt that it is absolutely essential to setup consistent code base, environments and packages on each of our accounts in DSMLP cluster for effective implementation of the project.

5.2 Challenges

The biggest challenge we faced was while training the network since it takes around an hour per epoch to train along with evaluating the metrics for the validation set at every epoch. This coupled with restricted access time to DSMLP cluster made training a bit challenging. We had to run background jobs to train effectively. Also, the model ckpt files including the weight

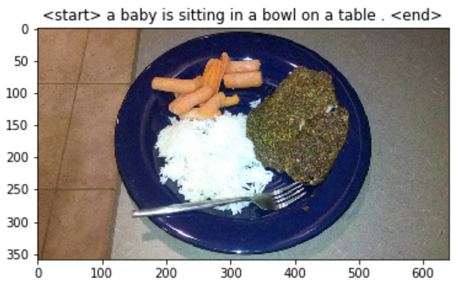


(a)

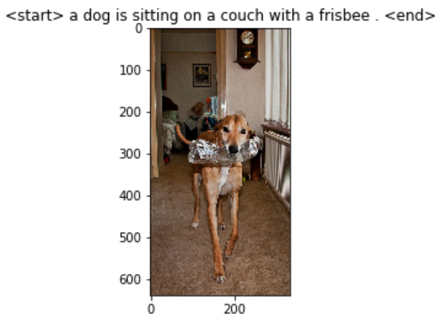


<start> a woman is sitting at a table with a cake on her face . <end>

(b)



(c)



(d)

Figure 8: Bad image captioning results

values were quite huge. It was difficult to share these files and we had to upload the files on google drive to collaborate. We couldn't make use of GitHub as the maximum file size allowed is 25 Mb and our model files were well above the limit.

5.3 Future Work

We observed that although better accuracy is obtained with a larger hidden layer feature size for LSTMs, the model tended to overfit. We can solve this by using some type of regularization including L2 regularization and dropouts. Another potential improvement can be by using attention model combined with the show tell model. The Show tell model imposes a limit on the length of the sequence that can be effectively learned. Attention model allows the network to learn where to pay attention in the input sequence for each item in the output sequence.

References

[1] Vinyals, O., Toshev, A., Bengio, S. Erhan, D. Show and tell: a neural image caption generator. In Proc. International Conference on Machine Learning - http://arxiv.org/abs/1502.03044 (2014).

[2] G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg, "Baby talk: Understanding and generating simple image descriptions," in CVPR, 2011

[3] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth, "Every picture tells a story: Generating sentences from images," in ECCV, 2010

[4] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollar, and C. L. Zitnick, "Microsoft coco captions: Data collection and evaluation server," arXiv preprint arXiv:1504.00325, 2015.

[5] https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning