Image Caption Generation using Deep Learning

John O'Connell
Khoury College of Computer Science
Northeastern University
Boston, MA 02115
oconnell.j@northeastern.edu

Abstract

This project demonstrates the ability of a computer to perform automatic image caption generation by leveraging strategies from the domains of computer vision and natural language processing. The group implemented a neural image captioning model that follows an encoder-decoder architecture consisting of a convolutional neural network (CNN) encoder outputting into a recurrent neural network (RNN) decoder. The model was trained on the Flickr8k dataset [1] and evaluated using Bilingual Evaluation Understudy (BLEU) metric [4]. The final model produced the following BLEU scores: BLEU-1: 0.556, BLEU-2: 0.324, BLEU-3: 0.206, BLEU-4: 0.125.

1 Introduction

When an image appears in front of us, our brains can quickly and efficiently decipher and annotate it. With the modern advances in machine learning, a deep learning architecture can be used to replicate this ability in machines. Image captioning is the task of describing the content of an image in words. This task lies at the intersection of computer vision and natural language processing. Most image captioning systems use an encoder-decoder framework, where an input image is encoded into an intermediate representation of the information in the image, and then decoded into a descriptive text sequence. The successful implementation of this task carries with it many benefits. For example, image captioning can be used to provide recommendations in image editing applications, help the visually impaired better understand their surroundings, and allow for improved monitoring of posts on social media. This project explores the deep learning models required to perform image captioning and showcases a successful application that can be used to accurately caption an image.

The successful use of an encoder-decoder model for this task was first demonstrated in 2015 by Vinyals et al. [4]. At that time, recent advances in machine translation had shown that translation can be done in a much simpler way using Recurrent Neural Networks (RNNs) and still reach state-of-the-art performance. An "encoder" RNN would read the source sentence and transform it into a rich fixed-length vector representation, which in turn was used as the initial hidden state of a "decoder" RNN that generated the target sentence. In the task of image captioning, we follow this elegant recipe, replacing the encoder RNN with a deep convolution neural network that transforms images into the vector representations required by the decoder.

2 Model

For this project we propose a neural and probabilistic framework to generate descriptions from images. The model makes use of a CNN to encode an input image into a fixed dimensional vector and uses this representation in an RNN to "decode" the desired output sentence. We are attempting to maximize the probability of the correct description given an input image in an end-to-end fashion – both for training and inference. We intend to maximize this probability using the following formula:

$$\theta^* = \underset{\theta}{argmax} \sum_{(I,S)} \log p(S \mid I, \theta)$$

Where θ is the parameters of our model, I is the input image, and S is the correct description for the image. Because S can be any sentence, it's length is not fixed. As we have seen this semester, it is common to apply the chain rule to sequences to calculate the joint probability over the entire sequence, $S_0 \ldots S_N$, where N is the length of the sequence. This application of the chain rule can be seen in the following formula:

$$\log p(S \mid I) = \sum_{t=0}^{N} \log p(S_t \mid I, S_0, \ldots, S_{t-1})$$

RNNs are well suited for this type of sequence generation due to their ability to capture sequential dependencies and process input data of variable lengths. In our case, we chose a specific type of RNN known as a Long-Short Term Memory (LSTM) net, which has shown state-of-the art performance on sequence tasks such as translation.

As for the representation of images, we use a CNN with the last layer removed to create rich feature vectors from our images. CNNs have been widely used and studied for image tasks and are currently state-of-the art for object recognition and detection. Our particular choice of CNN was the VGG-19 net, which will be discussed further in the following sections.

2.1 Dataset

The dataset chosen to train the model for this project was the Flickr8k dataset. This dataset is a benchmark collection for sentence-based image description and search, consisting of 8,091 images that are each paired with five different captions which provide clear descriptions of the salient entities and events. While there are larger datasets readily available such as the Flickr30k and COCO datasets, which contain 30k and 330k images respectively, the choice to go with the Flickr8k dataset came down to limited training time and computing power.

2.2 Data Preprocessing and Cleaning

The first step of preprocessing was to create a mapping of image IDs to captions. In their raw form, the captions are stored in a text file with one caption per line. To create a mapping of images to captions we used a Python dictionary where the keys were the image IDs and the item belonging to each key was an array of the captions associated with that image.

The next step was to clean the captions. Cleaning involved making all letters lowercase, removing any digits or special characters, deleting additional spaces, and adding start and stop characters.

We implemented start and stop characters similar to those seen throughout the semester; the start character was <s>, and the stop character was </s>. These unique characters help in making predictions with the model by giving us a consistent first character to feed to the model, and by allowing us to terminate the model prediction when the stop character is produced.

The final step in preprocessing was to tokenize the text. In order to do this, we used TensorFlow's built in tokenizer and fit it to our text. Fitting the tokenizer updates its internal vocabulary based on the captions provided and creates a vocabulary index based on word frequency.

2.3 CNN Feature Extractor

For the feature extractor, the group leveraged the existing VGG19 model. VGG19 is a deep CNN architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. The VGG19 model was trained on the ImageNet dataset, which contains over 1.2 million images. The model is known for its simplicity and effectiveness, and it achieved high performance on various computer vision tasks, including image classification and object recognition. The VGG19 architecture is characterized by using small 3x3 convolutional filters with a stride of 1 and a padding of 1. The network also employs max-pooling layers with 2x2 filters and a stride of 2 to down sample the spatial dimensions. The group chose to use a pretrained model as it reduces overall training time and greatly improves the overall model performance.

As previously mentioned, last layer of the VGG19 model was removed as it is a SoftMax layer used to make classification predictions on images. It was also necessary to reshape the images in our dataset to 224x224x3, the image size accepted by the model. Once this was done, we fed our images to the model and created a feature dictionary that mapped the image IDs to the rich, 4096-dimensional feature vector produced by the model for each image.

2.4 RNN Decoder

The overall structure of our decoder model can be seen in Figure 1 below.

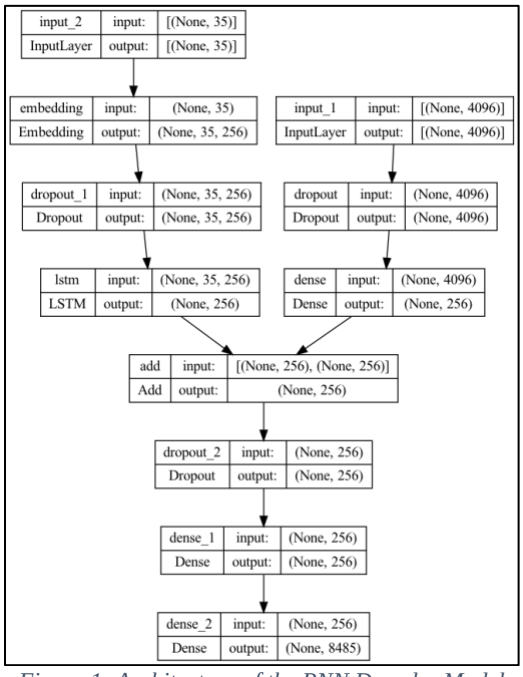


Figure 1: Architecture of the RNN Decoder Model

The model takes two inputs:

The first is the 4096-dimensional feature vector that is output from the VGG19 model. This input is fed through a dropout layer during training, which is used to add regularization to the data and avoiding over fitting. It then goes through a dense layer with a ReLU activation function that reduces the dimensions down to 256.

The second input is the fixed length caption sequence. In order to create this sequence, the caption is first transformed into a sequence of integers by the tokenizer, and then padded to the length of the longest caption in the data, 35 in our case. This sequence of integers is then fed through an embedding layer, which transforms it into a matrix of dense, 256-dimensional vectors. This matrix is then passed through a dropout layer during training, and finally passed through the LSTM net with a Tanh activation function. The LSTM net takes the input matrix and transforms it into a single 256-dimensonal vector.

The outputs from the two separate branches of the model are then added together, passed through another dropout layer during training, a dense layer with a ReLU activation function, and finally a SoftMax layer which produces a vector equal to the vocab size, 8,485 in our case. This SoftMax layer is used to determine the most probable next word in the input sequence.

3 Training

In order to train the model it was necessary to design a batch generator, which as its name suggests created batches of data in the correct format to feed to the model. This batch generator took the images and captions that were input and generated three outputs:

X1: The 4096-dimensonal feature vector of the image, to be fed into the model
X2: A "rolling" caption of the image, to be fed into the model
Y: The expected output of the model, which is X2 with the correct next word appended

An example of how this batch generator works can be seen in Figure 2 below.

X1 (Image Feature Vector)	X2 (Input Text Sequence)	Y (Expected Output Text)
Image Features (4096)	<s>	my
Image Features (4096)	<s> my	name
Image Features (4096)	<s> my, name	is
Image Features (4096)	<s> my, name, is	john
Image Features (4096)	<s> my, name, is, john	</s>

Figure 2: Example of Batch Generator Outputs

We used an 80/20 train test split, a batch size of 32, and trained the model for 20 epochs. It is important to note that the batch size in the context of this model is just the number of captions feed to the model at any one time. The number of steps per epoch was set to equal the length of the training data divided by the batch size, meaning that the model saw the entire training set in each epoch. We used the Adam optimizer and categorical cross-entropy loss for our loss function. The group explored changing all different hyperparameters of the model, but ultimately these values produced the best results. The full ablation study can be found in Appendix B.

4 Results

The train and validation losses, as well as the BLEU-1 scores recorded during training can be seen in Figures 3 and 4 below.

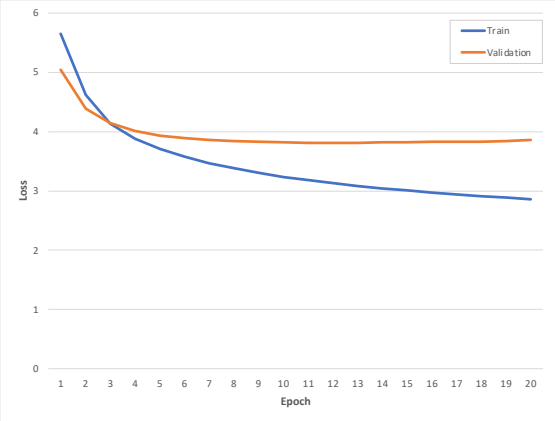


Figure 3: Training and Validation Loss vs Epoch

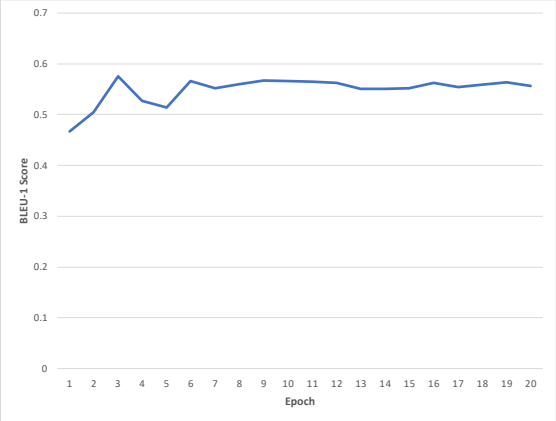


Figure 4: BLEU-1 Score vs Epoch

BLEU (Bilingual Evaluation Understudy) is a metric commonly e. The scores range between 0 and 1, with higher scores indicating better text generation quality. The BLEU scores for our final model at the 1 to 4-gram level were as follows: BLEU-1: 0.556, BLEU-2: 0.324, BLEU-3: 0.206, BLEU-4: 0.125. Examples of both successful and unsuccessful captions generated by our model can be seen in Appendix A.

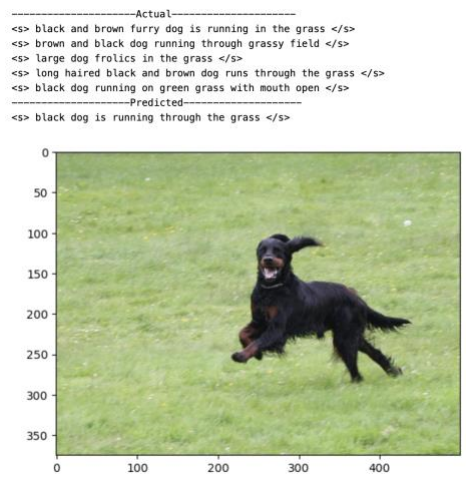
5 Conclusion / Future Work

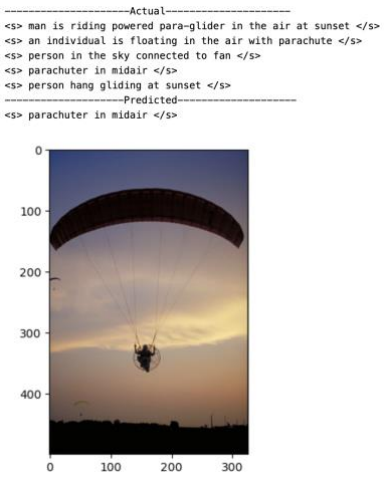
Overall, the group considers this project a success. We were able to successfully implement a neural image caption generator with our initial proposed architecture. This project demonstrates the impressive capability of the encoder-decoder architecture. It showcases a CNN's ability to represent an image as a rich feature vector, as well an LSTM net's ability to decode said feature vector into meaningful text. Typically, a BLEU-1 score above 0.5 is considered very high quality, fluent translations. Therefore, our models BLEU-1 of 0.556 is very impressive; especially considering it was trained on a relatively small dataset and used a pretrained CNN embedder. It is important to note that the BLEU score has its drawbacks. Namely, it doesn't do a good job fully capturing fluency, coherence or meaning. It is also sensitive to sentence length and favors shorter sentences. Additional tests may be necessary for a comprehensive evaluation of our model.

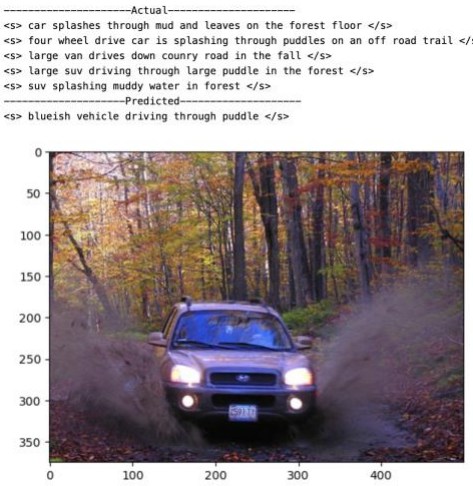
In the future, it would be interesting to replicate this project using a larger training dataset such as Flickr30k or COCO. Additionally, added computing power would provide the opportunity to further explore the model architecture including adding additional layers and an increased training period. Creating a custom trained encoder CNN is another option that would almost certainly improve the overall results of the mode. Ultimately, we are pleased with our results and learned a tremendous amount while completing this project.

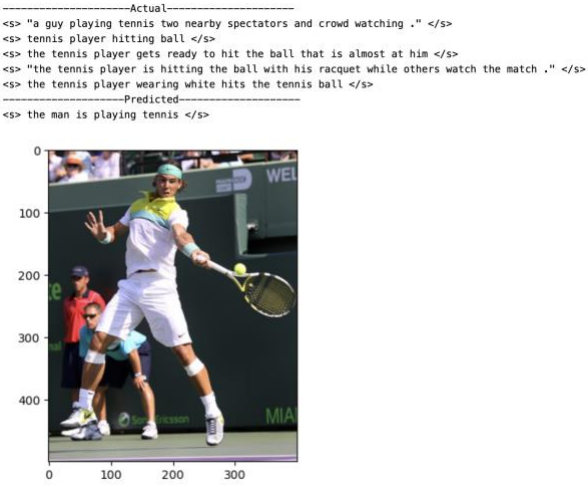
A. Appendix – Best Model Results

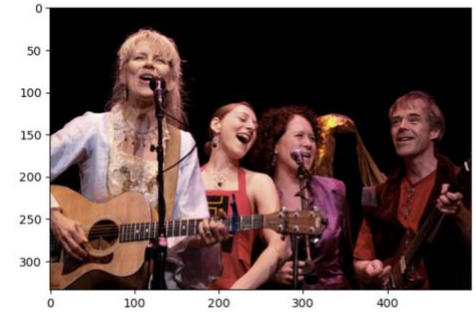
Successful captions:





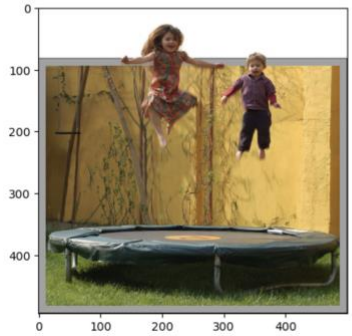






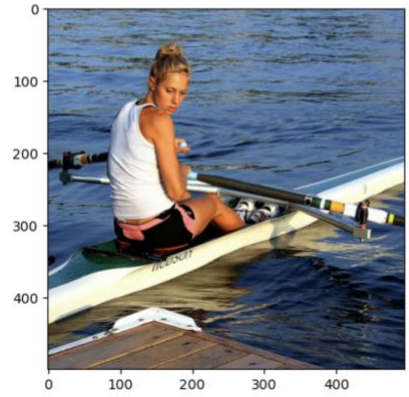
Unsuccessful captions:

----------Actual----------
<s> little boy and girl are jumping on trampoline in front of yellow building </s>
<s> "two children boy and girl in the air above trampoline ." </s>
<s> two children jumping on trampoline </s>
<s> two children jump on trampoline </s>
<s> two young children bounce on short trampoline </s>
----------Predicted----------
<s> man in black shirt is jumping off of rock </s>

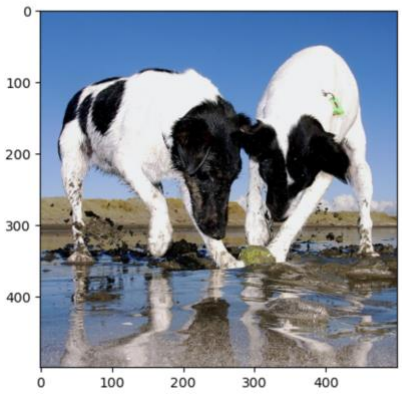


----------Actual----------
<s> an american footballer in red is being tackled by three players in white whilst other red players look on </s>
<s> an oklahoma player is tackled playing football </s>
<s> football player being tackled by group of other players on field </s>
<s> football players are tackling football player carrying the football </s>
<s> the football player is being tackled </s>
----------Predicted----------
<s> two men play soccer </s>

----------Actual----------
<s> girl is rowing in lake </s>
<s> girl sits on skinny canoe </s>
<s> woman sits in rowing scull and looks behind her at dock </s>
<s> woman sitting in rowboat </s>
<s> girl in thin rowboat leaving the dock of lake </s>
----------Predicted----------
<s> man in blue shirt is walking on the beach </s>



----------Actual----------
<s> the two dogs are white and black and are digging in the mud </s>
<s> two black and white dogs after something in the mud </s>
<s> two black and white dogs look at yellow ball in the wet sand </s>
<s> two dogs fight over ball that 's in puddle </s>
<s> two nearly identical dogs try to paw ball out of muddy water </s>
----------Predicted----------
<s> black dog is running through the snow </s>



B. Appendix – Ablation Study

The group explored many different models in order to find the optimal configuration for this project. The following is a list of the different models investigated, as well as a short description of the problems experienced with each. A picture of the GEN1 decoder architecture can be seen below the descriptions for reference. Further investigation may yield even better results, but due to the time it takes to train a single model the group was limited in their investigation efforts.

1. **VGG16 Encoder, GEN1 Decoder Architecture, 256 nodes per layer, 10 Training Epochs**
   BLEU-1 Score: 0.510348 BLEU-2 Score: 0.297888
   BLEU-3 Score: 0.190371 BLEU-4 Score: 0.102993
   Training Loss: 2.6438 Validation Loss: 4.3502
   This was the first model tested by the group. It demonstrated that the model architecture worked, but we believed it could perform better given a longer training period.

2. **VGG16 Encoder, GEN1 Decoder Architecture, 256 nodes per layer, 20 Training Epochs**
   BLEU-1 Score: 0.533349 BLEU-2 Score: 0.308247
   BLEU-3 Score: 0.191981 BLEU-4 Score: 0.115430
   Training Loss: 2.1470 Validation Loss: 4.7395
   At 20 epochs we began to see some increase in validation loss, but it wasn't significant yet. We also saw a clear improvement in training loss from the 10 epoch run.

3. **VGG16 Encoder, GEN1 Decoder Architecture, 256 nodes per layer, 30 Training Epochs**
   BLEU-1 Score: 0.496745 BLEU-2 Score: 0.278750
   BLEU-3 Score: 0.185229 BLEU-4 Score: 0.100145
   Training Loss: 1.8863 Validation Loss: 5.1446
   Even though the training loss continued to fall when we increased to 30 epochs, the validation loss was increasing significantly which suggested the model was starting to overfit to the training data.

4. **VGG19 Encoder, GEN1 Decoder Architecture, 256 nodes per layer, 10 Training Epochs**
   BLEU-1 Score: 0.528811 BLEU-2 Score: .300458
   BLEU-3 Score: 0.203952 BLEU-4 Score: .110293
   Training Loss: 2.5843 Validation Loss: 4.2925
   The first results from the VGG19 encoder seemed promising as it increase BLEU values across all metrics.

5. **VGG19 Encoder, GEN1 Decoder Architecture, 256 nodes per layer, 20 Training Epochs**
   BLEU-1 Score: 0.540275 BLEU-2 Score: 0.392047
   BLEU-3 Score: 0.218129 BLEU-4 Score: 0.137934
   Training Loss: 2.1143 Validation Loss: 4.5925
   Similar to the VGG16 encoder, the 20 epoch training time produced the most promising results from the model with the VGG19 encoder.

6. **VGG19 Encoder, GEN1 Decoder Architecture, 256 nodes per layer, 30 Training Epochs**
   BLEU-1 Score: 0.513823 BLEU-2 Score: 0.294101
   BLEU-3 Score: 0.193455 BLEU-4 Score: 0.109432
   Training Loss: 2.0563 Validation Loss: 4.7399
   Once again, when increasing the epochs the model began overfitting to the data.

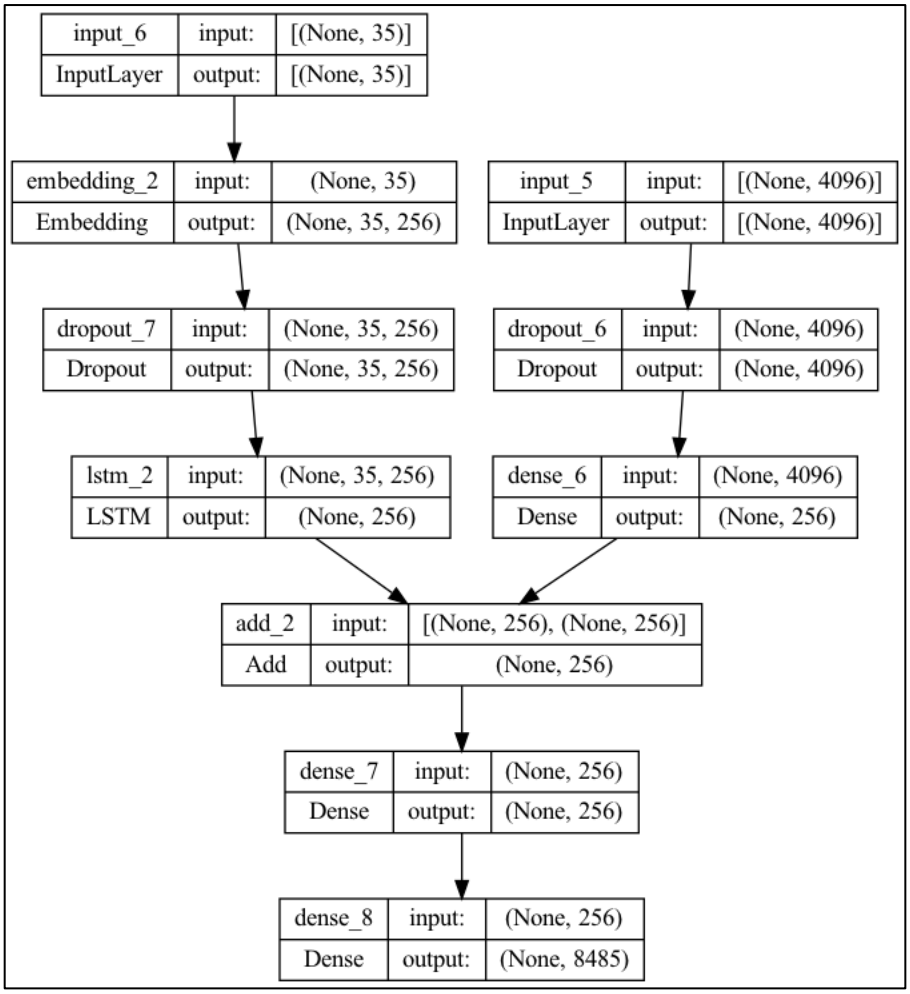
7. **VGG19 Encoder, GEN1 Decoder Architecture, 256 nodes per layer, 40 Training Epochs**
   BLEU-1 Score: 0.502922 BLEU-2 Score: 0.279041
   BLEU-3 Score: 0.183043 BLEU-4 Score: 0.099232
   Training Loss: 1.6522 Validation Loss: 5.4028
   We tried to keep pushing past the 30 epochs with the VGG19 encoder, but the model only seemed to overfit more and produce worse text.

8. **VGG19 Encoder, GEN1 Decoder Architecture, 512 nodes per layer, 20 Training Epochs**
   BLEU-1 Score: 0.500564 BLEU-2 Score: 0.274619
   BLEU-3 Score: 0.179341 BLEU-4 Score: 0.097263
   Training Loss: 1.5003 Validation Loss: 5.7982
   Overfitting occurred very quickly with this model when we switched to 516 nodes per layer. It appeared as though the best results came about 6 epochs in, at which point the validation loss began to rise quite significantly.

9. **VGG19 Encoder, GEN2 Decoder Architecture, 256 nodes per layer, 20 Training Epochs**
   BLEU-1 Score: 0.556309 BLEU-2 Score: 0.324236
   BLEU-3 Score: 0.206271 BLEU-4 Score: 0.125058
   Training Loss: 2.8570 Validation Loss: 3.8562
   This was the best performing model, all results from this model are detailed in the report.

10. **VGG19 Encoder, GEN2 Decoder Architecture, 256 nodes per layer, 30 Training Epochs**
    BLEU-1 Score: 0.532041 BLEU-2 Score: 0.308453
    BLEU-3 Score: 0.193502 BLEU-4 Score: 0.120352
    Training Loss: 2.4012 Validation Loss: 4.1447
    Once again, the 30 epoch training time began to overfit the model and produce worse BLEU score results.

This is the GEN1 Decoder Architecture referenced above. It uses one less dropout layer in training and a decreased dropout rate compared to the final architecture.



References

[1] Flickr8k Dataset. Retrieved July 2023, from https://www.kaggle.com/datasets/adityajn105/flickr8k

[2] Gupta, S. (2023, June 13). *Build Image Caption Generator using Deep Learning*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/

[3] Khandelwal, R. (2020, January 26). *Bleu - Bilingual Evaluation Understudy*. Towards Data Science. https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1

[4] Vinyals, O., Toshev, A., Bengio, S. & Erhan, D. (2015). *Show and tell: A neural image caption generator*. CVPR (p./pp. 3156-3164), : IEEE Computer Society. ISBN: 978-1-4673-6964-0