Show and Tell: A Neural Image Caption Generator

Reuven Birnbaum, David Raskin, Harris Nisar (Group 3)

1 INTRODUCTION

Image captioning involves automatically describing the contents of an image in natural sentences. This task has great potential for positive impact by, for example, helping visually impaired people better understand the content of images on the web. This task is challenging, combining fundamental aspects of computer vision with natural language processing. A caption for an image has to encoded using a convolutional neural network (CNN). Because the captions must be expressed in a natural language, a language model is also needed. This report summarizes our implementation [1] of the model presented in Show and Tell: A Neural Image Caption Generator [2] which approaches the caption generation problem by modeling the conditional probability of generating a sequence of words S given an image I, where S is target sequence of words, $S = \{S_1, S_2, ...S_N\}$ as shown in Eq. 1.

$$\theta^* = \arg\max_{\theta} \sum_{(I,S)} \log p(S|I;\theta) \tag{1}$$

A pre-trained convolutional neural network (CNN) generates image features which are passed as the first element to recurrent neural networks modeled with LSTM. The general architecture of the network is shown in Figure 1. After training, a particular element in the sequence (S_t) is modeled by looking at the image and the previous $t-1$ words in the sequence as in Eq. 2.

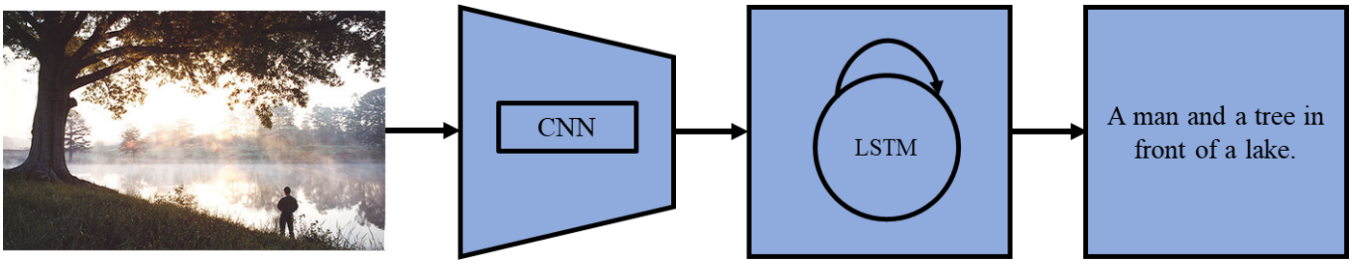


Figure 1: Network architecture of Show and Tell

$$\theta^* = \arg\max_{\theta} \sum_{t=0}^{N} \log p(S_t|I, S_0, ..., S_t - 1) \tag{2}$$

2 METHODS

2.1 Dataset

The dataset we used to train our model is the Flickr 8k dataset [3] which is a benchmark collection for sentence-based image descriptions. It consists of 8,000 images with five different captions for each image (40,000 captions). Figure 2 shows two example images with one of the captions for each of those images. We chose this dataset because of its (relatively) small size, allowing us to quickly train multiple models with varying hyperparameters. To generate training, validation and testing splits, we used splits generated by Andrej Karpathy [4]. This file split distribution is shown in Table 1.

Table 1: Shows the split distribution of the Karpathy splits for Flickr 8k dataset.

Split	Number of Captions
Training	30,000
Validation	5,000
Testing	5,000

Figure 2: Samples from the Flickr 8k dataset *Left caption:* A little baby plays croquet; *Right caption:* A brown dog running along a beach

2.2 Data Pre-Processing

We process image and caption data using standard techniques. Images were processed to comply with what is expected as input to our CNN. Namely, images were resized to 224x224 pixels, and re-scaled between 0-1. We normalize across the RGB channels using $\mu_{channel} = [0.485, 0.456, 0.406]$ and $\sigma_{channel} = [0.229, 0.224, 0.225]$ [5]. Captions are tokenized and converted to integer keys representing each unique word. For each batch, we pad the captions based on the max caption length in that particular batch to allow for a fixed sized input to the language model.

2.3 Model Architecture

Our model uses an encoder-decoder architecture. The encoder is a convolutional neural network known as ResNet101 [5], that has already been trained to perform well on image recognition tasks. Residual networks have connections which skip layers, allowing the network to perform better at larger depths. For our use case we modify the final layer of ResNet101 to be a fully connected layer so the output image feature length is the same as the embed size.

We then feed the image features as input to our decoder. Our decoder is an LSTM, which is used to predict the next word from the previous word. It keeps track of both a cell state and a hidden state, which

store information the model deems important from previous timesteps. We use an embedding layer that takes a tokenized word as input and computes a vector of our specified embed size from a single word token. This allows the model to group words with similar meaning by outputting similar vectors. This embedded word vector is fed as input to the LSTM at each timestep. At every timestep the output of the LSTM is then funneled through a linear layer going from the hidden state dimension to the dimension of our vocabulary. This gives a distribution of potential next words.

2.4 Training

During the train phase we used batches to train over the 6,000 training images, with 5 captions per image. Hence we feed batch size number of captions and their corresponding images as input to the model. During training we also include a dropout layer at the last step of the encoder and decoder to protect against overfitting. For the sake of simplicity we discuss training with a batch of size 1.

The image is first run through the encoder as discussed above and fed into the first input of the decoder. At this stage, the decoder predicts a distribution of potential words for the first word of the caption. In subsequent timesteps we feed in the correct previous word as input to the LSTM to predict the current word, irrespective of what the LSTM predicted at the previous timestep. This helps ensure that the model does not learn from improper previous words, which would carry over previous errors to the current timestep, and result in amplified loss. At the last timestep we feed in the correct final word in the caption, and the model is expected to predict the end of sentence token. After the whole caption has been fed into the LSTM, we compute cross-entropy loss between the list of predicted word distributions output by the LSTM and the list of tokenized words in the expected caption. We then update the parameters of the model with the Adam Optimization Algorithm. After each epoch of training we do validation over the 1000 validation images using the same process as training. Something to consider for the future in the validation phase would be to use inference in the decoder similar to what we do in the test phase. This could help ensure that our validation gives a more accurate measure of how our model is performing.

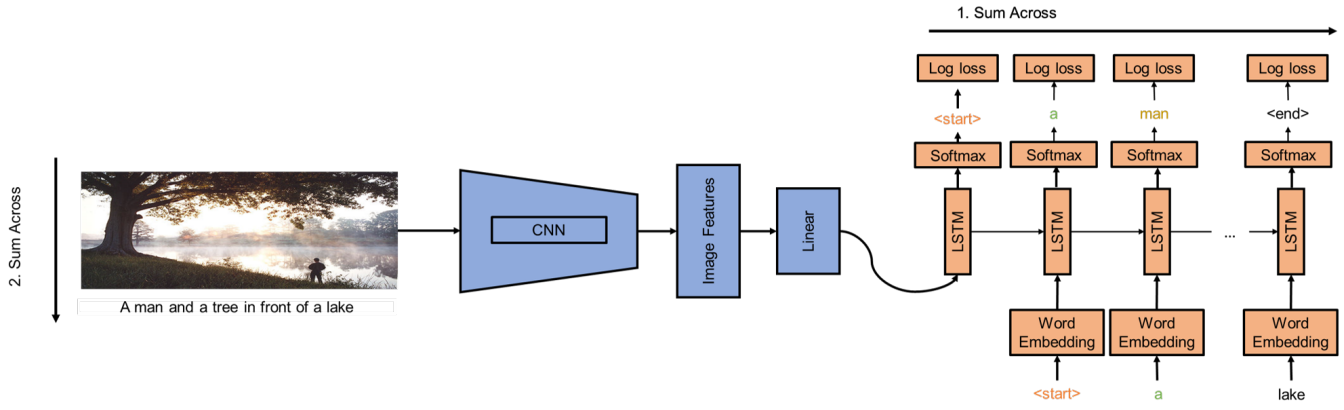


Figure 3: Our training architecture.

2.5 Inference

During inference, image features for new images are generated by passing new images to our trained network the same way as training except the dropout layers are deactivated. We generated the probability

distribution of the t^{th} element in the generated caption by passing the previous predicted word, $t-1$, as input to the LSTM cell (Figure 3).

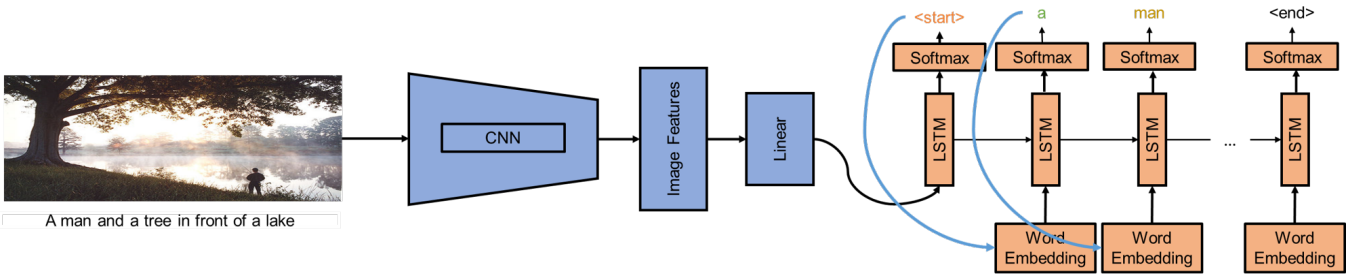


Figure 4: Our inference architecture

2.6 BLEU-Score

To evaluate our model it is necessary to have some way to compare the predicted captions with a phrase that a human reader would likely to choose to describe an image. One of the most effective methods is to have a person score the relavence of an image and it's inferred caption between zero and one. To remove any bias, we may further take the average over a large number of peoples relevancy score of the caption image pair. As you may imagine this would be an incredibly time consuming task, and as such we resolved to compute the BLEU-1 through BLEU-4 scores as done in the paper using the five reference captions .

BLEU-n is a natural language processing score that uses a set of references to create a modified precision scores based on multiple n-gram models. This modified precision score for a candidate, C, using a single n-gram model is calculated as

$$p_n = \frac{\sum_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum_{n\text{-}gram' \in C} Count(n\text{-}gram')} \tag{3}$$

Where $Count(n\text{-}gram)$ is the number of unique n-grams in the predicted caption, and $Count_{clip}(n\text{-}gram)$ is the lesser between the number of times a single n-gram occurs in a reference, and the maximum number of times that a n-gram appears in any of the references. This modified precision acts to reward candidates which match the references adequacy and fluency.

To form the BLEU-n score we take the average of precision scores from 1-gram to n-gram and add a brevity penalty, BP, i.e.,

$$BLEU\text{-}n = \frac{BP}{n}\sum_{m=1}^{n} p_m \quad \text{with } BP = \begin{cases} 1 & c > r \\ \exp(\frac{1-r}{c}) & c \leq r \end{cases} \tag{4}$$

This brevity penalty decreases the score if the candidate caption length, c, has a lower length then the length of its closest reference length, r. This score has been shown to correlate highly with human rankings and is a good metric to demonstrate the efficacy of our model.

3 RESULTS

As discussed the small size of our dataset allowed us to do a large number of hyper-parameter tuning that we performed on the NCSA Blue Water Supercomputer. We found the optimal model has architecture with an image embedding size of 512, a LSTM cell with a hidden size of 1024, dropout rate of 50%, and an initial learning rate of .003 that decreased by a factor of 10 every 30 epochs. The training was performed with batches of 32, and we trained on 60 epochs total. Below in Fig. 5, we can see the training and validation losses we achieved during this run. As we can see, we see the expected trend of training loss monotonically decreasing throughout the entire training, while the validation loss reaches a minimum (at epoch 36) before the model becomes over trained. We should also note that the validation loss is less than the training loss which is to be expected as we had dropout layers that were enabled during the training phases, but were not enabled during the evaluation phases.

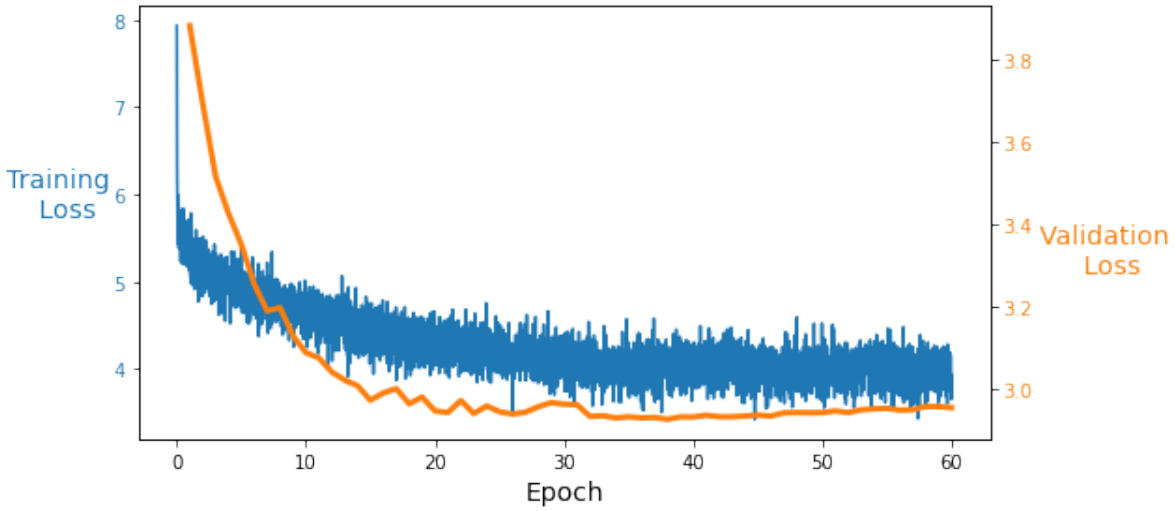


Figure 5: Training-Validation Loss over Epochs

We then used this optimal model to compute the BLEU-1,2,3,4 scores for each of the 1000 images from our testing partition. As can be see in Fig. 6, the higher BLEU-n values score worse, which is to be expected as they include higher n-gram models. We may also notice an interesting phenomenon that for $n > 1$ gram models, the distributions seems to have a break between the values close to 0 and the rest of the distribution. We hypothesize this occurs our captions are relatively short so that that there are few chances for higher n-grams to occur.

To summarize our results , we have listed the BLEU score means as well as results listed in the paper as well as the TA's results that were displayed in the CS547 Spring 2021 Project List document in Table 2. We have additionally shown three image-inferred caption pairs that achieved the highest BLEU-4 (Figs 7a-7c) score as well as the image-inferred caption pair with the lowest BLEU-1 score (Fig. 7d) . It should be noted that even the caption that performed the worst still had some level of fidelity as it recognizes a women in the picture, and our best performing pictures create extremely reliable captions. As such, we may qualitatively recognize that our optimized model does produce captions with good relevancy.

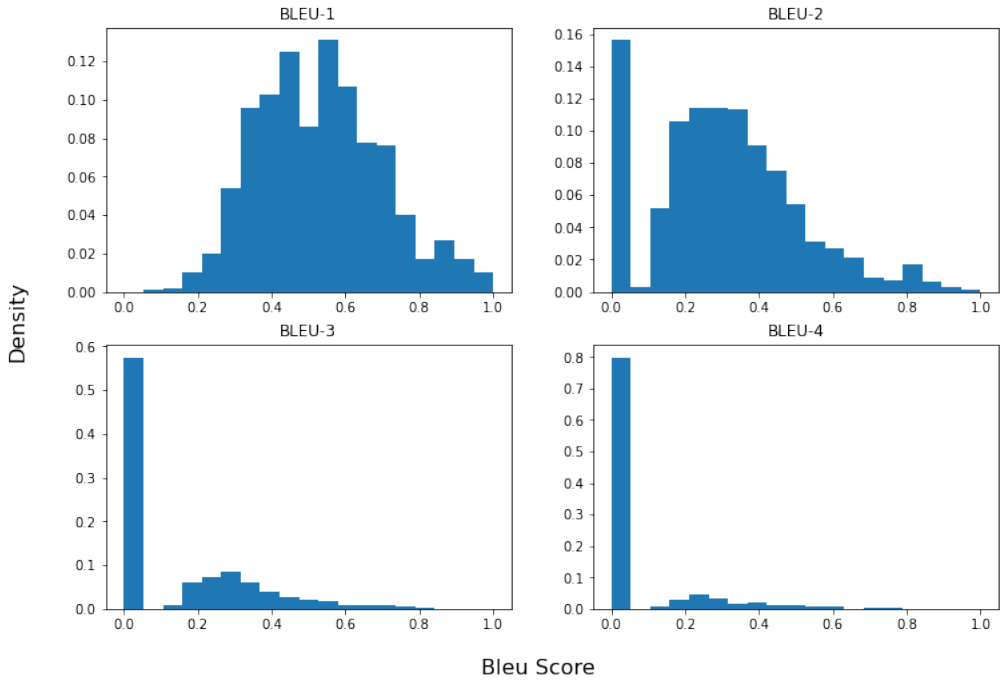


Figure 6: BLEU-n Score Distribution for our Test Partition

Table 2: Shows the split distribution of the Karpathy splits for Flickr 8k dataset.

Method	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Our Mean Score (Flickr 8k)	0.53	0.30	0.15	0.07
TA's Score (Flickr 30k)	0.539	0.348	0.22	0.143
Paper's Score	0.63 (Flickr 8k)	Not reported	Not reported	0.277 (MSCOCO)

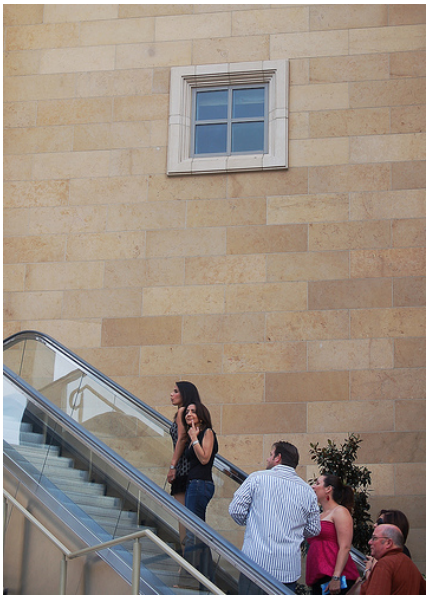
4 DISCUSSION

While the results of our training led to a reasonable model, improvements can be made to improve performance. One way we can potentially improve our results is by using a larger dataset, such as Flickr 30k or MSCOCO [6] [7]. Another area of improvement could be in our inference method. Currently, the model chooses the word with the highest likelihood from the output of the LSTM as input for the next timestep. An alternative to this greedy approach could be BeamSearch, a method proposed in Show And Tell [2] to perform better inference of captions given an image. In BeamSearch, at timestep t in the decoder, we retain the k best captions for the image up to word t based on their overall Bleu score. For each of these captions we consider the k best possible next words, and retain only the top k captions with $t + 1$ words. This allows us to consider the possibility that the model chooses words that score well early in the caption, but that lead the model to pick words that are poor choices later in the caption. Another technique that could help performance is by adding visual attention to our system, as proposed in [8]. Rather than compress an entire image into a static representation, attention allows for salient features to dynamically come to the forefront as needed. The model learns to look at specific parts of an image to predict the next word in the sequence based on what the model has already predicted. Finally, our word embeddings were trained completely from scratch. Instead a pre-trained networks such as Word2Vec [9]

(a) "a white dog is running through the snow."

(b) "two dogs play with each other in the grass."

(c) "two men are playing rugby."



(d) "a women sits at a picnic table eating with luggage."

Figure 7: 4 x 4

could have been fine tuned to generate these embeddings, similar to the convolutional network used in the encoder.

REFERENCES

[1] Harris Nisar Reuven Birnbaum, David Raskin. Cs 547 group project. https://github.com/ReuvenBirnbaum/CS547Group3, Spring 2021.

[2] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator, 2015.

[3] Micah Hodosh, Peter Young, and Julia Hockenmaier. Flickr8k dataset.

[4] Shravan Kumar and Andrej Karpathy. Karpathy splits, 2020.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] Lai Alice Hodosh Micah Young, Peter and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions.

[7] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.
[8] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention, 2016.
[9] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.