Project Report - iAssist (Vision Assistant)

SP21: CMPE-258 Sec 49 - Deep Learning, Professor Vijay Eranti
Team: Deep Dreamers

Haley Feng	Jocelyn Baduria	Princy Joy	Tripura Chandana Gayatri Gorla
haiyi.feng@sjsu.edu	jocelyn.baduria@sjsu.edu	princy.joy@sjsu.edu	tripurachandanagayatri.gorla@sjsu.edu
San Jose State University	San Jose State University	San Jose State University	San Jose State University

***Abstract*—Our goal for this project is to use deep learning to assist visually impaired people by giving them a description of their surroundings. The framework we created called iAssist is a web application that has the ability to generate captions when provided with an image and project the caption to audio.**

Keywords—deep learning, vision assist, image captioning, glove embedding

I. Introduction

Using deep learning to automatically describe the content of an image is a challenging task, but it may have a great impact on visually impaired individuals as it may help them understand their surroundings.

Figure 1. There are multiple ways to describe an image. For our captioning model, we input 5 different captions for every image we used for training.

Our model must not only be powerful enough to solve the computer vision challenges of determining which objects are in an image but must also be capable of expressing the image eloquently and in a rightful manner. In recent years, researchers made significant improvements in the field of image captioning. This includes using a combination of convolutional neural networks to obtain vectorized representation of images, and recurrent neural networks to decode those representations into natural language sentences. Aided with their work, we experimented with different popular encoder-decoder architectures and techniques to build our unique version of the caption generator model. Furthermore, we practiced MLOps by deploying our model for production such that users can upload their images through the iAssist web application and do inference with our model.

II. Related Works

Generating textual descriptions from photos is a challenging problem in Artificial Intelligence and it is extensively researched and studied, considering its applicability to a variety of domains. Conventional template-based approaches and retrieval-based approaches have fallen out in favor of the novel deep learning-based state-of-the-art methodologies. Our work draws inspiration from several papers published in the deep learning domain in recent years. The common theme of all these approaches is to produce an image model and a language model. Karpathy et al. [9] proposed a multi-modal RNN architecture that uses a visual-semantic alignment method. The method generates descriptions in a way that extracted visual features are aligned to particular parts of the description. The encoder-decoder architecture became popular after its success in language translation. Kiros et al. [10] were the first adopters of this architecture in the caption generation, they used a CNN-LSTM architecture. Vinyals et al. [11] proposed a similar architecture in the paper Show and Tell, they used GoogleNet. Xu et al.'s [2] paper 'Show, Attend and Tell' incorporates attention mechanisms into the encoder-decoder framework which gives importance to blocks of the image. We have taken inspiration from

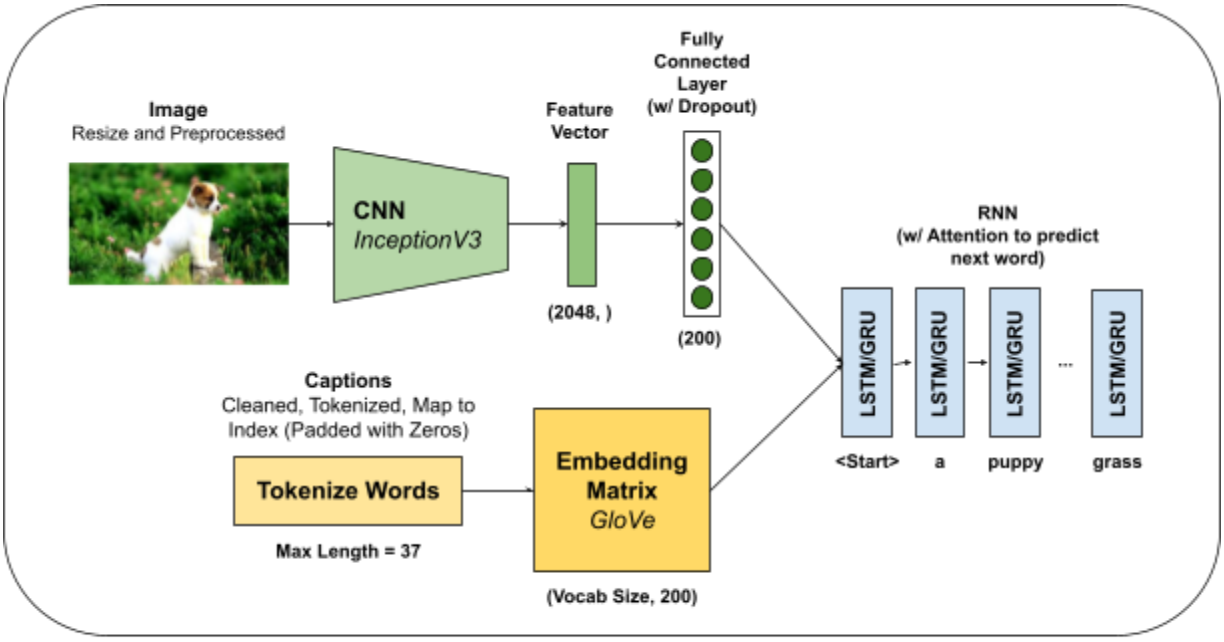


Figure 2. The overall image captioning model architecture using encoder and decoder. Images were preprocessed with InceptionV3 to extract for smaller feature set. Captions were first tokenized with top 5000 most commonly used word in the dataset which then get fed into the embedding layer in the decoder model that will map index to word and word to vector using GloVe embedding matrix.

several aspects of these architectures and created two models based on an encoder-decoder architecture with and without attention.

III. Implementation and Methodologies

A. Overview of Image Captioning Approach

We're using an Encoder-Decoder model which combines:

- CNN (InceptionV3) for image processing
- RNN with Glove Embedding for generating captions using the features learned by CNN

The overview of our architecture is shown in Figure 2 along with the specified shapes of our images and caption vectors at each stage during the encoding and preprocessing.

B. Image Extraction - Inception V3

Inception v3 is a convolutional neural network for assisting in image analysis and object detection. Originally introduced during the ImageNet Recognition Challenge, it was started as a module for Googlenet and now became a widely-used image recognition model that has shown to attain greater than 78.1% accuracy on the ImageNet dataset. It is useful as a transfer learning technique to help reduce time during training. For our implementation, we encoded our images with the last convolutional layer in the InceptionV3 architecture to generate a feature representation of size 2048 for each image.

C. GloVe Embedding

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

We implemented this transfer technique by first building an embedding matrix using the unique vocabularies (8,778 words) from our captions. If words were not found in the GloVe dictionary, then the vector for that word will be an array of zeros. In our decoder model, the embedding layer then takes the caption samples in indexed form and maps each integer to the corresponding word vector in the GloVe embedding matrix we built. By GloVe default, the number of features it generates for each

word is of size 200, thus the final shape for the embedding matrix is (8778, 200).

D. LSTM vs. GRU

The key difference between LSTM - Long Short Term Memory and GRU - Gated Recurrent Units is that LSTM has three gates (input, forget, output gates) whereas GRU has only two gates (reset and update) as shown in Figure 3. The reset gate is located between the previous activation layer and the next candidate activation layer to forget the previous state and the update gate decides how much candidate activation can be used in updating the state. GRU is relatively faster due to its less usage of training parameters and therefore less memory usage. LSTM is more accurate on datasets using longer sequences. LSTM has three values at the output (output, hidden state, internal cell state) whereas GRU has two values at the output (output and hidden state).

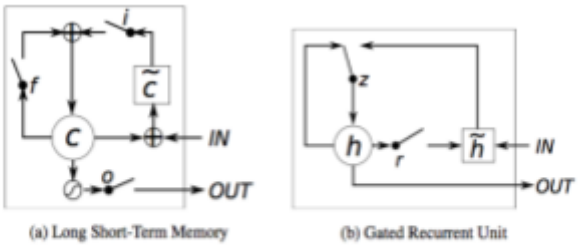


Figure 3. Illustration of a) LSTM and b) GRU. i,f,o are the input, forget and output gates respectively. c and ĉ are memory cells and new memory. r, z are reset and update gates. H and ĥ are activation and candidate activation.

E. Tensorflow Building with Attention

By using the TensorFlow model with attention, instead of compressing the image into static representation, we are dynamically selecting the important features on the forefront as per requirement. It's very important especially when there is a lot of clutter in images. This method is widely adopted but it has the drawback of losing important information which could be useful for richer, more descriptive captions. Most of the image caption generator models proposed recently, are based on recurrent neural networks and inspired by the successful use of sequence to sequence training with neural networks for machine translation. One major reason image caption generation is well suited to the encoder-decoder framework of machine translation is because it is analogous to “translating” an image to a sentence. Cho et al.

The basic idea of the Attention mechanism is to avoid attempting to learn a single vector representation for each sentence, instead, it pays attention to specific input vectors of the input sequence based on the attention weights. Bahdanau et al. proposed an attention mechanism that learns to align and translate jointly. It is also known as Additive attention as it performs a linear combination of encoder states and decoder states.

- All hidden states of the encoder (forward and backward) and the decoder are used to generate the context vector.
- The attention mechanism aligns the input and output sequences, with an alignment score parameterized by a feed-forward network. It helps to pay attention to the most relevant information in the source sequence.
- The model predicts a target word based on the context vectors associated with the source position and the previously generated target words.

The attention layer consists of :

- *Alignment score* - Measures how well the input position “j” and the output position “i” match. It is based on the previous decoder’s hidden state just before predicting the target word.

$$e_{ij.} = a(s_{i-1}, h_j) \qquad \text{Alignment Score}$$

- *Attention weights* - We apply a softmax activation function to the alignment scores to obtain the attention weights.

$$\alpha_{i_j} = \exp(e_{ij}) \Big/ \sum_{k=1}^{Tx} \exp(e_{ik}) \qquad \text{Attention weight}$$

- *Context vector* - Used to compute the final output of the decoder. The context vector c_i is the weighted sum of attention weights and the encoder hidden states (h_1, h_2, …,h□x), which

maps to the input sentence.

$$C_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j \quad \text{Context vector}$$

IV. Dataset and Preparation

A. Flickr 8k Dataset

For this project, we have chosen the Flickr 8K image captioning dataset. The dataset was picked over other popular datasets (e.g. MSCOCO, Flickr 30k) mainly for computational reasons as it is small enough for preprocessing and training. The dataset is properly labeled and contains 8092 JPEG images in various sizes and shapes. Each image has 5 different captions which makes a total of 40460 captions available.

B. Image Preparation

Every image is resized and applied with a downsampling filter before getting converted to a NumPy array with shape (1, 299, 299, 3). After conversion, they are reformatted with the InceptionV3 preprocessor and encoded for feature extraction. The final output after encoding is reshaped to an array of (2048,) and fed to the decoder model.

C. Text Preprocessing/Tokenization

Captions are cleaned and filtered by removing punctuations. We kept only the top 5000 most frequent vocabs such that when captions are tokenized, uncommon words are replaced by the unknown token <unk>. Since the maximum length of a caption in our dataset is 37, we padded each caption with 0 so that each caption is of the same size. One important thing to mention is that each caption includes a start and end token, this helps the model with beginning and ending the generating process.

V. Training Procedure

Across multiple experiments, some parameters stayed the same. This includes the number of units we chose for initializing the hidden state (512) and the number of batch sizes for every training step (64). Furthermore, we were limited to train our model between 20 to 25 epochs for each experiment with the lack of computational resources. With GPU, one epoch can take up to 80 seconds to train. Without GPU, one epoch can train for 30 minutes.

A. Loss Function - Sparse Categorical Cross-Entropy

With our loss function, we stuck with the common choice of categorical cross-entropy which it's used to predict each vocabulary probability during training.

B. Optimizer - Adam vs. RMSprop

We experimented with both Adam and RMSprop as our optimizers during training. RMSprop (Root Mean Square Propagation) is an adaptive learning rate method used in training neural networks. To simplify, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter.
Adam (Adaptive Moment Optimization) is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

C. Dropout, Batchnorm, Teaching Force

Some other common neural network techniques we used to prevent overfitting and smoothen the training process includes:

- Adding dropout (0.5) layer and batch normalization layer to the encoder and decoder model
- Applying the teacher forcing technique that uses ground truth instead of the decoder output as input for the next time step during the training step. This helps to keep the model on track.

D. Beam Search

An additional method we used to produce our captions after generating the possible words from the decoder model is beam search. With beam searching, the method picks several

words (denoted as beam index) with the highest probability vocab predictions and feeds them again to the model. In return, the model will continue considering words with the highest probability when making vocab predictions.

VI. Evaluation

A. BLEU Score

To evaluate the quality of our predicted captions, we used the Bilingual Evaluation Understudy Score (BLEU) that compares the number of matching n-grams from the predicted caption to the reference captions. Here, an n-gram is a sequence of n tokens or words from a sentence in which 1-gram is compared based on each word and 2-gram is compared based on word pair. For example, the BLEU score for 1-gram matches between the reference “Today is a good day” and the candidate “Today is a bad day” is equal to 80%. The highest BLEU score is 1 for a perfect match and 0 for no match.

B. GLEU Score

There are a few drawbacks in using the BLEU score such as not considering the different types of error in the predicted caption (i.e. synonyms, substitution). Additionally, it was designed to evaluate for larger corpus and not for single sentences. For these reasons, we also consider the GLEU score which is the newer version of BLEU released in 2016 by the team in Google. It considers the recall (number of matching n-grams over the total number of n-grams in the reference) and precision (number of matching n-grams over the number of total n-grams in the generated caption) and doesn’t have the limitation of BLEU. The interpretation of the score is the same as BLEU, with a perfect match denoted as 1 and a perfect mismatch as 0.

C. WER Score

Word Error Rate (WER) is a commonly used scoring metric in the field of automatic speech recognition to evaluate spoken language. It encapsulates the different types of error that can occur in a generated text, which include:

- Substitution - number of words needed to be substituted to match the reference
- Deletion - number of words dropped from the reference
- Insertion - number of extra words added compared to the reference

WER is calculated by summing up the above metrics and dividing it by the total number of correct words. We calculated the error rate for our implementation of image captioning by averaging the WER score between each reference to the generated caption. The smaller the value, the smaller the error from our prediction.

D. Method Comparison and Findings

After experimenting with using LSTM and GRU on the same architecture running a total of 50 epochs each, we concluded that both methods led to familiar total training error (Figure 4) and have the tendency to overfit (Figure 5) due to the size of our dataset.

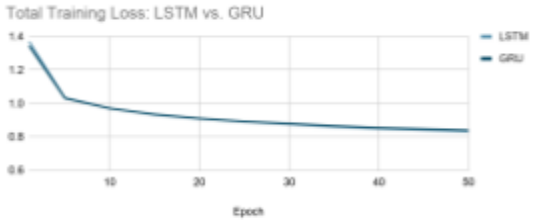


Figure 4. Total training loss comparison between LSTM and GRU.

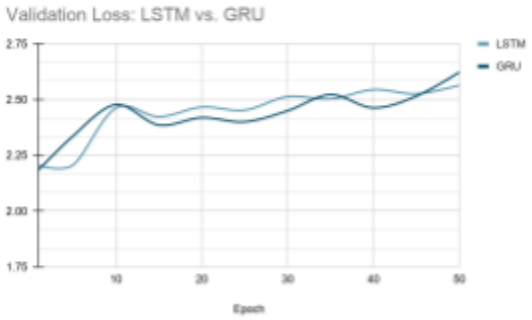


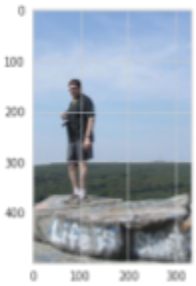
Figure 5. Validation loss comparison between LSTM and GRU

The paper “Show, Attend, and Tell” mentioned that the Flick8k worked best with RMSprop as an optimizer, but for our implementation, we found that the RMSProp optimizers result in a 13.8% higher training loss than the Adam optimizer. Additionally, we learned that adding a 0.5 dropout layer to the encoder model helps

TABLE I. Caption Evaluation

Implementation Method			BLEU Score					
TF	RNN	Generator	1-gram	2-grams	3-grams	4-grams	GLEU	WER
Functional API	*Vanilla*	*Regular*	**0.71**	**0.66**	**0.63**	**0.49**	**0.27**	**13.4**
Subclassing	*LSTM with Attention*	*Regular*	0.16	0.06	0.19	0.25	0.04	32.8
		Beam Search	0.13	0.06	0.19	0.25	0.03	32.4
	GRU with Attention	*Regular*	0.22	0.11	0.27	0.33	0.07	16.8
		Beam Search	0.13	0.10	0.11	0.06	0.07	32.2

Table 1. Compares the different model implementation results for one image using the caption evaluation metrics: BLEU, GLEU, and WER.



Reference Caption: a man in a dark shirt and shorts is standing on top of a high graffitied rock

Vanilla Model Generated Caption: a man in a red shirt is standing on a rock overlooking a waterfall

Figure 6. An example image from Flickr 8k with reference caption and vanilla model generated caption. Evaluation of the caption using different metrics is shown in Table 1.

decrease the training loss by 15%. Meanwhile, adding the dropout and batch norm layer in the decoder model did not have a notable effect. We also experimented with using more vocabularies when adjusting how many words to keep during the tokenizer stage and learned that more vocabularies produce more descriptive captions. However, it would result in the model producing more nonsense in the caption.

In terms of evaluating our captions using the metrics we mentioned, we discovered that the vanilla model produces the highest BLEU and GLEU score, and the lowest WER score. Table 1 shows the comparison between the different implementation and architectures we tried (LSTM vs. GRU, Beam search vs. Regular caption generator, TF Functional API vs. TF Subclassing) on the image from Figure 6. We deducted that the vanilla model did better due to simplicity of implementation in comparison to the subclass model which is more difficult to understand and debug.

VII. End to End Deployment

We have set up an MLOps framework to automate the cloud deployment on the Google Cloud Platform. A GitHub action workflow file is added to the repository that lists the jobs scheduled to run on an event. When the developer pushes the code to the GitHub repository, the action workflow will trigger a post-commit hook which will start the deployment process. Cloud build will create a docker image and push it to the container registry. The image will be rolled out to the Google Kubernetes engine to be deployed on the Kube pods and the application will start running and will be accessible at the external IP addresses to users. This saved us a lot of effort in the manual intervention every time we upgraded our code or model.

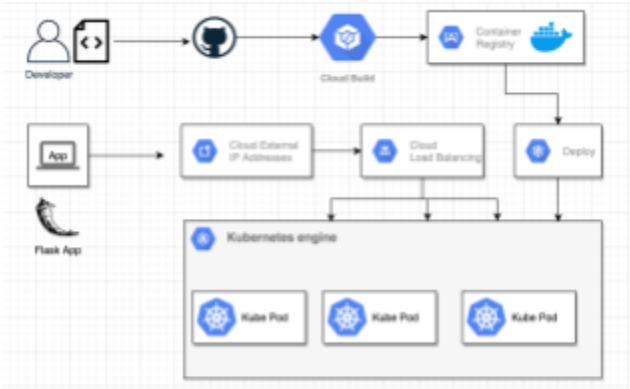


Figure 7. Our deployment pipeline using Github action workflow, flask, cloud build, and kubernetes.

VIII. Conclusion

Some of our team key learnings for this project include the following:

- How to build an encoder-decoder model with image and text captioning
- Transfer learning using CNN model Inception V3
- We learned to create subclass layers for customizing the model, but we also learned that it can get overly complicated and difficult to debug when compared to using Functional API
- We experimented with different model methodologies such as BEAM search, AI Platform with TFX Kubeflow pipeline, TFX Pipeline in AI Platform
- We learned that CICD was useful and easy to set up especially when there was a code/model upgrade

- Deploying the Flask web-app model in GCP Cloud.

There were also major challenges we encountered during the implementation process of our iAssist framework. We were faced with a quota limit in RAM in google colab and the model training process halted. This limited our usage of GPU in which causes us to run fewer epochs during training. During the TFX pipeline implementation, we have challenges on how to write the trainer code to deserialize the TFRecords to binary for ingestion and training. In terms of deployment, it was difficult to set up models in the Kubernetes (TFX Kubeflow) pipeline with resources/tutorials used having data loading inaccuracies and incomplete guidance.

For future work, our team would like to use a larger dataset with better compute resources such that we can experiment with more hyper-parameter tuning. We also want to experiment with the Transformer model in place of GRU/LSTM and compared the result between Soft/Hard attention. There is also CNN model call EfficienNet that's known to have better accuracy and efficiency we would like to try along with Self-Critical Sequence Training (SCST). Additionally, we can extend our project idea to other use cases such as image search/retrieval.

Acknowledgments

The authors of iAssist would like to thank the developers of Glove Embedding, InceptionV3 Convolution, and Image Captioning with Tensorflow. We acknowledge the support of professor Vijay Eranti throughout the project and course.

References

[1] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473, September 2014.

[2] Xu, Kelvin, et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, 2016, https://arxiv.org/pdf/1502.03044.pdf. Accessed 5 March 3031.

[3] Brownlee, Jason, editor. *How to Develop a Deep Learning Photo Caption Generator from Scratch*. 3030. *How to Develop a Deep Learning Photo Caption Generator from Scratch*, https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python

[4] Heaton, Jeff. Applications of Deep Neural Networks, https://github.com/jeffheaton/t81_558_deep_learning/blob/master/t81_558_class_10_4_captioning.ipynb.

[5] Brownlee, Jason. How to Prepare a Photo Caption Dataset for Training a Deep Learning Model. 2017. How to Prepare a Photo Caption Dataset for Training a Deep Learning Model, https://machinelearningmastery.com/prepare-photo-caption-dataset-training-deep-learning-model/.

[6] Brownlee, Jason. How to Develop a Deep Learning Photo Caption Generator from Scratch. 2019. How to Develop a Deep Learning Photo Caption Generator from Scratch, https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python

[7] CI/CD with Google Cloud https://cloud.google.com/kubernetes-engine/docs/tutorials/gitops-cloud-build

[8] A 2019 conference paper - Small Deep Learning Models for Hand Gesture Recognition - https://login.libaccess.sjlibrary.org/login?qurl=https://ieeexplore.ieee.org%2fdocument%2f9047350

[9] Karpathy, Andrej & Fei-Fei, Li. (2014). Deep Visual-Semantic Alignments for Generating Image Descriptions.

[10] Vinyals, Oriol & Toshev, Alexander & Bengio, Samy & Erhan, Dumitru. (2015). Show and tell: A neural image caption generator. 3156-3164. 10.1109/CVPR.2015.7298935. .

[11] R. Kiros, R. Salakhutdinov, and R. S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. In arXiv:1411.2539, 2014.