Image Caption Generator

Shree Murthy, Rahul Sura, Dylan Inafuku

2023-12-16

Table of Contents

1. Introduction **2**
1.1. Alt Text . . . 2

2. Analysis **2**
2.1 Flickr8k Dataset . . . 2
2.1.1. Dataset's Images . . . 3
2.1.2. Dataset's Captions . . . 4
2.2 Data Cleaning . . . 5

3. Methods **5**
3.1 Preprocessing and Tokenization . . . 6
3.2 Word Embeddings . . . 6
3.3 Transfer Learning . . . 7
3.4 Encoder/Decoder Structure . . . 7
3.4.1 Encoder . . . 8
3.4.2 Sequence Feature Layer . . . 9
3.4.3 Decoder . . . 9
3.5 Model Regularization . . . 10

4. Results **10**
4.1 Training Loss and Accuracy . . . 10
4.2 Testing Results . . . 11

5. Reflection **14**

6. References **15**

1. Introduction

This report aims to discuss the model building and evaluation of an Image Caption Generator trained on the Flickr 8k Dataset. The goal of this model is to take in an image and generate a caption that describes the image. This model is built using an Encoder/Decoder structure. The motivation for this model is to develop a model that can help create "Alt Text" to help visually impaired people understand images on the internet. This model can also be used to help people who are learning a new language understand the meaning of an image.

1.1. Alt Text

Alt text is a short description of an image that is used by screen readers to describe images to visually impaired people. The goal is to make the internet more accessible and useable for everyone. HubSpot has a great article on how to write alt text for images. It explains that alt text should be short yet descriptive. The goal of alt text would be to help people access wide variety of websites without worrying about not being able to understand the images. Over 2.2 billion people have a vision impairment or blindness. This is a huge number of people who are not able to access the internet in the same way as people who are not visually impaired. Furthermore, users who use a screen reader report they are ~61% less likely to retain all the information presented by an image. With this information in mind, we sought to create a model that could be helpful generating captions for images to help people who are visually impaired understand images on the internet.

2. Analysis

This section aims to discuss the dataset, data cleaning, and preprocessing. The dataset, again, will be the Flickr8k dataset. This section will also conduct some Exploratory Data Analysis (EDA) to help understand the pictures in the dataset as well as the format of the captions.

Note: The original creators provided a public domain license to enable anyone to reference the Flickr8k Dataset

2.1 Flickr8k Dataset

The dataset contains ~8000 images of various different stock images. These images have ~5 captions attached with them (located in the captions.txt file). First let's start by examining the images in the dataset.

2.1.1. Dataset's Images

Below are the randomly sampled images from the dataset:

Figure 1: Flickr8k Images

Based on the images above we can notice a few things. We notice a lot of pictures were taken outdoors (either in nature or in the city). This indicates that the model is mostly going to learn about the outdoor environment and related features, such as grass, buildings, the sky, and more. Furthermore, we notice a decent number of the images had animals. While this may not be accurate across the entire dataset, it is still vital information because it informs us that the dataset has variety when it comes to what's being depicted. The model shouldn't be biased to just humans, it requires a diverse set of information to help train it effectively and ensure it is knowledgeable about all of our world's features. Also, the images are augmented differently, they are all different shapes and sizes (some vertical and some horizontal). Moreover, the pictures have some blurry backgrounds and are taken during different seasons of the year, such as the one taken in the snowy terrain.

2.1.2. Dataset's Captions

Now, let's analyze the structure of the image captions. These captions are located in the captions.txt file and have at least 5 captions per image in the dataset. Below is a table with 5 random captions.

Image	**Caption**
1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg	A girl going into a wooden building .
1003163366_44323f5815.jpg	a man sleeping on a bench outside with a white and black dog sitting next to him .

Image	Caption
1012212859_01547e3f17.jpg	"A dog shakes its head near the shore , a red ball next to it ."
1287475186_2dee85f1a5.jpg	Little boy sitting in water with a fountain coming up through his lap .
1287475186_2dee85f1a5.jpg	A boy sitting in water .

Based on the image captions above we can see there is a lot of variation happening, meaning there will be a decent amount of data cleaning taking place before we tokenize the captions. First we notice the captions aren't standardized, i.e they aren't all starting with capital letters, some have quotes and the others don't. This indicates that there wasn't much emphasis placed on creating standardized captions, rather they placed them into the document without any cleaning. Furthermore, the lengths of the captions are of varying degrees. Some are really short, such as the second caption and last caption. The differing lengths are extremely useful for our problem statement, we don't want captions to 'over stay their welcome.' Essentially, we want the captions to be short and detailed to ensure the features of the image are explained properly without any loss in information.

2.2 Data Cleaning

For data cleaning our outline was very clear. We didn't want a lot of noise with the actual words itself which may create a scrambled caption. We knew that our model wasn't going to create great captions, but we could at least make sure that the semantics of the caption itself (i.e. misplaced punctuation) could be kept at a minimum. The first step we took in isolating the captions was to create a dictionary with the keys being set to the image ids (i.e the filename(s)) and the values were set to the caption for that image id. The goal was to ensure that we can have a safe and secure way to just analyze the captions and not influence the model by passing in the image ids. The next step was to clean up the captions, essentially standardize, the words. This was achieved by our data cleaning function in the `model.py` file called `cleaning`. The method took a dictionary of image names and their corresponding captions, and then did miscellaneous things to it. Some of those things include removing punctuation, as mentioned, as well as making all the words lower case, trimming surrounding spaces. This will lessen the bias the model will have towards certain words and characters.

3. Methods

This section aims to discuss the preprocessing and model building. The preprocessing will discuss the tokenization of the captions and the model building will discuss the transfer learning

process and the Encoder/Decoder structure. Furthermore, this section will discuss the model's regularization techniques and the loss function used to train the model.

3.1 Preprocessing and Tokenization

The first of the preprocessing steps, inspired from our CPSC-393 classwork, was adding two custom tokens/tags for each caption. Towards the beginning of the caption, we would add something called `'startseq '` and at the end, we would add `' endseq'`, still following the standardization conventions mentioned in section 2 for the sake of consistency. These two tags enabled us to help train the model in order to recognize that at the beginning of the sequence, there is a logic-sounding start, and at the end of the sequence, there is a logical end. This mitigates the chances of the model generating a caption that starts or ends with something like the word "and". Since every single caption prepended and appended these two custom tokens uniformly, it reinforced the model to understand the start and end a little better. Also, we preferred this method because we didn't want to feed in a specific variable that represents how many words the model will output. We want a model to literally prepare a caption with no help from the external user; thus, applying a start and end token to each caption ensures the model will generate a caption until the end token is reached.

Next, we tokenized the captions. This was done using the `Tokenizer` class from the `tensorflow.keras.preprocessing.text` module. We appended all the captions into a list and then passed the list into the Tokenizer class. This creates our vocabulary of unique words. We then used the `Tokenizer` class's `fit_on_texts` method to create a dictionary of words and their corresponding integer values. This dictionary is used to convert the captions into a sequence of integers. This is done using the `Tokenizer` class's `texts_to_sequences` method. This method takes in a list of captions and converts each caption into a sequence of integers. This is done by looking up each word in the caption and finding its corresponding integer value in the dictionary.

3.2 Word Embeddings

Word Embeddings are a vital tool to our project. Word Embeddings are a way to represent words as vectors. This is done by using a pre-trained model that has learned the semantics of words. The model we used was the GloVe model. This model was trained on 6 billion tokens and has 400,000 words in its vocabulary. The model we used was the 200-dimensional model. This means that each word is represented as a 200-dimensional vector. This is extremely useful because it allows us to represent words as vectors and then use those vectors to train our model. The GloVe model is combined with our vocabulary to create an embedding matrix. This embedding matrix is specific to our vocabulary instead of all the words within the GloVe model. From here, we can use this for the embedding layer in our model which will be discussed later.

3.3 Transfer Learning

Transfer learning is a vital tool to our project. Transfer learning is the process of using a pre-trained model and then using it for a different task. In our case, we are using the InceptionV3 model. In Tensorflow, the `InceptionV3` class is derived from the `tensorflow.keras.applications` module. This model was trained on the ImageNet dataset which contains over 14 million images and 20,000 categories. We modified the model to extract features from the images. The model was modified to ensure we are only retaining the hidden representations of the model. The hidden representations are what provide us the important features of the Flickr8k. Below is a code snippet showcasing how we modified the InceptionV3 model to only retain the hidden representations.

The following code is found in the `feature_extraction.py` file.

```
# Create a new model, by removing the last layer (output layer) from the inception v3
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224,224,3))

# Remove classification layers
pooled_output = base_model.layers[-1].output
encoded_features = GlobalAveragePooling2D()(pooled_output)

# Create updated model w/o classification layers
feature_extraction = Model(inputs=base_model.input, outputs=encoded_features)

# Freeze layers in base model
for layer in base_model.layers:
    layer.trainable = False # ensures these layers aren't retrained
```

This class takes in the input shape, which is set to (224, 224, 3) and the weights, which is set to `imagenet`. This ensures that the model is using the weights from the ImageNet dataset. The model is then used to extract features from the images using the `predict` method from the `InceptionV3` class. Once prepared for training, we pass in the images from the Flickr8k dataset we receive a 2048-dimensional vector for each image. Prior to the images being passed in they are resized to fit the (224, 224, 3) shape that was set for the InceptionV3 model. The resulting features are serialized using the `'pickle'` python library such that it can be referenced in the model.py file. This vector is then used to train the model.

3.4 Encoder/Decoder Structure

The meat of our project is reliant on an encoder/decoder structure with a sequence feature layer in-between to represent the LSTM layers that learn based on the embedding layer and

the captions. Before we dive into each part of the model, let's take a look at the model's summary.

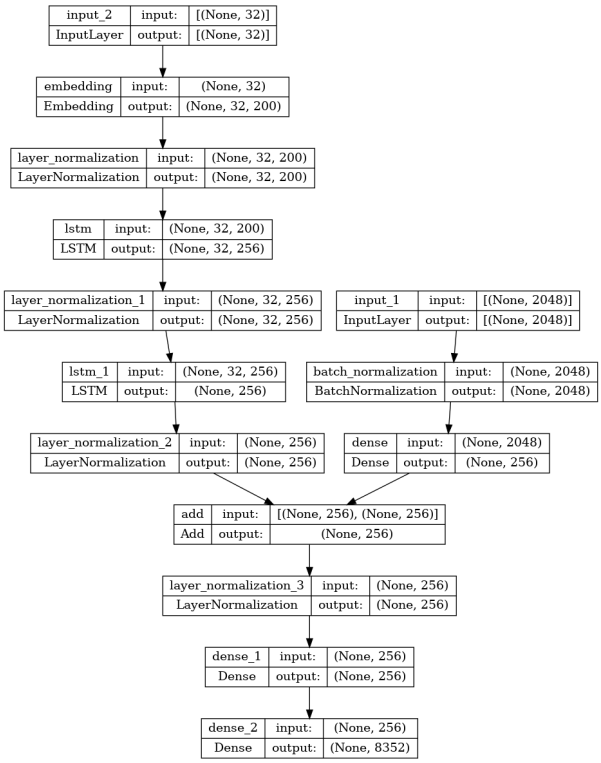

Figure 2: Model Summary

The picture indicated showcases the how the different layers get fused. On the right side we have the encoder model that accepts the 2048-dimensional vector representing an image's encoded_features. On the left side we have the LSTM layers being trained on the embedding layer and embedding matrix. Finally the two sides fuse together to create the output layer that produces the probabilities for the next word in the sequence.

3.4.1 Encoder

The Encoder is formatted as follows:

```
# Encoder
inputs1 = Input(shape=(2048,))
fe1 = BatchNormalization()(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
```

It is straightforward, the encoder accepts the extracted features from the InceptionV3 model. The features are then passed through a BatchNormalization layer to normalize the features. This is done to ensure the model is not biased towards certain features. The features are then passed through a Dense layer with a ReLU activation function.

3.4.2 Sequence Feature Layer

The Sequence Feature Layer is formatted as follows:

```
# Sequence Feature Layer
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = LayerNormalization()(se1)
se3 = LSTM(256, return_sequences=True)(se2)
norm = LayerNormalization()(se3)
se4 = LSTM(256)(norm)
norm2 = LayerNormalization()(se4)
```

The Sequence Feature Layer is where the LSTM layers are located. The first step is to pass in the captions. The captions are passed through an Embedding layer. From there layer normalization is used and then passed into the an LSTM layer with return_sequences set to True. This is done to ensure the LSTM layer returns the hidden state for each input time step. The hidden state is then passed through another layer normalization layer. This is then passed through another LSTM layer with return_sequences set to False. This is done to ensure the LSTM layer returns the hidden state for the last input time step. The hidden state is then passed through another layer normalization layer.

3.4.3 Decoder

Finally the two sides are fused together to create the decoder. The decoder is formatted as follows:

```
# Decoder
decoder1 = add([fe2, norm2])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
```

A very simple structure that aims to take the two sides (sequence feature and the encoder) and fuse them together. The two sides are passed into a Dense layer to ensure the model can learn the features from both sides. The output layer is then passed into a Dense layer with a

softmax activation function. This is done to ensure the model can learn the probabilities for the next word in the sequence.

3.5 Model Regularization

This section aims to discuss the model's regularization techniques. The model uses the following regularization techniques: Layer Normalization, and Batch Normalization. Both these techniques are used to ensure the model is not biased towards certain features. This is done to ensure the model is learning the features from the images and the captions. Layer Normalization is used to normalize the caption's features. Layer Normalization is preferred to Batch Normalization for recurrent structures because it normalizes over the feature dimension (it calculates the mean and variance for each instance separately) whereas batch normalization depends on the batch size and normalizes over the batch dimension. Batch Normalization is used to normalize the image's features to ensure the model is not biased towards certain features in the images. I recommend reading more about this topic in the Understanding Batch Normalization, Layer Normalization and Group Normalization by implementing from scratch article.

We tried using Dropout layers; however, this didn't adjust the accuracy of the model. The accuracy was very low when we used these layers. Once we introduced the normalization layers, the accuracy increased significantly, the resulting accuracy is discussed in the results section. We assume the dropout layers didn't work for the encoder because we already extracted the features from the pretrained model; thus, normalization just smoothed out the data. For the sequence feature layer it doesn't make sense to use dropout layers because it could lead to having holes in the sequential data meaning semantics and syntax could be lost. Hence, we opted to use normalization layers instead of dropout layers.

4. Results

This section aims to discuss the results of the model. First let's analyze the loss and accuracy for the training set. The model was split 80/20 for the training and testing set. The model was trained for 35 epochs using the Adam optimizer and a categorical_crossentropy loss function. We chose categorical_crossentropy because we are dealing with the probability of the next word in the sequence. When predicting the next word in a sequence, the goal is to generate a probability distribution over the vocabulary and pick the word with the highest probability. Categorical crossentropy loss helps the model learn to generate these probabilities more accurately based on the context of the preceding words in the sequence.

4.1 Training Loss and Accuracy

Metric	Training Set
Loss	0.2735
Accuracy	0.9052

The training loss and accuracy are very good. With the low loss and high accuracy it means the model can predict the next word with satisfcatory results. This is a good sign because it means the model is learning the features from the images and the captions. However, when we test the model with the testing set OR random images that aren't in the dataset at all it is important to evaluate the results based on the sentence structure and the semantics of the caption.

4.2 Testing Results

To test the model we created a method that asks the user to provide a number 1-10 (while there are 1600 images in the test set we limited the options to 10 for efficiency purposes). We tested the model using human analysis and BLEU score. BLEU score is a metric used to evaluate the quality of machine translated text. The higher the BLEU score the better the translation. The BLEU score is calculated by comparing the machine translated text to the human translated text. The score is calculated by comparing the n-grams of the machine translated text to the n-grams of the human translated text. To get a broad range of BLEU score results we tested it using 1-gram, 2-gram, 3-gram, and 4-gram.

We tested this feature by running it on 10 images. Below are the images with the best and worst outcome as well the BLEU score for the overall iteration we tested on.

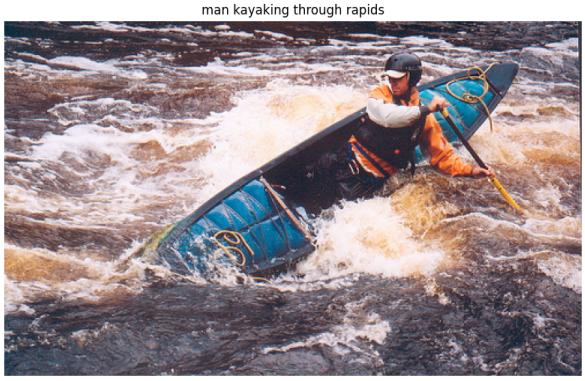

Figure 3: Best Image

Figure 4: Worst Image

Metric	BLEU Score
1-gram	0.547170
2-gram	0.329081
3-gram	0.218637
4-gram	0.0000

The captions generated showcase the best and worst that the model is able to showcase. For the first picture is was spot on and determined the caption correctly based on the features and the activity. However, for the second one it completely misses the mark and assumes people are in a picture who aren't even present within the picture (it looks like an animal race of some sort). This goes to show that while the training model was effective, it still has a lot of room for improvement. While these are the extreme cases, most of the generated captions were in in-between these two extremes. Most of the images accurately predicted the main topic (i.e a child or an animal doing some task); however, it failed at noticing key events in the background to project the full context. This could be a limitation of the model's training not showcasing these events since a lot of these images are unique and there aren't many events that re-present themselves in the dataset (ex: the Worst Image image is a unique event that may not have been captured in the dataset in augmented ways). Using human analysis we can tell that the captions aren't an ideal representation of the image but most of the time it is able to judge and capture some of the vital features of the image.

The BLEU score also showcases this, the 1-gram BLEU score is decent, but as the n-gram increases the score decreases. This indicates that as the sequence continues it isn't able to

effectively predict the next word. However, BLEU score isn't designed to capture the semantic meaning of the original sequence and the generated sequence. Hence, we used human analysis to determine the effectiveness of the model. In the future, metrics such as METEOR or CIDr could be used since they account better for semantic meaning. Nevertheless, the BLEU score was a solid indicator if the model was able to generate words that were on topic for the image.

Overall, the model didn't blow us away with the generated captions; however, based on limitations of the dataset we feel this is a satisfactory conclusion for our initial efforts. This is a great baseline to start with and build upon.

5. Reflection

When tackling the problem of image captioning, we first analyzed what we already knew. Throughout CPSC 393, we learned CNNs, LSTMs, and transfer learning models. However, we never really put all those ideas together (i.e Model Fusion) which was a necessary step for this project. The closest thing we learned to multiple models working together were transformers. With that initial knowledge, we sought to blend CNN and LSTMs together to create the image caption model. Also, we could use transfer learning to help with the mundane tasks like feature extractions. Once, we figured out this baseline we built an effective model that could generate captions for the images. Nevertheless, this approach still had some limitations. The dataset we used had over 8000 images; however, most of the pictures were of people or household animals. Furthermore, most of the people were white or had pale complexions. This could've easily poisoned our model training stage because it could assume that when it sees a person they are automatically white, or have a pale complexion. This is not the ideal when trying to create a model that can be used by everyone and for any type of pictures. Also, the pictures were mainly outdoors and people doing activities. This also limits the model's ability to learn because it may not learn what features are important to tell the interior of a house, school, workplace environment, etc. Our world has various activities and tasks that people and animals do. While, this model showcases the capabilities to train on an image dataset to generate somewhat accurate captions, it is not the best model to use for a wide variety of images.

To improve this model, it would be ideal to have two things: a training environment with access to a large number of GPUs. We trained this model on one NVIDA GPU, while it trained quickly we didn't have extra GPUs to handle a model that could be trained on a larger image dataset (maybe like 100,000 images with various features).

We hope this model creates a simple baseline for future researchers to use a simple, yet efficient, model design and tweak the hyperparameters and dataset information to create a more robust and effective model that can be used by all to generate alt text at an effective rate.

Overall, given our problem statement, i.e generate alt text, we created a sufficient model that can hold up to the task. However, we recognize there are limitations to this approach and hope that future researchers, as well as ourselves, can improve upon this model.

6. References

References are placed in order of appearance in the report.

1. HubSpot. HubSpot
2. Sharif, et. al. Understanding Screen-Reader Users' Experiences with Online Data Visualizations
3. Flickr8k Dataset. Flickr8k Dataset
4. Hockenmaier, et. al. Flickr8k Original Authors
5. GloVe Model. GloVe Model
6. Batch Normalization vs Layer Normalization. Understanding Batch Normalization, Layer Normalization and Group Normalization by implementing from scratch