CAPTION GENERATOR

Deep Learning Model to Automatically Describe Photographs

Team Members:

Sahithi Reddy Paspuleti
Mugdha Bajjuri
Sushmitha Boddireddy

Objective:

Aim of this project is to develop a deep learning model to generate a caption, given an image.

Introduction:

Caption generation is a challenging artificial intelligence problem where a textual description must be generated for a given photograph.

Caption generation requires both methods from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order.

Image captioning requires to recognize the important objects, their attributes and their relationships in an image. It also needs to generate syntactically and semantically correct sentences. Deep learning-based techniques are capable of handling the complexities and challenges of image captioning.

By seeing the above picture, different people may give different captions. For example, “children having fun near pool” or “people near a pool” or some people may tell “people enjoying seeing the fishes in the pool”. All these captions are suitable for the image. It is easy for a human being to caption the images by

looking at it. But for a machine to give a caption by analyzing the image needs training. So now this is possible with deep learning provided we have a required dataset.

Data Collection/Description:

We found that, a good dataset to use when getting started with image captioning is the Flickr8K Dataset. It is a dataset for automatic image description and grounded language understanding.
The downloaded data consists of two folders:

1. Flickr8k_Dataset
2. Flickr8k_text

We have downloaded the dataset from https://www.kaggle.com/shadabhussain/flickr8k

Image file Description:

Flickr8k_Dataset contains 8000 images in JPEG format.

Text files Description:

Flickr8k_text contains a number of files containing different sources of descriptions for the photographs. The dataset has pre-defined splitting as follows:

Training Set – 6000 images

Dev Set – 1000 images

Test Set – 1000 images

One of the files is "Flickr8k.token.txt" which contains the name of each image along with its 5 captions.

Every line consists of <image name> #caption number <caption>. Here the caption number varies from 0 to 4 as there are five captions for each image.

Data Preparation:

Preparing Photo data:

In our scenario, input to the model are images. These images cannot be sent directly to the model. We must send them to the model in the form of a vector. We need to convert every image into a fixed size vector and feed that to a neural network. In our project, we used the Oxford Visual Geometry Group (VGG), model that won the ImageNet competition in 2014.

Keras provides this pre-trained VGG model directly. So, we used this model to form an input vector. We pre-computed the photo feature using the pre-trained VGGmodel and saved them to file. We can load these features later and then send them to our model for interpretation of the photo. In this way we can train our models faster and use less memory. The last layer in the VGG class is used to classify the photo. As we are not interested in the classification of the photos, we can pop the layer. We have reshaped the images to

preferred size of 224*224-pixel image. We have photo_features_extraction function to do all of these. Here we load the photo to feed it in to VGG and then extract features from VGG. The image features are a 1-dimensional 4,096 element vector. We put this resultant dictionary in filename called features.plk. So, we can extract features from this file and use for our future use.

Preparing Text Data:

Understanding of Text data:

Here in Image processing, the text is something the model needs to predict, which means text is the output we expect from the model. So, while training the model captions(text) will be the target variable that the model is learning to predict. After downloading the dataset, we see one of the files "Flickr8k.token.txt" containing name of the image along with its 5 relevant captions as you can see in Fig 2.

For above image, we can see 5 captions in below screenshot. The same exists for all the images we have.

```
1000268201_693b08cb0e.jpg#0     A child in a pink dress is climbing up a set of stairs in
1000268201_693b08cb0e.jpg#1     A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2     A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3     A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4     A little girl in a pink dress going into a wooden cabin .
```

Fig.2

Preparing of data to train the model:

As you can see in Fig 2, every line contains "imagename.jpg#i" "caption". The i value will be 0<=i<=4 for every image. Now we created a dictionary named "mapping" to load the descriptions using load_photo_descriptions() function. This will contain the name of the image without the .jpg extension as keys and list of 5 captions for the corresponding image as values. After the function is executed the dictionary looks as below:

mapping['100268201_693b08cb0e']=[' A child in a pink dress is climbing up a set of stairs in an entry way .',' A girl going into a wooden building .',' A little girl climbing into a wooden playhouse .',' A little girl climbing the stairs to her playhouse .',' A little girl in a pink dress going into a wooden cabin .']

Data Cleaning:

Generally, while sending some text data to a model, we need to first clean the data to get good results. Cleaning of data means we prepare the raw data in such a way it fits a machine learning model. This is one of the important pre-processing steps before we start training the model in deep learning. We have done cleaning of data using "clean_image_descriptions()" function.

1. Tokenized the words in the descriptions.

2. We have removed special characters in the tokens.
3. Converted all the tokens to lower case letters.
4. Removed tokens which contains number attached in them.
5. Stored the descriptions as string in desc_list[i].
6. 8092 descriptions were loaded.

Converting the loaded descriptions to vocabulary of words:

Then, for about 40,000 captions available, we have created a vocabulary for all the unique words present using "convert_to_vocabulary()" function. The output was 8763, which means we have 8763 unique words across all the captions we have.

Loading the new descriptions after processing:

Now the descriptions after cleaning are stored using "save_captions()" function. We have appended the list in descriptions and saved in descriptions.txt

Deep Learning model:

Loading Training Set:

Flickr_8k.trainImages.txt consists of the names of the training images. Now, we loaded the descriptions of these images from "descriptions.txt" (saved on the hard disk) in the Python dictionary "train_descriptions". While loading we are adding two tokens at start and end of the caption. The two tokens added are 'endseq' and the 'startseq'. After that, we load the train_features with the features loaded in features.plk.

The descriptions in the text needed to be converted into numbers before it is fed to the model. Keras provides the Tokenizer class in order to map the words to a unique number. So, for that we covert the dictionary of cleaned descriptions to a list of descriptions using to_lines() function. Then using the generate_tokens() we fit a tokenizer to each description and also, we calculated the maximum length of the description among all the captions which we will be using in the future.

Preprocessing the captions:

In our project, Captions are the targets which are needed to be predicted by the model. We followed a procedure to predict the caption word by word given an image. Each word should be converted into a fixed size vector. So, the input to the model and the target which will be something as follows:

Image feature Vector	Partial caption	Target word

We have the Image feature vector which is given by the model. We have to generate the partial caption.

Generation of Partial Caption:

Groundwork on how to generate captions:

Consider we have the following three images.

A kid with red cap playing in the park
(Image 1)

A kid sleeping in bed
(Image 2)

A kid with red cap sleeping in bed
(Image 3)

Let us consider if we want to train the first two images and use the third image for testing.
We have to build the vocabulary for the first two captions. The vocab for the first image will be {A, kid, with, red, cap, playing, in, the, park}. A unique index is given to each word in the vocabulary i.e. startseq-1, A-2, kid-3, with-4, red-5, cap-6, playing-7, in-8, the-9, park-10, endseq-11.

Let us take the first image vector and the corresponding caption is "startseq A kid with red cap playing in the park endseq". For the first time we provide the image vector and the first word as input and try to predict the second word. And the image feature vector of the first image be Image_1.

So, for the first time:

Input= Image_1 + startseq= output= A

Thus, the corresponding data matrix of the caption is as follows:

Image feature vector	Partial caption	Target word
Image_1	starriest	A
Image_1	Startseq A	Kid
Image_1	startseq A kid	With
Image_1	startseq A kid with	Red
Image_1	startseq A kid with red	Cap
Image_1	startseq A kid with red cap	Playing
Image_1	startseq A kid with red cap playing	In
Image_1	startseq A kid with red cap playing in	The
Image_1	startseq A kid with red cap playing in the	Park
Image_1	startseq A kid with red cap playing in the park	Ends

So here one image plus caption is not a single data point here, here it is multiple data points depending on the length of the caption.

Similarly, we can construct the data matrix for the second image also. Both the image vector and partial caption.

Therefore, we employed the above process on our captions.

Here we are processing sequences, we employed a Recurrent Neural Network to read these captions.

We are not going to pass the English word, so we created sequences of indices where each index represents a unique word. All this is done in the generate_sequences() method which returns the sequences of images, sequences of words and the target word. The data points will look as below:

Image feature vector	Partial caption	Target word
Image_1	[1]	2
Image_1	[1,2]	3
Image_1	[1,2,3]	4
Image_1	[1,2,3,4]	5
Image_1	[1,2,3,4,5]	6
Image_1	[1,2,3,4,5,6]	7
Image_1	[1,2,3,4,5,6,7]	8
Image_1	[1,2,3,4,5,6,7,8]	9
Image_1	[1,2,3,4,5,6,7,8,9]	10
Image_1	[1,2,3,4,5,6,7,8,9,10]	11
Image_2	[2]	3
Image_2	[2,3]	6
Image_2	[2,3,6]	1
Image_2	[2,3,6,1]	5
Image_2	[2,3,6,1,5]	4
Image_2	[2,3,6,1,5,4]	7

To make sure that each sequence is of equal length we did zero padding. The length of the zero appended depends on the length of the caption which is maximum. So, we appended zeros until the length of the sequence is equal to maximum length of the caption given in the data set. So, after padding with zeros the resultant data point will look like below

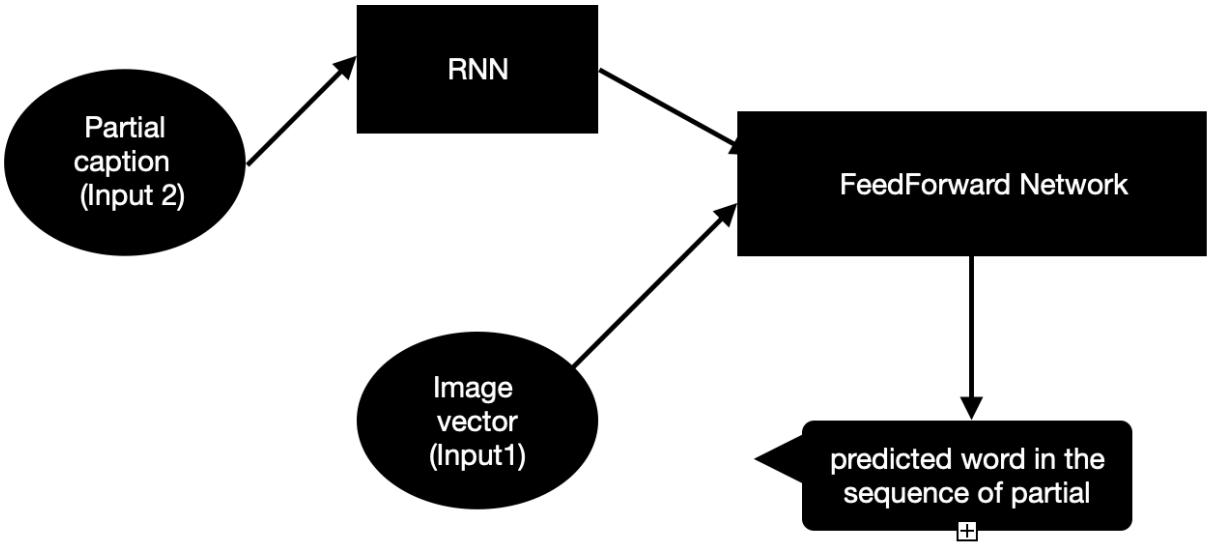
Image feature vector	Partial caption	Target word
Image_1	[1,0,0…..0]	2
Image_1	[1,2,0,0….0]	3
Image_1	[1,2,3,0,0….0]	4
Image_1	[1,2,3,4,0,0….0]	5
Image_1	[1,2,3,4,5,0,0….0]	6
Image_1	[1,2,3,4,5,6,0,0….0]	7
Image_1	[1,2,3,4,5,6,7,0,0….0]	8
Image_1	[1,2,3,4,5,6,7,8,0,0….0]	9
Image_1	[1,2,3,4,5,6,7,8,9,0,0….0	10
Image_1	[1,2,3,4,5,6,7,8,9,10,0,0….0],	11
Image_2	[2,0,0….0]	3
Image_2	[2,3,0,0….0]	6
Image_2	[2,3,6,0,0….0]	1
Image_2	[2,3,6,1,0,0….0]	5
Image_2	[2,3,6,1,5,0,0….0]	4
Image_2	[2,3,6,1,5,4,0,0….0]	7

In the similar way the dev data is also preprocessed to test the trained model.

We now have enough data to train the model.

Defining the model:

Input consists of two parts. One is the image vector and the other is the partial caption. So we used the Functional API to create the merge models.



The sequence of partial captions is fed to the special Recurrent Neural Network i.e. LSTM.

Why we used LSTM?

LSTMs are used to avoid long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

The image vector obtained by preprocessing the photos by VGG model and the output from the RNN(LSTM) are merged together and processed by a Dense Layer to make prediction.

The Photo Feature Extractor model expects input photo features to be a vector of 4,096 elements. These are processed by a Dense layer to produce a 256-element representation of the photo. In order to ignore the padded values, the sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer. Then it is fed to the LSTM layer with 256 memory units. Both the input models produce a 256-element vector and to reduce over fitting we added a dropout layer with 0.5 dropout.

The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence. We compiled the model using adam optimizer.

Fitting the model:

We fitted the model on to the training data set.

First, we monitored the skill of the trained model on the holdout development dataset. When the skill of the model on the development dataset improves at the end of an epoch, we saved the whole model to file. At the end of the run, we used the saved model with the best skill on the training dataset as our final model. We did this by using a ModelCheckpoint pre-defined in Keras and specified it to monitor the minimum loss on the validation dataset and saved the model to a file that has both the training and validation loss in the filename. We then specified the checkpoint in the call to fit() via the callbacks argument. We also specified the development dataset in fit() via the validation_data argument. We have fitted the model for 10 epochs

Model Summary

```
Dataset: 6000
Descriptions: train=6000
Photos: train=6000
Vocabulary Size: 7579
Description Length: 34
Dataset: 1000
Descriptions: test=1000
Photos: test=1000
Model: "model_6"
```

Layer (type)	Output Shape	Param #	Connected to
input_10 (InputLayer)	(None, 34)	0	
input_9 (InputLayer)	(None, 4096)	0	
embedding_4 (Embedding)	(None, 34, 256)	1940224	input_10[0][0]
dropout_7 (Dropout)	(None, 4096)	0	input_9[0][0]
dropout_8 (Dropout)	(None, 34, 256)	0	embedding_4[0][0]
dense_10 (Dense)	(None, 256)	1048832	dropout_7[0][0]
lstm_4 (LSTM)	(None, 256)	525312	dropout_8[0][0]
add_4 (Add)	(None, 256)	0	dense_10[0][0] lstm_4[0][0]
dense_11 (Dense)	(None, 256)	65792	add_4[0][0]
dense_12 (Dense)	(None, 7579)	1947803	dense_11[0][0]

```
Total params: 5,527,963
Trainable params: 5,527,963
Non-trainable params: 0
```

None

We have observed the following results after fitting the model:

```
Train on 306404 samples, validate on 50903 samples
Epoch 1/10
 - 1726s - loss: 4.5311 - val_loss: 4.0807

Epoch 00001: val_loss improved from inf to 4.08075, saving model to model-ep001-loss4.531-val_loss4.081.h5
Epoch 2/10
 - 1178s - loss: 3.8672 - val_loss: 3.9211

Epoch 00002: val_loss improved from 4.08075 to 3.92109, saving model to model-ep002-loss3.867-val_loss3.921.h5
Epoch 3/10
 - 1234s - loss: 3.6514 - val_loss: 3.9002

Epoch 00003: val_loss improved from 3.92109 to 3.90022, saving model to model-ep003-loss3.651-val_loss3.900.h5
Epoch 4/10
 - 1248s - loss: 3.5440 - val_loss: 3.8913

Epoch 00004: val_loss improved from 3.90022 to 3.89130, saving model to model-ep004-loss3.544-val_loss3.891.h5
Epoch 5/10
 - 1239s - loss: 3.4805 - val_loss: 3.8987

Epoch 00005: val_loss did not improve from 3.89130
Epoch 6/10
 - 1235s - loss: 3.4403 - val_loss: 3.9228

Epoch 00006: val_loss did not improve from 3.89130
Epoch 7/10
 - 1192s - loss: 3.4158 - val_loss: 3.9503

Epoch 00007: val_loss did not improve from 3.89130
Epoch 8/10
 - 1207s - loss: 3.3951 - val_loss: 3.9654

Epoch 00008: val_loss did not improve from 3.89130
Epoch 9/10
 - 1149s - loss: 3.3849 - val_loss: 3.9707

Epoch 00009: val_loss did not improve from 3.89130
Epoch 10/10
 - 1206s - loss: 3.3810 - val_loss: 4.0396

Epoch 00010: val_loss did not improve from 3.89130
```

After 5th epoch the validation loss did not improve from 3.89130. Now we are using the model at the 4th epoch to evaluate or the predict the new picture. This prediction or evaluating the model is done in the evaluating the model block.

Evaluating the model:

We are evaluating the model by using the holdout test dataset. We followed the process of first generating descriptions for all the photos in the test dataset followed by evaluating those predictions with a standard cost function.
The function named "create_desc()" given a prepared photo as input, first generates a description for a photo using a trained model. This involves passing in the start description token "*startseq*", generating one word, then calling the model recursively with generated words as input until the end of sequence token is reached "*endseq*" or the maximum description length is reached.
It then calls the function "word_for_id()" in order to map an integer prediction back to a word.
Then we have the function named "evaluate_model()" which will evaluate a trained model against a given dataset of photo descriptions and photo features. The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text.
As said before, we have used BLEU metrics for evaluating the model. A learned metric should be capable of correctly distinguishing human written captions from machine generated ones. What the BLEU score does is given a machine generated translation, it allows to automatically compute a score that measures how good is that machine translation. And the intuition is as long as the machine generated translation is pretty close to any of the references provided by humans, then it will get a high BLEU score. BLEU scores range between 0 and 1. Here, we compared each generated description against all of the reference descriptions for the photograph. We then calculated BLEU scores for 1, 2, 3 and 4 cumulative n-grams. A higher score close to 1.0 is better, a score closer to zero is worse.

Grams	Weights	BLEU scores
1	1.0, 0, 0, 0	0.534015
2	0.5, 0.5, 0, 0	0.275758
3	0.3, 0.3, 0.3, 0	0.181298
4	0.25, 0.25, 0.25, 0.25	0.077716

Generating a new caption:

Whenever a new photograph is fed to the model, it needs to generate a new caption which describes the photograph.
First, we have loaded the Tokenizer from *tokenizer.pkl* and defined the maximum length (34) of the sequence to generate, needed for padding inputs.
Then we have downloaded a photo onto our local directory and have loaded the same photograph for which we want to extract features and get the description to the model.
We have defined a function "extract_photo_features" where we have re-defined the model and used VGG model to extract features, converted the image pixels to a NumPy array and then re-shaped the image. Now we have pre-processed the image to send for VGG model using "preprocess_input(image)" function. We have sent the results to "feature" variable and have returned the variable to "extract_photo_features" function.

Results:

Captions generated on Example images:

Model was able to generate meaningful captions for an image in most of the cases. However, in some cases it missed some of the key features in the images this may be due to the lack of tokens in the dictionary to find the event.

Image	Caption generated

For below cases, model was able to perfectly find out the details in an image

	**Caption generated** Man is surfing in the water Image name: surf.png
	**Caption generated** Man in red shirt is standing on the beach Image name: beach.jpg

For below images, model is not able to capture all of the key features.

	**Caption generated** Man in blue shirt is standing on the beach **Human provided caption** Man is '*doing backflip on wakeboard'* near the beach Image name: Backflip.png
	**Caption generated** man in black shirt is standing on the street **Human provided caption** Man in black shirt *is 'playing guitar'* Image name: guitar.png
	**Caption generated** Man in blue shirt is standing on the street **Human provided caption** Man is '*Holding bananas*' on the street Image name: bananas.png

	**Caption generated** dog is running through the grass **Human provided caption** Dog is '*jumping over bar'* Image name: dog.png

We have taken a special case, where to confuse the model with an ambiguous image of a rabbit's face morphed on a lion and it detects a dog.

	**Caption generated** Dog is running through the snow. Image name: morphed.png

Some of the challenges we faced:

Building models on our workstation using a CPU

Our dataset is about 1GB in size and running all the images on our CPU is difficult. So, we have decided on training our deep learning model on Google colab GPU. We did all the pre required steps and made google colab ready to use. At the photo data preparation step, after 2 hours of execution, we received an error saying "Your session crashed after using all available RAM"
Our data is huge. After the RAM is maxed out, we immediately get the prompt of crash and option to increase my RAM. When an option of increasing the RAM is selected, we would have to perform all the execution again from start which taken couple of more hours but still the extra RAM provided was also not sufficient. Google Colab gives free 25 Gb as Ram space. But for our model we need more which we weren't able to get from the amazing google colab.
Next, we proceeded with trying AWS EC2 GPU but launching the proper AMI along with the required hardware configurations is not possible with a free tier AWS account. We would have to pay for the AWS monthly account.
Finally, we ended up training the model on our workstation which took about 16hours to execute.

Results and accuracy aren't very great

Model's accuracy can be boosted when a larger dataset is used. Larger dataset meaning the words in the vocabulary of the model increase significantly which we believe would provide better results. Also using relatively newer architectures would increase the accuracy in predicting which in turn would reduce the error rate in the language generation.

Conclusion:

We have presented an end to end neural network system that is capable of viewing an image and generating a descriptive caption in English depending on the words generated in the dictionary based on tokens in captions in train images. The model has a convolutional neural network encoder and a LSTM decoder that helps in generation of sentences.

Model shows decent results on Flickr 8k dataset. We evaluate the accuracy of the model on the basis of BLEU scores. We would be interested to try this on the bigger dataset.

References:

- Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." Proceedings of the IEEE Image Captioning conference on computer vision and pattern recognition. 2015 - Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan
- "Merge-model" described by Marc Tanti, et al. 2017
- Where to put the Image in an Image Caption Generator https://arxiv.org/abs/1703.09137 - Marc Tanti, Albert Gatt, Kenneth P. Camilleri
- https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/ - Jason Brownlee