CS-1390 Final Project

Manya Sachdev, Pratham Singh
December 21, 2021

1 INTRODUCTION

This project is about creating an image caption generator. Image to sentence conversion is a deep learning problem which which combines Computer Vision and Natural Language Processing. Some of the applications include but are not limited to:

1. Image to speech conversion for visually impaired people
2. Image based search service
3. Autonomous driving

We can approach this problem by identifying the inputs and outputs. The input to an image caption generator is some kind of image like a matrix or a tensor. The output is a sentence, basically a sequence. A sequence is a set of variables that has some defined ordering to it, because one word has to come after another and in that order to have some meaning.

We can use neural networks to solve this problem. Neural networks are basically mathematical functions that transform one kind of variable to a variable of another kind. It could be vectors to vectors as we would see in classification problems or vectors to scalars as we would see in regression networks.

2 LITERATURE SURVEY

In 2014, research scientists on the Google Brain team trained a machine learning system[4] to automatically produce captions that accurately describe images. Further development of that system led to its success in the Microsoft COCO 2015 image captioning challenge, a competition to compare the best algorithms for computing accurate image captions, where it tied for first place. They later made the system open-source and it is available as a library in tensorflow.

3 Project Description and Survey

We initially wanted to create a lyrical caption generator. People who use Instagram or other photo-sharing platforms commonly use song lyrics or quotes to caption their pictures when they post it. The same people struggle to find such captions and surf around looking for them. The idea was to design a product for such people. A lyrical caption generator would have taken an image as an input and the output would have been a famous quote or song lyric related to the picture which could have been used as a caption for a picture being posted.

Unfortunately, we were unable to implement a lyric-based image caption generator due to lack of time and resources and ended up making a normal image caption generator.

4 Dataset Specification

We have used the Flickr 8K data set to build these models. It contains over eight thousand images, each of which are annotated with five different captions which provide description of the entities in the image. This data set contains a training data set of six thousand images, validation data set of one thousand images and test data set of one thousand images.

5 Method

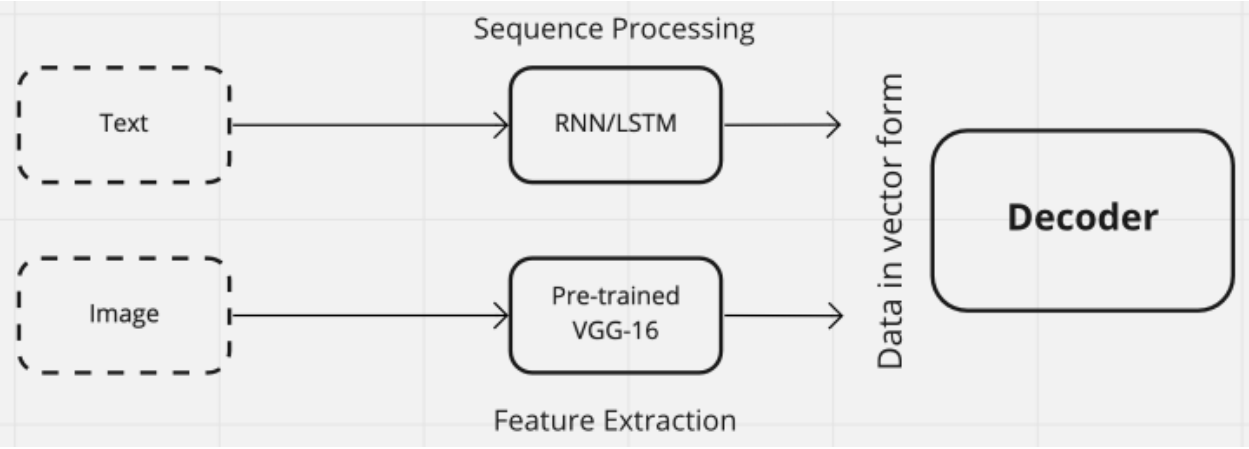

Figure 5.1: Caption Generator Model

6 Implementation details

6.1 Feature Extraction

We will be using a pre-trained Visual Geometry Group (VGG-16)[3] model provided by Keras to extract features from the images in the data set. The number 16 in VGG-16 depicts the number of weight layers in the model. It is a CNN based model containing 3x3 convolution layers stacked on top of each other in increasing depth. We chose this over other models like ResNet because the VGG-16 model is relatively small and can be trained faster. In the feature_ extraction() function, we remove the first layer of the VGG-16 model as it is for image classification purposes and we do not need to classify images.

```
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
```

We then use this function to extract features from each image in the dataset and store the features in a pickle file called extracted_ features.pkl.

6.2 Defining Vocabulary and Loading Descriptions

We now clean the descriptions of the images by removing any punctuation, numbers or redundant words. We also convert the descriptions to a vocabulary of words so that the embedding layer of our underlying LSTM model can understand these word tokens to generate captions for the images. We store the result in a file called descriptions.txt.

We now load a pre-defined set of image identifiers and the load the descriptions while wrapping them with tokens ('start' and 'end') and store them in a dictionary with their corresponding image ID. The wrapping is done because in building the sequence to sequence architecture, 'end' sequence is fed as the input to the decoder cell and 'start' sequence is the input to the encoder cell.

6.3 Tokenization and Sequence Creation

We need to encode the text into numbers now so that our model can understand the data. No machine learning model can take text-based data as input directly, hence the conversion. In order to encode the data, we create mappings of the text to integer values (using one-hot encoding). Keras provides a tokenizer class which helps us do the same. We also create arrays of sequences so as to train our LSTM model later.

6.4 Model Definition

Now, we will merge the VGG-16 model as well as the LSTM model and process them through a dense layer in order to predict the caption.

We first have an input layer for the features on which we use a Dropout layer (50%) to address over-fitting of data. We then use a Dense layer with a 'relu' (Rectifier Linear Unit) activation model to produce. 256 element representation of the image.

```
inputs1 = Input(shape=(4096,))
        fe1 = Dropout(0.5)(inputs1)
        fe2 = Dense(256, activation='relu')(fe1)
```

Then we input the sequences or descriptions in another input layer and feed them to an Embedding layer. We now use a Dropout layer (50%) for regularization and pass it through the LSTM layer.

```
inputs2 = Input(shape=(max_length,))
        se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
        se2 = Dropout(0.5)(se1)
        se3 = LSTM(256)(se2)
```

Now, the decoder model merges both the above vector by performing an add operation and passes it through two Dense layers for the prediction of the next word in the sequence.

```
decoder1 = add([fe2, se3])
        decoder2 = Dense(256, activation='relu')(decoder1)
        outputs = Dense(vocab_size, activation='softmax')(decoder2)
```

Now, the model is bind together using the Model function from keras and is compiled using the 'adam' optimizer because it is computationally efficient, has little memory requirement, is invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data.

```
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

6.5 Model Training

The above defined image based model needs a large amount of memory to be trained. We needed at least 32GB of RAM to train the model directly. Due to lack of memory, we decided to train the model progressively. Keras supports progressive loading of datasets. We create a stand-alone generator function which yields one batch of dataset at a time. As we know, a particular deep learning model is fit on multiple epochs where one epoch depicts one pass through the entire training data set of images. One epoch is comprised of multiple batches of data and model weights get updated at the end of each batch.

```
def data_generator(descriptions, photos, tokenizer, max_length, vocab_size):
        while 1:
                for key, desc_list in descriptions.items():
                        photo = photos[key][0]
                        in_img, in_seq, out_word = create_sequences(tokenizer, max_length,
                                                        desc_list, photo, vocab_size)
                        yield [in_img, in_seq], out_word
```

Using this, we save a large amount of memory and are able to run this on our machine without the use of a GPU.

6.6 Model Evaluation

We use BLEU[2] Score to evaluate our model. BLEU Score stands for Bilingual Evaluation Understudy Score and it summarizes how close a particular generated text is to the expected text using a score between 0 to 1. BLEU Score is majorly prevalent in machine translation but it can be used to evaluate other types of models related to image captioning, text summarizing, speech recognition, etc., that is, any problem where we have different target variations of an input text sequence. We use the sentence_ blue() function from the NLTK library to calculate the BLEU score.

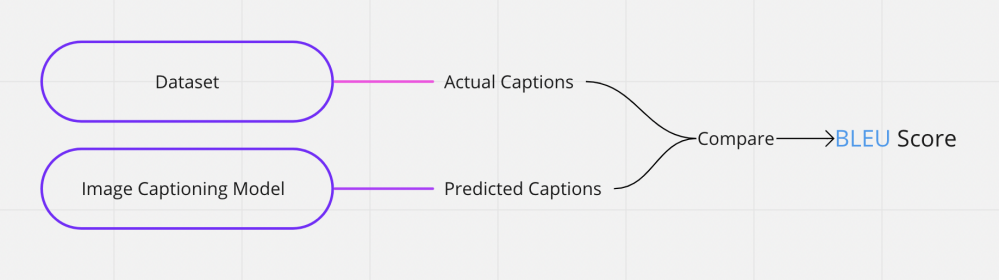

Figure 6.1: BLEU Score

```
Dataset: 6000
Descriptions: train=6000
Vocabulary Size: 7579
Description Length: 34
Dataset: 1000
Descriptions: test=1000
Photos: test=1000
BLEU-1: 0.483238
BLEU-2: 0.247558
BLEU-3: 0.166703
BLEU-4: 0.074481
```

Figure 6.2: Evaluation Output

7 OBSERVATIONS

To finally predict the caption of a new image, we first extract the features of this image using the existing VGG-16 model and then generate a description a caption for the image using the model of our choice.

Figure 7.1: Input image

```
startseq man in red shirt is riding bike down the street endseq
```

Figure 7.2: Output sequence

As we can see that the predicted caption is not entirely accurate but does a decent job. This was derived using the model created after the 12th epoch iteration. The results can be further improved if we train the model for more epochs.

8 CONCLUSION

In conclusion, we feel that the model was a success as it was able to predict the caption of the input image with good accuracy. Although we faced a lot of issues while training the model (project crashed multiple times), we were able to do it successfully using progressive loading. There is a lot of scope for improvement of the results. Possible options are training the model with a larger dataset, using more epochs or using a better GPU-based virtual machine on cloud (using Amazon EC2).

If we had more time and resources, we could have implemented the original idea that was so make a lyrical caption generator. The data required to make such a model was not easily available. In retrospection, we can scrape the data (song lyrics) from the web manually and curate the dataset on our own. We can then use Natural Language Processing to find similar lyrics to the caption generated by the existing model and display the result to the user via a web-app. We plan on implementing this in the future and building a product around the same.

REFERENCES

[1] J. Brownlee. Develop a deep learning caption generation model. [Online]. Available: https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/

[2] T. W. Kishore Papineni, Salim Roukos and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. [Online]. Available: https://aclanthology.org/P02-1040.pdf

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. [Online]. Available: https://arxiv.org/abs/1409.1556

[4] O. Vinyals *et al.* Show and tell: A neural image caption generator. [Online]. Available: https://arxiv.org/pdf/1411.4555.pdf