Image Captioning using Deep Learning Techniques

By: Prakruthi Rao
Advisor: Dr. Yingshu Li
IMAGE CAPTIONING

- Process of generating captions for an image.
- Describes the image by recognizing:
  - The objects present in the image.
  - The relationship between the objects.
  - The object attributes.
- Image Captioning combines Computer Vision and Natural Language Processing fields.
- More complex than image classification and image recognition.
- Deep Learning Techniques are used to achieve this.
A group of people sitting at a produce market.

A group of people sitting at a fruit stand.
GOAL

Implementing Deep Learning Algorithms and obtain the feature vector of the images. Apply LSTM and bi-directional LSTM to the model.

Images

Predict the caption
SYSTEM FUNCTIONALITIES

Front-End Implementation

ReactJS

Back-End Implementation

Python
- **Image captioning**:
  - Input – Image.
  - Output – A syntactically correct caption.

- **Video Captioning (Base for future work)**
  - Input – Video – capture a video frame.
  - Output – Caption for the video frame
  - Method – Sending the video frame as an input to the image captioning model.
DATASET - Flickr-8k Dataset
IMPLEMENTATION

Data Loading

Data Pre-processing

Model Architecture

Training

Inference
DATA LOADING

- Flickr8k
  - Flickr8k_dataset - consists of images.
  - Flickr8k_text - consists of images with 5 captions each
<table>
<thead>
<tr>
<th>Image Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000268201_693b08cb0e.jpg#0</td>
<td>A child in a pink dress is climbing up a set of stairs in an entry way.</td>
</tr>
<tr>
<td>1000268201_693b08cb0e.jpg#1</td>
<td>A girl going into a wooden building.</td>
</tr>
<tr>
<td>1000268201_693b08cb0e.jpg#2</td>
<td>A little girl climbing into a wooden playhouse.</td>
</tr>
<tr>
<td>1000268201_693b08cb0e.jpg#3</td>
<td>A little girl climbing the stairs to her playhouse.</td>
</tr>
<tr>
<td>1000268201_693b08cb0e.jpg#4</td>
<td>A little girl in a pink dress going into a wooden cabin.</td>
</tr>
</tbody>
</table>
- The `flickr8k_text` file is loaded.
- Data cleaning is performed
  - Removed special characters
  - Words containing numbers
  - Convert into lower-case
- A set of unique words i.e. corpus is created.

def clean_description_data(descriptions):
    # Replace all punctuations to empty character
    remove_punctuation = str.maketrans('', '', string.punctuation)
    for key, list_of_descriptions in descriptions.items():
        for i in range(len(list_of_descriptions)):
            description = list_of_descriptions[i]
            description = description.split()
            # convert all the words to lowercase
            description = [word.lower() for word in description]
            # Remove all punctuations
            description = [w.translate(remove_punctuation) for w in description]
            # Remove any word whose length is less than 1
            description = [word for word in description if len(word)>1]
            # Remove any word whose contents are not alphabets
            description = [word for word in description if word.isalpha()]
        list_of_descriptions[i] = ' '.join(description)
DATA PRE-PROCESSING

Data pre-processing

Image dataset pre-processing

Text dataset pre-processing
Feature Extraction

Achieved by Transfer Learning.
- InceptionResNetV2 – a pre-trained model.
- This model is implemented on the ImageNet dataset.
- The ImageNet dataset is an image classification dataset.
In order to get the feature vector, we stop the process 2 steps before the final step.
This will give us the model to obtain the Image Feature Vector.
Flickr Images → InceptionResNetV2 pre-trained model → Feature Vector (1356 length)
TEXT DATASET PRE-PROCESSING

- The captions along with the image name are stored in a dictionary.
- Two tokens are added to each caption:
  - captStrt \(\rightarrow\) added at the beginning of a caption
  - captEnd \(\rightarrow\) added at the end of a caption
- Two dictionaries are created
  1) Dict1 \(\rightarrow\) word to index
  2) Dict2 \(\rightarrow\) index to word
Data Embedding is done using the pre-trained GloVe Method.
Caption → GloVe model → Fixed-size Vector (200 length)
MODEL ARCHITECTURE

- **Image Vector**
  - Dense Layer
  - Bidirectional LSTM SoftMax
  - Predicted next word

- **Caption Vector**
  - LSTM
- **Image model**
  - Uses a dense layer along with Relu Activation

- **Caption model**
  - Long Short-Term memory model

- **Final Merge model**
  - Bi-directional Long Short-Term memory model
  - Softmax Activation

- The final model is compiled using RMSprop().

```python
from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector, Merge, 
                     Activation
import data_generator
from keras.models import Sequential
from keras.layers.wrappers import Bidirectional
from keras.optimizers import RMSprop

def train_the_data(class_name, max_length, vocab_size, embedding_dim, train_descriptions, train_features, wordtoix):
    image_model = Sequential([
        Dense(embedding_dim, input_shape=(1536,), activation='relu'),
        RepeatVector(max_length)
    ])

    caption_model = Sequential([
                                Embedding(vocab_size, embedding_dim, input_length=max_length),
                                LSTM(256, return_sequences=True),
                                TimeDistributed(Dense(300))
                            ])

    # Merging the image and caption model and moving on to other layers.
    caption_generator_model = Sequential([[
                                            Merge([image_model, caption_model], mode='concat', concat_axis=1),
                                            Bidirectional(LSTM(256, return_sequences=False)),
                                            Dense(vocab_size),
                                            Activation('softmax')
                                        ]])

caption_generator_model.compile(loss='categorical_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])
```
Bi-directional - LSTM

LSTM
Once the model is compiled, it is ready to be trained with the images.

Data generator is applied in order to save memory.

The training is done in batches. Here it is 3 images per batch.

A total of 2000 epochs takes place.

Training is done in batches.

The weights are obtained once the model is trained through Backpropagation.
epochs = 10
count_pics_per_batch = 3
steps = len(train_descriptions) // count_pics_per_batch

for i in range(epochs):
    generator = data_generator.data_generator(train_descriptions, train_features, wordtoix, max_length,
                                              count_pics_per_batch, vocab_size)
    print(generator)
    caption_generator_model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)

caption_generator_model.save('Model/model_' + class_name + '.h5')
New images are tested against the trained model.
- Image vector is calculated for the new image using InceptionResNetV2.
- The vector along with the weights and the partial caption are sent to the model.
- The model generates a probability of words as output.
- Beam Search is implemented to select one word.
- Searches level by level like BFS.
- The word given as output is appended to the partial caption and the process continues.
import numpy as np
from keras.preprocessing import sequence

This function runs the model multiple times, meanwhile generating the caption one word at a time

def beamsearch(image_feature_vector, caption_generator_model, max_length, word_to_index, index_to_word, graph):
    beam_index = 3
    start = [word_to_index["start"]]
    start_word = [[start, 0.0]]

    while len(start_word[0][0]) < max_length:
        temp = []
        for s in start_word:
            partial_caption = sequence.pad_sequences([[s[0]], max_length, padding='post')
            with graph.as_default():
                predictions = caption_generator_model.predict([image_feature_vector, partial_caption])
            word_predictions = np.argsort(predictions[0])[-beam_index:]

            for w in word_predictions:
                next_cap, prob = s[0][1], s[1]
                next_cap.append(w)
                prob *= predictions[0][w]
                temp.append([next_cap, prob])

        start_word = temp
        # Sorting the words according to the probabilities
        start_word = sorted(start_word, reverse=True, key=lambda x: x[1])
        # Get the words with highest probabilities
        start_word = start_word[-beam_index:]

    start_word = start_word[-1][0]
    intermediate_partial_caption = [index_to_word[i] for i in start_word]

    generated_caption = []

    for i in intermediate_partial_caption:
        print(i)
        if i != 'end':
            generated_caption.append(i)
        else:
            break

    generated_caption = ' '.join(generated_caption[1:]
    return generated_caption
CONTINUES TILL CAPTEND IS PREDICTED
APPLICATIONS

AID TO THE VISUALLY IMPAIRED
SELF-DRIVING CARS
SOCIAL MEDIA, CAPTIONS FOR IMAGES IN PRESENTATIONS
SECURITY LIKE CCTV CAMERAS
DEMO
THANK YOU!!