DS-GA 1019
Image Caption Generator using Keras Optimization

Group 16: Wanyi Yang (wy815), Haohang Yan (hy2664), Yunming Chen (yc4042)
Jordan Tian (yt2639), Weidong Liu (wl2841), Jiaxin Li (jl14333)

1 Introduction

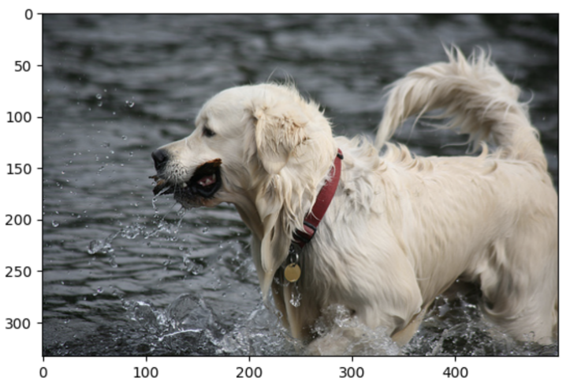
Image caption generator is a cutting-edge application of artificial intelligence that has gained significant attention in recent years. It is a system that automatically generates textual descriptions or captions for images by analyzing the visual features and context of the images. The process involves training the model on a large dataset of images and their corresponding captions using advanced techniques of deep learning and natural language processing.

The accuracy of the image caption generator model depends heavily on the size of the training dataset. Therefore, the training process can be quite time-consuming and computationally expensive. Our project aims at implementing various optimization techniques to reduce the training time and computational requirements while still achieving high accuracy and performance levels.

2 Dataset

The Flickr8k dataset consists of 8091 images, and all of them are associated with a single token.txt file containing the image ID along with five different captions for each image. These captions provide clear descriptions of the salient objects and events in the images. To train our model, we use a total of 8000 images, reserving the remaining 91 images for testing purposes.

Here is an example image with its corresponding captions.



```
['a wet white dog is splashing in the water with a stick in its mouth ',
 'a white dog is in the water ',
 'a white dog treads the water with sticks in its mouth ',
 'a white dog with red collar splashes in the water ',
 'the dog is wading through shallow water while holding something in its mouth ']
```

3 Model Training Workflow

In the process of training our image captioning model, we begin with raw data consisting of data points, each with one image corresponding to several sentences. We utilize two TensorFlow models for image preprocessing: a GloVe model for embedding descriptions and an Xception model for embedding images. While obtaining image and description features, we also create additional variables such as a word dictionary, description length, and an image-description dictionary.

To generate training data for the model, we develop a data generator, which is fed into the model for every epoch. Each generated data point consists of inputs x1 and x2, along with its output y. For instance, if the image caption is "an airplane is on the runway", the first data point will be (x1: image_feature, x2: an), with the output (y: airplane). The second data point will be (x1: image_feature, x2: an airplane), with the output (y: is). The output y represents the word we aim to predict after x2, based on the image feature x1.

By optimizing the data generator, we reduce the training time for each epoch. Once trained, our RNN-LSTM model predicts the probabilities of words following a given sequence based on image features. Using these probabilities, we employ greedy search and beam search techniques to generate a description for an image based on the predicted word probabilities.

4 Optimization Methods

4.1 Data loading (Cython, Multithreading)

We create a function to parse image descriptions from a document, generating a dictionary with image IDs as keys and description lists as values. We implement multithreading for data loading using ThreadPoolExecutor with 64 workers, removing punctuation and converting words to lowercase. The dataset is split into training and testing sets to evaluate model performance. Data loading processes are optimized using Cython, which translates Python code into C/C++ for improved performance Cython allows direct access to C libraries and functions, which can significantly improve performance for computationally-intensive tasks or tasks that require interfacing with low-level system components.

4.2 Data Preprocessing (Vectorization, Multithreading)

Image descriptions are cleaned by removing punctuation and converting words to lowercase for consistency. We build a vocabulary and create word-to-index and index-to-word dictionaries for encoding and decoding. To handle varying description lengths, we determine the maximum description length to pad or truncate the sequences accordingly.

A numpy matrix with pre-trained GloVe embeddings is generated for better semantic understanding, enabling element-wise operations on arrays without explicit loops, and optimized for reduced memory usage and enhanced computational efficiency. Lastly, the feature extraction function reads and preprocesses training images before utilizing the Xception model to extract image features. This process is parallelized across 24 threads, each equipped with its own Xception model instance. This enables simultaneous feature extraction from multiple images, as opposed to sequential processing.

4.3 Model Generator (PlaidML, Generator, Multithreading, Mini-batch)

PlaidML is configured as the Keras backend to facilitate GPU acceleration, as it enhances deep learning tasks by harnessing the parallel processing capabilities of GPUs. A custom neural network model is designed for image captioning. The model accepts two inputs: image features derived from the Xception model and a sequence of words. The architecture comprises dropout layers, dense layers, an LSTM layer, and a final softmax layer that yields a probability distribution over the vocabulary.

The word embedding layer is initialized with pre-trained word embeddings, and its weights are set to non-trainable. The model is compiled using categorical cross-entropy loss and the Adam optimizer.

The data generator function serves as an iterator that produces input-output pairs in mini-batches, using image features and a partial caption sequence as input, and the subsequent word in the caption sequence as output. The model is trained for 15 epochs with a batch size of 3. This generator function enables on-the-fly training data generation while optimizing memory usage. The function's concurrent execution of multiple threads allows for efficient data batch preparation for training by processing several descriptions and images simultaneously. Furthermore, mini-batch training conserves memory efficiency and enables more frequent weight updates for the model.

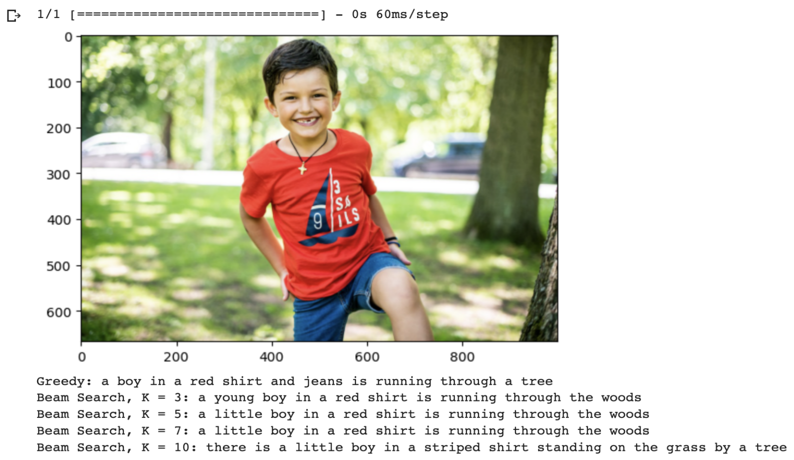
4.4 Search method (Greedy search & Beam search)

Initially, we use greedy search to generate captions for each test image, where the algorithm generates the output sequence token by token, selecting the token with the highest probability at each step. We then tried beam search which considers multiple candidate sequences in a parallel manner. Beam search maintains a set of top K candidate sequences, it expands these sequences by evaluating all possible next tokens and selects the most likely sequence based on the cumulative score of the probability of considering each token in the sequence. By considering multiple candidate sequences, beam search avoids getting trapped in local optimization and provides more accurate captions.

It is worth noting that beam search introduces a higher computational cost. Although the runtime performance of the algorithm is not optimized, the beam search greatly improves the quality of the test results, especially when considering more complex relationships and contexts in the images.

5 Result

After training the model, we conducted a test using a set of images. One of the results is displayed below. Based on the results, encoding the image takes 60ms, obtaining the output using greedy search requires 0.79s, and generating the output using beam search with k=10 takes 22.04s. Clearly, beam search is slower than greedy search; however, it offers superior performance, as evidenced by the caption, "there is a little boy in a striped shirt standing on the grass by a tree."



```
1/1 [==============================] - 0s 60ms/step
```

```
Greedy: a boy in a red shirt and jeans is running through a tree
Beam Search, K = 3: a young boy in a red shirt is running through the woods
Beam Search, K = 5: a little boy in a red shirt is running through the woods
Beam Search, K = 7: a little boy in a red shirt is running through the woods
Beam Search, K = 10: there is a little boy in a striped shirt standing on the grass by a tree
```

6 Conclusion

In this project, we aimed to finish an image caption generator and use multiple techniques to enhance the efficiency and performance of the code.

Key optimization methods employed in the project included Cython, GPU acceleration, multithreading, vectorization, beam search and the use of generators. We use multithreading for tasks like data loading, preprocessing, and other CPU-bound tasks that are not directly related to the GPU-accelerated computations. The GPU is utilized by the deep learning framework for training, while the CPU workers will handle other tasks concurrently. By carefully integrating these methods into the whole process, we achieved significant improvements in performance. The total execution time for the entire code was reduced from 151 minutes to 69 minutes, representing a 54.3% decrease in duration. In the preprocessing stage, the runtime decreased from 50 minutes to 8 minutes, while in the training phase, the runtime was reduced from 101 minutes to 61 minutes. The reduction in time not only saves computational resources but also enables faster development and deployment of the image caption generator model.

The success of this optimization project has important implications for the future development of AI-based image captioning systems. By leveraging these optimization techniques, developers can efficiently create and improve image caption generator models that provide more accurate and context-sensitive textual descriptions for images. This in turn can drive further development in various fields such as social media, e-commerce and assistive technologies.

7 References

Ysthehurricane. “Image Caption Generator Tutorial.” *Kaggle*, Kaggle, 6 Sept. 2021, https://www.kaggle.com/code/ysthehurricane/image-caption-generator-tutorial.

Gautam, Tanishq. “Implementation of Attention Mechanism for Caption Generation on Transformers Using Tensorflow.” *Analytics Vidhya*, 20 Jan. 2021, https://www.analyticsvidhya.com/blog/2021/01/implementation-of-attention-mechanism-for-caption-generation-on-transformers-using-tensorflow/.

Ranjan, Atul. “Image Captioning with an End-to-End Transformer Network.” *Medium, Python in Plain English*, 13 June 2022, https://python.plainenglish.io/image-captioning-with-an-end-to-end-transformer-network-8f39e1438cd4.

Gautam, Tanishq. “Create Your Own Image Caption Generator Using Keras!” *Analytics Vidhya*, 4 Nov. 2020, https://www.analyticsvidhya.com/blog/2020/11/create-your-own-image-caption-generator-using-keras/.

Bhandari, Aniruddha. “Image Augmentation on the Fly Using Keras Imagedatagenerator!” *Analytics Vidhya*, 16 Aug. 2020, https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/.

“Tf.keras.preprocessing.image.imagedatagenerator .” *TensorFlow*, https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.