Online Code: Image Caption Generator using CNN and LSTM | Kaggle

This report outlines the modifications made to the code to improve its suitability and performance for my final project. The objective was to create a more robust and efficient image captioning model.The online project provides a foundational approach to image caption generation using a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) model. This model is implemented in Python and utilizes the Flickr8K dataset to recognize the context of an image and describe it in natural language.The report will detail key changes made to this base implementation to enhance its performance and scalability, incorporating concepts of High-Performance Computing (HPC) and parallel processing.

Base Model Analysis:

The online project serves as a fundamental Proof of Concept (PoC) for an image captioning model, demonstrating a successful, working implementation. The project's methodology involves extracting features from images using a restructured VGG16 model and then feeding these into an LSTM network, which is responsible for generating the captions. The process includes essential data preparation steps such as cleaning and preprocessing the captions by converting them to lowercase and adding 'startseq' and 'endseq' tags, as well as using a tokenizer to transform words into numerical sequences and padding to ensure uniform lengths. After training the model for 20 epochs with a batch size of 32, its performance is evaluated using the BLEU score. The model's results, with a BLEU-1 score of 0.526863 and a BLEU-2 score of 0.295591, are considered a good outcome, establishing a solid foundation for further modifications.

Modifications and changes made by me:

To evolve the project from a simple PoC to a more optimized and production-ready solution, several modifications were implemented, primarily focusing on improving performance and adding more comprehensive data analysis.

HPC and Parallelism:

The most significant changes were the integration of HPC and parallel computing concepts, which are absent in the initial online project. My project incorporated the following:

Mixed Precision: The policy was set to mixed_float16, which uses a combination of float16 and float32 data types. This can significantly speed up training on compatible hardware like GPUs.

XLA JIT Compilation: The tf.config.optimizer.set_jit(True) command was used to enable Just-In-Time compilation, optimizing the TensorFlow graph for better performance.

Distributed Training: The tf.distribute.MirroredStrategy() was implemented to leverage parallel computing, allowing the model to be trained efficiently across multiple GPUs.

Code and Data Pipeline Refinements

The data handling and code structure were improved to be more robust and efficient.

Data Generator: A tf.data.Dataset.from_generator was used to create a more efficient input pipeline for the model, which helps avoid session crashes and handles data processing in parallel. The batch size was also increased from 32 to 256 to take advantage of the distributed training setup.

Evaluation Script: A corrected, batched greedy decode evaluation function was implemented, which is more robust and suitable for a graph-compatible loop using tf.while_loop. This allowed for a more comprehensive evaluation, including BLEU-3 and BLEU-4 scores, which were not provided in the original project.

Reason for Changes:

The modifications and enhancements implemented in my project were driven by the need to transform a basic proof-of-concept into a more efficient, scalable, and analytically comprehensive solution. The primary reasons for these changes were to significantly reduce training time and resource consumption, which are major limitations of deep learning projects, especially when scaling up to larger datasets. By incorporating concepts like mixed precision and XLA JIT compilation, the project was optimized for high-performance computing (HPC), enabling faster training and inference. Furthermore, the introduction of a more robust data pipeline using tf.data.Dataset and a generator function addressed the issue of handling data in larger batches efficiently without causing system crashes.

The use of tf.distribute.MirroredStrategy was a crucial step towards parallel computing, allowing the model to leverage multiple GPUs for synchronous training, which is essential for working with extensive datasets in the future. The addition of detailed data analysis and visualizations was intended to provide a deeper, more professional understanding of the model's behavior and the dataset's characteristics, moving beyond a simple pass/fail evaluation based on a single metric.

Conclusion

While the online project successfully demonstrates the core principles of an image caption generator, the modifications in my project transform it into a more efficient and analytical solution. The integration of HPC features, parallel computing, and a refined data pipeline resulted in a more scalable model. The addition of comprehensive visualizations and detailed data analysis provides a deeper understanding of both the dataset and the model's behavior, making the final project more complete.