Exploring and Analyzing Compression Algorithms in Pre-Processing Pipelines

Adrian David Castro Tenemaya
adrian.castro[at]tum.de
Technische Universität München

October 4, 2021

Abstract

Deep learning models are leading to several challenges to resource optimization, as the amount of data, they require to be trained increases along with their complexity. Consequently, data-intensive steps such as preprocessing are being studied to enhance the performance of training pipelines. One of these studies [1] has reportedly achieved good performance by building an open-source profiling library that can automatically decide suitable preprocessing strategies based on a cost model, and the results it has achieved show the potential of pipeline tuning. Compressing data is a process that is being used for different systems to save storage and network bandwidth. This however leads to additional processing power overhead on the system that receives the compressed data, which has to run a potentially very slow decompression algorithm.

The preprocessing pipeline can be split into steps which are run *once*, called "offline" from this point onwards, and steps which are performed *every iteration*, called "online". The set of preprocessing steps depends on the dataset and the model input, but generally, any transformation is a step, like cropping an image or encoding a word. A preprocessing *strategy* is processing up to (and including) a step offline, and the remainder of the pipeline is executed online.

We choose to explore the type of data that was used to analyze the original study's preprocessing pipelines. In particular, we will start with images (Computer Vision problems). Selecting this approach allows us to avoid unnecessary additional work, as the datasets and parts of the software are already present and working.

Our starting approach is simple: at every step n, the output data will be compressed using an algorithm, and at the next step $n + 1$, the input data from the step n is decompressed. Our intuition is that we reduce the strain on the bandwidth and disk usage, at the cost of processing performance.

In the case a particular algorithm at a particular step of the preprocessing pipeline does not yield particular benefits to the overall performance of the system, then we can consider applying other algorithms. For example, it is very well possible that certain algorithms used at different steps of the pipeline could carry more performance benefits than others. Some can be more efficient when used against structured data.

Example of possible compression algorithms and formats that can be applied are the ones based of `LZ77`, which is one of the most widely used algorithms for data compression, although it is ill-advised to use them as one-size-fits-all solutions. Another very common compression algorithm is `gzip` for dealing with file data. It is often seen as one of the low-hanging fruits to pick when trying to reduce strain on bandwidth in HTTP communications. Furthermore, compression algorithms that target floating-point and integer data, which can have the most impact when dealing with multidimensional data, such as `SIMD-BP128` or `varint-G81U` [2] can be used, and other algorithms that can be applied according to the nature of the input data.

Finally, the guided research presented here aims to provide further insights into the above-mentioned profiling library by exploring the effects of data compression on storage and network usage. We will provide an in-depth analysis of the impact of compression algorithms applied to different data representations of preprocessing pipelines.

References

[1] Alexander Isenko. *Where is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines*. 2021.

[2] D. Lemire and L. Boytsov. "Decoding billions of integers per second through vectorization". In: *Software: Practice and Experience* 45.1 (May 2013), pp. 1–29. ISSN: 1097-024X. DOI: 10.1002/spe.2203. URL: http://dx.doi.org/10.1002/spe.2203.