Performance Optimization for Computer Vision and Natural Language Processing

Austin Xiong
High Performance Machine Learning
Columbia Univeristy
New York, USA
ax2156@columbia.edu

Capucine Leroux
High Performance Machine Learning
Columbia Univeristy
New York, USA
cpl2146@columbia.edu

***Abstract*—This paper describes the process of optimizing two deep learning models. The first one is a Convolutional Neural Network (CNN) model called Visual Geometry Group (VGG) and the second one is a Natural Language Processing (NLP) sentiment analysis model using Long Short-Term Memory (LSTM).**

I. Introduction

We profiled and optimized two deep learning models through various techniques introduced in E6998 - High Performance Machine Learning, taught by Professor Parijat Dube and Professor Kaoutar El Maghraoui.

II. Models and Data Description

A. CIFAR-10

For the CNN model, the data used for training was the CIFAR-10 [1] dataset, a set of 60,000, 32x32 color images with 10 classes.

B. VGG-11

The first model is a model called VGGNet [2] developed by the company Visual Geometry Group. This deep learning model uses a classical CNN architecture, utilizing small 3x3 kernels with the only other components being pooling and fully connected layers.

The specific VGGNet we used was VGG-11 which uses 11 layers, as opposed to deeper versions such as VGG-16 which uses 16 layers.

C. IMDB

For the LSTM model, the dataset [3] used for training recollects 50K movie reviews from the IMDB database for natural language processing or Text analytics. This dataset was made for binary sentiment classification, providing a set of 25,000 highly polar movie reviews for training and 25,000 for testing.

D. LSTM Model

The second model is an LSTM model for sentiment analysis, written by Affandy Fahrizain in 2022 [4]. It takes numerical embeddings of movies reviews as input, and returns whether the review is positive or negative.

III. Training and Profiling Methodology

A. Training

The training of VGG-11 on the CIFAR-10 dataset was done using a virtual machine from Google Cloud Platform (GCP). The VM instance was equipped with one Nvidia Tesla P100 GPU for training and inference.

The training of the LSTM model was done using Google Colab, with a V100 GPU and a High-RAM configuration.

B. Profiling Methodology

Both models were analyzed using PyTorch's [4] profiler. Both CPU and GPU times were analyzed during inference, and both the CPU's and GPU's memory consumption were analyzed during training.

IV. Performance Tuning Methodology

A. Data Loaders

The number of workers passed to PyTorch's Data Loader class specifies the number of subprocesses to be used for data loading for model use during training. Setting the number of workers to be zero, which is the default, causes data loading to be handled by the main process, which as one can expect will increase time spent during training since the main process needs to both train and load the data.

Thus, we know that more workers might decrease the training time, and so we ran tests to find the optimal number of workers.

B. Batch size

The batch size is the number of samples processed in a single forward/backward pass of a neural network during training. Larger batch sizes can result in faster computation time because more samples are processed in parallel, and the overhead of transferring data to and from the GPU or CPU is reduced. However, there is a limit to the speedup that can be achieved by increasing the batch size, and beyond a certain point, the benefits may diminish or even reverse.

We ran a few tests on the LSTM model and noticed a big influence of the batch size on the overall training time.

C. Quantization

As deep learning networks become larger and more complex, reducing precision via quantization can decrease memory consumption, which is becoming more important as these models are put on embedded systems that have less computing power.

We quantized the VGG-11 model from 32-bit precision (Float) to 8-bit precision (Int) via Pytorch's quantization API.

D. Pruning

The main bottleneck of the LSTM model is memory. Pruning weights can reduce the size of the model and thus the amount of memory required to store the model. As the pruned model is sparser, this can even reduce computation time.

We used a structured magnitude-based weight pruning of 90% on the LSTM model, evaluating magnitude with the L2 norm.

E. TorchScript

Torchscript traces a model over one inference iteration and converts it to a script, which can allow for faster inference.

Torchscript also gives the benefit of serializing a model for use in non-Python environments, making them easily portable to various production environments.

V. Experimental Results

A. VGG results

For the VGG-11 model, the best number of workers for dataloaders turned out to be 4. As confirmed by the memory consumption profiler, the time spent during data loading decreased drastically as the number of workers was increased from 0 to 4. On the CPU this brought training time from around 27 minutes to 26 minutes.

However, on the GPU, when the number of workers was increased from 0 to 4, the training time decreased from 127.16 seconds to 49.18 seconds. As we can see, training on the GPU greatly benefits from having an optimal number of workers.

As expected from quantizing a float-32 model to int-8, the model size decreased from 37 MB to 9 MB. The quantization also led to a 0.3% increase in test accuracy.

After TorchScripting the model, the inference speed on the CPU only saw an increase by 3%, however on the GPU the inference speed increased by 130%.

B. LSTM results

If increasing the number of workers had a positive impact on the VGG training time, it did not on the LSTM. In this case, loading movements slowed down the model consistently with the number of workers. However, increasing the batch size inferred a good speed up. The optimal parameters found were 0 number of workers and a batch size of 128. This parameter change induced a training speedup ratio of 1.3 on each epoch in average.

The pruning had both a positive impact on memory and time usage. We first tested a few different pruning ratios to test the sensitivity of each layer. We found that the pruning had little impact on the test accuracy, and decided to prune 90% of weights based on their L2-magnitude. The structured pruning divided the number of learnable parameters from 34,731,777 to 3,528,662, that is to say a compression ratio of 9.84. It even reduced the overall time by more than half, which can be explained by the fact that a sparser model takes less time to be trained and evaluated.

VI. Conclusion

In conclusion, our experiments showed us the power of quantization and pruning techniques to reduce model size and even training time. Increasing the number of workers of the dataloader can be a huge boost for speed as we saw with the VGG model, but it is not always the case, LTSM model being a counter-example. Other optimizations such as finding optimal batch size or using Torchscript also helped induce faster inference.

References

[1] A. Krizhevsky, 'Learning Multiple Layers of Features from Tiny Images', pp. 32–33, 2009.

[2] K. Simonyan and A. Zisserman, 'Very Deep Convolutional Networks for Large-Scale Image Recognition', arXiv [cs.CV]. 2015.

[3] N. Lakshmipathi, IMDB Dataset, https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews, 2019.

[4] A. Fahrizain, LSTM Model, https://www.kaggle.com/code/affand20/imdb-with-pytorch, 2022.

[5] A. Paszke et al., 'Automatic differentiation in PyTorch', 2017.