TensorFlow Runtime Optimisation Report

Introduction

Deep learning models are increasingly utilized for complex tasks, requiring efficient training pipelines and robust performance. This task sought to optimize the runtime of the model pipeline while stabilizing validation loss. Techniques such as mixed precision training, TensorFlow caching, and L2 regularization were employed to achieve these objectives. The methods were rigorously tested over **50 epochs** and with a **subset of classes** to accommodate limited computational resources, ensuring a focused evaluation of the optimizations. The optimized approach resulted in notable improvements in training efficiency and model generalization. It achieved **lower training loss** and **higher training accuracy** in fewer epochs compared to the original implementation, further emphasizing its efficiency. Additionally, alternative architectures, including EfficientNet V2 B0, were tested to explore their suitability. This report discusses the achieved results, the methodologies applied, and potential directions for further refinement.

Achievements of the Optimization

The optimized model delivered significant improvements in runtime, reducing training time from **5462.38 seconds to 5029.70 seconds**, a 7.9% improvement. This reduction was achieved through software optimizations without altering hardware configurations. By implementing mixed precision training, the model utilized 16-bit floating-point calculations for non-critical operations, reducing memory usage and computational overhead.

In addition to runtime improvements, the optimized model exhibited **lower training loss** and **higher training accuracy** within fewer epochs compared to the original implementation. By epoch 10, the optimized model achieved a training loss of **1.2064** and an accuracy of **69.40%**, whereas the original model required 16 epochs to reach comparable training performance. This faster convergence highlights the effectiveness of the applied optimizations in accelerating model learning.

The model architecture was also refined, reducing the total trainable parameters from **7,342,504 to 6,098,464**. Despite this reduction, the model retained its accuracy, highlighting the importance of architectural efficiency in achieving computational gains while mitigating overfitting risks.

Validation loss stability was also addressed using L2 regularization. Initially, mixed precision and TensorFlow caching introduced numerical instability, leading to increased validation loss. L2 regularization effectively mitigated this issue by penalizing large weights during training, ensuring smoother loss convergence and improved model generalization. The final validation accuracy of **90.09%** demonstrated a successful balance between efficiency and robust performance.

Testing and Analysis of EfficientNet V2 B0

EfficientNet V2 B0 was evaluated as a potential base model to improve runtime efficiency further. This architecture is theoretically designed for faster computations and efficient resource utilization, making it an appealing candidate. However, testing revealed an **increase in runtime** compared to the original optimized model. This unexpected result was attributed to the preprocessing requirements and the complexity of intermediate operations during the forward and backward passes. While the architecture maintained competitive accuracy, its slower runtime rendered it less suitable for this task's goals, where training efficiency was paramount.

The evaluation of EfficientNet V2 B0 highlighted the importance of aligning model selection with task objectives. Although theoretically advantageous, the practical implementation highlighted challenges that need consideration, such as compatibility with optimization techniques and overall computational demands.

Methodologies for Optimization

The optimized training pipeline incorporated multiple strategies to achieve efficiency and stability. Mixed precision training was a foundation of the improvements, enabling faster computations by reducing the precision of non-critical calculations. This approach minimized memory consumption and allowed for larger batch sizes without sacrificing accuracy.

TensorFlow caching complemented this by improving data-loading efficiency. By caching datasets in memory, the pipeline avoided frequent disk I/O operations, which are a common bottleneck in deep learning workflows. This integration ensured that data retrieval was both quick and consistent, further reducing runtime.

To address the challenges introduced by mixed precision and caching, L2 regularization was implemented in the dense layers. This strategy penalized large weights, reducing overfitting and stabilizing validation loss across epochs. These methodologies collectively balanced computational efficiency with robust model performance, providing a scalable solution for complex tasks.

Future Directions

Despite the success of the optimizations, further refinements can target validation loss reduction while retaining runtime efficiency. Advanced regularization techniques, such as DropConnect, could provide additional stabilization by randomly zeroing individual weights rather than entire neurons. Similarly, implementing adaptive L2 regularization could dynamically adjust penalties based on loss trends, offering more targeted regularization.

Exploring hybrid precision training is another promising avenue. This approach selectively applies 32-bit precision to critical layers while using 16-bit precision elsewhere, potentially addressing the numerical stability concerns observed in some layers without significantly impacting runtime.

Learning rate scheduling techniques, such as cosine annealing or warm restarts, could also enhance model convergence. These strategies dynamically adjust learning rates during training, preventing plateaus and oscillations in validation loss. Additionally, advanced data augmentation strategies could improve generalization by diversifying the training dataset while leveraging TensorFlow caching to manage augmented data efficiently.

Finally, automated hyperparameter tuning tools, such as Optuna or Keras Tuner, could systematically optimize model configurations, ensuring robust performance across a range of tasks. Improvements in dataset management, including hybrid caching strategies for memory-constrained environments, could also enhance scalability and adaptability.

Conclusion

This project successfully demonstrated how mixed precision training, TensorFlow caching, and L2 regularization can optimize runtime and stabilize validation loss in deep learning models. The improvements in training time, parameter efficiency, and validation stability showcased the effectiveness of these techniques in addressing computational challenges.

The analysis of EfficientNet V2 B0 highlighted the importance of practical testing when selecting architectures for optimization. While theoretically advantageous, its increased runtime made it unsuitable for this project's objectives. These findings underscore the need for aligning optimization techniques with project goals and computational constraints.

Future work should focus on further stabilizing validation loss and exploring adaptive precision and learning rate strategies. By building on the foundations established in this project, deep learning models can achieve greater efficiency, scalability, and robustness, ensuring their applicability across diverse real-world scenarios.