Empirical Analysis of Early Stopping and Learning Rate Scheduling for Deep Learning on Resource-Constrained Hardware

Karl Satchi Navida
navidake@national-u.edu.ph
College of Computing & Information Technology
National University
Manila, Philippines

Abstract

Efficient training of deep neural networks on resource-constrained consumer hardware presents unique challenges that differ fundamentally from datacenter environments. While most optimization research targets high-performance computing infrastructure, practitioners with limited resources face hardware constraints that can alter the effectiveness of traditional optimization strategies. This paper provides the first systematic empirical analysis of how training optimizations behave under consumer hardware constraints, specifically examining the interactions between early stopping criteria and learning rate adaptation strategies. We evaluate these techniques across three representative consumer platforms: CPU-only systems and entry-level GPUs, using CIFAR-10 as our benchmark task. Our findings shows that optimization strategies effective on high-end hardware do not necessarily maintain their relative performance under resource constraints, and we provide practical guidelines for practitioners operating under strict computational budgets. The study introduces resource-normalized performance metrics and establishes reproducible evaluation protocols for assessing training optimizations on consumer hardware.

CCS Concepts

• **Computing methodologies** → **Supervised learning**; **Neural networks**; *Machine learning algorithms.*

Keywords

Early Stopping, Learning Rate Scheduling, Resource-Constrained Training, Consumer Hardware, Deep Learning Optimization

1 Introduction

1.1 Background of the Study

The proliferation of deep learning applications has created a significant disparity between research conducted on high-performance computing infrastructure and the reality faced by practitioners with limited resources. While most optimization research targets datacenter environments with abundant computational resources, a substantial portion of the deep learning community operates on consumer-grade hardware with strict constraints on memory, processing power, and energy consumption.

Current optimization literature predominantly focuses on scaling to larger models and datasets, often overlooking the unique challenges present in resource-constrained environments. These challenges include thermal throttling, limited memory bandwidth, and the need to balance training time with energy consumption—factors that can fundamentally alter the effectiveness of traditional optimization strategies.

The assumption of unlimited computational resources pervades much of the optimization literature, leading to techniques that may not translate effectively to environments where every computational cycle and memory access carries a direct cost. This disconnect between research assumptions and practical constraints creates a gap that limits the applicability of state-of-the-art optimization techniques for a large portion of the deep learning community.

1.2 Statement of the Problem

This study addresses a key gap in understanding how training-time optimization strategies perform under the hardware constraints typical of low-cost consumer systems. Most prior research assumes high-end GPUs with ample memory, stable thermal performance, and unlimited runtime—conditions rarely met by students, independent researchers, or practitioners operating under tight resource budgets.

This work explores whether resource constraints introduce significant differences in how optimization strategies behave, particularly in terms of energy efficiency and convergence performance.

Specifically, we investigate the following research questions:

1.2.1 ***Primary Research Question.*** How do early stopping and learning rate scheduling strategies affect training efficiency and energy usage on low-cost consumer hardware?

1.2.2 ***Secondary Questions.***

(1) Which strategy—`EarlyStopping`, `ReduceLROnPlateau`, or both—yields better convergence and accuracy in constrained environments?
(2) How do resource-normalized metrics (e.g., accuracy-per-watt, convergence-per-cost) reflect training performance across platforms?
(3) What platform-specific behaviors (e.g., CPU/GPU load, power draw) influence the effectiveness of these strategies?
(4) What practical recommendations can be made for training deep learning models on entry-level CPUs and GPUs?

These questions address the broader challenge of translating optimization research from high-performance environments to the practical realities of training on everyday hardware.

1.3 Significance of the Study

This study contributes to a clearer understanding of deep learning training under resource-constrained environments, focusing

on practical and energy-aware optimization strategies. The key contributions are:

1.3.1 ***Empirical Characterization.*** We conduct a systematic evaluation of early stopping and learning rate scheduling techniques on two common consumer hardware configurations: CPU-only systems and entry-level discrete GPUs. Using the CIFAR-10 dataset [9] as a benchmark, we highlight training behavior and convergence patterns that differ significantly from those typically observed in high-end or cloud-based environments.

1.3.2 ***Resource-Aware Guidelines.*** Our findings support actionable recommendations for practitioners working within hardware-limited contexts, such as students, solo researchers, or developers using entry-level systems. These guidelines emphasize efficiency, cost-awareness, and platform-specific considerations.

1.3.3 ***Efficiency Metrics.*** We apply and validate resource-normalized metrics—such as accuracy-per-watt and convergence-per-peso (PHP)—to quantify the trade-offs between training performance, energy use, and cost. These metrics offer more relevant indicators of optimization effectiveness in real-world low-resource settings.

1.3.4 ***Transparent Evaluation Protocol.*** The study follows a reproducible and detailed protocol for evaluating training strategies, including environment logging (e.g., temperature, CPU/GPU power draw) and per-epoch metric collection. This framework can guide future experiments in the emerging subfield of sustainable and accessible deep learning.

1.3.5 ***Interaction Effects.*** This study highlights the critical need to investigate how optimization strategies interact under resource-constrained conditions. By empirically evaluating combinations (e.g., EarlyStopping with ReduceLROnPlateau), this work is significant for revealing that these strategies do not combine linearly, often leading to synergistic or diminishing returns. This understanding underscores the critical need for context-aware hyperparameter tuning in such environments, moving beyond isolated evaluations.

1.4 Scope and Limitations

This study focuses specifically on image classification tasks using convolutional neural networks (CNNs), trained on consumer hardware representative of typical academic and individual research settings. We examine two primary optimization strategies—early stopping and learning rate scheduling—chosen for their widespread applicability, implementation simplicity, and potential for significant impact under resource constraints.

The hardware platforms studied represent two common scenarios: CPU-only training on modern laptops and entry-level discrete GPU training. These platforms capture the majority of resource-constrained training situations while preserving experimental control and reproducibility. Although a free-tier cloud environment was initially included in the research plan, it was ultimately dropped due to excessive limitations encountered during experimentation. Specifically, the volatility of GPU runtime availability in the free tier forced us onto a CPU-only session—an approach that, if continued, would have exceeded the project's timeline.

Our analysis is limited to training-phase optimizations and does not address inference optimization, model architecture design, or distributed training scenarios. The findings may not generalize to other domains such as natural language processing or fundamentally different architectures such as transformers, though the methodology could be extended to these areas in future work.

The study also does not address data loading optimization, storage constraints, or network bandwidth limitations, focusing instead on computational and memory constraints during the training process itself.

2 Background and Related Work

2.1 Training Optimization Under Resource Constraints

The conventional approach to deep learning optimization assumes abundant computational resources, enabling extensive hyperparameter search and long training runs. However, resource-constrained environments introduce qualitatively different challenges that are often overlooked in mainstream optimization research.

Thermal Management and Performance Variability: Consumer hardware typically lacks the sophisticated cooling systems found in datacenter environments. Sustained high utilization can trigger thermal throttling, dynamically reducing clock speeds and computational throughput [8, 13]. This phenomenon can fundamentally alter training dynamics, making some optimization strategies counterproductive.

Unlike datacenter GPUs that maintain consistent performance under thermal management systems, consumer GPUs may experience performance degradation of 20-40% during extended training sessions [12]. This variability affects the reproducibility of optimization strategies and suggests that techniques designed for consistent hardware performance may require adaptation.

Memory Hierarchy Effects: Consumer GPUs often have limited VRAM and slower memory interfaces compared to professional accelerators. This constraint affects optimal batch sizes, gradient accumulation strategies, and the feasibility of certain optimization techniques that rely on maintaining large buffers or historical information [20].

The memory bandwidth limitations of consumer hardware can create bottlenecks that shift the optimal balance between computation and memory operations. Optimization strategies that assume high memory bandwidth may become counterproductive when memory access becomes the limiting factor.

Energy Constraints and Cost Considerations: Unlike datacenter environments where energy costs are amortized across many users, individual practitioners must consider energy consumption as a direct cost. This consideration changes the optimization objective from pure performance to efficiency-aware metrics that account for power consumption and training time trade-offs [16].

2.2 Early Stopping in Practice

Early stopping is a fundamental regularization technique that prevents overfitting by halting training when validation performance plateaus. While the theoretical foundations are well-established [14], practical implementation involves numerous design choices that can significantly impact effectiveness under resource constraints.

Validation-Based Stopping Criteria: Traditional early stopping relies on monitoring validation loss or accuracy, requiring practitioners to set aside validation data and computational resources for periodic evaluation [1]. In resource-constrained settings, both the data allocation and computational overhead of validation can represent significant costs.

Recent work by Mahsereci et al. [11] introduced validation-free early stopping using gradient statistics, which is particularly relevant for resource-constrained settings. However, their approach has not been evaluated on consumer hardware where computational constraints may affect the reliability of gradient-based stopping criteria.

Patience and Restoration Strategies: The choice of patience parameter (number of epochs to wait before stopping) and model restoration strategy (whether to restore the best weights) significantly influence both final model quality and resource utilization. Longer patience periods provide more opportunities for recovery from local minima but consume additional resources that may be critical under constraints [2].

Gap in Current Understanding: Most early stopping research evaluates stopping criteria in isolation, without considering their interaction with other optimization components or hardware constraints. The optimal patience parameter on a high-performance GPU may be suboptimal on consumer hardware where training dynamics are affected by thermal throttling and variable performance.

2.3 Learning Rate Scheduling Strategies

Learning rate adaptation has evolved from simple step decay to sophisticated adaptive methods, but the choice of scheduling strategy interacts complexly with hardware characteristics in ways that are poorly understood.

Cyclical Learning Rates: Smith's cyclical learning rates [15] demonstrated significant improvements on standard benchmarks, but these results were obtained on high-performance hardware with consistent computational throughput. Consumer hardware with variable performance due to thermal management may require different cycle lengths or amplitude choices.

The effectiveness of cyclical schedules depends on the assumption that training dynamics remain consistent across cycles. Hardware-induced performance variability may disrupt this assumption, suggesting the need for adaptive cycle parameters that account for actual rather than nominal computational capacity.

Adaptive Methods and Memory Overhead: Modern adaptive optimizers such as Adam [7] and RMSprop [18] maintain per-parameter learning rate information, increasing memory requirements substantially. On memory-constrained hardware, this overhead may offset the convergence benefits, suggesting different trade-offs than those observed in unconstrained environments.

The memory requirements of adaptive optimizers scale with model size, creating a tension between optimization effectiveness and resource utilization that is particularly acute on consumer hardware with limited VRAM.

Warm Restarts and Resource Budgets: Loshchilov and Hutter's warm restarts [10] showed promising results by periodically resetting the learning rate and allowing the optimization to escape local minima. However, the optimal restart schedule may depend critically on the available computational budget.

A restart schedule designed for unlimited training time may be suboptimal when training must complete within a fixed time window or energy budget, suggesting the need for budget-aware scheduling strategies.

2.4 Hardware-Aware Optimization

The computer systems community has produced substantial work on hardware-aware optimization, but this research has largely focused on inference rather than training, and on mobile/embedded devices rather than consumer desktop hardware.

Architectural Adaptations: Work on efficient architectures such as MobileNets [5], SqueezeNet [6], and EfficientNet [17] focuses on designing models that are inherently efficient. However, these approaches address model architecture rather than training procedures, and do not consider how training strategies should be adapted for the target hardware.

Compiler and System-Level Optimizations: Frameworks such as TVM [3] and TensorRT [19] optimize individual operations and kernel execution but do not address higher-level training strategies or their interaction with hardware constraints. These optimizations are orthogonal to our focus on algorithmic training strategies.

Energy-Aware Training: Recent work has begun to address energy consumption in training [4, 16], but primarily from an environmental perspective focusing on large-scale training rather than as a practical constraint for individual researchers with limited budgets.

2.5 Research Gap and Positioning

Despite extensive research on both training optimization and hardware-efficient deep learning, there exists a significant gap in understanding how these domains interact in practical resource-constrained scenarios.

Evaluation Methodology Gap: Most optimization research uses high-end hardware and focuses on final accuracy or convergence speed, ignoring resource utilization metrics that are critical for constrained environments. Standard benchmarking practices do not capture the trade-offs between optimization effectiveness and resource consumption.

Interaction Effects: Training optimizations are typically evaluated in isolation, but their effectiveness may be fundamentally altered by hardware constraints such as thermal throttling or memory limitations. The assumption of independent effects may not hold under resource constraints.

Practical Guidelines: Existing work provides either theoretical analysis or high-level architectural recommendations, but lacks practical guidance for practitioners with specific hardware constraints and budget limitations.

Reproducibility and Standardization: The lack of standardized evaluation protocols for resource-constrained training makes it difficult to compare results across studies or translate research findings to practical applications.

Our work addresses these gaps by providing a systematic empirical analysis of training optimization strategies specifically designed

for and evaluated on consumer hardware, with explicit consideration of resource constraints and practical applicability. This focus on the intersection of optimization effectiveness and resource efficiency represents a novel contribution to the field.

3 Methodology

3.1 Dataset and Preprocessing

We conducted experiments on the CIFAR-10 image classification dataset, which contains 50,000 training images and 10,000 test images (32×32 color) across 10 distinct classes [9]. The dataset was loaded using the `tensorflow.keras.datasets.cifar10` API. Pixel values were then normalized to the [0,1] range, and per-channel standardization was applied.

From the 50,000 original CIFAR-10 training images, 40,000 were allocated for training and the remaining 10,000 were designated as the validation set. The original 10,000 CIFAR-10 test images served as the final evaluation test set. A fixed random seed was applied across all run iterations to ensure the reproducibility of data partitioning and batching order during both training and validation phases.

To enhance model generalization, the training images were augmented through the `tensorflow.keras.preprocessing.image.ImageDataGenerator` API. The augmentation pipeline included rotations, width and height shifts, shearing, zooming, and horizontal flipping.

3.2 Model Architecture

The model employed in this study is a Residual Neural Network (ResNet), specifically a ResNet18 architecture adapted for CIFAR-10 classification. ResNet architectures are particularly well-suited for image classification tasks due to their ability to train very deep networks through residual connections that mitigate the vanishing gradient problem.

3.2.1 ***Detailed Architecture.*** The ResNet18 model consists of 11,173,962 total parameters, with 11,156,778 trainable parameters, as illustrated in Figure 1. The architecture is structured as follows:

Input Layer: The network accepts input images of shape (32, 32, 3), corresponding to CIFAR-10's RGB images with 32×32 pixel resolution.

Initial Convolutional Block: The first layer is a Conv2D layer with 32 filters of size 3×3, followed by BatchNormalization and ReLU activation. This layer maintains the spatial dimensions while extracting initial feature representations.

Residual Block Groups: The network contains four residual block groups with progressively increasing channel depths:

- **Group 1 (64 channels):** Contains two residual blocks (1a and 1b), each with Conv2D layers using 3×3 kernels, BatchNormalization, and ReLU activations. Skip connections enable direct gradient flow.
- **Group 2 (128 channels):** Begins with a downsampling residual block (2a) using stride-2 convolution to reduce spatial dimensions from 32×32 to 16×16, followed by a standard residual block (2b).
- **Group 3 (256 channels):** Similarly structured with a downsampling block (3a) reducing dimensions to 8×8, followed by block 3b.
- **Group 4 (512 channels):** Final residual group with downsampling block (4a) reducing to 4×4 spatial resolution, followed by block 4b.

Global Average Pooling: Reduces the spatial dimensions from $4 \times 4 \times 512$ to $1 \times 1 \times 512$ by computing the average across spatial dimensions, significantly reducing parameters compared to fully connected alternatives.

Classification Head: A Flatten layer prepares the features for the final Dense layer, followed by Dropout (p=0.5) for regularization. The output Dense layer contains 10 neurons corresponding to CIFAR-10's class labels, with no activation function applied (logits output).

3.2.2 ***Architecture Choice Justification.*** ResNet18 was selected for several compelling reasons. First, residual connections enable training of deeper networks without suffering from vanishing gradients, allowing the model to learn more complex feature representations. Second, the architecture's moderate depth (18 layers) provides an optimal balance between model capacity and computational efficiency for CIFAR-10's relatively small image size. Third, ResNet architectures have demonstrated consistent state-of-the-art performance on image classification benchmarks, making them a reliable baseline for comparative studies. Finally, the systematic downsampling through stride-2 convolutions efficiently reduces spatial dimensions while preserving important feature information, which is particularly important for the small 32×32 input images in CIFAR-10.

3.3 Experimental Setup

3.3.1 ***Hardware and Software Environment.*** **Hardware:** All experiments were conducted on a standard gaming laptop, Acer Nitro 5 with RTX 3050 Ti Laptop and 11th generation Intel i5, representing entry-level hardware for deep learning applications. The primary computational resource utilized was an RTX 3050 Ti Laptop GPU with 4GB VRAM capacity, accompanied by Intel i5-11400H CPU with 12 logical cores (6 physical cores with hyper-threading). Training was performed using the local workstation platform with different Miniconda environments configured for GPU and CPU-only experiments.

Software: The experimental environment was built using Python 3.9.21 with the following key dependencies to ensure reproducibility:

- TensorFlow: 2.10.1 (GPU)
- TensorFlow CPU: 2.10.1 (CPU)
- Scikit-learn: 1.6.1
- NumPy: 1.26.4
- CUDA: 11.2 (GPU)
- cuDNN: 8.1 (GPU)
- Dotenv: 0.9.9

3.3.2 ***Training Regimes and Strategies.*** Four distinct training strategies were implemented and compared to evaluate their effectiveness on model performance:

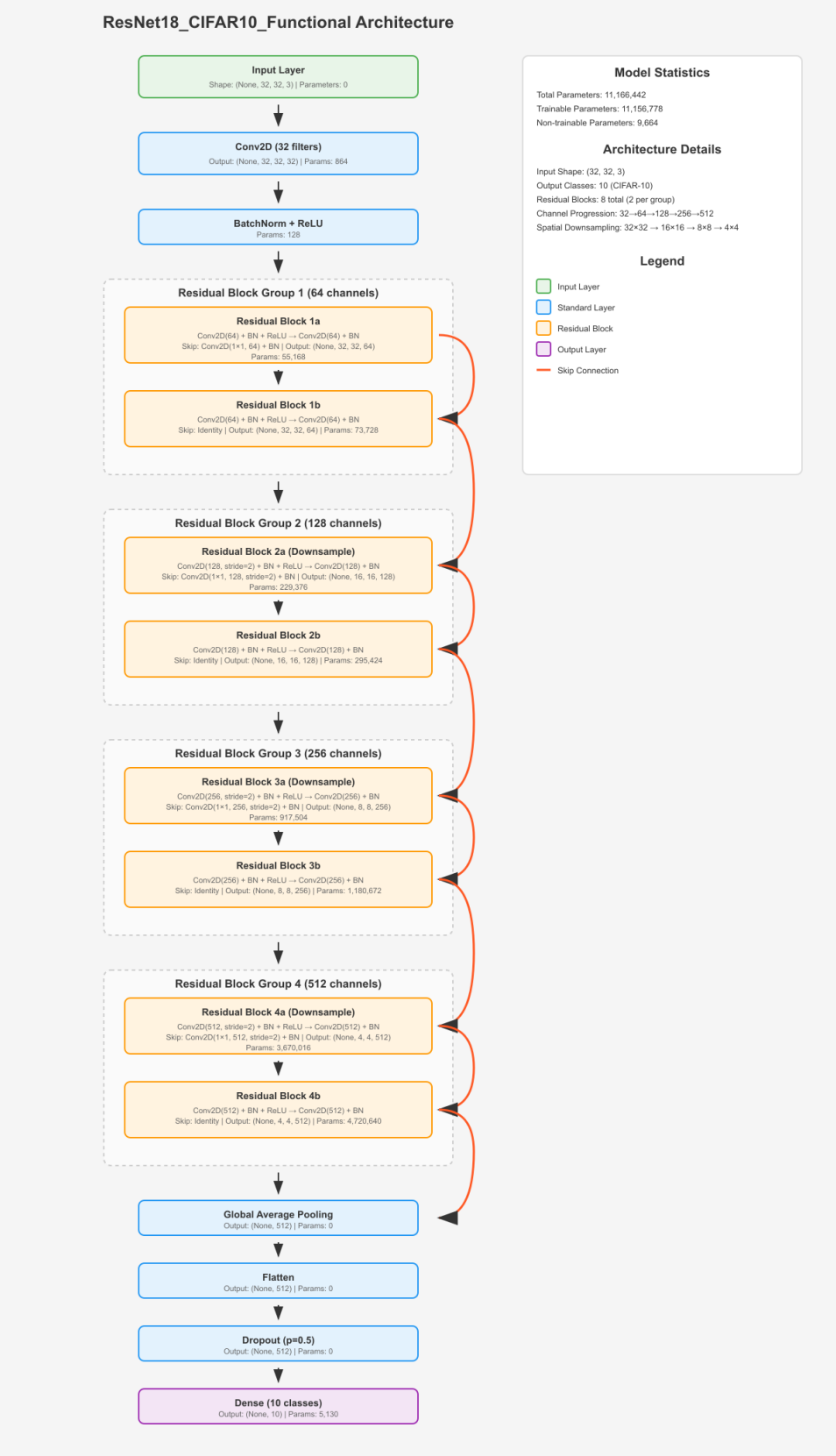

Figure 1: ResNet18 architecture used for CIFAR-10 experiments. The network consists of an initial convolutional layer followed by four residual block groups with 64, 128, 256, and 512 channels respectively. Each residual block includes Conv2D-BatchNorm-ReLU layers with skip connections. Spatial downsampling occurs through stride-2 convolutions. Global Average Pooling reduces spatial dimensions before the final Dense layer with 10 outputs for classification.

Baseline Training: The baseline approach employed standard training without any adaptive mechanisms. The model was trained for a fixed number of XXX epochs using a constant learning rate of XXX. No early stopping or learning rate scheduling was applied, providing a control condition for comparison.

Early Stopping Strategy: Early stopping was implemented to prevent overfitting by monitoring validation loss. The training process was configured with a patience of 10 epochs, meaning training would terminate if the validation loss failed to improve for 10 consecutive epochs. The callback was set to verbose mode (level 2) to provide detailed output during training, and the best model weights were automatically restored upon early termination.

Learning Rate Scheduling Strategy: `ReduceLROnPlateau` learning rate scheduling policy was used to adaptively reduce the learning rate when training plateaus. The configuration included:

- Monitor metric: validation loss
- Patience: 5 epochs
- Reduction factor: 0.5
- Minimum learning rate: 1e-6 (0.000001)
- Mode: 'min' (default for loss monitoring)
- Verbose: level 2 for detailed output

Combined Strategy: The integrated approach combined both the `EarlyStopping` and the learning rate scheduling mechanisms. Early stopping parameters remained consistent with the individual strategy (patience: 10 epochs, monitor: validation loss, mode: 'min', verbose: 2, restore best weights). The `ReduceLROnPlateau` scheduler operated with the same configuration as the individual LR scheduling strategy (patience: 5 epochs, factor: 0.5, min_lr: 1e-6, verbose: 2), with both mechanisms running concurrently during training. This combination aimed to leverage the benefits of both adaptive learning rate adjustment and overfitting prevention. The experimental design included four configurations: `EarlyStopping` only, `ReduceLROnPlateau` only, combined `EarlyStopping` + `ReduceLROnPlateau`, and baseline training without either mechanism. Due to time constraints, CPU experiments were limited to a single run per configuration, while GPU experiments conducted two runs per configuration for improved reliability.

3.3.3 ***Hyperparameters.*** All experiments utilized consistent hyperparameters to ensure fair comparison across training strategies:

- **Batch size:** 64
- **Optimizer:** Adam
- **Initial learning rate:** 1e-4 (0.0001)
- **Optimizer parameters:**
  - Learning rate: 1e-4 (0.0001)
  - Beta1 (Adam): 0.9 (default)
  - Beta2 (Adam): 0.999 (default)
  - Epsilon (Adam): 1e-7 (default)
  - Weight decay: 0 (default)
- **Maximum epochs:** 100
- **Loss function:** Sparse Categorical Crossentropy
- **Metrics:** `accuracy`
- **Data Augmentation parameters:**
  - Rescale: 1./255 (pixel normalization)
  - Rotation range: 20 degrees
  - Width shift range: 0.2 (20% of total width)
  - Height shift range: 0.2 (20% of total height)
  - Shear range: 0.2
  - Zoom range: 0.2 (20% zoom in/out)
  - Horizontal flip: True
  - Seed: [your seed value]
- **Validation split:** 20% of training data
- **Random seed:** 42 (for reproducibility; seed increases by 1 every same configuration run)

Each training strategy was executed twice for GPU and ones for CPU, with different a fixed seed to account for training variability, and results were averaged across runs to provide robust performance estimates.

3.4 Data Collection and Performance Metrics

(1) Profiling and Monitoring: Explain how you collected data on training performance and resource utilization.
   (a) Wall-Clock Time: How is it measured (e.g., start-stop timers, epoch-by-epoch).
   (b) Resource Utilization: Detail the tools used (e.g., `psutil` for CPU/RAM, `nvidia-smi` for GPU, TensorFlow Profiler/TensorBoard). Specify the metrics collected (e.g., CPU %, RAM usage, GPU utilization %, VRAM usage, power draw).

(2) Evaluation Metrics: Define all metrics used to evaluate the models and experiments:
   (a) Classification Performance: Accuracy, F1-score, precision, recall (if applicable), generalization gap, epochs to reach target accuracy.
   (b) Training Efficiency: Wall-clock time, time per epoch, throughput (e.g., images/second).
   (c) Resource Utilization: Peak CPU/GPU/VRAM, average CPU/GPU power consumption.
   (d) Convergence Diagnostics: Loss curves, learning rate trace, stability metrics (e.g., validation accuracy variance).
   (e) Composite Scores: Clearly define any custom metrics like "Speed-Accuracy Index" and "Resource Efficiency" with their formulas.

3.5 Experimental Protocol

(1) Number of Runs: State how many times each experiment (each training regime on each hardware setup) will be repeated to ensure statistical robustness, and how results will be aggregated (e.g., average, standard deviation).

(2) Reproducibility of Experiments: Reiterate steps taken to ensure that your experiments can be replicated by others (e.g., fixed random seeds for weight initialization, consistent data loading).

(3) Ethical Considerations (if applicable): While less common for this type of study, if there are any ethical considerations related to data privacy, energy consumption implications, etc., they would go here.

References

[1] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. *Neural networks: Tricks of the trade* (2012), 437–478. doi:10.1007/978-3-642-35289-8_26

[2] Rich Caruana, Steve Lawrence, and C Lee Giles. 2001. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems* 13 (2001).

[3] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*. 578–594.

[4] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research* 21, 248 (2020), 1–43.

[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[6] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. In *arXiv preprint arXiv:1602.07360*.

[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] Jonathan Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. 2011. Web extra appendix: implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing* 33, 3 (2011), 46–54.

[9] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report. University of Toronto.

[10] Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).

[11] Maren Mahsereci, Lukas Balles, Christoph Lassner, and Philipp Hennig. 2017. Early stopping without a validation set. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2238–2247.

[12] Sparsh Mittal. 2014. A survey of techniques for improving energy efficiency in embedded computing systems. *Journal of Systems and Software* 95 (2014), 208–237. doi:10.1016/j.jss.2014.05.021

[13] Massoud Pedram and Inkwon Hwang. 2012. Energy-efficient datacenters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 10 (2012), 1465–1484. doi:10.1109/TCAD.2012.2212898

[14] Lutz Prechelt. 1998. Early stopping-but when? *Neural Networks: Tricks of the trade* (1998), 55–69. doi:10.1007/978-3-642-35289-8_5

[15] Leslie N Smith. 2017. Cyclical learning rates for training neural networks. *2017 IEEE winter conference on applications of computer vision (WACV)* (2017), 464–472. doi:10.1109/WACV.2017.58

[16] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019), 3645–3650. doi:10.18653/v1/P19-1355

[17] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[18] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012), 26–31.

[19] Han Vanholder. 2016. Efficient inference with tensorrt. In *GPU Technology Conference*, Vol. 1. 2.

[20] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*.