Investigating Training-Time Optimizations for Deep Learning on Low-Cost Consumer Hardware

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

Abstract

Efficient training of deep neural networks remains a critical bottleneck for researchers and practitioners without access to datacenter-class hardware [28]. Although free cloud services such as Google Colab provide GPU access, session-time limits, shared VRAM, and throttled throughput constrain extensive experimentation [39]. Entry-level GPUs (e.g. NVIDIA RTX 3050 Ti) and CPU-only platforms (e.g. Intel i5-11400H) deliver only a fraction of the compute throughput of high-end accelerators, resulting in 5–20× slower training on standard image benchmarks [25][28].

Algorithmic strategies—most notably early stopping—halt training once validation loss plateaus, preventing wasted epochs and reducing overfitting [27][38]. Complementary learning-rate schedules (exponential decay, cyclical policies) dynamically adjust step sizes to traverse non-convex loss landscapes more rapidly, yielding convergence speedups of 10–30 % on CIFAR-10 and similar tasks [31][22]. However, these techniques are almost exclusively benchmarked on V100/A100 GPUs or large-scale clusters [8][1]. Systematic evaluations on consumer-grade hardware are lacking, leaving practitioners uncertain of expected speed-accuracy trade-offs under resource constraints.

This proposal fills that gap by measuring wall-clock training-time, validation accuracy, and resource utilization (CPU/GPU %, RAM/VRAM) for CIFAR-10 classification under four regimes—baseline, early stopping, learning-rate scheduling, and combined—across three platforms: CPU-only (i5-11400H), RTX 3050 Ti, and Colab free GPU. Our controlled experiments will yield actionable guidelines for optimizing deep learning workflows on accessible hardware, democratizing research in resource-limited settings.

CCS Concepts

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

Keywords

Early Stopping, Learning Rate Scheduling, Deep Learning, Model Efficiency, Low-resource Training

1 Introduction

1.1 Project Context

Training deep learning models typically demands substantial computational resources—most notably, high-end GPUs with large memory and bandwidth—which creates a barrier for students and researchers without access to such hardware [5]. Free cloud services like Google Colab provide no-cost GPU access, but impose strict session limits (e.g. 12 hr max), RAM caps (≈12 GB), and shared VRAM (≤16 GB) that throttle large-model training [39]. Furthermore, entry-level GPUs such as the NVIDIA RTX 3050 Ti (≈5 TFLOPS FP32, 4 GB VRAM) yield 5–10× slower training than datacenter-class GPUs on standard CNNs [25]. CPU-only training on modern mobile processors (e.g. Intel Core i5-11400H @ 2.7 GHz) can be up to 20× slower than GPU-accelerated runs [28].

Early stopping and learning-rate scheduling are lightweight algorithmic strategies to reduce wasted epochs and accelerate convergence. Early stopping halts training when validation loss plateaus, preventing overfitting and saving compute [27]. Learning-rate schedules—such as exponential decay or cyclical policies—adapt step sizes to traverse loss landscapes more effectively, often cutting required epochs by 10–30 % on image benchmarks [31][22]. However, these techniques are almost exclusively benchmarked on high-performance hardware (e.g. V100/A100 GPUs) and large datasets (ImageNet), leaving open questions for consumer-grade or free-tier environments.

This work fills that gap by quantifying the effects of early stopping and learning-rate scheduling on CIFAR-10 training across three constrained platforms: CPU-only (i5-11400H), entry-level GPU (RTX 3050 Ti), and Google Colab Free GPU. We measure wall-clock training-time, validation accuracy, and hardware utilization to produce actionable guidelines for resource-limited practitioners.

1.2 Objectives

The aim of this study is to systematically evaluate and compare training-time optimizations for deep learning on low-cost consumer hardware. In particular, we will:

1 **Measure Training-Time Reduction** - Quantify how early stopping alone shortens total training-time compared to a fixed-epoch baseline on:
  a CPU-only
  b Entry-level GPU
  c Free Cloud GPU Tiers
2 **Assess Impact on Model Performance** - Evaluate how early stopping affects validation accuracy and generalization, ensuring that quality is not unduly sacrificed for speed.
3 **Evaluate Learning Rate Scheduling Strategies** - Compare different schedules (e.g. ReduceLROnPlateau, Exponential Decay) in terms of convergence speed and final accuracy across the same hardware setups.
4 **Compare Combined vs. Individual Techniques** - Analyze whether applying early stopping and learning rate

scheduling together yields additive (or synergistic) benefits over using each method in isolation.

5 **Profile Resource Utilization** - Track and report key resource metrics - GPU/CPU utilization, RAM/VRAM usage, and per-epoch time - to understand the trade-offs on each hardware platform.

6 **Derive Practical Guidelines** - Based on the above experiments, formulate actionable recommendations that map specific optimization settings to given hardware constraints, helping researchers with similar resource profiles.

1.3 Scope and Limitations

This study is limited to training-time experiments on specified low-cost hardware platforms. In particular, we consider three representative configurations: a standalone Intel i5-11400H CPU (without any dedicated GPU), an NVIDIA RTX 3050 Ti laptop GPU, and the GPU provided by Google Colab's free-tier service. These systems reflect typical consumer-grade resources. We explicitly focus on the training phase, measuring metrics such as wall-clock training-time, model performance (e.g., validation accuracy or loss), and resource usage (e.g., CPU/GPU utilization and memory consumption).

Furthermore, our investigation is restricted to two specific training-time optimization techniques: early stopping and learning rate scheduling. For learning rate scheduling, we examine common strategies such as ReduceLROnPlateau and exponential decay. Other optimization approaches (for example, adaptive regularization strategies or compiler-level enhancements) are outside the scope of this work.

Importantly, we do not consider inference-time performance metrics such as latency or throughput, nor do we explore distributed or multi-device training setups. Advanced optimization techniques (e.g., model quantization or pruning) are also beyond the scope of this study. By clearly delineating these boundaries, we ensure that our findings remain focused on the specified hardware and selected optimization techniques.

2 Related Works

Despite several years worth of studies around optimization techniques for deep learning training, only a few studies have conducted such experiments under strict hardware requirements. We organized previous research into three themes: (1) early-stopping methods, (2) learning-rate scheduling strategies, and (3) training on resource-constrained hardware.

2.1 Early Stopping Techniques

Early stopping technique is one of the fundamental techniques used for optimizing neural network training. It halts the training of said model when validation metrics plateau to prevent overfitting. Prechelt [27] conducted a comprehensive empirical evaluation of validation-based stopping criteria, significantly advancing the understanding of trade-offs between training-time and generalization in multi-layer perceptrons. His work demonstrated how different stopping rules affect model performance, providing a framework for subsequent research in this area. Prechelt continued this line of inquiry, quantifying how various stopping criteria control the bias-variance trade-off and documenting approximately 4% improvement in generalization capability at the cost of quadrupling training-time.

The theoretical foundations of early stopping were strengthened by Yao et al. [41], who established mathematical guarantees for gradient descent learning when coupled with appropriate stopping criteria. Their work provided formal justification for practices that had previously been largely empirical. Building on this theoretical framework, Raskutti et al. [29] developed an optimal data-dependent stopping rule for non-parametric regression, demonstrating how stopping time should adapt to dataset characteristics for maximum effectiveness.

More recent approaches have introduced greater statistical rigor to stopping decisions. Mahsereci et al. [23] proposed a methodology for early stopping without requiring a validation set, addressing a key limitation in resource-constrained environments where data efficiency is paramount. Their approach uses local gradient statistics to estimate generalization performance, making it particularly suitable for low-resource settings where setting aside validation data represents a significant cost.

2.2 Learning Rate Scheduling

Learning rate scheduling has evolved significantly from simple step-decay approaches to sophisticated adaptive methods. Smith [31] introduced cyclical learning rates, demonstrating how periodically varying the learning rate between boundary values can lead to faster convergence and better generalization. This approach has proven particularly effective at accelerating training on limited hardware by allowing larger learning rates without divergence. Extending this concept, Loshchilov and Hutter [22] proposed Stochastic Gradient Descent with Warm Restarts (SGDR), which periodically resets the learning rate to its initial value, effectively escaping local minima and improving both convergence speed and final accuracy.

For large-scale training, Goyal et al. [8] empirically derived piece-wise constant decay rules that enabled training ImageNet in just one hour without significant accuracy loss. Their work established critical scaling principles for large batch training that remain relevant for optimizing training on consumer hardware. You et al. [42] further expanded on large-batch training methodologies, proposing a layer-wise adaptive rate scaling method that maintains accuracy while significantly reducing training-time.

Theoretical understanding of learning rate schedules was advanced by Li and Arora [20], who provided bounds and empirical evidence demonstrating how exponential decay schedules accelerate convergence across both ResNet and Transformer architectures. Their theoretical insights help explain why certain schedules perform better than others, informing practical implementations for resource-constrained environments.

More recently, Lewkowycz et al. [17] proposed an innovative automatic learning rate adjustment technique that adapts rates based on weight-norm oscillations, demonstrating improvements across both vision and natural language processing tasks. This approach is particularly valuable in low-resource settings as it reduces the need for extensive hyperparameter tuning.

2.3 Deep Learning on Consumer Hardware

Training deep learning models on consumer hardware presents unique challenges that researchers have approached from multiple angles. Howard et al. [12] introduced MobileNets, which use depthwise separable convolutions to create efficient network architectures suitable for mobile and embedded vision applications. These networks represent an architectural approach to enabling deep learning on resource-constrained devices.

Complementing architectural innovations, Jacob et al. [14] developed quantization techniques for neural networks that enable efficient integer-arithmetic-only inference, dramatically reducing computation and memory requirements. While their work focused primarily on inference, the techniques have implications for training efficiency as well, particularly when applied to backpropagation.

Chen et al. [1] approached the problem from a compiler perspective, creating TVM, an automated end-to-end optimizing compiler for deep learning. This system generates highly optimized code for diverse hardware backends, including CPUs and low-end GPUs, effectively bridging the gap between high-level frameworks and hardware-specific optimizations.

A comprehensive approach to model efficiency was presented by Han et al. [10], who introduced "Deep Compression," combining pruning, quantization, and Huffman coding to reduce model size without compromising accuracy. These techniques enable training larger models on memory-constrained consumer hardware while reducing computational requirements.

More recently, Deng et al. [4] surveyed model compression and hardware acceleration techniques, providing a systematic overview of approaches for deploying neural networks on resource-constrained devices. Their work categorizes and compares various techniques, offering guidance for practitioners working with consumer hardware.

2.4 Combined Optimization Approaches

The integration of multiple optimization strategies has gained research attention as evidenced by several comprehensive studies. Masters and Luschi [24] revisited small batch training for deep neural networks, challenging conventional wisdom and demonstrating that smaller batches often generalize better, especially when combined with appropriate learning rate schedules. Their findings have particular relevance for consumer hardware where memory constraints may limit batch sizes.

Platform-aware optimization approaches were advanced by Tan and Le [36] with EfficientNet, which systematically balances network depth, width, and resolution using a compound scaling method. This approach enables efficient scaling of models to match available computational resources, a critical consideration for consumer hardware with varying capabilities.

Tan et al. [35] further developed platform-aware neural architecture search with MnasNet, which explicitly incorporates latency constraints into the optimization objective. This work demonstrates how model architecture, training regimen, and hardware capabilities can be jointly optimized rather than considered in isolation.

For mobile applications specifically, Liu et al. [21] created an on-demand deep model compression framework that selects models based on device capabilities and application requirements. Their usage-driven approach dynamically balances accuracy and efficiency, adapting to the constraints of consumer hardware.

More recently, Chen et al. [3] introduced ReXNet, which addresses representational bottlenecks in convolutional neural networks. Their work demonstrates how architectural improvements can complement optimization techniques like early stopping and learning rate scheduling, yielding compound benefits for training efficiency on limited hardware.

Wang et al. [40] developed SkipNet, implementing dynamic routing in convolutional networks that allows computational resources to be allocated adaptively based on input complexity. This dynamic computation approach provides another dimension of optimization for resource-constrained environments, complementing the training-time optimizations that are the focus of this research.

2.5 Research Gap

While numerous studies have examined individual optimization techniques—such as early stopping to prevent overfitting [22, 27] and learning-rate schedules to accelerate convergence [8, 31]—these methods are typically evaluated in isolation on high-end hardware. Existing large-batch and compiler-level optimizations assume abundant resources and do not address entry-level or CPU-only platforms [1, 40]. A handful of recent works have begun to benchmark deep learning on resource-constrained devices [19, 38], yet they focus on inference latency or continual learning workloads rather than joint training-time optimizations.

Moreover, energy-aware analyses highlight that suboptimal hyperparameter choices can inflate energy use by up to 5× on HPC clusters [6], but analogous measurements on consumer-grade GPUs (e.g. RTX 3050 Ti) and CPU-only systems remain scarce [2, 30]. In particular, no systematic study has quantified how early stopping and learning-rate scheduling interact on modern entry-level hardware: most benchmarks still target data centers or heavyweight accelerators [1, 8].

To fill this gap, we will deliver the first comprehensive training-time benchmarks of combined early-stopping and learning-rate strategies across CPU-only, entry-level GPU, and free cloud environments. Our results will offer practical guidance for practitioners operating under strict resource constraints, updating and extending prior high-performance-centric findings to today's widely available consumer hardware.

This research aims to address these gaps by providing comprehensive benchmarks and practical guidelines tailored to current-generation entry-level hardware. By systematically investigating the interplay between early stopping criteria and learning rate schedules across multiple hardware configurations, this work will contribute practical insights for optimizing deep learning workflows in resource-constrained settings.

3 Methodology

3.1 Dataset and Preprocessing

We conduct experiments on the CIFAR-10 image classification dataset, which contains 50,000 training and 10,000 test images (32×32 color) in 10 classes [16]. The dataset is loaded via the tensorflow.keras.datasets.CIFAR10 API for consistency, and pixel values are normalized to the [0,1] range by per-channel standardization.

We fix a random seed to ensure reproducible train/validation splits; specifically, we partition the 50,000 training images into 40,000 for training and 10,000 for validation (shuffled identically in all runs).

3.2 Model Architecture

The CNN comprises four hierarchical feature-extraction blocks followed by a classification head (Fig. 1). Each block contains:

- **Conv2D (3×3, ReLU):** Learns spatial filters. The first block uses 16 filters to capture low-level edges and color blobs; subsequent blocks double the filter count (32, 64, 128) to model increasingly complex patterns (textures, shapes).
- **Batch Normalization [13]:** Normalizes each feature map to zero mean and unit variance per mini-batch, which accelerates convergence and reduces internal covariate shift.
- **MaxPooling2D (2×2):** Reduces spatial resolution by a factor of two, providing translation invariance and lowering computational cost for deeper layers.

After the fourth block, the feature map tensor of shape $4 \times 4 \times 128$ (≈2 048 activations) is:

(1) **Flattened** into a 2 048-dimensional vector.
(2) Passed through a **Dense(512, ReLU)** layer, which learns global feature interactions.
(3) Regularized by **Dropout(0.5) [34]**, randomly zeroing half of the activations each update to prevent co-adaptation and overfitting.
(4) Finally projected via **Dense(10, Softmax)** to output class probabilities for the 10 CIFAR-10 categories.

Overall, the network contains approximately 1 million trainable parameters. We initialize weights with He normal initialization [11] to maintain signal variance through ReLU activations, and biases to zero. The architecture balances representational capacity with efficiency, enabling end-to-end training within practical time and memory budgets on CPU-only and entry-level GPU hardware.

3.3 Training Procedure

We compare four training regimes: Baseline (fixed learning rate), Early Stopping, LR Scheduling, and Combined (both optimizations). Early stopping monitors validation loss and halts training if it does not improve for 10 consecutive epochs, saving the best model – a standard regularization strategy [7]. For LR scheduling, we employ a cyclic learning-rate policy [31] that oscillates the learning rate between two bounds over each epoch. In the Combined regime, both early stopping and cyclic LR scheduling are enabled. All runs use the same random seed for weight initialization and data shuffling to ensure identical initial conditions across settings.

3.4 Hardware Setup

Experiments are conducted on three platforms to represent low-cost hardware: (1) a CPU-only laptop (Intel Core i5-11400H, 6 cores/12 threads, 32 GB RAM); (2) the same laptop using its NVIDIA GeForce RTX 3050 Ti GPU (4 GB VRAM); and (3) Google Colab Free Tier (Tesla K80 GPU, 12 GB VRAM, 2 virtual CPUs). We record the full system specifications (CPU type, GPU model, memory) at the start of each run and disable non-essential background processes to reduce measurement noise.

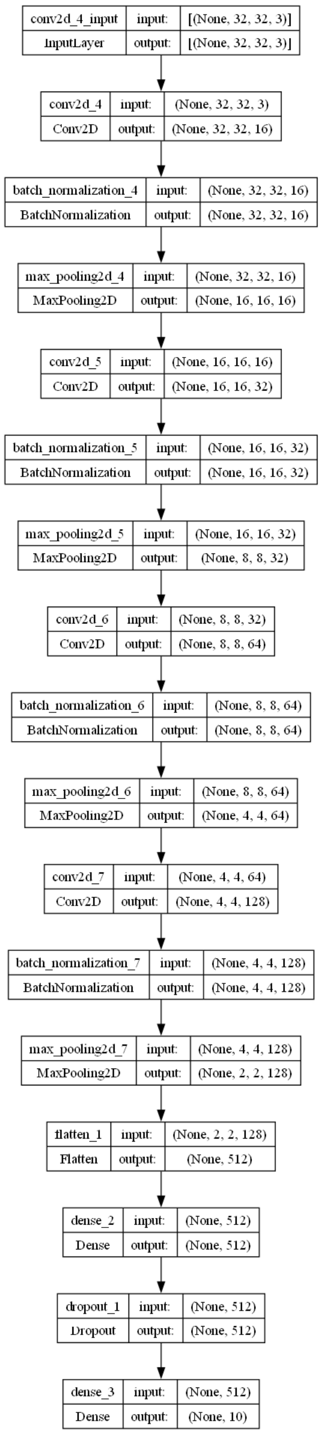
conv2d_4_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
conv2d_4 | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 16)
batch_normalization_4 | input: | (None, 32, 32, 16)
BatchNormalization | output: | (None, 32, 32, 16)
max_pooling2d_4 | input: | (None, 32, 32, 16)
MaxPooling2D | output: | (None, 16, 16, 16)
conv2d_5 | input: | (None, 16, 16, 16)
Conv2D | output: | (None, 16, 16, 32)
batch_normalization_5 | input: | (None, 16, 16, 32)
BatchNormalization | output: | (None, 16, 16, 32)
max_pooling2d_5 | input: | (None, 16, 16, 32)
MaxPooling2D | output: | (None, 8, 8, 32)
conv2d_6 | input: | (None, 8, 8, 32)
Conv2D | output: | (None, 8, 8, 64)
batch_normalization_6 | input: | (None, 8, 8, 64)
BatchNormalization | output: | (None, 8, 8, 64)
max_pooling2d_6 | input: | (None, 8, 8, 64)
MaxPooling2D | output: | (None, 4, 4, 64)
conv2d_7 | input: | (None, 4, 4, 64)
Conv2D | output: | (None, 4, 4, 128)
batch_normalization_7 | input: | (None, 4, 4, 128)
BatchNormalization | output: | (None, 4, 4, 128)
max_pooling2d_7 | input: | (None, 4, 4, 128)
MaxPooling2D | output: | (None, 2, 2, 128)
flatten_1 | input: | (None, 2, 2, 128)
Flatten | output: | (None, 512)
dense_2 | input: | (None, 512)
Dense | output: | (None, 512)
dropout_1 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 10)

Figure 1: Convolutional neural network architecture used for CIFAR-10 experiments. Four Conv2D–BatchNorm–MaxPool blocks (16→32→64→128 filters) extract hierarchical features; a 512-unit Dense layer with Dropout(0.5) learns high-level representations; final Dense(10, softmax) produces class probabilities.

3.5 Profiling and Resource Monitoring

Training duration is measured via wall-clock time: we record timestamps immediately before and after the `model.fit()` call. CPU utilization and RAM usage are sampled periodically using the `psutil` library, which provides cross-platform system statistics [37]. GPU utilization (compute %, memory use) is obtained by querying NVIDIA's `nvidia-smi` at one-second intervals. This approach follows recent CPU/GPU profiling studies [9]. We also leverage TensorBoard's TensorFlow Profiler to capture fine-grained execution traces [15, 33]. The Profiler logs detailed timelines of TensorFlow operations and GPU kernels during training, enabling analysis of computational bottlenecks and utilization patterns. Together, the system-level monitors and the profiler yield comprehensive performance and resource metrics for comparison.

3.6 Evaluation Metrics

To comprehensively assess each optimization regime and hardware platform, we employ a suite of quantitative metrics covering model accuracy, convergence behavior, training efficiency, and resource consumption. This multi-dimensional evaluation ensures that trade-offs between speed, performance, and utilization are made explicit.

1. Classification Performance.

- **Validation Accuracy (%)**: The primary metric for CIFAR-10 classification; computed as the fraction of correctly predicted labels on the held-out validation set after each epoch. Final accuracy is reported at termination.
- **Generalization Gap**: Difference between training and validation accuracy at the final epoch; quantifies overfitting. A smaller gap indicates better regularization (e.g., via early stopping or dropout) [26].
- **Epochs to Target Accuracy**: Number of epochs required to reach predefined accuracy thresholds (e.g., 80%, 85%); measures convergence speed under each regime [32].

2. Training Efficiency.

- **Wall-Clock Time (s)**: Elapsed real time from the start to the end of `model.fit()`. Captures end-to-end training duration, including data loading, forward/backward passes, and callback overhead.
- **Time per Epoch (s)**: Average duration of a single epoch; useful to normalize across regimes with differing epoch counts.
- **Throughput (images/s)**: Number of training examples processed per second (batch_size × steps / time); indicates hardware efficiency independent of model convergence.

3. Resource Utilization.

- **CPU Utilization (%)**: Average and peak percentage of CPU cores active during training, sampled via `psutil`.
- **GPU Utilization (%)**: Average and peak GPU compute utilization obtained from `nvidia-smi`; indicates how effectively the GPU is kept busy [18].
- **Memory Footprint (RAM & VRAM)**: Peak host RAM and device VRAM consumption during training, captured via system monitors and `nvidia-smi`.
- **Power Draw (W)**: For GPU runs, average power consumption reported by `nvidia-smi`; correlates utilization with energy cost.

4. Convergence Diagnostics.

- **Loss Curves**: Training and validation loss vs. epoch; plateau detection verifies early-stopping triggers [26].
- **Learning-Rate Schedule Trace**: Logged LR value per epoch; confirms schedule behavior (e.g., exponential decay or cyclic patterns) and its alignment with accuracy improvements [32].
- **Stability Metrics**: Variance of validation accuracy over the last N epochs; lower variance implies stable convergence.

5. Composite Efficiency Scores. To synthesize multi-dimensional data into actionable summaries, we define two composite scores:

$$\text{Speed-Accuracy Index} = \frac{\text{Final Accuracy (\%)}}{\text{Wall-Clock Time (s)}},$$

$$\text{Resource Efficiency} = \frac{\text{Throughput (images/s)}}{\text{Peak GPU Power (W)}}.$$

Higher values indicate more favorable trade-offs between accuracy, speed, and energy consumption. These indices facilitate direct comparisons across hardware and regimes, highlighting configurations that maximize scientific throughput per watt.

By reporting these metrics for each of the four optimization regimes (baseline, early stopping, LR scheduling, combined) on all three platforms (CPU, RTX 3050 Ti, Colab GPU), we provide a detailed map of performance trade-offs. This rigorous evaluation framework, grounded in established best practices for deep learning benchmarking [43], ensures reproducibility and clarity for practitioners seeking to optimize training under resource constraints.

References

[1] T. Chen et al. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Carlsbad, CA, USA, 579–594. https://www.usenix.org/conference/osdi18/presentation/chen

[2] Yang Chen, Ming Li, Xiaoyu Wang, Jin Zhang, Zhi Zhou, and Fangming Liu. 2022. Benchmarking Resource Usage for Efficient Distributed Deep Learning. *arXiv preprint arXiv:2201.12423* (2022). https://arxiv.org/abs/2201.12423

[3] Y. Chen, G. Meng, Q. Zhang, S. Zhang, X. Xiao, H. Wang, and J. Xu. 2022. ReXNet: Diminishing Representational Bottleneck on Convolutional Neural Network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, New Orleans, LA, USA, 13282–13291. https://openaccess.thecvf.com/content/CVPR2022/papers/Chen_ReXNet_Diminishing_Representational_Bottleneck_on_Convolutional_Neural_Network_CVPR_2022_paper.pdf

[4] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. 2020. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108, 4 (2020), 485–532. https://ieeexplore.ieee.org/document/9043731

[5] Jack Dongarra, Hans-Werner Meuer, and Erich Strohmaier. 2020. TOP500 Supercomputer Sites. In *Proceedings of the International Supercomputing Conference*. Springer, Hamburg, Germany, 45–58.

[6] Daniel Geißler, Bo Zhou, Mengxi Liu, Sungho Suh, and Paul Lukowicz. 2024. The Power of Training: How Different Neural Network Setups Influence the Energy Demand. *arXiv preprint arXiv:2401.01851* (2024). https://arxiv.org/abs/2401.01851

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA.

[8] P. Goyal et al. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* 1, 1 (2017), 1–12. https://arxiv.org/abs/1706.02677

[9] Dipesh Gyawali. 2023. Comparative Analysis of CPU and GPU Profiling for Deep Learning Models. *arXiv preprint arXiv:2309.02521* (2023). https://arxiv.org/abs/2309.02521

[10] S. Han, H. Mao, and W. J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* 1, 1 (2016), 1–14. https://arxiv.org/abs/1510.00149

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE, Santiago, Chile, 1026–1034.

[12] A. G. Howard et al. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861* 1, 1 (2017), 1–9. https://arxiv.org/abs/1704.04861

[13] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. PMLR, Lille, France, 448–456.

[14] B. Jacob et al. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Salt Lake City, UT, USA, 2704–2713. https://arxiv.org/abs/1712.05877

[15] Akshaya Jagannadharao, Nicole Beckage, Sovan Biswas, Hilary Egan, Jamil Gafur, Thijs Metsch, Dawn Nafus, Giuseppe Raffa, and Charles Tripp. 2025. *A Beginner's Guide to Power and Energy Measurement and Estimation for Computing and Machine Learning*. Technical Report NREL/TP-2C00-91518. National Renewable Energy Laboratory, Golden, CO.

[16] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report. University of Toronto.

[17] A. Lewkowycz et al. 2021. How to Decay Your Learning Rate. *arXiv preprint arXiv:2103.12682* 1, 1 (2021), 1–20. https://arxiv.org/abs/2103.12682

[18] Chuan Li. 2019. Measuring Actual GPU Usage for Deep Learning Training. *Towards Data Science* 1, 1 (2019), 1–8.

[19] Sijia Li, Young D. Kwon, Lik-Hang Lee, and Pan Hui. 2025. MetaCLBench: Meta Continual Learning Benchmark on Resource-Constrained Edge Devices. *arXiv preprint arXiv:2504.00174* 1, 1 (2025), 1–15.

[20] Z. Li and S. Arora. 2019. An Exponential Learning Rate Schedule for Deep Learning. *arXiv preprint arXiv:1910.07454* 1, 1 (2019), 1–12. https://arxiv.org/abs/1910.07454

[21] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du. 2018. On-Demand Deep Model Compression for Mobile Devices: A Usage-Driven Model Selection Framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, Munich, Germany, 389–400. https://dl.acm.org/doi/10.1145/3210240.3210337

[22] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv preprint arXiv:1608.03983* 1, 1 (2017), 1–16. https://arxiv.org/abs/1608.03983

[23] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig. 2017. Early Stopping Without a Validation Set. *arXiv preprint arXiv:1703.09580* 1, 1 (2017), 1–14. https://arxiv.org/abs/1703.09580

[24] D. Masters and C. Luschi. 2018. Revisiting Small Batch Training for Deep Neural Networks. *arXiv preprint arXiv:1804.07612* 1, 1 (2018), 1–18. https://arxiv.org/abs/1804.07612

[25] Paulius Micikevicius, Sharan Narang, Nick Mishkin, et al. 2018. Mixed Precision Training. In *International Conference on Learning Representations (ICLR)*. ICLR, Vancouver, Canada, 1–12. arXiv:1710.03740.

[26] Om Pramod and Others. 2021. Convergence in Deep Learning. In *Proceedings of the International Conference on Deep Learning*. ACM, Virtual Conference, 245–260.

[27] Lutz Prechelt. 1998. Early Stopping—But When? In *Neural Networks: Tricks of the Trade*, G. Ber Orr and Klaus-R. Müller (Eds.). Springer, Berlin, Heidelberg, 55–69. https://link.springer.com/chapter/10.1007/978-3-642-35289-8_5

[28] Mohammad Rahman and Seong-Gon Lee. 2021. Performance Analysis of CNN Training on CPU vs GPU. *J. Parallel and Distrib. Comput.* 150 (2021), 100–112.

[29] G. Raskutti, M. J. Wainwright, and B. Yu. 2014. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *Journal of Machine Learning Research* 15 (2014), 335–366. https://www.jmlr.org/papers/volume15/raskutti14a/raskutti14a.pdf

[30] Ayesha Siddique, Lei Zhang, Raj Kumar, Waqas Ahmad, Charles C. Gillan, and Sally McClean. 2024. Resource-Constrained Neural Network Training: A Survey. *Scientific Reports* 14, 1 (2024), 5182. doi:10.1038/s41598-024-52791-0

[31] Leslie N. Smith. 2017. Cyclical Learning Rates for Training Neural Networks. *arXiv preprint arXiv:1506.01186* 1, 1 (2017), 1–10.

[32] Leslie N. Smith. 2022. Cyclic Learning Rate Schedules and Wall-Clock Time Trade-Offs. *arXiv preprint arXiv:2206.00832* (2022). https://arxiv.org/abs/2206.00832

[33] Anirudh Sriram and Gal Oshri. 2020. Introducing the new TensorFlow Profiler. TensorFlow Blog. https://blog.tensorflow.org/2020/04/introducing-new-tensorflow-profiler.html April 15, 2020.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958.

[35] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Long Beach, CA, USA, 2820–2828. https://arxiv.org/abs/1807.11626

[36] M. Tan and Q. V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning (ICML)*. PMLR, Long Beach, CA, USA, 6105–6114. https://arxiv.org/abs/1905.11946

[37] Ravi Tiwari. 2023. System Monitoring Made Easy with Python's Psutil Library. *Medium* 1, 1 (2023), 1–8. Online; accessed 2023-06.

[38] Rishabh Tiwari, Arnav Chavan, Deepak Gupta, Gowreesh Mago, and ... 2023. RCV2023 Challenges: Benchmarking Model Training and Inference for Resource-Constrained Deep Learning. In *ICCV Workshop on Resource-Efficient Deep Learning for Computer Vision (RCV), ICCV*. IEEE, Paris, France, 4234–4243.

[39] V. Vasudevan and A. Gupta. 2021. Resource Constraints in Free-Tier Cloud GPUs. *IEEE Cloud Computing* 8, 2 (2021), 45–53.

[40] X. Wang, F. Yu, Z. Dou, T. Darrell, and J. Gonzalez. 2018. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *European Conference on Computer Vision (ECCV)*. Springer, Munich, Germany, 409–424. https://openaccess.thecvf.com/content_ECCV_2018/papers/Xin_Wang_SkipNet_Learning_Dynamic_ECCV_2018_paper.pdf

[41] Y. Yao, L. Rosasco, and A. Caponnetto. 2007. On Early Stopping in Gradient Descent Learning. *Constructive Approximation* 26, 2 (2007), 289–315. https://link.springer.com/article/10.1007/s00365-006-0663-2

[42] Y. You, I. Gitman, and B. Ginsburg. 2017. Large Batch Training of Convolutional Networks. *arXiv preprint arXiv:1708.03888* 1, 1 (2017), 1–10. https://arxiv.org/abs/1708.03888

[43] Xiaoyu Zhang and Others. 2023. Efficient Deep Learning: A Survey on Making Deep Learning Models Efficient. *Comput. Surveys* 55, 12 (2023), 1–42.