

Accelerating Solar Panel Snow Detection: CPU vs. GPU Training Performance of VGG-19 and ResNet-50

Abdulkadir Turkan^{*ID}, Batuhan Hangun^{**ID}, Onder Eyecioglu^{†ID}

^{*} Department of Computer Engineering, Bolu Abant Izzet Baysal University, 14030 Bolu, Turkiye

^{**} Department of Computer Engineering, Yildiz Technical University, 34220 Istanbul, Turkiye

(abdulkadirturkan@gmail.com, batuhanhangun@gmail.com, onder.eyecioglu@ibu.edu.tr)

[†] Corresponding Author: Onder Eyecioglu, Department of Computer Engineering, Bolu Abant Izzet Baysal University, 14030 Bolu, Turkiye, onder.eyecioglu@ibu.edu.tr

Received: 06.04.2026 Accepted: 27.04.2026

Abstract- Solar panel snow detection is critical for maintaining photovoltaic system efficiency in cold climates. While deep learning models such as VGG-19 and ResNet-50 achieve near-perfect classification accuracy for this task, their training demands substantial computational resources. This study presents a systematic performance analysis of CNN training on the TRUBA national high-performance computing infrastructure, comparing multi-threaded CPU parallelization (1–112 threads on Intel Xeon Platinum 8480+) with GPU acceleration (NVIDIA V100). Experiments reveal that the model architecture fundamentally determines CPU scaling behavior: VGG-19, dominated by large fully connected layers, achieves 28× speedup at 56 threads (serial fraction $f \approx 1.8\%$), whereas ResNet-50, composed of many small residual operations, scales to only 1.56× ($f \approx 63.4\%$). GPU training provides 163–1067× acceleration over single-threaded CPU execution for both architectures. Classification quality (accuracy and F1-score ≥ 0.995) remains invariant across all hardware configurations, confirming that training infrastructure selection is purely a performance optimization decision.

Keywords solar panel snow detection, CNN training, GPU acceleration, CPU parallelization, Amdahl's law

1. Introduction

1.1. Impact of Snow Accumulation on Photovoltaic Performance

Solar energy has emerged as one of the fastest-growing renewable energy sources worldwide; global photovoltaic (PV) capacity exceeded 1,400 GW at the end of 2023 [1, 2]. One of the fundamental challenges of solar energy systems is their susceptibility to geographical and climatic conditions. For example, snow accumulation on solar panels in cold climate regions significantly impacts energy production and remains a major operational issue. In an experiment conducted by Quan et al. on how snow depth affects the energy production efficiency of photovoltaic modules, they demonstrated that when snow depth reaches 1 cm, the energy production efficiency of the entire photovoltaic module drops to 7.1% of normal levels, and when snow depth reaches 6 cm, the energy production capacity of the photovoltaic module drops to zero [3]. Additionally, other studies have shown that, depending on geographical location, panel tilt, and snowfall intensity, snow cover can reduce panel efficiency by 0.3–13% annually and by up to 25–40% during winter months [4, 5].

Therefore, timely detection of snow cover is critical for activating cleaning mechanisms and minimizing energy losses.

1.2. Deep Learning for Solar Panel Monitoring

Deep learning (DL)-based image classification has emerged as a powerful approach for automated solar panel inspection, progressively addressing monitoring tasks ranging from manufacturing defect detection to field-condition assessment. Deitsch et al. [6] compared hand-crafted features with CNN-based classifiers for detecting defective photovoltaic cells in electroluminescence images, finding that CNNs achieved accuracy above 88%. Akram et al. [7] proposed a CNN architecture for automatic defect detection in PV cells, demonstrating that deep learning can effectively replace manual visual inspection. Li et al. [8] extended CNN-based defect analysis to large-scale PV farms using thermal infrared imagery, identifying hot spots, cracks, and other module-level faults. More recently, integrated approaches combining aerial imaging with ground-based maintenance have been explored; for instance, Kishor et al. [9] employed Region-based CNN (R-CNN) segmentation combined with

drone imagery and reported a detection accuracy of 92.3% using CNN-LSTM models for solar panel soiling assessment.

While most of the studies listed above focus on electroluminescence or infrared imaging for defect detection, snow cover detection from visible light images has been relatively understudied. In our previous work [10], convolutional neural network (CNN) architectures, particularly VGG-19 [11], ResNet-50 [12], and InceptionV3, were shown to reliably classify solar panel images into three snow cover categories (*all_snow*, *no_snow*, and *partial*) and achieve F1 scores of 0.99, 0.91, and 1.00, respectively. That study established the classification viability of CNNs for snow detection; however, it did not investigate the computational cost of training these models, a critical consideration for practical deployment where models must be periodically retrained as environmental conditions evolve or new data becomes available.

1.3. Computational Cost of CNN Training

While CNN models deliver high classification accuracy, their training demands substantial computational resources, presenting a major bottleneck for operational deployment. Studies on computational complexity have shown that the training and runtime of a CNN are directly dependent on hyperparameters such as the number of convolutional layers, filter sizes, batch size, and optimization algorithms. For example, it has been found that the runtime of models with different configurations ranges from 1,208 to 1,645 seconds, and that for shallow architectures the primary computational load stems from the abundance of operations in the convolutional layers rather than the fully connected layers [13]. Hardware complexity analyses quantitatively reinforce these constraints; a standard 5-layer CNN model requires the storage of approximately 126,500 parameters and around 181,000 multiply-accumulate (MAC) operations during the inference phase [14]. Even with transfer learning using pre-trained weights from ImageNet [15, 16], a single training epoch of ResNet-50 on our snow detection dataset takes approximately 894 seconds on a single CPU thread, whereas VGG-19 takes over 10,000 seconds. In field-deployed solar monitoring systems, models may need to be periodically retrained to adapt to new panel types, changing climate conditions, or expanded image datasets. Therefore, understanding the relationship between hardware configuration and training efficiency is critical for operational planning.

1.4. HPC Acceleration Approaches

Modern high-performance computing (HPC) platforms offer two primary acceleration pathways for DL training: multi-threaded CPU parallelization via optimized linear algebra libraries such as Intel MKL [17], and GPU acceleration through NVIDIA CUDA [18] and cuDNN [19].

CPU-based parallelization. Multi-threaded CPU acceleration has proven effective for both CNN training and the computationally intensive preprocessing stages that feed DL pipelines. At the kernel level, when the Winograd algorithm and OpenMP-based SIMD (Single Instruction,

Multiple Data) parallelization were applied to the convolution layers, which account for the majority of the computational load, the complexity of multiplication operations was reduced, resulting in approximately a 30-fold performance increase in GFLOPS compared to a single core on 64-core Intel processors [20]. Multi-core tests conducted on models such as AlexNet and VGG16 similarly confirm that parallelization increases overall throughput by reducing CPU wait times [21]. Beyond training itself, CNN-based systems also benefit from parallelized data preparation. Hangun and Bayar [22] demonstrated that OpenMP-based parallelization of local and global image enhancement algorithms on systems with 48 physical cores achieves up to 38-fold speedup over sequential execution and operates up to 53.6 times faster than competing methods in the literature. This directly reduces the preprocessing bottleneck for datasets fed into CNN models. The efficiency of CPU thread utilization depends critically on the ratio between computational and synchronization overhead. Operations involving large matrices (e.g., FC layers in VGG-19) can efficiently distribute the workload across many threads, whereas workloads consisting of many small operations limit scalability by incurring thread-per-operation allocation and barrier synchronization costs [23]. The complexity of parallel programming extends beyond performance; hybrid MPI and OpenMP models introduce structural challenges such as deadlocks and race conditions. In this context, deep learning itself has been applied to address such challenges. A custom CNN model using abstract syntax tree (AST) tokenization successfully detected synchronization errors in parallel programs (MPI/OpenMP) with 97% accuracy and 99% AUC score without requiring manual feature engineering [24], illustrating the growing synergy between DL and parallel computing.

GPU-based acceleration. GPU parallelization provides decisive speedups across the entire DL pipeline, from preprocessing to training. At the preprocessing level, Hangün and Eyecioğlu [25] showed that CUDA-based GPU acceleration provides significant performance advantages in fundamental image processing operations such as resizing, thresholding, histogram equalization, and edge detection. These operations constitute the preprocessing steps applied before CNN models, thereby directly optimizing the deep learning pipeline. At the convolution kernel level, Jordà et al. [26] achieved speedups of up to 2.29× by developing cuConv, a custom CUDA convolution kernel surpassing cuDNN, and improved end-to-end inference time in ResNet-50 and Inception-v4 models. For domain-specific applications, Dong et al. [27] presented a GPU-optimized CNN framework for hyperspectral image classification that achieved 5.9× training speedup by combining CUDA multistream execution with mini-batch gradient descent. While the superiority of GPUs for DL workloads is well documented in the literature [28–30], the degree of acceleration depends heavily on model architecture characteristics [23].

Architecture-dependent scaling and Amdahl's Law. Critically, the parallelization benefit from both CPU threading and GPU acceleration is fundamentally constrained by model architecture. Architectures dominated by large matrix

multiplications (e.g., the fully connected layers in VGG-19) exhibit different parallelization behavior than architectures composed of many small, heterogeneous operations (e.g., the residual blocks in ResNet-50). This architecture-dependent scaling behavior is well-predicted by Amdahl's Law [31, 32], yet empirical validation on real HPC systems with production DL frameworks remains limited.

1.5. Research Gap, Objectives, and Contributions

Despite the extensive body of work reviewed above, several important gaps remain in the literature that motivate the present study. First, existing HPC benchmarking studies for deep learning predominantly focus on large-scale datasets (e.g., ImageNet) and do not systematically evaluate acceleration for domain-specific tasks such as solar panel monitoring, where dataset size, image resolution, and class distribution differ substantially. Second, while individual CPU threading and GPU acceleration studies exist, no prior work has systematically compared CPU thread scaling with GPU acceleration on the same HPC infrastructure using architecturally distinct models under identical training protocols. Third, the effect of Non-Uniform Memory Access (NUMA) topology on DL training performance has rarely been empirically quantified. Fourth, whether hardware configuration affects classification quality (not only training speed) has not been rigorously verified across a comprehensive range of configurations.

To address these gaps, this study presents a systematic analysis of CNN-based classification of snow accumulation patterns on solar panels using VGG-19 and ResNet-50 on the TRUBA national HPC infrastructure [33], with the Snow Covered Solar Panels Dataset [10, 34] comprising three classes: no snow, partial snow, and all snow. The following research questions guide this investigation:

- How does increasing the number of CPU threads affect the training performance of VGG-19 and ResNet-50, and to what extent does the model architecture determine CPU scaling behavior?
- For each architecture, how much speedup does GPU (CUDA) acceleration provide compared to multi-threaded CPU training?
- To what extent does hardware configuration (CPU thread count, or CPU vs. GPU) affect CNN classification quality in terms of accuracy and F1-score.
- Can Amdahl's Law accurately predict the scaling limits observed in empirical measurements?

The main contributions and innovations of this work are as follows:

1. **First systematic CPU–GPU comparison on a single HPC platform for domain-specific DL:** A comparative analysis of CNN training performance across CPU (1–112 threads) and GPU (NVIDIA V100) configurations on the TRUBA national HPC infrastructure, using two architecturally distinct models (VGG-19 and ResNet-50) for a real-world solar panel snow detection task, providing

empirical evidence that has been absent from the literature.

2. **Empirical validation of Amdahl's Law for DL training:** Quantitative demonstration that VGG-19 exhibits a serial fraction of $f \approx 1.8\%$ (maximum speedup $28 \times$ at 56 threads), while ResNet-50 exhibits $f \approx 63.4\%$ (maximum speedup $1.56 \times$), revealing an $18\times$ difference in CPU scaling attributable solely to model architecture.
3. **Quantification of NUMA topology effects on DL training:** Evidence that cross-socket memory access introduces significant performance degradation beyond the single-socket boundary (56 threads), with VGG-19 losing 36% of its peak speedup at 112 threads.
4. **Hardware-invariant classification quality:** Confirmation that classification quality (accuracy ≥ 0.997 , F1-score ≥ 0.995) remains constant across all 18 tested hardware configurations, establishing that training platform selection is purely a computational efficiency decision with no impact on model accuracy.

The remainder of this paper is organized as follows. Section 2 describes the experimental methodology, including the dataset, models, hardware platforms, experimental design, and evaluation metrics. Section 3 presents the results covering CPU scaling analysis, GPU performance, and comparative evaluation. Section 4 discusses the practical implications, and Section 5 concludes the paper.

2. Methodology

To address the research questions of this study, two complementary experiments were designed on the TRUBA national high-performance computing infrastructure [33]. Two architecturally distinct CNN models (VGG-19 [11] and ResNet-50 [12]) were selected as benchmarks because they represent opposite ends of the parallelization spectrum: While VGG-19 is dominated by a small number of large fully connected (FC) layers whose weight matrices map directly to highly parallelizable BLAS operations, ResNet-50 distributes its computation across many small, heterogeneous residual operations with sequential dependencies. Both models were initialized with ImageNet pre-trained weights, and a three-class solar panel snow detection dataset derived from our previous work [10] was used. In the first experiment, CPU multithreading scalability was evaluated using PyTorch's [35] OpenMP/MKL backend. In the second experiment, GPU training was performed using the CUDA/cuDNN backend. To ensure that the performance differences observed in both experiments could be attributed solely to hardware configuration, the same hyperparameters, data splits, and evaluation protocols were used. Additionally, the classification performance of both CNN models was determined using the accuracy, cross-entropy loss, and F1 score metrics. The dataset, model configurations, hardware platforms, experimental design, and evaluation metrics are detailed in the following subsections.

2.1. Dataset

The dataset consists of solar panel images captured under three snow coverage conditions: *all_snow* (fully covered), *no_snow* (clean panels), and *partial* (partial coverage). The original dataset, derived from surveillance camera footage of a rooftop solar panel installation [34], contains 394 unique source images distributed as follows: 97 *all_snow*, 270 *no_snow*, and 27 *partial* images. These original images were expanded to 3,000 samples through offline data augmentation (random horizontal and vertical flips, rotation up to 15° , brightness and contrast adjustment, and color jitter), consistent with the augmentation strategy used in our previous work [10]. The per-class augmentation ratios ranged from approximately $7.5\times$ (*no_snow*) to $8.3\times$ (*partial*), as detailed in Table 1. We acknowledge that augmentation was applied to the entire image pool before splitting; consequently, augmented variants of the same source image may appear across different splits. This protocol introduces potential data leakage that likely inflates the absolute classification accuracy values reported

here. However, since the present study focuses on hardware-induced *variation* in classification quality across configurations rather than absolute accuracy, this limitation does not affect our central conclusions: identical training protocols on identical data should produce equivalent generalization, allowing us to isolate the effect of hardware configuration. A leak-free evaluation protocol with source-level splitting is left for future work.

All images were resized to 192×192 pixels and normalized using ImageNet channel statistics (μ , σ). The dataset was divided into 70% training, 15% validation, and 15% test sets using stratified sampling to preserve class proportions. The exact class distribution across all splits is reported in Table 1. Additionally, stochastic augmentation transforms (random horizontal flip, rotation up to 10° , and color jitter with brightness and contrast factors of 0.2) were applied online to the training set only during each epoch, further diversifying the training samples without altering the validation or test distributions.

Table 1. Dataset class distribution before and after augmentation, and per-split sample counts. Stratified sampling preserves class proportions across train (70%), validation (15%), and test (15%) set

Class	Original	Aug. ratio	Augmented	Train	Val	Test	Proportion
<i>all_snow</i>	97	$\sim 7.7\times$	749	524	112	113	25.0%
<i>no_snow</i>	270	$\sim 7.5\times$	2,027	1,418	304	305	67.6%
<i>partial</i>	27	$\sim 8.3\times$	224	156	33	35	7.5%
Total	394	$\sim 7.6\times$	3,000	2,098	449	453	100%

2.2. Models

In this study, two CNN models with different architectures, both initialized with weights pre-trained on the ImageNet dataset, were selected. The first selected model, VGG-19 [11], features a 19-layer sequential architecture consisting of 16 convolutional layers organized into five blocks, along with an adaptive average pooling layer and three FC layers. In line with the study's objective, the model's final FC layer was modified to produce three-class outputs (Figure 1). With approximately 143 million parameters, VGG-19 is characterized by massive FC weight matrices (512×4096 and 4096×4096), which significantly dominate both

computational and memory requirements. The second model, ResNet-50 [12], offers a fundamentally different computational profile compared to VGG-19. This 50-layer residual network architecture consists of four main stages: an initial base structure (7×7 convolution, batch normalization, and max-pooling), followed by four stages containing 3, 4, 6, and 3 bottleneck blocks, respectively (Figure 2). Each of these bottleneck blocks includes 1×1 , 3×3 , and 1×1 convolutions, supported by batch normalization and skip connections. In this model, where a total of 25.6 million parameters are distributed across numerous small operations, the final classification layer is configured to map the 2,048 features obtained to three output classes.

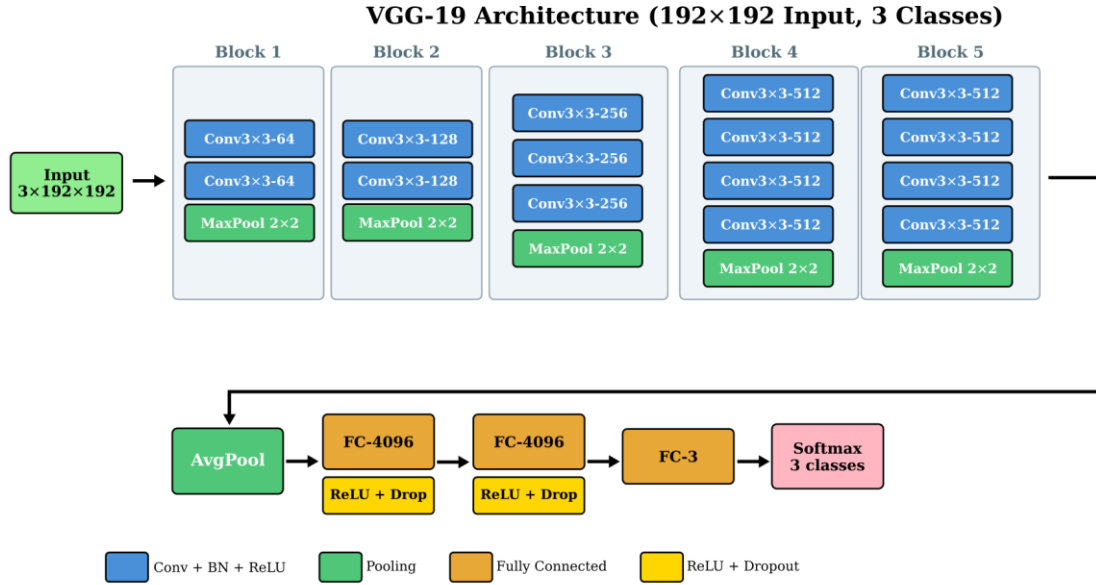


Fig. 1. VGG-19 architecture adapted for 192×192 input with 3-class output. The network consists of five convolutional blocks with max pooling, followed by three fully connected layers with dropout.

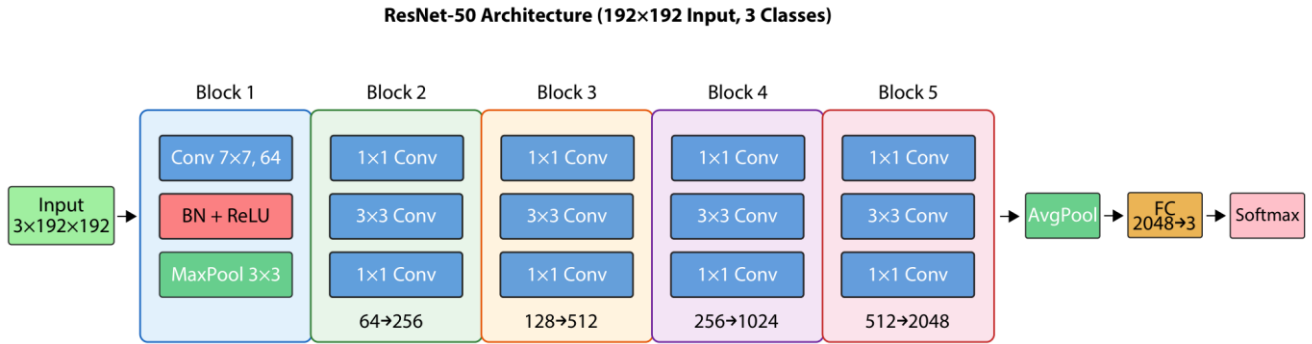


Fig. 2. ResNet-50 architecture with stem, four bottleneck stages, and 3-class output. Skip connections bypass each bottleneck block, enabling gradient flow through residual paths.

Both models were trained using the Adam optimizer with learning rate $\eta = 10^{-4}$ and weight decay $\lambda = 10^{-4}$. A class-weighted cross-entropy loss was employed to account for class imbalance, with weights inversely proportional to class frequency. A ReduceLROnPlateau scheduler reduced the learning rate by a factor of 0.5 after 3 epochs without validation loss improvement. Early stopping with a patience of 7 epochs was applied to prevent overfitting.

2.3. Hardware Platforms

All experiments were performed on the TRUBA national HPC infrastructure [33]. Table 2 summarizes the two hardware configurations used.

Table 2. Hardware platform specifications

Specification	TRUBA (orfoz Node)	TRUBA (akya-cuda Node)
Processor	Xeon Platinum 8480+	Xeon Gold 6248R
Cores	2×56 = 112	2×20 = 40
NUMA domains	8	—
GPU	None	V100-SXM2-16GB
CUDA version	—	12.0
PyTorch	2.1.0	2.1.0

The CPU platform features a dual-socket Intel Xeon Platinum 8480+ node with 8 Non-Uniform Memory Access (NUMA) domains. Each socket provides 56 physical cores,

and the two sockets are interconnected via UPI. The GPU platform is equipped with an NVIDIA V100-SXM2-16GB

with 5,120 CUDA cores, 16 GB HBM2 memory, and a thermal design power (TDP) of 300 W.

2.4. Experimental Design

Two complementary experiments were designed to investigate the effect of hardware parallelization on training

performance while keeping all other variables, optimizer, learning rate, batch size, loss function, and data splits constant. The end-to-end training pipeline common to both experiments is illustrated in Fig. 3

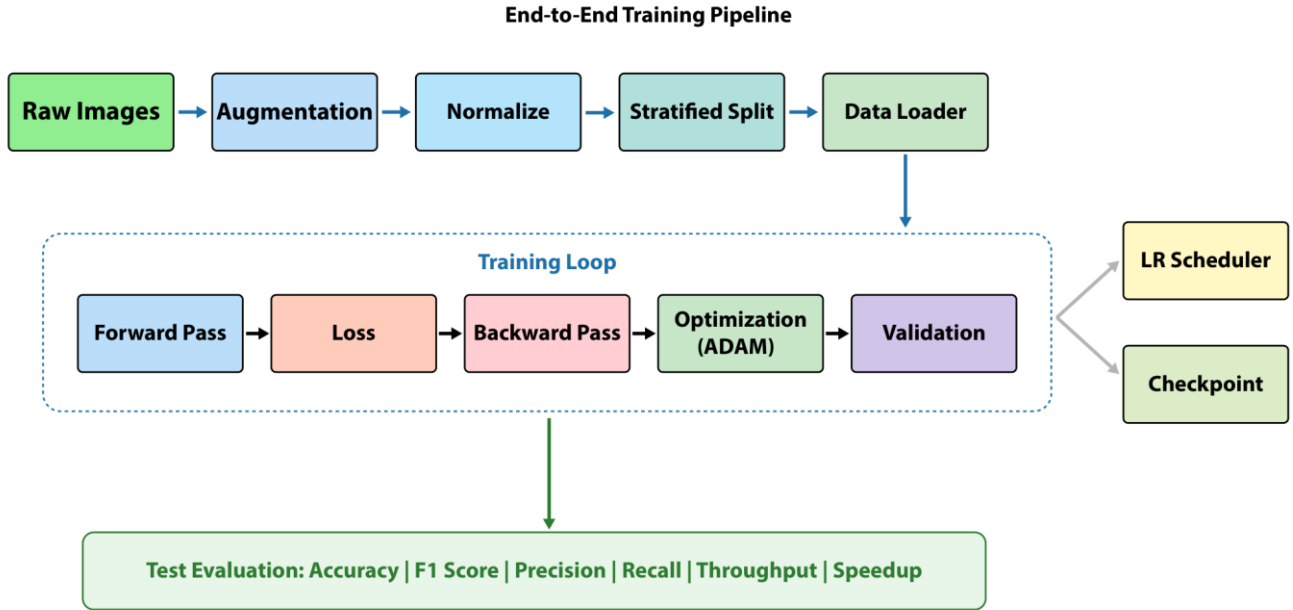


Fig. 3. End-to-end training pipeline: data augmentation, stratified split, training loop with validation, and final test evaluation.

2.4.1. CPU Multi-threaded Scaling Experiment

CPU thread counts were varied across {1,2,4,7,14,28,56,112}, spanning from single-threaded execution to full dual-socket utilization. Threading was controlled via `torch.set_num_threads()`, which configures both OpenMP and MKL thread pools. The DataLoader was set to `num_workers=0` to isolate intra-operator parallelism from data loading parallelism. Batch size was fixed at 32 with a maximum of 15 epochs per configuration.

2.4.2. GPU CUDA Experiment

GPU training utilized PyTorch's CUDA backend [35], which offloads convolution and linear algebra operations to the cuDNN [19] and cuBLAS libraries running on NVIDIA's CUDA Toolkit 12.0. All model parameters and input tensors were transferred to the GPU via `model.cuda()` and automatic device placement; the cuDNN auto-tuner was enabled to select the fastest kernel for each layer configuration. GPU training used 2 CPU threads (fixed) for the DataLoader. The batch size was 32, and training ran for up to 30 epochs.

2.5. Evaluation Metrics

2.5.1. Classification Metrics

The following standard metrics were computed on the held-out test set to assess model classification quality.

Accuracy measures the fraction of correctly classified samples over the total number of samples:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

where TP , TN , FP , and FN denote true positives, true negatives, false positives, and false negatives, respectively. Cross-entropy loss quantifies the discrepancy between predicted probability distributions and ground-truth class labels. For a C -class problem with class-imbalance weighting, the loss for a batch of N samples is:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C w_c y_{ic} \log \hat{p}_{ic} \quad (2)$$

where w_c is the class weight inversely proportional to class frequency, $y_{ic} \in \{0,1\}$ is the one-hot ground-truth indicator, and $\hat{p}_{ic}$ is the predicted softmax probability for class c . Precision and recall for class c are:

$$\text{Precision}_c = \frac{TP_c}{TP_c+FP_c}, \quad \text{Recall}_c = \frac{TP_c}{TP_c+FN_c} \quad (3)$$

F1-score is the harmonic mean of precision and recall, providing a balanced measure under class imbalance:

$$F1_c = 2 \cdot \frac{\text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c} \quad (4)$$

The macro-averaged F1-score averages $F1_c$ uniformly across all C classes regardless of class frequency. Per-class scores are also reported to assess performance on the minority class (*partial*: 224 samples out of 3,000).

2.5.2. HPC Performance Metrics

The following metrics were collected to quantify training efficiency: average epoch time (seconds), training throughput (images/second), speedup $S(p) = T(1)/T(p)$, parallel efficiency $E(p) = S(p)/p$, CPU/GPU utilization (%), peak memory usage (RSS for CPU, `nvidia-smi` for GPU), and average GPU power draw (watts). All reported timings represent the mean across the steady-state epochs of a single training run per configuration, with the first epoch excluded to remove kernel auto-tuning and cache warm-up effects. Per-epoch variability within each run is reported via box plots in Section 3.1.5. We acknowledge that single-run measurements provide point estimates without across-run confidence intervals, and that multi-replicate runs would strengthen the statistical robustness of the speedup estimates; this remains a limitation of the present study.

A note on metric definitions is warranted. CPU utilization is reported as a fraction of *total system cores* (112 on the orfoz node), so a single fully-utilized thread corresponds to approximately $1/112 \approx 0.9\%$. This explains the 0.9% value reported for ResNet-50 at 1T, where one thread is fully active but the remaining 111 cores are idle. Peak RSS memory for CPU is sampled via `psutil` at maximum observed value during training, including PyTorch framework overhead, shared libraries, and per-thread workspace buffers, but excluding kernel page cache. GPU memory is reported as the peak allocated value via `nvidia-smi`.

2.5.3. Performance Modeling: Amdahl's Law

Amdahl's Law [31] provides a fundamental bound on parallel speedup. For a program with serial fraction f , the maximum speedup with p processors is:

$$S(p) = \frac{1}{f + \frac{1-f}{p}} \quad (5)$$

This implies an upper bound of $S_{\max} = 1/f$ regardless of processor count. Hill and Marty [32] extended this to multicore architectures, showing that even small serial fractions significantly limit scaling on many-core processors. Gustafson [36] proposed an alternative scaling law where the problem size grows with processor count, yielding more optimistic predictions for data-parallel workloads. Ben-Nun and Hoefler [23] identified DL-specific serial bottlenecks (batch normalization, gradient synchronization, and memory allocation) that increase the effective serial fraction. The serial fraction f is estimated from observed speedup by rearranging Eq. (5):

$$f = \frac{\frac{1}{S(p)} - \frac{1}{p}}{1 - \frac{1}{p}} \quad (6)$$

where $S(p)$ is the observed speedup at p threads.

3. Results and Analysis

In this section, the experimental findings of the study are presented and analyzed in detail. First, CPU multi-threaded scaling behavior is examined, covering training duration and throughput, validation of Amdahl's Law, the effects of

NUMA topology, memory scaling, and epoch duration variability. Next, GPU training performance on the NVIDIA V100 architecture is evaluated based on metrics such as throughput, resource utilization, and power consumption. These analyses are followed by a direct CPU–GPU comparison that quantifies the performance gain (speedup) provided by GPU acceleration compared to the best CPU configurations. In the final stage, the classification quality obtained across all 18 tested model-hardware combinations (8 CPU thread counts $\times$ 2 models + 1 GPU $\times$ 2 models) is examined. The fact that both models achieve an accuracy of ≥ 0.997 and a macro-averaged F1 score of ≥ 0.995 across all configurations (a significant improvement over the F1 scores of 0.99 for VGG-19 and 0.91 for ResNet-50 reported in our previous work [10]) confirms that the choice of training platform in deep learning is a purely hardware-based decision driven by computational efficiency, with no impact on model quality [15].

3.1. CPU Multi-threaded Scaling

This subsection examines how intra-operator thread count affects training performance on the dual-socket Xeon Platinum 8480+ node. Training time, throughput, and speedup are presented first, followed by an Amdahl's Law analysis of the serial fraction, an investigation of cross-socket NUMA effects at 112 threads, memory scaling trends, and epoch-time variability.

3.1.1. Training Time and Throughput

Comprehensive CPU scaling results for the ResNet-50 and VGG-19 models are presented in Table 3 and Table 4, respectively. Although both models were trained on the same hardware, dataset, and hyperparameters, their scaling behaviors differ fundamentally. When the number of threads is increased from 1 to the single-socket limit of 56, ResNet-50's epoch time decreases from 894.2 seconds to 572.4 seconds, recording only a modest 36% reduction. VGG-19's epoch time drops from 10,052 seconds to 356.7 seconds, achieving a 96% reduction and a striking 28-fold ($28\times$) speedup. This significant disparity directly reflects the architectural differences discussed in Section 2.2: VGG-19's massive FC weight matrices (512×4096 and 4096×4096) align efficiently with multi-threaded BLAS operations, whereas ResNet-50's numerous small residual operations cannot fully saturate the available threads.

These contrasting behaviors are clearly illustrated in Figure 4. In panel (a), where the epoch time is plotted on a logarithmic scale against the number of threads, VGG-19 exhibits a nearly linear and steep decline from 1 to 56 threads, ResNet-50 exhibits a much shallower decline profile, and the curve's flattening after 4 threads indicates that the parallelizable portion has reached early saturation. On the other hand, due to cross-socket NUMA overhead, both models experience a performance drop at the 112-thread level (where the curve breaks upward). This pattern is confirmed in panel (b), which shows the corresponding throughput (images/second) scaling: While VGG-19's throughput increases 28-fold from 0.21 to 5.88 images/second, ResNet-50's throughput only increases from 2.35 to 3.67

images/second (a 1.56-fold improvement), highlighting just how limited the benefit of additional hardware resources is in architectures with a high serial fraction.

Table 3. ResNet-50 CPU multi-threaded training results

T	Epoch (s)	img/s	S	E (%)	CPU (%)	Mem (MB)	F1
1	894.2	2.35	1.00	100.0	0.9	3999	1.000
2	740.1	2.83	1.21	60.4	1.8	4745	1.000
4	654.5	3.21	1.37	34.2	3.6	4829	1.000
7	618.7	3.39	1.45	20.6	6.2	5195	1.000
14	584.3	3.59	1.53	10.9	12.4	5220	1.000
28	572.6	3.66	1.56	5.6	24.7	5755	1.000
56	572.4	3.67	1.56	2.8	49.3	5926	0.995
112	617.7	3.40	1.45	1.3	97.3	6294	1.000

Table 4. VGG-19 CPU multi-threaded training results

T	Epoch (s)	img/s	S	E (%)	CPU (%)	Mem (MB)	F1
1	10052	0.21	1.00	100.0	47.4	5396	1.000
2	4012	0.52	2.51	125.3	16.3	5474	1.000
4	2753	0.76	3.65	91.3	18.4	7550	1.000
7	1285	1.63	7.82	111.7	6.8	5519	0.995
14	1165	1.80	8.63	61.7	13.0	7753	1.000
28	614.5	3.41	16.36	58.4	26.0	10060	0.995
56	356.7	5.88	28.18	50.3	51.6	12410	1.000
112	555.3	3.78	18.10	16.2	97.2	12410	1.000

GPU vs CPU Training Performance — VGG-19 and ResNet-50

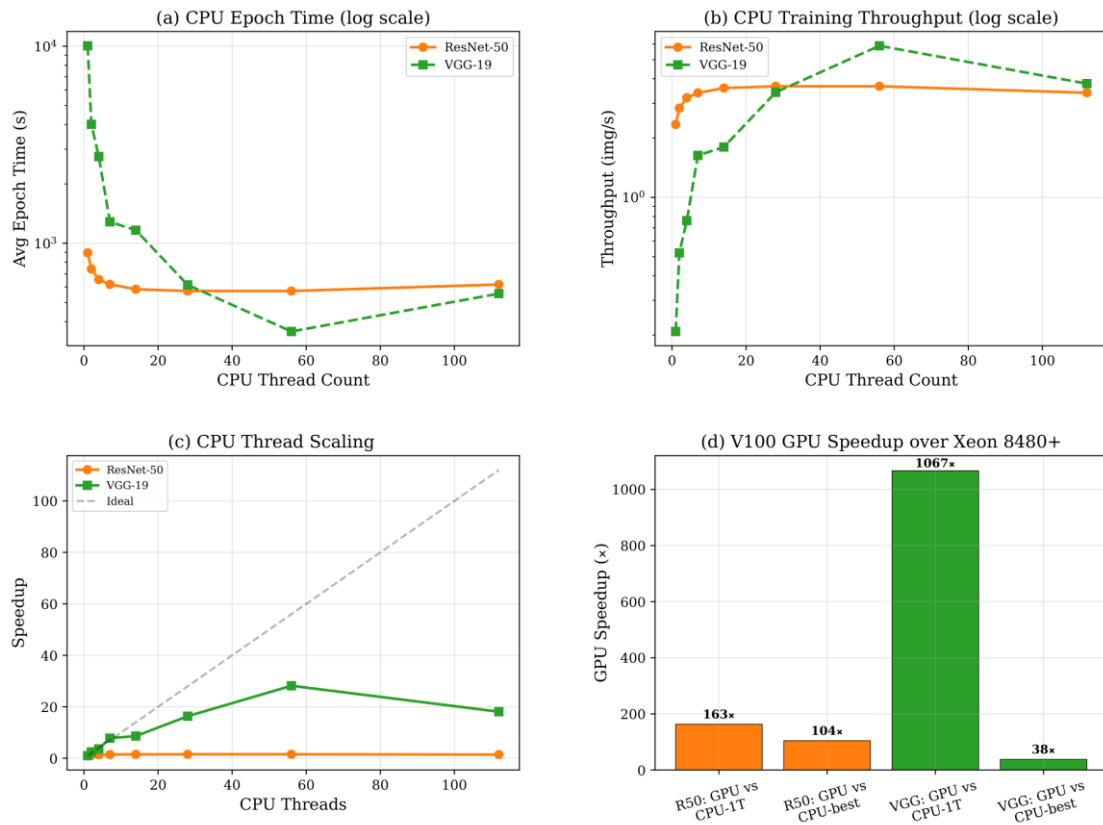


Fig. 4. Combined performance summary: (a) epoch time vs. thread count, (b) throughput scaling, (c) speedup with Amdahl curves, and (d) GPU speedup ratios.

3.1.2. Amdahl's Law Analysis

The serial fractions, calculated as $f \approx 0.634$ for ResNet-50 and $f \approx 0.018$ for VGG-19, differ by roughly 35-fold and directly explain the contrasting scaling behavior of the two models. Note that this 35 $\times$ ratio in serial fractions translates to an 18 $\times$ ratio in observed maximum speedup (28.2 $\times$ vs. 1.56 $\times$); both quantities reflect the same underlying architectural difference but through different mathematical lenses. According to Amdahl's Law, ResNet-50's theoretical maximum speedup is $1/0.634 \approx 1.58 \times$, while VGG-19's theoretical upper limit is calculated as $1/0.018 \approx 55.7 \times$. The experimentally observed maximum acceleration values of 1.56 $\times$ and 28.2 $\times$ indicate that ResNet-50 has effectively reached the Amdahl limit, while the gap between VGG-19's observed 28.2 $\times$ speedup and its 55.7 $\times$ theoretical ceiling suggests that, on a hypothetical larger single-socket processor or a NUMA-optimized dual-socket implementation, further scaling would be possible. With current cross-NUMA penalties, however, 56 threads represents the practical optimum on this platform.

It should be noted that the estimated serial fractions encompass not only intrinsic architectural sequential dependencies (e.g., batch normalization, residual block ordering) but also pipeline-level serial work including data loading (with `num_workers=0`), validation passes, and PyTorch framework overhead. The reported f values therefore represent an effective serial fraction at the training-loop level rather than a pure architectural property. This effect is most pronounced for ResNet-50, whose smaller per-batch compute makes data-loading overhead a proportionally larger fraction of epoch time, which may inflate the apparent serial fraction beyond what is attributable to architecture alone. Disentangling these contributions through finer-grained profiling is left for future work.

The overlap between the observed acceleration values and the theoretical Amdahl curves adapted to the estimated scaling rates is presented in Figure 4 (c). In this graph, the two models exhibit strikingly different profiles: VGG-19's curve shows a continuous and steep rise across 2, 7, 14, 28, and 56 threads (T), closely matching the predicted Amdahl estimate for $f = 0.018$; and even exhibits super-linear acceleration at the 2T ($S = 2.51$, 125% efficiency) and 7T ($S = 7.82$, 112% efficiency) levels before converging to the theoretical slope at high thread counts. This observed super-linear behavior is consistent with cumulative cache effects: as the number of threads increases, the combined L2/L3 cache capacity grows proportionally, allowing more of VGG-19's massive FC weight matrices to be cached, thereby reducing main memory access frequency. We note, however, that direct verification of this explanation would require cache hit-rate measurements (e.g., via `perf` hardware counters), which were not collected in this study. In contrast, ResNet-50's curve flattens almost immediately, reaching $S = 1.37$ at the 4T level, and then plateaus around $S = 1.56$, strictly adhering to the Amdahl ceiling of $1/0.634 \approx 1.58$ starting from 28 threads. This rapid saturation point reflects the dominance of serial operations such as batch normalization, element-wise additions, and dependencies within sequential residual blocks that cannot

benefit from additional threads. Consequently, the dramatic divergence between these two scaling profiles, despite using identical hardware, datasets, and training protocols, demonstrates that the model architecture is the primary determinant of CPU parallelization efficiency. The implications of this finding for architecture-aware HPC task scheduling will form the focus of future research.

3.1.3. Non-Uniform Memory Access (NUMA) Effects

As described in Section 2.3, the CPU platform comprises a dual-socket Intel Xeon Platinum 8480+ with 8 NUMA domains (4 per socket) interconnected via UPI. At 56 threads, execution is confined to a single socket where all memory accesses are local; at 112 threads, the workload spans both sockets, introducing remote memory accesses that traverse the Ultra Path Interconnect (UPI) interconnect with significantly higher latency.

This cross-socket penalty manifests differently for the two architectures. ResNet-50 speedup drops from 1.56 $\times$ (56T) to 1.45 $\times$ (112T), a 7% loss. This modest degradation occurs because ResNet-50's many small operations generate relatively little data movement per operation, limiting the impact of increased memory latency. VGG-19, by contrast, drops from 28.2 $\times$ to 18.1 $\times$, a 36% loss that eliminates over one-third of the peak speedup. This disproportionate penalty arises because VGG-19's FC layers require frequent access to large weight matrices (512×4096 , 4096×4096); when threads on the remote socket must fetch these buffers across UPI, the resulting latency directly increases the per-operation execution time of the dominant BLAS calls. As shown in Section 3.1.4, VGG-19's peak RSS memory also reaches 12.4 GB at 112T (130% growth from 1T), further increasing the fraction of data that resides in remote NUMA domains.

These results establish 56 threads (matching the single-socket physical core count) as the optimal configuration. Allocating threads beyond the socket boundary not only fails to improve performance but actively degrades it, an important practical guideline for HPC job scheduling on multi-socket nodes.

It should also be acknowledged that the cross-socket performance penalty observed at 112 threads may be compounded by thermal effects. On the Xeon Platinum 8480+, sustained dense floating-point computation across many cores can trigger AVX-512 frequency reduction or thermal throttling, particularly at full dual-socket utilization. Disentangling NUMA-induced memory latency from thermal-induced frequency reduction would require per-core frequency monitoring (e.g., via `turbostat` or hardware performance counters), which was not collected in this study. The 36% performance loss for VGG-19 at 112 threads should therefore be understood as a combined effect of these two mechanisms, with NUMA effects likely dominant given VGG-19's heavy reliance on large weight-matrix accesses.

3.1.4. Memory Scaling

In addition to execution time, the multi-threaded training process directly affects memory consumption because each OpenMP/MKL thread allocates private buffers for

intermediate computation results. According to the trend visualized in Figure 5, ResNet-50's peak RSS memory is 4.0 GB with 1 thread (1T) and rises to 6.3 GB with 112 threads (112T), showing a 57% increase, while VGG-19's memory consumption jumps from 5.4 GB to 12.4 GB, growing by 130% and more than doubling its single-threaded memory footprint. We note that VGG-19's peak RSS values are identical at 56T and 112T (12,410 MB), suggesting that the reported peak may reflect a memory-allocator pool ceiling or sampling resolution limit rather than precise working-set measurement. The relative growth trend, however, is consistent with the architectural differences between the two models. This imbalance is fully consistent with the architectural features discussed throughout this section: VGG-

19's FC layers contain massive weight matrices of 512×4096 and 4096×4096 elements, and each requires per-thread workspace copies during multi-threaded GEMM (Generalized Matrix Multiplication) operations, whereas ResNet-50's smaller convolutional kernels impose proportionally much less overhead. This peak value of 12.4 GB, achieved by VGG-19 on 112 threads, is particularly noteworthy given the NUMA effects analyzed in Section 3.1.3. As the total memory footprint grows, an increasingly larger portion of the data is inevitably located in distant NUMA domains, a situation that multiplies the cross-socket latency penalty and directly contributes to the 36% performance loss observed with 112 threads.

CPU Peak Memory (RSS) vs Thread Count (Xeon Platinum 8480+)

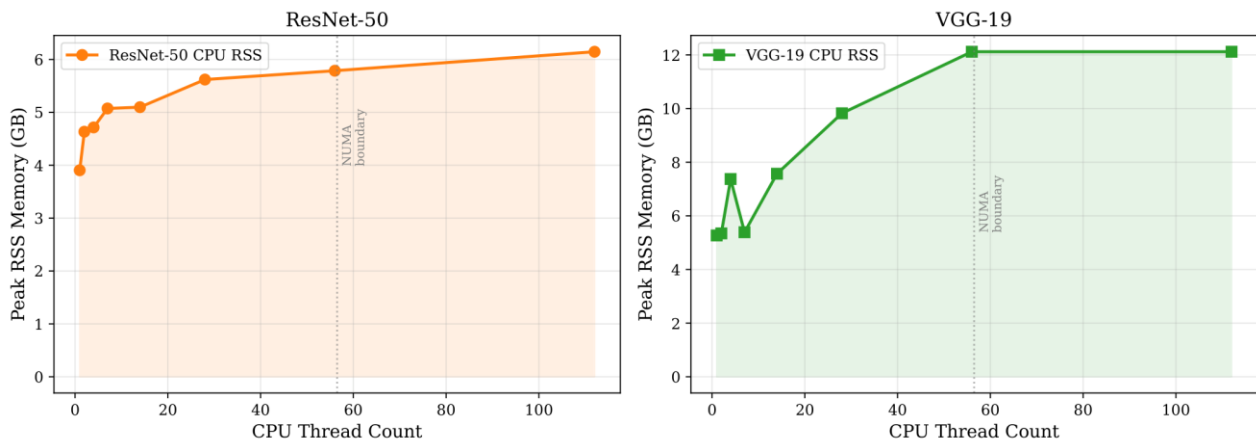


Fig. 5. CPU peak RSS memory vs. thread count. VGG-19 exhibits 130% memory growth from 1T to 112T due to large FC layer buffers, while ResNet-50 grows 57%.

3.1.5. Epoch Time Variability

Beyond average performance, the consistency of training duration across epochs is also of great importance for workload scheduling on shared HPC clusters. Figure 6 presents the training times per epoch across all thread configurations using box plots. It is observed that VGG-19 exhibits a notably higher relative variability at low thread counts (1T–4T); this can be attributed to the much larger cumulative impact of system-level external factors (such as operating system scheduling or background I/O operations on shared nodes) on the long absolute epoch durations (2,753–

10,052 s). When the number of threads was increased to 56 (56T), both models were observed to converge to stable, low-variance epoch durations (356.7 s for VGG-19 and 572.4 s for ResNet-50). This convergence indicates that the parallelized workload is utilizing (saturating) the available cores at full capacity and has become less sensitive to external system interventions. On the other hand, in all tested configurations, the first epoch consistently runs slower than subsequent epochs due to one-time kernel auto-tuning in oneDNN/MKL and CPU cache warm-up effects. However, this temporary cost is quickly amortized over the remaining periods and does not affect steady-state transaction volume measurements.

CPU Epoch Time Variability

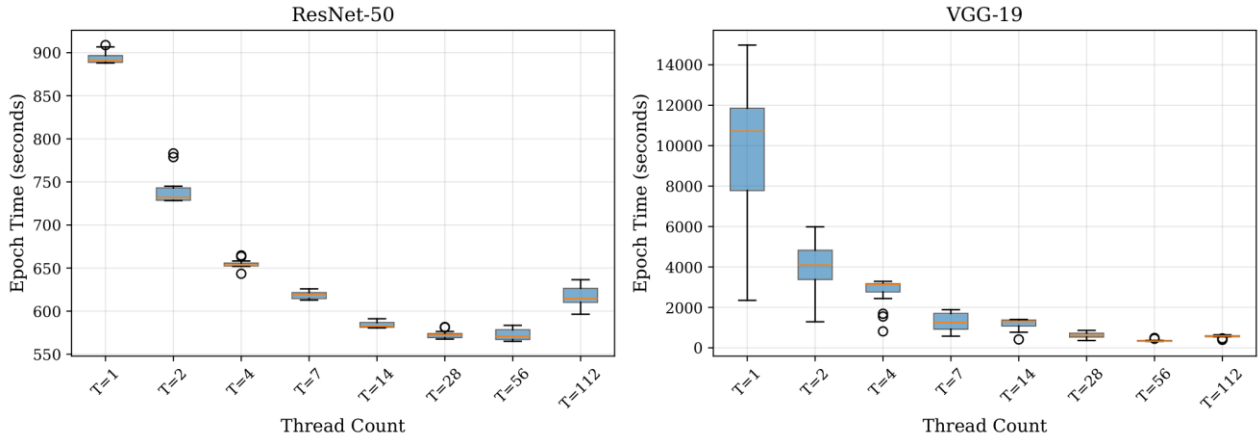


Fig. 6. CPU epoch time variability across thread configurations, shown as box plots for ResNet-50 and VGG-19.

3.2. GPU Training Performance

Having established the CPU scaling behavior and its architectural limitations in Section 3.1, this subsection evaluates training performance on the NVIDIA V100 GPU, where the parallelization paradigm shifts from thread-level BLAS concurrency to warp-level Single Instruction, Multiple Threads (SIMT) execution across 5,120 CUDA cores. Table 5 summarizes the results. ResNet-50 achieves 382.7 img/s (5.48 s/epoch), while VGG-19 reaches 222.7 img/s (9.42 s/epoch), meaning ResNet-50 attains 72% higher throughput than VGG-19 on the GPU. This reversal of the CPU scaling pattern,

where VGG-19 consistently outperformed ResNet-50 in multi-threaded scaling, reflects an intrinsic difference in how each architecture maps onto GPU hardware: ResNet-50's many small independent operations can be efficiently pipelined across the V100's streaming multiprocessors, whereas VGG-19's large FC layers, though highly parallelizable on CPU via BLAS, present fewer but larger kernel launches that leave some GPU resources idle between successive layers. Both models achieve perfect classification (accuracy and F1-score of 1.000), confirming that GPU training introduces no quality degradation.

Table 5. GPU training performance (NVIDIA V100-SXM2-16GB)

Model	Epoch (s)	img/s	GPU (%)	Mem (MB)	Power (W)	Acc	F1
ResNet-50	5.48	382.7	91.6	5335	232.4	1.000	1.000
VGG-19	9.42	222.7	94.3	5859	196.8	1.000	1.000

3.2.1. GPU Resource Utilization

Fig. 7 provides a detailed breakdown of GPU resource utilization during training. Both models achieve high compute utilization (91.6% for ResNet-50 and 94.3% for VGG-19), indicating that neither model is memory-bound or I/O-bound; the GPU's computational units are effectively saturated throughout the training loop. Peak GPU memory usage ranges from 5,335 MB (ResNet-50) to 5,859 MB (VGG-19), consuming only approximately 33% to 37% of the V100's 16 GB HBM2 capacity. This substantial headroom suggests that increasing the batch size beyond 32 could further improve throughput by better amortizing kernel launch overhead, a direction to be explored in future work. It should also be noted

that all GPU experiments were performed in single-precision (FP32) mode without automatic mixed precision (AMP). Consequently, the V100's 640 tensor cores, which accelerate FP16/FP32 mixed-precision matrix multiplications, were not utilized. Enabling AMP could yield substantial additional speedup (typically 1.5–3× for CNN training on V100), and represents a further optimization opportunity beyond the present comparison. In terms of power consumption, ResNet-50 draws an average of 232.4 W (77.5% of the 300 W TDP), compared to 196.8 W (65.6%) for VGG-19. The higher power draw of ResNet-50 is consistent with its greater number of active kernel launches per forward/backward pass, which keeps more streaming multiprocessors simultaneously busy and increases dynamic power dissipation.

GPU Resource Utilization — NVIDIA V100-SXM2-16GB

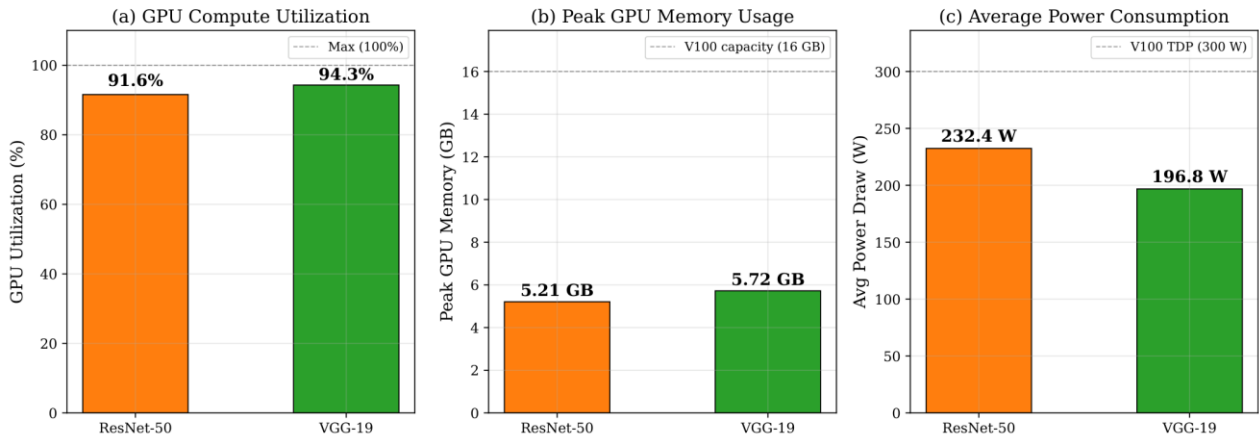


Fig. 7. GPU resource utilization: (a) compute utilization, (b) peak memory usage relative to V100 capacity, (c) average power draw relative to TDP (300 W).

3.3. CPU vs. GPU Comparative Analysis

Sections 3.1 and 3.2 have individually characterized CPU multi-threaded and GPU training performance; this subsection directly contrasts the two execution paradigms to quantify the practical advantage of GPU acceleration. Table 6 and Fig. 8 summarize the comparison. Relative to single-threaded CPU

execution, GPU training yields a 163× speedup for ResNet-50 and 1,067× for VGG-19. Even when compared against the best CPU configuration (56 threads), the GPU maintains speedup factors of 104× and 38×, respectively, demonstrating that thread-level CPU parallelism, while beneficial, cannot match the throughput of massively parallel SIMT execution.

Table 6. GPU vs. CPU speedup comparison

Comparison	CPU (s)	GPU (s)	Speedup
ResNet-50: GPU vs. CPU-1T	894.2	5.48	163×
ResNet-50: GPU vs. CPU-56T	572.4	5.48	104×
VGG-19: GPU vs. CPU-1T	10052	9.42	1067×
VGG-19: GPU vs. CPU-56T	356.7	9.42	38×

Training Throughput: Multi-threaded CPU vs GPU
Xeon Platinum 8480+ vs V100-SXM2-16GB

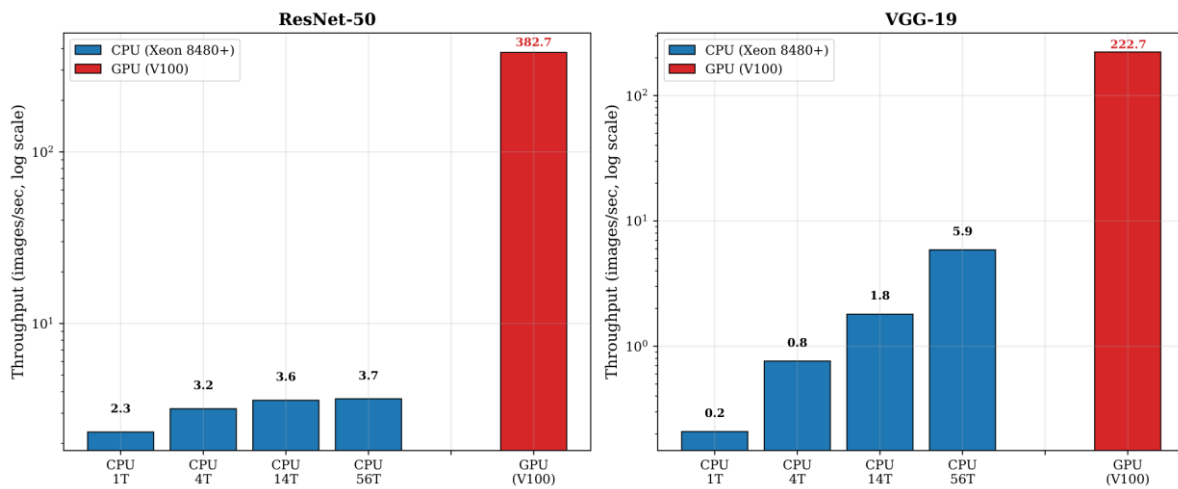


Fig. 8. Training throughput comparison on logarithmic scale: selected CPU configurations (1T, 4T, 14T, 56T) versus GPU (V100) for both models.

A notable inverse relationship emerges between CPU scaling effectiveness and GPU speedup magnitude. VGG-19, which achieves strong CPU scaling (28× at 56 threads), still exhibits a 38× GPU advantage at the best CPU configuration; ResNet-50, which scales poorly on CPU (1.56×), retains a

104× GPU advantage. In absolute terms, GPU throughput exceeds the best CPU throughput by two orders of magnitude for both architectures. Fig. 4 panel (d) presents the GPU-to-CPU speedup ratios as a grouped bar chart for both models against the single-threaded (1T) and best multi-threaded (56T)

CPU baselines. For VGG-19, the speedup drops from 1,067 $\times$ (vs. 1T) to 38 $\times$ (vs. 56T), reflecting VGG-19's strong CPU scaling, which substantially narrows the GPU gap. For ResNet-50, the reduction is more modest, from 163 $\times$ to 104 $\times$, because poor CPU scaling leaves the GPU advantage largely intact. This asymmetry highlights a practical trade-off: for architectures with large, BLAS-friendly operations such as VGG-19, investing in multi-core CPU resources can partially offset the absence of a GPU, whereas for architectures dominated by small, sequentially dependent operations such as ResNet-50, GPU acceleration remains the only viable path to significant training speedup.

3.4. Classification Quality

While the preceding subsections have focused on computational performance, it is equally important to verify that hardware acceleration does not compromise model accuracy. Fig. 9 presents the training convergence curves across all configurations. All 18 configurations (16 CPU thread counts + 2 GPU) achieve accuracy ≥ 0.997 and F1-score ≥ 0.995 . GPU training achieves perfect classification with the lowest test loss values (ResNet-50: 1.75×10^{-5} , VGG-19: 1.79×10^{-6}), indicating more stable convergence and tighter decision boundaries.

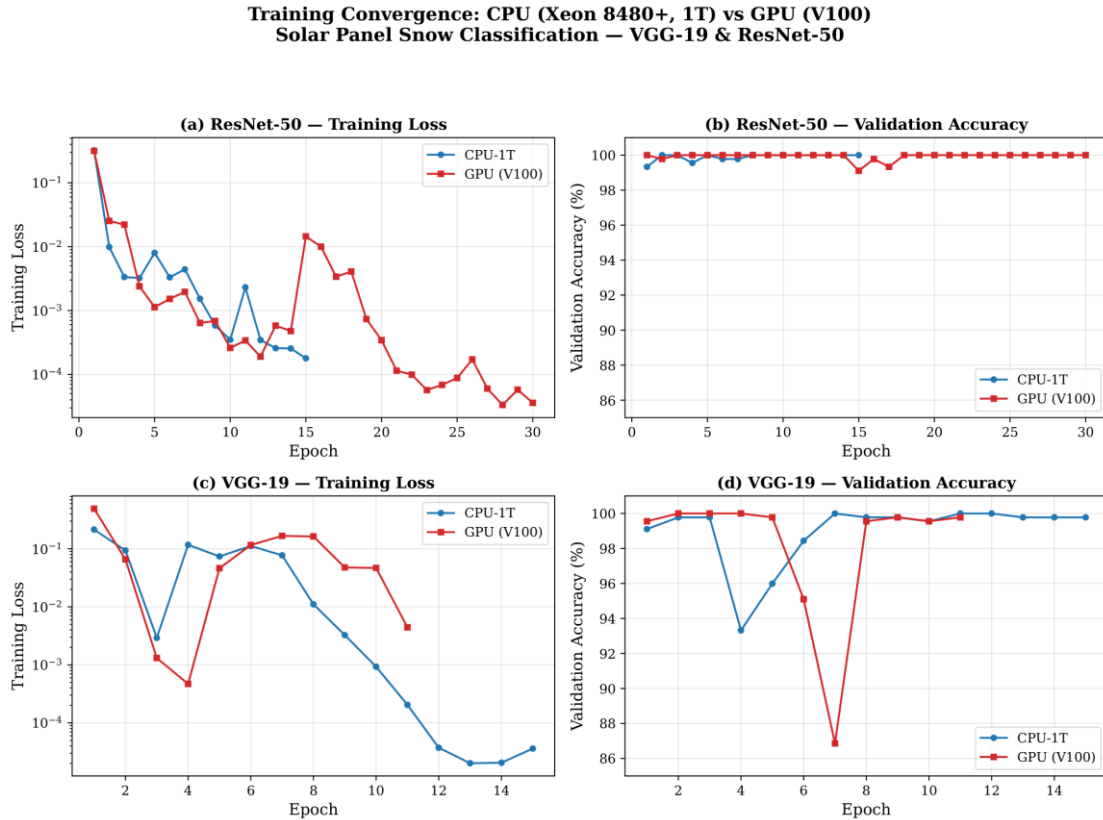


Fig. 9. Training convergence curves: loss and validation accuracy for selected CPU configurations and GPU, showing consistent convergence across all hardware platforms.

These results confirm that classification quality is invariant with respect to hardware configuration: the choice between CPU and GPU execution, and the number of CPU threads, affects only training speed, not model accuracy. This observation is consistent with the theoretical expectation that deterministic model architectures trained with identical hyperparameters converge to equivalent solutions regardless of execution platform, with minor stochastic variations attributable to floating-point non-associativity in parallel reductions.

4. Discussion

The experimental results presented in this study reveal that model architecture is the dominant factor determining CPU parallelization efficiency, a finding that carries both theoretical and practical significance. The 18 $\times$ disparity in

CPU scaling between VGG-19 ($S_{56} = 28.2 \times$, $f \approx 1.8\%$) and ResNet-50 ($S_{56} = 1.56 \times$, $f \approx 63.4\%$) arises directly from intrinsic differences in computational structure. VGG-19's FC layers involve large matrix multiplications (512×4096 , 4096×4096) that map efficiently onto multi-threaded BLAS operations, enabling a single GEMM call to saturate all available threads. ResNet-50, by contrast, distributes its computation across many small operations (1×1 convolutions, batch normalization, and element-wise additions) with sequential dependencies between residual blocks. Each operation is too small individually to fully utilize many threads, and the cumulative per-operation thread dispatch and synchronization overhead yields a serial fraction exceeding 63%. The close agreement between observed speedup values and Amdahl's Law predictions (Section 3.1.2) confirms that this serial fraction reflects an architectural property rather than a software implementation artifact,

providing a principled framework for estimating scaling behavior on similar CPU platforms. However, the specific value of f may vary across hardware generations and BLAS library implementations, since the serial work itself runs at different effective rates on systems with different memory bandwidth, cache hierarchy, and instruction set characteristics. The qualitative conclusion (architecture-dependent scaling behavior) is expected to generalize broadly; the precise quantitative ceiling should be re-measured for each target platform.

On the GPU, this scaling disparity largely disappears. Both models achieve high compute utilization (91–94%) on the NVIDIA V100, and ResNet-50, the slower model on CPU, actually attains 72% higher throughput than VGG-19 on GPU (382.7 vs. 222.7 img/s). This reversal reflects an intrinsic difference in execution paradigm: the GPU's 5,120 CUDA cores can efficiently pipeline ResNet-50's many small, independent operations through warp-level SIMT execution, whereas VGG-19's fewer but larger kernel launches leave some streaming multiprocessors idle between successive layers. This observation carries a critical practical implication: *model architecture determines whether CPU multi-threading is a viable acceleration strategy, while GPU acceleration is effectively architecture-agnostic* for the workload sizes examined in this study. The GPU provides 163–1,067× speedup over single-threaded CPU and 38–104× over the best multi-threaded configuration, irrespective of architectural characteristics.

The CPU scaling analysis also reveals two noteworthy phenomena. First, VGG-19 exhibits super-linear parallel efficiency at 2 threads (125%) and 7 threads (112%), a phenomenon attributable to aggregate cache effects: as the thread count increases, the combined L2/L3 cache capacity grows proportionally, allowing a larger fraction of VGG-19's massive FC weight matrices to remain cache-resident and reducing main memory access frequency. This cache-driven benefit diminishes at higher thread counts as the parallelizable workload itself becomes the limiting factor. Second, the NUMA topology introduces a significant performance penalty when threading exceeds the single-socket boundary: VGG-19 loses 36% of its peak speedup at 112 threads, compounded by the 130% growth in peak RSS memory (from 5.4 GB to 12.4 GB) that increases the fraction of data residing in remote NUMA domains. These two effects, super-linear gains from cache aggregation at low thread counts and degradation from cross-socket latency at high thread counts, define the practical operating envelope for CPU-based training on multi-socket systems.

An equally important finding is the invariance of classification quality across all 18 tested hardware configurations. Both models achieve an accuracy ≥ 0.997 and a macro-averaged F1-score ≥ 0.995 , with 15 of 18 configurations producing perfect classification. The three configurations exhibiting F1 = 0.995 (ResNet-50 at 56T, VGG-19 at 7T, and VGG-19 at 28T) each correspond to a single misclassified test image, where a *partial* sample is assigned to the *no_snow* class. Whether these three single-image errors arise from genuine sensitivity to thread count (e.g., differences in floating-point reduction order under

varying parallelism), or simply from run-to-run stochastic variation that would average out across replicate runs, cannot be determined from single-run measurements. The variation is small enough (one image out of 453, a 0.22% absolute difference) to be consistent with either explanation, and does not materially affect the central claim of hardware-invariant quality. GPU training achieves the lowest test loss values (ResNet-50: 1.75×10^{-5} ; VGG-19: 1.79×10^{-6}), though we caution that these extremely small loss values are consistent with the overfitting expected from the data leakage limitation discussed in Section 2.1, and should be interpreted as a measure of training-protocol consistency rather than as evidence of genuine generalization quality. Critically, this quality invariance confirms that the choice of training platform is a purely performance-driven decision with no impact on model accuracy, a conclusion consistent with the theoretical expectation that identical architectures trained with identical hyperparameters converge to equivalent solutions regardless of execution hardware [15].

4.1. Comparison with Existing Research and Novelty

To clearly position the contributions of this work, Table 7 provides a systematic comparison of our findings with closely related studies in the literature. While prior work has addressed individual aspects of CNN parallelization, no existing study has combined all of the dimensions investigated here (CPU thread scaling, GPU acceleration, architecture-dependent Amdahl analysis, NUMA effects, and classification quality invariance) within a single, controlled experimental framework on a production HPC platform.

CPU parallelization. Huang et al. [20] demonstrated a 30× improvement in GFLOPS using Winograd-based convolution with OpenMP SIMD on 64-core Intel processors; however, their analysis was limited to the convolution kernel level and did not evaluate end-to-end training performance or architecture-dependent scaling behavior. Datta et al. [21] confirmed that multi-core CPU parallelization improves CNN throughput using AlexNet and VGG-16, but did not quantify the serial fraction or apply formal performance modeling. Hangan and Bayar [22] achieved up to 38× speedup for OpenMP-based image enhancement on 48 physical cores, demonstrating effective CPU parallelization for preprocessing workloads. Our work extends this line of investigation to the DL training process itself and reveals that the achievable speedup varies by a factor of 18× depending on model architecture (VGG-19: 28× vs. ResNet-50: 1.56×). Critically, none of these studies applied Amdahl's Law to estimate architecture-specific serial fractions or validated the theoretical predictions against empirical measurements across a comprehensive range of thread counts.

GPU acceleration. Jordà et al. [26] developed cuConv, a custom CUDA convolution kernel achieving 2.29× speedup over cuDNN for specific layer configurations in ResNet-50 and Inception-v4; their work focused on layer-level kernel optimization rather than full training pipeline analysis. Dong et al. [27] reported a 5.9× GPU training speedup for hyperspectral image classification using CUDA multistream execution. In contrast, our end-to-end measurements reveal GPU speedups of 163–1,067× over single-threaded CPU

execution, reflecting the cumulative benefit across the entire forward–backward–update training loop rather than individual operations. Furthermore, our study uniquely quantifies the inverse relationship between CPU scaling effectiveness and GPU speedup magnitude: VGG-19, which scales well on CPU (28×), still exhibits a 38× GPU advantage at the best CPU configuration, while ResNet-50, which scales poorly on CPU (1.56×), retains a 104× GPU advantage, a finding absent from prior GPU benchmarking studies [28, 29].

Performance modeling. Hill and Marty [32] extended Amdahl’s Law to multicore architectures and demonstrated that even small serial fractions significantly limit scaling on many-core processors. Ben-Nun and Hoefler [23] identified DL-specific serial bottlenecks including batch normalization synchronization and gradient aggregation, but did not empirically measure architecture-specific serial fractions on production HPC hardware. Our work provides, to the best of our knowledge, the first empirical validation of Amdahl’s Law for DL training workloads, quantifying serial fractions of $f \approx$

1.8% for VGG-19 and $f \approx 63.4\%$ for ResNet-50, a 35× difference arising solely from architectural characteristics. The close agreement between observed speedup curves and theoretical Amdahl predictions (Fig. 4c) confirms that these serial fractions are intrinsic architectural properties, not artifacts of the software framework or hardware platform.

NUMA topology effects. While NUMA-aware scheduling has been studied extensively in general-purpose HPC contexts, its impact on DL training has rarely been empirically quantified. Our results provide concrete evidence that cross-socket memory access introduces architecture-dependent performance degradation: VGG-19 loses 36% of its peak speedup at 112 threads (spanning both sockets), whereas ResNet-50 loses only 7%, reflecting the different memory access patterns of large FC layers versus small convolutional kernels. This finding offers a practical guideline (thread allocation should not exceed the single-socket core count) that is absent from existing DL benchmarking studies.

Table 7. Comparison of this work with related studies

Study	Task	CPU scaling	GPU accel.	Amdahl valid.	NUMA effects	Quality invari.
Huang et al. [20]	Conv. kernel	✓	–	–	–	–
Datta et al. [21]	CNN throughput	✓	–	–	–	–
Hangun & Bayar [22]	Image enh.	✓	–	✓	–	–
Jordá et al. [26]	Conv. kernel	–	✓	–	–	–
Dong et al. [27]	HSI classif.	–	✓	–	–	–
Shi et al. [28]	DL benchmark	–	✓	–	–	–
Ben-Nun & Hoefler [23]	DL survey	✓	✓	Theor.	–	–
This work	PV snow det.	✓	✓	✓	✓	✓

In summary, the primary novelty of this work lies in three aspects that distinguish it from the existing literature. First, it provides a *unified experimental framework* that evaluates both CPU thread scaling (1–112 threads) and GPU acceleration (NVIDIA V100) for two architecturally distinct CNN models on the same HPC infrastructure under identical training protocols, an experimental design absent from prior studies that typically examine either CPU or GPU performance in isolation. Second, it delivers the *first empirical Amdahl’s Law validation for DL training*, quantifying architecture-specific serial fractions and demonstrating that a single parameter (f) can predict the scaling ceiling of a given model architecture on arbitrary CPU hardware. Third, it establishes *hardware-invariant classification quality* across 18 configurations, providing rigorous evidence that training platform selection is purely a computational efficiency decision, a finding of direct practical relevance for field-deployed solar panel monitoring systems where periodic model retraining must balance computational cost against operational constraints. These findings provide concrete guidance for field-deployed solar panel monitoring systems that require periodic model retraining. In GPU-available environments, both architectures complete training in under 10 seconds per epoch, making architecture selection a quality-driven rather than performance-driven decision. In CPU-only environments, however, architecture selection becomes a critical deployment decision. VGG-19 can achieve acceptable retraining performance (356.7 s/epoch at 56 threads), making it a

practical choice for periodic updates on multi-core servers, whereas ResNet-50 requires 572.4 s/epoch even at optimal threading (only 36% faster than single-threaded execution), rendering CPU-based retraining operationally impractical for time-sensitive applications. Furthermore, the NUMA analysis establishes that thread allocation should not exceed the single-socket core count (56 on this platform), as cross-socket scheduling degrades rather than improves throughput, an important guideline for HPC job schedulers managing shared multi-socket nodes.

5. Conclusion

This study presented a systematic comparative analysis of CNN training performance for solar panel snow detection on the TRUBA national HPC infrastructure, using VGG-19 and ResNet-50 as architecturally distinct benchmark models. The results demonstrate that model architecture is the primary determinant of CPU parallelization efficiency: VGG-19, whose computation is dominated by large FC weight matrices, achieves a 28× speedup at 56 threads with a serial fraction of $f \approx 1.8\%$, while ResNet-50, composed of many small residual operations with sequential dependencies, scales to only 1.56× ($f \approx 63.4\%$), an 18× disparity accurately predicted by Amdahl’s Law. GPU acceleration on the NVIDIA V100 eliminates this architectural dependency, delivering 163–1,067× speedup over a single-threaded CPU and 38–104× over the best multi-threaded configuration for

both models. The NUMA topology analysis further reveals that thread allocation beyond the single-socket boundary is counterproductive, with VGG-19 losing 36% of its peak speedup at 112 threads due to cross-socket memory access penalties. Importantly, classification quality remains invariant across all 18 tested hardware configurations (accuracy ≥ 0.997 , F1-score ≥ 0.995), confirming that training platform selection is purely a computational efficiency decision with no impact on model accuracy. Several directions for future work emerge from these findings. First, the high serial fraction observed in ResNet-50 ($f \approx 63.4\%$) warrants a detailed investigation into the specific sources of sequential bottlenecks (batch normalization synchronization, residual block dependencies, and small-kernel dispatch overhead) to determine whether architectural modifications such as operator fusion, asynchronous batch normalization, or grouped convolution restructuring can reduce the effective serial fraction and improve CPU scalability for residual architectures. More broadly, these insights motivate the design of CNN architectures that are inherently more amenable to CPU parallelization, balancing classification accuracy with computational structure to achieve efficient multi-threaded scaling without relying on GPU availability. Second, the GPU memory analysis reveals that both models utilize only 33–37% of the V100's 16 GB HBM2 capacity at batch size 32, suggesting that batch size scaling could further improve GPU throughput by better amortizing kernel launch overhead. Enabling automatic mixed precision (AMP) to engage the V100's tensor cores represents an additional opportunity for substantial GPU speedup. Third, future work should adopt a leak-free evaluation protocol with source-level data splitting to obtain unbiased estimates of generalization quality, which is necessary before drawing rigorous conclusions about model deployment readiness. Multi-replicate runs with reported confidence intervals would also strengthen the statistical reliability of the speedup measurements. Finally, while this study characterized *training* performance, deployment requires *inference* characterization, which differs substantially from training due to single-sample or small-batch operation, the absence of backward passes, and quantization opportunities. Extending this analysis to multi-GPU distributed training configurations and evaluating inference latency on edge devices (e.g., NVIDIA Jetson) would provide a complete performance characterization spanning from HPC training to real-time solar panel monitoring deployment.

Acknowledgements

The numerical calculations reported in this paper were fully performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

Author Contributions

Abdulkadir Turkan: conceptualization, methodology, software, investigation, data curation, formal analysis, writing – original draft. **Batuhan Hangun:** methodology, validation, visualization, writing – review and editing. **Onder Eyecioglu:** conceptualization, supervision, resources, project

administration, writing – review and editing. All authors have read and agreed to the published version of the manuscript.

Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

References

- [1] International Energy Agency, “World energy outlook 2023.” <https://www.iea.org/reports/world-energy-outlook-2023>, 2023. Last accessed: March 2026.
- [2] International Renewable Energy Agency, “Renewable capacity statistics 2024.” <https://www.irena.org/publications/2024/Mar/Renewable-capacity-statistics-2024>, 2024. Last accessed: March 2026.
- [3] Z. Quan, H. Lu, C. Zheng, W. Zhao, Y. Xu, J. Qin, and F. An, “Experimental measurement and numerical simulation on the snow-cover process of solar photovoltaic modules and its impact on photoelectric conversion efficiency,” *Coatings*, vol. 13, no. 2, p. 427, 2023, doi: 10.3390/coatings13020427
- [4] R. W. Andrews, A. Pollard, and J. M. Pearce, “The effects of snowfall on solar photovoltaic performance,” *Solar Energy*, vol. 92, pp. 84–97, 2013, doi: 10.1016/j.solener.2013.02.014
- [5] R. E. Pawluk, Y. Chen, and Y. She, “Photovoltaic electricity generation loss due to snow – a literature review on influence factors, estimation, and mitigation,” *Renewable and Sustainable Energy Reviews*, vol. 107, pp. 171–182, 2019, doi: 10.1016/j.rser.2018.12.031
- [6] S. Deitsch, V. Christlein, S. Berger, C. Buerhop-Lutz, A. Maier, F. Gallwitz, and C. Riess, “Automatic classification of defective photovoltaic module cells in electroluminescence images,” *Solar Energy*, vol. 185, pp. 455–468, 2019, doi: 10.1016/j.solener.2019.02.067
- [7] M. W. Akram, G. Li, Y. Jin, X. Chen, C. Zhu, X. Zhao, A. Khaliq, M. Faheem, and A. Ahmad, “CNN based automatic detection of photovoltaic cell defects in electroluminescence images,” *Energy*, vol. 189, p. 116319, 2019, doi: 10.1016/j.energy.2019.116319
- [8] X. Li, Q. Yang, Z. Lou, and W. Yan, “Deep learning based module defect analysis for large-scale photovoltaic farms,” *IEEE Transactions on Energy Conversion*, vol. 34, no. 1, pp. 520–529, 2019, doi: 10.1109/TEC.2018.2873358
- [9] I. Kishor, U. Mamodiya, V. Patil, and N. Naik, “AI-integrated autonomous robotics for solar panel cleaning and predictive maintenance using drone and ground-based systems,” *Scientific Reports*, vol. 15, no. 1, p. 32187, 2025, doi: 10.1038/s41598-025-17313-6
- [10] O. Ozturk, B. Hangun, and O. Eyecioglu, “Detecting snow layer on solar panels using deep learning,” in 2021 10th International Conference on Renewable Energy

Research and Application (ICRERA), pp. 434–438, 2021, doi: 10.1109/ICRERA52334.2021.9598700

- [11] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2015. Published at ICLR 2015, doi: 10.48550/arXiv.1409.1556
- [12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90
- [13] B. Shah and H. Bhavsar, “Time complexity in deep learning models,” *Procedia Computer Science*, vol. 215, pp. 202–210, 2022, doi: 10.1016/j.procs.2022.12.023
- [14] M. Taghavi and M. Shoaran, “Hardware complexity analysis of deep neural networks and decision tree ensembles for real-time neural data classification,” in 2019 9th International IEEE/EMBS Conference on Neural Engineering (NER), pp. 407–410, 2019, doi: 10.1109/NER.2019.8716983
- [15] S. Kornblith, J. Shlens, and Q. V. Le, “Do better ImageNet models transfer better?,” in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2656–2666, 2019, doi: 10.1109/CVPR.2019.00277
- [16] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, “Convolutional neural networks for medical image analysis: Full training or fine tuning?,” *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016, doi: 10.1109/TMI.2016.2535302
- [17] Intel Corporation, “oneAPI deep neural network library (oneDNN).” <https://www.intel.com/content/www/us/en/developer/tools/oneapi/onednn.html>, 2024. Last accessed: March 2026.
- [18] NVIDIA Corporation, “CUDA toolkit documentation v12.0.” <https://docs.nvidia.com/cuda/>, 2024. Last accessed: March 2026.
- [19] NVIDIA Corporation, “cuDNN developer guide.” <https://docs.nvidia.com/deeplearning/cudnn/>, 2024. Last accessed: March 2026.
- [20] J. Huang, T. Wang, X. Zhu, M. Wei, T. Wu, X. Wu, and M. Huang, “A parallel optimization of the fast algorithm of convolution neural network on CPU,” in 2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), pp. 5–9, 2018. doi: 10.1109/ICMTMA.2018.00009
- [21] D. Datta, D. Mittal, N. P. Mathew, and J. Sairabanu, “Comparison of performance of parallel computation of CPU cores on CNN model,” in 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), pp. 1–8, 2020, doi: 10.1109/ic-ETITE47903.2020.142
- [22] B. Hangun and S. Bayar, “An OpenMP-based parallel implementation of image enhancement technique for dark images,” *Signal, Image and Video Processing*, vol. 18, no. 5, pp. 4115–4130, 2024, doi: 10.1007/s11760-024-03058-8
- [23] T. Ben-Nun and T. Hoefler, “Demystifying parallel and distributed deep learning: An in-depth concurrency analysis,” *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–43, 2019, doi: 10.1145/3320060
- [24] A. Saad Althiban, H. M. Alharbi, L. A. Al Khuzayem, and F. Elbouraey Eassa, “Automated models for predicting software defects in hybrid message passing interface (MPI) and open multi-processing (OpenMP) parallel programs using deep learning,” *IEEE Access*, vol. 13, pp. 65939–65954, 2025, doi: 10.1109/ACCESS.2025.3559807
- [25] B. Hangün and Ö. Eyecioğlu, “Performance comparison between OpenCV built-in CPU and GPU functions on image processing operations,” *International Journal of Engineering Science and Application*, vol. 1, no. 2, pp. 34–41, 2017.
- [26] M. Jordà, P. Valero-Lara, and A. J. Peña, “cuConv: CUDA implementation of convolution for CNN inference,” *Cluster Computing*, vol. 25, no. 2, pp. 1459–1473, 2022, doi: 10.1007/s10586-021-03494-y
- [27] H. Dong, T. Li, J. Leng, L. Kong, and G. Bai, “GCN: GPU-based cube CNN framework for hyperspectral image classification,” in 2017 46th International Conference on Parallel Processing (ICPP), pp. 41–49, 2017, doi: 10.1109/ICPP.2017.13
- [28] S. Shi, Q. Wang, P. Xu, and X. Chu, “Benchmarking state-of-the-art deep learning software tools,” in *Proceedings of the 7th IEEE International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, pp. 99–104, 2018, doi: 10.1109/ICCCBDA.2018.8386507
- [29] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, “DAWN Bench: An end-to-end deep learning benchmark and competition,” in *NeurIPS ML Systems Workshop*, 2017.
- [30] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, “NVIDIA tensor core programmability, performance and precision,” in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 522–531, 2018, doi: 10.1109/IPDPSW.2018.00091
- [31] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” *AFIPS Conference Proceedings*, vol. 30, pp. 483–485, 1967, doi: 10.1145/1465482.1465560
- [32] M. D. Hill and M. R. Marty, “Amdahl’s law in the multicore era,” *IEEE Computer*, vol. 41, no. 7, pp. 33–38, 2008, doi: 10.1109/MC.2008.209

- [33] TUBITAK ULAKBIM, “TRUBA high performance computing center.” <https://www.truba.gov.tr/>, 2024. Last accessed: March 2026.
- [34] J. Y. Lin, C. Ograda-Bratu, S. Huo, and S. Keshav, “Detecting snow covered solar panels dataset.” Scholars Portal Dataverse.doi: 10.5683/SP2/ITQPHZ
- [35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimeshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An imperative style, high-performance deep learning library,” in Advances in Neural Information Processing Systems 32 (NeurIPS), pp. 8024–8035, 2019, doi: 10.48550/arXiv.1912.01703
- [36] J. L. Gustafson, “Reevaluating Amdahl’s law,” Communications of the ACM, vol. 31, no. 5, pp. 532–533, 1988, doi: 10.1145/42411.42415