Pareto Optimization of CNN Models via Hardware-Aware Neural Architecture Search for Drainage Crossing Classification on Resource-Limited Devices

Yuke Li*
University of California, Merced
Merced, California, USA
yli304@ucmerced.edu

Jiwon Baik*
University of California, Santa Barbara
Santa Barbara, California, USA
jiwon.baik@geog.ucsb.edu

Md Marufi Rahman
University of North Texas
Denton, Texas, USA
mdmarufirahman@my.unt.edu

Iraklis Anagnostopoulos
Southern Illinois University
Carbondale, Illinois, USA
iraklis.anagno@siu.edu

Ruopu Li
Southern Illinois University
Carbondale, Illinois, USA
ruopu.li@siu.edu

Tong Shu†
University of North Texas
Denton, Texas, USA
tong.shu@unt.edu

ABSTRACT

Embedded devices, constrained by limited memory and processors, require deep learning models to be tailored to their specifications. This research explores customized model architectures for classifying drainage crossing images. Building on the foundational ResNet-18, this paper aims to maximize prediction accuracy, reduce memory size, and minimize inference latency. Various configurations were systematically probed by leveraging hardware-aware neural architecture search, accumulating 1,717 experimental results over six benchmarking variants. The experimental data analysis, enhanced by nn-Meter, provided a comprehensive understanding of inference latency across four different predictors. Significantly, a Pareto front analysis with three objectives of accuracy, latency, and memory resulted in five non-dominated solutions. These standout models showcased efficiency while retaining accuracy, offering a compelling alternative to the conventional ResNet-18 when deployed in resource-constrained environments. The paper concludes by highlighting insights drawn from the results and suggesting avenues for future exploration.

CCS CONCEPTS

• **Computer systems organization → Multiple instruction, multiple data**.

KEYWORDS

Hardware-aware neural architecture search, Pareto optimization

*Both authors, Yuke Li and Jiwon Baik, contributed equally to this research.

†Corresponding author: Tong Shu (ORCID: 0000-0001-8617-1772), Department of Computer Science and Engineering, University of North Texas, Denton, TX 76207, USA.

This research work is done in the Smart High-performance and Ubiquitous Systems (SHU'S) lab at the University of North Texas.

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624258

ACM Reference Format:
Yuke Li, Jiwon Baik, Md Marufi Rahman, Iraklis Anagnostopoulos, Ruopu Li, and Tong Shu. 2023. Pareto Optimization of CNN Models via Hardware-Aware Neural Architecture Search for Drainage Crossing Classification on Resource-Limited Devices. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3624062.3624258

1 INTRODUCTION

In recent years, the field of feature detection out of remotely sensed data over the outdoor landscape has witnessed a surge in interest driven by applying advanced technologies to real-world scenarios. The potential to uncover and analyze geographical features holds significant implications across various domains, from environmental management to urban planning and disaster response. This emerging trend has sparked research endeavors aimed at harnessing the power of deep learning and remote sensing data to achieve accurate and efficient classification of crucial geographical attributes.

Notably, detecting hydrographic features has garnered substantial attention within this landscape. High-Resolution Digital Elevation Models (HRDEMs) have emerged as a valuable resource for capturing intricate geographical details. Accurately identifying and classifying hydrographic features, such as drainage crossings, are paramount due to their implications for water resource management, infrastructure development, and environmental assessment. A foundation of previous works has been established in this realm, where researchers have contributed to the advancement of hydrographic feature detection. These efforts underscore the significance of accurate and efficient classification methods that leverage the rich information contained within HRDEMs. While previous works [13, 31, 33] had used deep learning architectures to solve classification-related problems, to our best knowledge, no study was done on proposing optimized neural network architecture based on accuracy and inference time trade-off for this problem. With an optimized deep learning architecture, it will be convenient for real-time analysis of the desired problem. Moreover, it can also open a research window to determine the correlation of different

neural architectures or input feature combinations that affect the performance of neural networks on that specific dataset.

This paper aims to extend this line of research by presenting a comprehensive methodology for accurately detecting and classifying drainage crossings using deep learning and remotely sensed data. Our approach capitalizes on the insights offered by Residual Networks (ResNets), a class of deep neural networks known for their exceptional performance in image-related tasks [10]. Specifically, the ResNet-18 model is employed due to its balance between computational efficiency and expressive power. To make the generated convolutional neural network (CNN) efficiently run on target devices, nn-Meter [41] was introduced, a framework designed to predict inference latency on various embedded platforms with limited computational resources. This framework aids in optimizing model configurations for real-world deployment scenarios, accounting for hardware variations that can significantly impact performance. Furthermore, our research leverages the principles of Pareto front analysis, a key concept in multi-objective optimization, to explore the trade-offs between accuracy, latency, and memory usage. By identifying non-dominated solutions, this paper strives to pinpoint optimal model configurations that balance these objectives. While pursuing genuine Pareto optimal solutions remains a challenge due to the complexity of model configuration spaces, our approach provides valuable insights into the interplay of objectives within this context.

In the subsequent sections, we delve into the details of our methodology, encompassing the ResNet-18 model architecture, the neural network intelligence (NNI) framework [23] and the nn-Meter tool [22], and the Pareto Front analysis. We also present our experimental setup and results, showcasing the effectiveness of our approach in accurately detecting drainage crossings within diverse geographical regions. Overall, this paper contributes to the evolving landscape of geographical feature detection by offering a robust and comprehensive methodology that embraces deep learning, latency prediction, and multi-objective analysis. Our research aims to advance the accuracy and efficiency of hydrographic feature detection, facilitating informed decision-making in various domains that rely on accurate geographical attribute identification.

In summary, our contributions are as follows:

(1) Our study exploits ResNet-18 mode for drainage crossing detection with six input data combinations (two image channels with three different batch sizes).
(2) We use NNI to explore different model architectures with five-fold evaluations, which results in 1,717 NNI experiment results. And based on the NNI experiment results, we use nn-Meter to predict latency across 4 predictors provided.
(3) We also discuss the obtained five non-dominated solutions (which were identified with lower inference latency, lower model memory usage, and comparable inference accuracy with the original ResNet-18 model.), and discuss the potential optimizations for this study in future work.

2 BACKGROUND

2.1 Study Region and Datasets

There has been a growing interest in applying advanced technologies to real-world geographical feature detection in recent years. Previous works present a notable contribution in this field by focusing on the detection of hydrographic features using High-Resolution Digital Elevation Models (HRDEMs) [18, 38]. The study encompasses four distinct study regions spanning the continental US: the West Fork Big Blue Watershed in Nebraska, the Vermilion River Watershed in Illinois, the Maple River Watershed in North Dakota, and the Sacramento-Stone Corral Watershed in California. Both HRDEMs and Aerial Orthophotos from each of the study regions were obtained. These datasets, which exhibit varying spatial resolutions yet are standardized to 1 meter, constitute fundamental elements for their feature detection methodology. The HRDEM dataset offers elevation values, while the Aerial Orthophotos comprise values for Red, Green, Blue, and Near Infrared channels for each spatial cell. Leveraging this data, the Normalized Difference Vegetation Index (NDVI) [26] and the Normalized Difference Water Index (NDWI) [21] were calculated based on equations 1 and 2, respectively. Including these additional features becomes a central concern of their research, potentially enhancing the accuracy of hydrographic feature detection. The training data build involves object segmentation, enabling the identification of distinct features within the geographical datasets. Specifically, 6,034 training images containing drainage crossings were detected through this segmentation process. To ensure balanced data for training, an equivalent number of training images without drainage crossings were obtained through random spatial sampling. Consequently, comprehensive training data consisting of 12,068 images was constructed. The comprehensive details regarding the datasets from each study region are illustrated in Table 1.

$$NDVI = \frac{NIR - RED}{NIR + RED} \tag{1}$$

$$NDWI = \frac{GREEN - NIR}{GREEN + NIR} \tag{2}$$

2.2 CNN and Neural Architecture Search

With the wide application of deep learning techniques to scientific research [27, 28, 37], CNNs represent a groundbreaking development in computer vision. These architectures have demonstrated remarkable aptitude in image classification and object detection tasks. CNNs leverage their inherent structure to automatically extract hierarchies of features from input images [15, 17]. Through convolutional layers, these networks apply filters to capture local patterns, and subsequent layers amalgamate these features to identify more intricate structures. This intrinsic ability to capture spatial hierarchies has positioned CNNs as fundamental tools in modern computer vision applications.

Another pivotal advancement in CNN architecture materialized with the inception of Residual Networks (ResNets). ResNets address the challenges of training deep networks by introducing residual blocks. These blocks enable networks to learn residual functions, simplifying the approximation of the desired mapping. The integration of skip connections within residual blocks mitigates the vanishing gradient predicament, thereby facilitating the training of deeper networks [10, 40]. Among the spectrum of ResNet variations, ResNet-18 stands out due to its adept equilibrium between model depth and computational efficiency. Possessing 18 layers,

Locations	DEM Source	DEM resolution	True sample	False sample	Total sample	Aerial Orthophoto Source
Nebraska	Nebraska Department of Natural Resource	1m	2022	2022	4044	USGS National Agriculture Imagery Program (NAIP) (1m resolution)
Illinois	Illinois Geospatial Data Clearinghouse	0.3m	1011	1011	2022	
North Dakota	North Dakota GIS Hub Data Portal	0.61m	613	613	1226	
California	USGS	1m	2388	2388	4776	

Table 1: Data Sources and Study Regions

ResNet-18 has demonstrated noteworthy proficiency in various image classification tasks.

Popular CNN architectures, such as ResNet [10], MobileNet [12], ResNext [39] are designed by experts, which requires a vast amount of expertise in the domain. This can be a limiting factor for researchers to design an efficient neural network based on limited domain knowledge, time, and resources. To address this issue, neural architecture search (NAS) has been proposed [7]. It emerges as a paradigm shift, automating the process of architecting neural networks [11, 24, 25]. Rather than relying on human intuitions, NAS harnesses computational techniques to explore the expansive landscape of potential architectures, identifying those that exhibit optimal performance for specific tasks. The three key concerns for NAS are the search space definition (i.e., all model architectures to be possibly explored), a search algorithm (i.e., how to select model architectures, such as, gradient-based methods [36]), and performance estimation (i.e., evaluating the performance of the selected model architectures). The NNI framework provides an adaptable environment for conducting NAS, facilitating researchers and practitioners in the efficient exploration and enhancement of neural architectures [20, 41]. The convergence of the established ResNet-18 architecture with the NNI framework yields a potent synthesis of deep learning model training and automated architecture search. By subjecting ResNet-18 to NAS through NNI, a prospect emerges to unveil potential enhancements tailored to the specific task at hand. This amalgamation of proven architecture and automated architectural exploration offers a promising avenue for advancing model capabilities in our targeted domain. Furthermore, hardware-aware neural architecture search (HW-NAS) incorporate hardware resources on target devices into consideration.

3 METHODOLOGY

3.1 Baseline CNN Model: ResNet-18

In the realm of deep learning, Residual Networks (ResNets) have demonstrated their prowess in achieving SOTA performance across an array of tasks, most notably within the realm of image classification [40]. To adeptly and precisely classify locations where drainage crossings occur, leveraging the insights provided by remotely sensed images, the ResNet-18 model was deemed the most fitting cornerstone. ResNet-18, a member of the ResNet family, comprises 18 layers and maintains a relatively lightweight and lower memory footprint than its deeper counterparts, making it particularly well-suited for scenarios where computational resources are constrained.

As shown in Figure 1, the ResNet-18 model architecture is composed of an initial convolutional layer followed by four residual blocks with two convolutional layers for each block. The four residual blocks allow the network to learn residual functions with inputs.

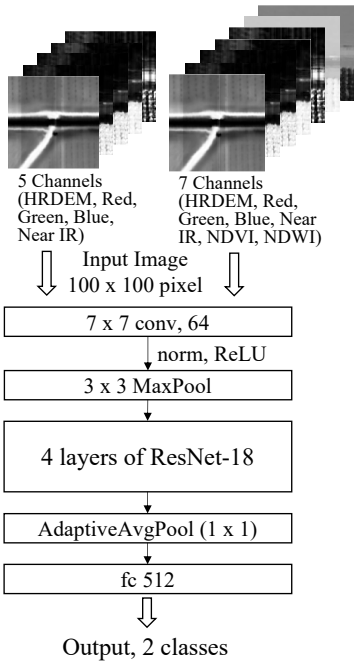

Figure 1: ResNet-18 model architecture with two types of input: data with 5 channels and 7 channels.

After the four residual blocks, there is an average pooling layer and a fully connected layer to produce a final output for categorizing the given image into relevant drainage crossing classifications.

3.2 Neural Architecture Search: NNI

The exploration of neural architecture through the neural network intelligence (NNI) [23] framework unfolds within a dynamic search space, visually depicted in Figure 2. This space is meticulously defined by the range of options available for fine-tuning the foundational model, ResNet-18, to cater to the specific challenges of classifying drainage crossings. Encompassing hyperparameters and model architecture, this search space encapsulates the essence of our endeavor.

In the realm of hyperparameters, the scope is characterized by two primary dimensions. Firstly, the batch size, a pivotal hyperparameter influencing training dynamics, is investigated across three discrete values—8, 16, and 32. Secondly, the number of input image channels assumes significance, offering a choice between 5 and 7 channels, as succinctly portrayed in Figure 1. This dual exploration underscores hyperparameters' critical role in shaping model convergence and computational efficiency.

Figure 2 reveal the architectural modeling, which consists of:

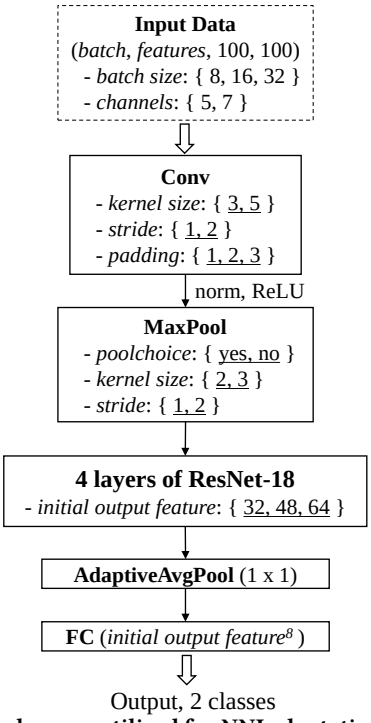

Figure 2: Search space utilized for NNI adaptations of ResNet-18, whose values are denoted by underscores. The configuration showcases combinations of input data characterized by varying batch sizes and image channels.

- **Initial convolutional layer:** the initial convolutional layer of ResNet-18 becomes a focal point for adaptation. Here, three key parameters—kernel size, stride, and padding come into play, each of which has distinct search space options with the numbers 2, 2, and 3, respectively. This orchestrated interplay of parameters empowers the model's ability to capture intricate features within the data.
- **Max Pooling layer:** as a subsequent layer, the Max Pooling layer, augments this architectural complexity. Configured immediately after the initial convolutional layer, this layer's structure presents three essential parameters for modification. Firstly, the decision to incorporate pooling or not holds significance, followed by the choice of kernel size and stride if pooling is to be included. This layer's adaptive nature speaks to its potential for enhancing feature representation within the model.
- **Four Backbone Blocks of Convolutional Layers:** A closer examination of the architecture reveals the backbone's intricacies, consisting of four convolutional layers in ResNet-18. Significantly, the choice of initial output feature size for these layers—drawn from the set 32, 48, 64—serves as the bedrock upon which the model's hierarchy is established. With each subsequent layer, there is an exponential growth in the number of filters, which dictates both the model's expressive depth and complexity. Notably, the ripple effect of these initial filter sizes extends to the terminal fully connected layer, wherein its input filter size mirrors that of the originating layer but is amplified by a factor of four. This methodically structured design underscores the depth of our research and highlights our pursuit to find an optimal balance between performance efficacy and computational efficiency.

Ultimately, the culmination of these architectural considerations funnels into the final fully connected layer, primed to render a binary output—a decisive verdict on the presence or absence of drainage crossings within the input image. This comprehensive architectural exploration within the NNI framework not only highlights the richness of NAS, but also underlines our commitment to optimizing model performance for the specific task at hand.

As an outcome, the defined search space yields 288 distinct model configurations for every combination of input data. It's worth noting that certain configurations may coincide due to the 'no pool' option for the pooling layer selection, where the kernel and stride settings are rendered irrelevant. Consequently, the NNI experiments with six input data combinations culminate in 1717 valid outcomes that are subsequently subjected to Pareto front analysis, details of which will be elaborated upon in the succeeding sections.

Given the dimensions of our task and inspired by the favorable outcomes of a CNN model in the reference paper by Wu et al. [38], it is anticipated that a streamlined architecture, less complex than the baseline ResNet-18, would effectively address our objective—an efficient and accurate ResNet-based model on the resource-limited hardware. This informed expectation propels our quest to unearth architectural configurations optimized for classification efficiency.

In deep learning model evaluation, assessing a model's performance on a dataset often involves partitioning the data into disjoint subsets for training and testing. One commonly employed technique for this purpose is k-fold cross validation. This method systematically divides the dataset into k subsets or folds. In each iteration, one fold serves as the validation set, while the remaining $k-1$ folds are used for training the model. This process is repeated k times, with each fold being utilized as the validation set once. The performance metrics obtained from each iteration are then aggregated to provide a more comprehensive understanding of the model's performance. This technique offers several advantages, including better utilization of available data, reduced variance in performance estimation, and a more robust evaluation of model generalization [9, 14, 32]. In the context of evaluation using the NNI framework, a 5-fold cross-validation approach was introduced to rigorously assess our model's performance. Specifically, the average accuracy over the 5-fold cross-validation process was calculated to derive a more reliable estimate of the model's performance. This approach enhances the reliability of evaluation by mitigating the potential impact of data variability within a single train-test split. The average accuracy obtained over multiple cross-validation folds provides a more stable and representative measure of how well the model generalizes to new, unseen data. By integrating k-fold cross validation into NNI evaluation strategy, a robust assessment of the model's performance that considers its consistency across different subsets of the dataset is strengthened. In this work, the NNI run on NVIDIA A100 GPU [2] with NNI Retiarii v2.10 [20].

3.3 Inference Latency Prediction: nn-Meter

nn-Meter emerges as a versatile and comprehensive framework designed to assess and comprehend the inference performance of

Hardware name	Device	Framework	Processor	±10% Accuracy
cortexA76cpu	Pixel4	TFLite v2.1	CortexA76 CPU	99.00%
adreno640gpu	Mi9	TFLite v2.1	Adreno 640 GPU	99.10%
adreno630gpu	Pixel3XL	TFLite v2.1	Adreno 630 GPU	99.00%
myriadvpu	Intel Movidius NCS2	OpenVINO2019R2	Myriad VPU	83.40%

Table 2: Hardware Performance Comparison of nn-Meter Predictors [22]

neural network models in embedded devices. In response to the escalating complexity of modern neural architectures and the pressing need for rigorous model analysis, nn-Meter stands as a critical solution. The nn-meter tool offers users access to four distinct latency predictors, each tailored to specific computational environments. These predictors are named as follows: cortexA76cpu, adreno640gpu, adreno630gpu, and myriadvpu. Each predictor is designed to simulate the inference latency of neural network models within a particular computational context, taking into account the unique characteristics of the hardware and software setup, which is detailed in Table 2. By encompassing a range of computational environments from mobile CPU and mobile GPU, to Intel VPU, nn-meter allows for a comprehensive assessment of how inference latency might vary across different platforms. Recognizing that inference latency can significantly differ based on the computational environment, the process of predicting and understanding latency becomes paramount. As neural network models are deployed across various devices and hardware configurations, it's imperative to account for these differences to ensure optimal performance. This is where the predictive capabilities of nn-meter's various predictors come into play. By simulating inference under distinct computational conditions, these predictors enable users to anticipate latency variations and make informed decisions when deploying models. Predicting latency using different settings allows developers and data scientists to uncover potential bottlenecks or advantages associated with specific hardware setups. To facilitate effective model configuration evaluations, a thorough and comparative analysis of inference latency is essential. Consequently, nn-meter employs all four predictors to forecast latency values in distinct computational environments. These predictors capture the intricacies of various hardware and software setups, providing a nuanced understanding of how different configurations impact inference performance. To arrive at a comprehensive evaluation metric, the average latency value is derived from the predictions generated by the four predictors. This approach ensures that the model configurations are assessed comprehensively across a spectrum of potential computational scenarios, thereby enabling more informed decisions when selecting the optimal configuration for deployment. In this work, the nn-Meter run on NVIDIA RTX 2970 super GPU [3] with Microsoft nn-Meter v2.0 [22].

3.4 Pareto Front Analysis

Pareto front analysis, rooted in the Pareto optimality principle, is a pivotal concept in multi-objective optimization. In scenarios with multiple conflicting objectives, it focuses on identifying a set of solutions that strike optimal trade-offs. These solutions form the Pareto front—a collection of non-dominated points where no solution is superior in all objectives without deteriorating in others. By showcasing the trade-off spectrum, decision-makers can navigate complex decision spaces, aiding them in making informed choices that align with their priorities and preferences [6]. This concept finds application across diverse domains, offering insights into the intricate balance of competing criteria [5].

- **Non-dominated solutions:** These solutions lie on the Pareto front and present optimal compromises between conflicting objectives, showcasing a balance where improving one objective inevitably leads to a deterioration in others. Identifying non-dominated solutions allows decision-makers to explore trade-offs and select configurations that best align with their desired outcomes.
- **Dominated solutions:** The solutions are inferior to at least one other in all specified objectives. They are excluded from the Pareto front because they can be improved without drawbacks in other objectives. Dominated solutions provide a benchmark against which non-dominated solutions are compared, allowing researchers to assess the performance improvements achieved by the Pareto optimal solutions.

In diverse domains, the application of this concept provides valuable insights into the delicate balance between competing criteria. This paper employs Pareto front analysis to emphasize three key objectives in model configurations, quantified by NNI [20, 23] and nn-Meter [22, 41]: maximizing prediction accuracy (in percents), minimizing inference latency (in milliseconds), and minimizing memory consumption (in megabytes). The goal is identifying the optimal model configuration that excels across these dimensions.

Considering prediction accuracy, inference latency, and memory usage is crucial when designing neural network models, especially for resource-constrained environments. These three factors represent fundamental aspects of model accuracy and efficiency that directly impact the practicality and usability of the model in real-world applications. Accuracy is a fundamental metric that measures the model's ability to correctly classify input data. In many cases, accuracy is a machine learning model's primary goal, as it determines its reliability and effectiveness. However, achieving high accuracy can come at the cost of increased model complexity, leading to higher memory usage and longer inference times. Balancing accuracy with other factors ensures that the model is not overly complex and remains efficient, even in scenarios where computational resources are limited.

Inference latency refers to the time it takes for a model to process an input and produce an output. In real-time applications, such as autonomous vehicles, robotics, or mobile devices, low inference latency is crucial to ensure timely responses. High inference latency can lead to delays, making the model less practical for time-sensitive tasks. Optimizing inference latency is especially important for applications requiring quick decisions, such as object detection in real-time video feeds. Memory usage pertains to the amount of memory required to store and execute the model. Models with high memory requirements may not be deployable on devices with

limited memory capacity, such as edge or IoT (Internet of things) devices. Efficient memory usage is particularly important in scenarios where hardware resources are constrained, as it enables the model to run smoothly without causing memory-related issues.

Through Pareto front search, non-dominated solutions were identified among the given model configurations. However, it's important to note that while these solutions are non-dominated, they might not guarantee true Pareto optimal outcomes in a general concept. The introduction of additional model configuration options could potentially reveal superior solutions. Notably, as model configuration involves integer parameters, achieving a continuous Pareto front is unfeasible [4, 29, 30]. Thus, pursuing genuine Pareto optimal solutions would require exploring all feasible model configurations, a task not addressed in this study. Nonetheless, our research successfully extracted non-dominated solutions from our experimentation efforts. After identifying the three-dimensional Pareto optimal solutions encompassing our objectives, we further explored model configurations, considering factors such as the number of channels, batch size, and specific model settings. These findings shed light on the intricate interplay of multiple objectives within the context of our study.

4 EXPERIMENTS

We tested six variants of the ResNet-18 benchmark with parameter combinations of two distinct input channels and three different batch sizes. This extensive exploration resulted in a total of 1,717 validate unique model configurations, each with its respective NNI outcome. The three-dimensional representation of all the Pareto front analysis results, visualizing the three objectives, can be seen in Figure 3. As shown in Table 3, among the 1717 model outcomes, the inference accuracy ranged from 76.19% to 96.13%. The inference latency spanned a range from 8.13 ms to 249.56 ms, while the model memory usage fluctuated between 11.18 MB and 44.69 MB according to the model complexity. From our Pareto front analysis, out of the 1,717 model configurations, we obtained five non-dominated solutions. These are prominently marked as red dots in Figure 3. To emphasize the connections among the non-dominated solutions, they are normalized within their respective ranges.

	Inference Accuracy	Inference Latency	Memory Usage
Min	76.19 %	8.13 ms	11.18 MB
Max	96.13 %	249.56 ms	44.69 MB

Table 3: The objective value ranges

In the analysis of the NNI results combined with the Pareto front, we identified five non-dominated solutions that met three distinct objectives: inference accuracy, inference latency, and model memory. These solutions are comprehensively detailed in Table 4, in which column names are explained as follows:

- 'channels': the input channels of the images.
- 'batch': the batch size for training and inference.
- 'accuracy': the average accuracy out of 5-fold cross-validation during NNI search.
- 'latency': provides the average inference latency for each model across four predictors from nn-Meter.
- 'lat_std': the standard deviation of the inference latency across four predictors from nn-Meter.

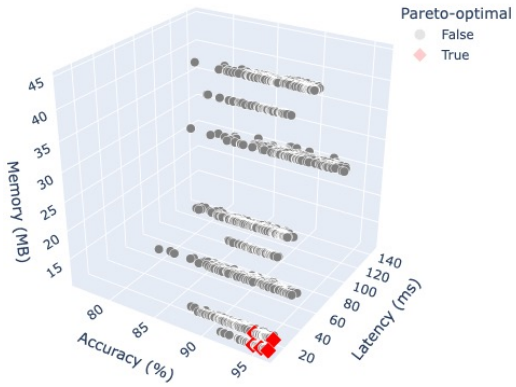

Figure 3: Pareto front analysis result

- 'memory': the memory requirement to store the model in the onnx file format.

Additionally, the rest of the various parameters, namely 'stride', 'padding', 'pool_choice', 'initial_output_feature', 'kernel_size', 'stride_pool', and 'kernel_size_pool', form the backbone of the search space for NNI, which is introduced in Figure 2.

In the review of model configurations, those with the smallest initial output feature, kernel size, and padding values were chiefly responsible for the least model memory footprint, registering at 11.18 MB. Two standout non-dominated results emerged: one showcasing the shortest inference latency at 8.13 ms, and the other pointing out the highest inference accuracy at 96.13%. Beyond these, three additional non-dominated results neither achieved the peak accuracy nor the minimum latency.

Upon comparing these solutions with the findings of the reference study [38] whose accuracy ranges from 95.92% to 97.43%, a significant observation arises: despite the reduction in training epochs by half, from 10 to 5, in response to time limitations imposed by the multiple NNI testing, the accuracy remains on par.

Also, we compare our identified non-dominated models with the conventional ResNet-18 model. We evaluated these models in terms of prediction accuracy, inference latency, latency standard deviation, and memory usage for six benchmark variants with two distinct input image channels and three batch sizes. In the NNI experiments, we train each model for five epochs and evaluate them with 5-fold cross validation. The results are shown in Table 5.

Upon comparison, it became evident that all our non-dominated models surpassed the general ResNet-18 in several aspects: they boasted lower inference latencies, displayed more consistent latency performance across varied hardware or predictors, and required less memory for model storage. Notably, while making these improvements, we ensured that the inference accuracy remained on par with, if not better than, the general ResNet-18 model.

Moreover, while the five non-dominated results exhibit varied model configurations, it is shown that they have some commonalities. For instance, in Figure 4, all these non-dominated models consistently utilize the smallest kernel size, the least number of channels, larger stride, and minimal padding size. These common traits provide valuable insights into model design tailored for resource-limited hardware on image classification applications.

channels	batch	accuracy	latency (ms)	lat_std	memory (MB)	kernel_size	stride	padding	pool_choice	kernel_size_pool	stride_pool	initial_output_feature
7	16	96.13	8.19	4.59	11.18	3	2	1	0	3	2	32
5	16	95.45	8.23	4.66	11.18	3	2	1	0	2	2	32
7	8	95.79	18.3	16.02	11.18	3	2	1	1	3	2	32
5	8	94.68	8.13	4.53	11.18	3	2	1	0	3	2	32
5	8	93.97	18.24	15.96	11.18	3	2	1	1	3	1	32

Table 4: The Pareto optimal solutions for three objectives: the inference accuracy, the inference latency, and the model memory.

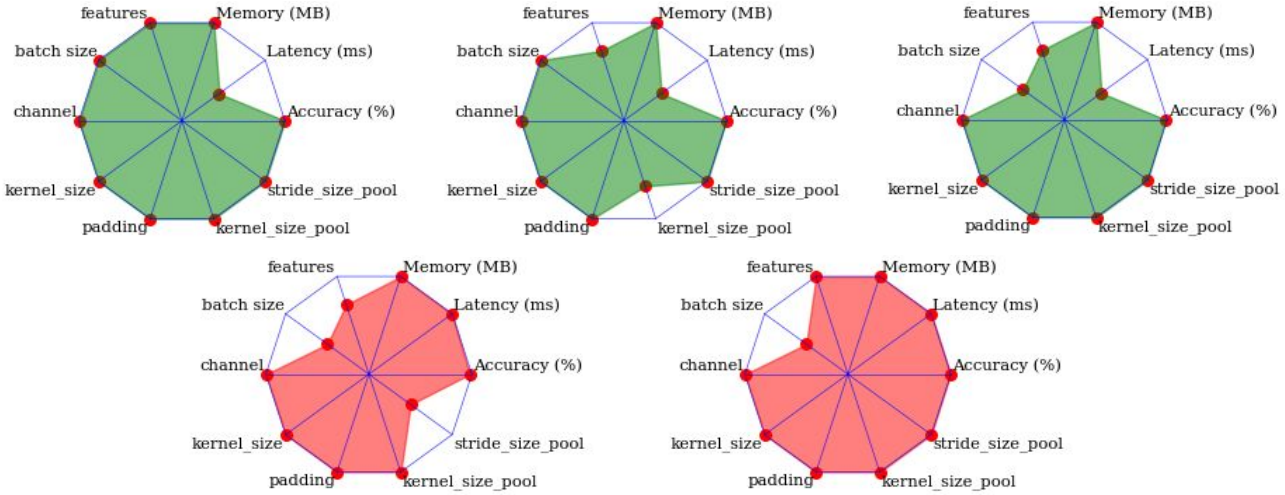

Figure 4: Radar Plots Revealing 5 Non-Dominated Solutions: Mapping Model Configurations and Objectives. (Red circles represent configurations without Pooling layers, while green circles indicate configurations with Pooling layers.)

channels	batch	accuracy	latency (ms)	lat_std	memory (MB)
5	8	92.9	31.91	20.36	44.71
5	16	93.6	31.91	20.36	44.71
5	32	89.67	31.91	20.36	44.71
7	8	94.76	32.46	20.96	44.73
7	16	95.37	32.46	20.96	44.73
7	32	94.51	32.46	20.96	44.73

Table 5: Evaluation on six ResNet-18 benchmark variants.

5 DISCUSSION

Through our NNI experiments and subsequent Pareto front analysis, we discerned two significant observations concerning the experimental design:

(1) **Duration of the NNI Experiments:** The NNI experiments tend to be time-consuming. As an illustration, the experiment with 5 input channels and batch size 8 spanned 9 hours and 20 minutes, while the one for 7 input channels and the same batch size lasted an extensive 29 hours and 3 minutes.
(2) **Potential for Search Space Optimization:** Insights from the Pareto front analysis and the non-dominated results suggest potential streamlining opportunities in the search space. For instance, confining the padding size to 1 can effectively curtail the combination permutations in the search space.

The presented findings suggest several potential pathways for refining NNI experiments. One promising strategy involves using the NVIDIA Nsight system [1] to profile NNI experiments' resource utilization and computational load. Such data would be invaluable for adjusting various experimental configurations, from the number of trials to the nuances of the search space, ultimately paving the way for a more resource-efficient NAS for the development of critical and streamlined neural network models [34, 42]. Additionally, improving the capacity of hardware-aware NAS through utilizing parallel execution on multi-GPU platforms [8, 16], along with leveraging advanced network interconnect clusters [19, 35], could ensure enhanced efficiency in model experimentation and design.

6 CONCLUSION

In conclusion, this paper addresses a crucial challenge in the realm of deep learning: the optimization of neural network models for efficient image classification on computational resources-limited hardware. Through a comprehensive exploration using techniques like NNI and nn-Meter, this paper has delved into the intricate interplay between model architecture, accuracy, inference latency, and memory usage. The results unveiled a series of non-dominated solutions that showcase a delicate balance of these objectives, offering a tangible step forward in the quest for efficient yet accurate models.

Pareto front analysis has provided a deeper understanding of the trade-offs and synergies between accuracy and efficiency. These non-dominated solutions, consistently surpassing the conventional ResNet-18 model in multiple dimensions without compromising accuracy, present a promising avenue for efficient image classification. Their shared traits, such as small kernel sizes, minimal padding, and optimized channel counts, shed light on design principles tailored for resource-constrained environments.

In a world increasingly reliant on edge computing and mobile devices, the findings of this study hold significance beyond the realm of academia. This paper offers practical insights for deploying neural network models in real-world scenarios with limitations on computational resources, fostering a bridge between cutting-edge research and practical implementation. Through meticulous experimentation, rigorous analysis, and thoughtful interpretation, this paper contributes to the ongoing journey of making deep learning more accessible and effective for a broader range of applications.

ACKNOWLEDGMENTS

This research is sponsored by National Science Foundation under Grant No. OAC-2306184 with the University of North Texas and Grant No. BCS-1951741 with Southern Illinois University. We appreciate the computing resources from PADSYS.

REFERENCES

[1] 2018. https://developer.nvidia.com/nsight-systems
[2] 2023. https://www.nvidia.com/en-us/data-center/a100/
[3] 2023. https://www.nvidia.com/content/geforce-gtx/GEFORCE_RTX_2070_SUPER_User_Guide.pdf
[4] Jiwon Baik and Alan T Murray. 2022. Locating a Facility to Simultaneously Address Access and Coverage Goals. *Papers in Regional Science* 101, 5 (2022), 1199–1217.
[5] Jared L Cohon. 2004. *Multiobjective Programming and Planning*. Vol. 140. Courier Corporation.
[6] Jared L Cohon, Richard L Church, and Daniel P Sheer. 1979. Generating Multiobjective Trade-offs: An Algorithm for Bicriterion Problems. *Water Resources Research* 15, 5 (1979), 1001–1010.
[7] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.
[8] Yuan Feng and Hyeran Jeon. 2023. Understanding Scalability of Multi-GPU Systems. In *Proc. of the 15th Workshop on General Purpose Processing Using GPU (GPGPU)*. Montreal, Canada, 36–37.
[9] Seymour Geisser. 1975. The Predictive Sample Reuse Method with Applications. *Journal of the American statistical Association* 70, 350 (1975), 320–328.
[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 770–778.
[11] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems* 212 (2021), 106622.
[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861* (2017).
[13] Umair Iqbal, Johan Barthelemy, and Pascal Perez. 2022. Prediction of Hydraulic Blockage at Culverts from a Single Image Using Deep Learning. *Neural Computing and Applications* 34, 23 (2022), 21101–21117.
[14] Ron Kohavi et al. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proc. of Intl. Joint Conf. on Artificial Intelligence (IJCAI, Vol. 14)*. Montreal, Canada, 1137–1145.
[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems* 25 (2012).
[16] Turja Kundu and Tong Shu. 2023. HIOS: Hierarchical Inter-Operator Scheduler for Real-Time Inference of DAG-Structured Deep Learning Models on Multiple GPUs. In *Proc. of IEEE International Conference on Cluster Computing (Cluster)*. Santa Fe, NM, USA, 12 pages.
[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 7553 (2015), 436–444.
[18] Ruopu Li, Zhenghong Tang, Xu Li, and Jessie Winter. 2013. Drainage Structure Datasets and Effects on LiDAR-Derived Surface Flow Modeling. *ISPRS Intl. Journal of Geo-Information* 2, 4 (2013), 1136–1152.
[19] Yuke Li, Hao Qi, Gang Lu, Feng Jin, Yanfei Guo, and Xiaoyi Lu. 2022. Understanding Hot Interconnects with an Extensive Benchmark Survey. *BenchCouncil Trans. on Benchmarks, Standards and Evaluations* 2, 3 (2022), 100074.
[20] Lingxiao Ma, Zhiqiang Xie, Zhi Yang, Jilong Xue, Youshan Miao, Wei Cui, Wenxiang Hu, Fan Yang, Lintao Zhang, and Lidong Zhou. 2020. Rammer: Enabling Holistic Deep Learning Compiler Optimizations with rTasks. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Virtual, 881–897.
[21] Stuart K McFeeters. 1996. The Use of the Normalized Difference Water Index (NDWI) in the Delineation of Open Water Features. *Intl. Journal of Remote Sensing* 17, 7 (1996), 1425–1432.
[22] Microsoft. 2023. nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices. https://github.com/microsoft/nn-Meter.
[23] Microsoft. 2023. NNI: An Open Source AutoML Toolkit for Neural Architecture Search, Model Compression and Hyper-Parameter Tuning. https://github.com/microsoft/nni.
[24] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *Proc. of AAAI Conf. on Artificial Intelligence*, Vol. 33. 4780–4789.
[25] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–34.
[26] John Wilson Rouse, Rüdiger H Haas, John A Schell, and Donald W Deering. 1974. Monitoring Vegetation Systems in the Great Plains with ERTS. *NASA Spec. Publ* 351, 1 (1974), 309.
[27] Tong Shu, Yanfei Guo, Justin Wozniak, Xiaoning Ding, Ian Foster, and Tahsin Kurc. 2021. Bootstrapping In-Situ Workflow Auto-tuning via Combining Performance Models of Component Applications. In *Proc. of ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*. St. Louis, MO, USA, 1–15.
[28] Tong Shu, Yanfei Guo, Justin Wozniak, Xiaoning Ding, Ian Foster, and Tahsin Kurc. 2021. POSTER: In-Situ Workflow Auto-tuning through Combining Component Models. In *Proc. of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. Seoul, South Korea, 467–468.
[29] Rajendra Solanki. 1991. Generating the Noninferior Set in Mixed Integer Biobjective Linear Programs: An Application to a Location Problem. *Computers & Operations Research* 18, 1 (1991), 1–15.
[30] Nidamarthi Srinivas and Kalyanmoy Deb. 1994. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation* 2, 3 (1994), 221–248.
[31] Larry Stanislawski, Tyler Brockmeyer, and E Shavers. 2018. Automated Road Breaching to Enhance Extraction of Natural Drainage Networks from Elevation Models through Deep Learning. *The Intl. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 42 (2018), 597–601.
[32] Mervyn Stone. 1974. Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society: Series B (Methodological)* 36, 2 (1974), 111–133.
[33] Sameerah Talafha, Di Wu, Banafsheh Rekabdar, Ruopu Li, and Guangxing Wang. 2021. Classification and Feature Extraction for Hydraulic Structures Data Using Advanced CNN Architectures. In *Intl. Conf. on Transdisciplinary AI (TransAI)*. 137–146.
[34] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. 2020. Leaky DNN: Stealing Deep-Learning Model Secret with GPU Context-Switching Side-Channel. In *Proc. of IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN)*. 125–137.
[35] Adam Weingram, Yuke Li, Hao Qi, Darren Ng, Liuyao Dai, and Xiaoyi Lu. 2023. xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning. *Journal of Computer Science and Technology* 38, 1 (2023), 166–195.
[36] Yu Weng, Tianbao Zhou, Yujie Li, and Xiaoyu Qiu. 2019. NAS-Unet: Neural Architecture Search for Medical Image Segmentation. *IEEE Access* 7 (2019), 44247–44257.
[37] Justin M. Wozniak, Philip Davis, Tong Shu, Jonathan Ozik, Nicholas Collier, Ian Foster, Thomas Brettin, and Rick Stevens. 2018. Scaling Deep Learning for Cancer with Advanced Workflow Storage Integration. In *Proc. of the 4th Workshop on Machine Learning in HPC Environments (MLHPC) in conjunction with ACM/IEEE SC*. Dallas, TX, USA, 114–123.
[38] Di Wu, Ruopu Li, Banafsheh Rekabdar, Claire Talbert, Michael Edidem, and Guangxing Wang. 2023. Classification of Drainage Crossings on High-Resolution Digital Elevation Models: A Deep Learning Approach. *GIScience & Remote Sensing* 60, 1 (2023), 2230706.
[39] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated Residual Transformations for Deep Neural Networks. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 1492–1500.
[40] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide Residual Networks. *arXiv preprint arXiv:1605.07146* (2016).
[41] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices. In *Proc. of Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys)*. 81–93.
[42] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. 2021. Stealing Neural Network Structure Through Remote FPGA Side-Channel Analysis. *IEEE Trans. on Information Forensics and Security* 16 (2021), 4377–4388.

A SUMMARY OF EXPERIMENTS REPORTED

A.1 Introduction

This document contains the experiment description, the hardware configuration, and the software setup in this paper. In the first part, we will describe how the experiments are organized. In the second part, we will present the source of the datasets, the software versions, the hardware we used for the experiments, and the processes to reproduce the results shown in the paper. An overview of the process of the project are described in the following:

(1) Data downloading and data processing.
(2) Neural Architecture Search experiments running.
(3) nn-Meter experiments running based on the NAS results.
(4) nn-Meter results gathering, Pareto-front analysis, and figure drawing.

A.2 Figures Usage

The input image figures in Figure 1 can be drawn by Artifact 8.
We provide an interactive version of the Figure 3 with the link https://jiwonbaik96.github.io/dlgpu/pareto. The figure can be generated by running the Artifact 7.
Figure 4 can be generated by running the Artifact 7.

A.3 Platform Usage

The NNI experiments are running on the local setup with:

- NVIDIA A100 PCIe 40GB
- Intel(R) Xeon(R) Gold 6330 CPU 2.00GHz
- AlmaLinux 8.5
- Python 3.9.17
- Pytorch 2.0.1 + cu117
- NNI Retiarii v2.10

The nn-Meter and Pareto front analysis are running on the local setup with:

- NVIDIA GeForce RTX 2070 super
- AMD Ryzen™ 5 3600
- Windows 11
- Python 3.10.9
- Pytorch 2.0.1 + cu117
- Microsoft nn-Meter v2.0

A.4 Experiments Usage

The experiments in this project are divided into three parts: dataset, searching, and prediction. Their corresponding artifacts are explained below:

- `Dataset`: the datasets are described in Section 2.1. Using Artifacts 1, 2, 3, and 4 to download and generate the corresponding data files for training and testing.
- `Searching`: searching includes the implementation of the baseline model introduced in Section 3.1 and the NNI described in Section 3.2. Using Artifact 5 to run the search experiment.
- `Prediction`: prediction contains the latency prediction by nn-Meter introduced in Section 3.3 and Pareto Front Analysis described in Section 3.4 in this project. Using Artifact 6 and 7, respectively.

B CREATED OR MODIFIED ARTIFACTS

B.1 Artifact 1

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/ClippedSample_4Areas.zip
Artifact name: ClippedSample_4Areas.zip

B.2 Artifact 2

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/DataRead.py
Artifact name: DataRead.py

B.3 Artifact 3

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/VICalculation.py
Artifact name: VICalculation.py

B.4 Artifact 4

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/channels.ipynb
Artifact name: step0.merge_data.ipynb

B.5 Artifact 5

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/step1.NNI.ipynb
Artifact name: step1.NNI.ipynb

B.6 Artifact 6

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/step2.nn-Meter.ipynb
Artifact name: step2.nn-Meter.ipynb

B.7 Artifact 7

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/step3.pareto_front.ipynb
Artifact name: step3.pareto_front.ipynb

B.8 Artifact 8

Persistent ID: https://github.com/SHUs-Lab/SHDA23YL/blob/main/channels.ipynb
Artifact name: channels.ipynb