Accuracy-Constrained Efficiency Optimization and GPU Profiling of CNN Inference for Detecting Drainage Crossing Locations

Yicheng Zhang
University of California Riverside
Riverside, California, USA
yzhan846@ucr.edu

Dhroov Pandey
University of North Texas
Denton, Texas, USA
dhroovpandey@my.unt.edu

Di Wu
Southern Illinois University
Carbondale, Illinois, USA
di.wu@siu.edu

Turja Kundu
University of North Texas
Denton, Texas, USA
turjakundu@my.unt.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

The accurate and efficient determination of hydrologic connectivity has garnered significant attention from both academic and industrial sectors due to its critical implications for environmental management. While recent studies have leveraged the spatial characteristics of hydrologic features, the use of elevation models for identifying drainage paths can be influenced by flow barriers. To address these challenges, our focus in this study is on detecting drainage crossings through the application of advanced convolutional neural networks (CNNs). In pursuit of this goal, we use neural architecture search to automatically explore CNN models for identifying drainage crossings. Our approach not only attains high accuracy (over 97% for average precision) in object detection but also excels in efficiently inferring correct drainage crossings within a remarkably short time frame (0.268 ms). Furthermore, we perform a detailed profiling of our approach on GPU systems to analyze performance bottlenecks.

CCS CONCEPTS

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

KEYWORDS

CNN inference throughput, GPU profiling, SPP-Net

ACM Reference Format:
Yicheng Zhang, Dhroov Pandey, Di Wu, Turja Kundu, Ruopu Li, and Tong Shu. 2023. Accuracy-Constrained Efficiency Optimization and GPU Profiling of CNN Inference for Detecting Drainage Crossing Locations. 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/10.1145/3624062.3624260

*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/10.1145/3624062.3624260

1 INTRODUCTION

Hydrologic connectivity plays a vital factor in environmental management such as nutrient transport, watershed-scale water quality, and eco-protection. One of the commonly used tools for spatially characterizing hydrologic connectivity is digital elevation models (DEMs). Despite its widespread usage, studies have indicated a limitation in using elevation models to delineate drainage features, mainly due to the influence of flow barriers such as bridges, roads, and culverts (Figure 1). This influence leads to inaccuracies in simulated drainage lines, including incorrect flow directions and premature termination near these structures [27].

While incorporating drainage crossing locations into elevation models has been proven to enhance the accuracy of modeled drainage networks, researchers frequently encounter challenges related to inferior data quality and inaccessibility of data quality and access [1]. However, recent progress in deep learning techniques for fast and accurate detection of terrain features [14, 41] provides a unique opportunity to detect the locations of drainage crossing. Indeed, while deep learning techniques are widely applied to scientific research [23, 24, 33], the convolutional neural network (CNN) has shown its ability to fuel progress in automatic feature selection, extraction, and generalization [21]. However, as CNN models continue to grow in depth, the parameters and inference time also increase. This can make it challenging for researchers to achieve both fast and accurate detection with high efficiency.

In this study, we propose a novel approach to designing a set of optimized CNN models for detecting drainage crossing locations. To enhance the robustness of the network and improve the detection accuracy and efficiency, SPP-Net, a network structure with a spatial pyramid pooling (SPP) strategy, has been applied to our CNN models because of its superior multi-level feature extraction [30]. The neural network intelligence (NNI) toolkit [19] was utilized to perform neural architecture search (NAS) on these network structures and hyperparameters, and the inference latency of each multi-branch model was minimized by the inter-operator scheduler (IOS) [3]. Then, an accuracy-constrained inference efficiency optimization was constructed to identify the most efficient model within a set of relatively accurate models for a large volume of inferences. Finally, a further analysis of the hardware utilization of

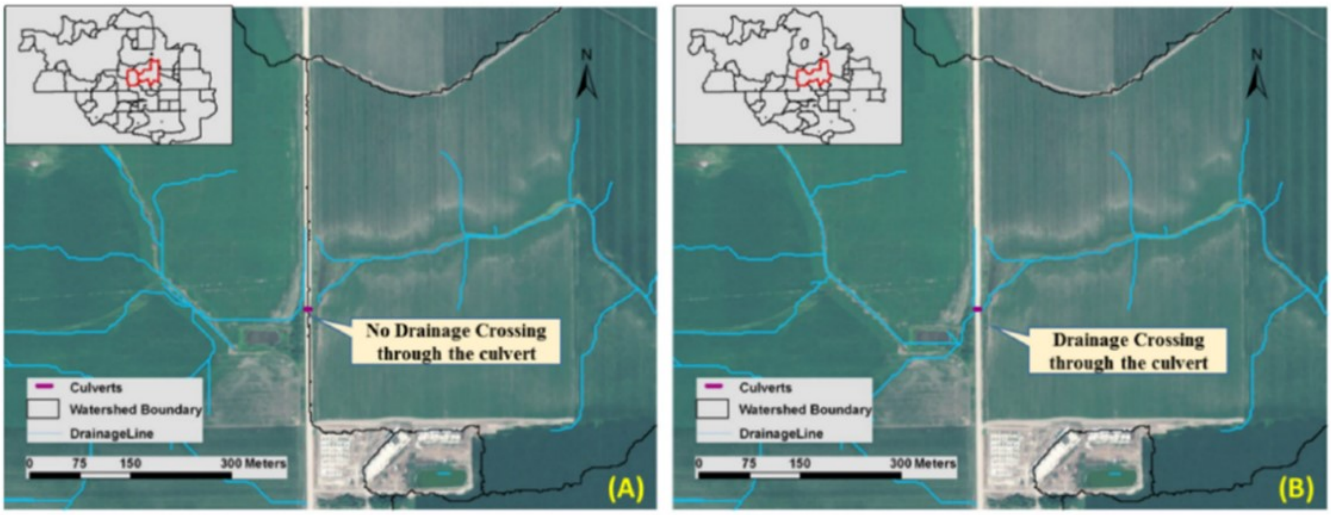

Figure 1: (A) Did not Incorporate Culvert Information, and (B) Incorporated Culvert Information in Modeling [13].

the most efficient model was implemented. The main contributions of this work are as follows.

- We achieve the accurate detection of drainage crossing by SPP-Net based CNN models, which proves that SPP-Net is a promising tool to improve simulated hydrologic connectivity.
- We show that NNI provides the power to hyperparameter optimization and neural architecture search for deep learning, which makes the turning workflow organized by aggregating and comparing tuning results.
- We use IOS to optimize the hardware utilization of CNN inference and find the CNN model with the best inference efficiency under the prediction accuracy constraint.
- We provide a profiling-based performance analysis for the deep learning inference and identify performance bottlenecks.

2 MOTIVATION AND BACKGROUND

2.1 Digital Dams and Drainage Crossing

Traditional methods for delineating hydrologic flowlines, such as digitizing surface water information from aerial photography and paper maps, are costly and inefficient. As an efficient alternative, DEM-based hydrologic delineation has been widely adopted to develop stream features. However, 'digital dams' have appeared to be prominent in such a kind of elevation-based models, which seriously affects the accurate delineation of stream flowlines. Digital dam is the obstruction caused by introduced elevation data of artificial embankments (e.g., bridges, roads, and culverts) and unexpected depressions (e.g., artificial depressions associated with interpolation processing), where DEM-derived flow routing is segmented or misled because of unidentified embankment underpasses and anomalous terrain characteristics [16]. It has been proved that combining with drainage crossing locations could reduce the adverse effects of flow barriers and improve the spatial connectivity of HRDEM-derived hydrologic features [13]. However, the existing drainage crossing dataset is either unavailable or exhibits an inconsistent format and low quality.

Thus, developing high-quality drainage crossing location datasets and modeling methods is essential to improve the accuracy of elevation-derived hydrologic features and their connectivity. The traditional methods to remove these drainage structures and determine breached channels include manual digitization [22] and field collection [29], which are often inefficient and costly, especially for large study areas. Some other existing approaches, such as the selective drainage method, were launched to overcome these issues, using a breaching algorithm to cut through the flow obstructions and connect drainage network [16]. The shortcoming of this geomorphic approach is that it relies on the manual selection of those depressions caused by drainage structures underneath where water is flowing. This selection is mostly based on knowledge-based approaches, which depend on the source and scale of data and the specificity of landscapes [20]. Sometimes, the selection may be arbitrary.

The discussion above indicates there are still challenges to obtaining reliable drainage crossing location datasets on large scale. Since the ability to produce results that are comparable to human expert performance in fields such as image classification and computer vision [9], CNN models have been regarded as potential tools to solve the digital dam problem. Existent studies also support that CNN techniques can be utilized to facilitate the identification of drainage crossing locations. Iqbal et al. [7] automated the process of manual visual blockage classification of culverts by applying CNN models. Wu et al. [34] developed different CNN models for classifying the images that contain the locations of drainage crossing. As the sizes of the drainage structures in the original dataset are inconsistent, SPP-net, a network structure with a spatial pyramid pooling strategy, has been applied to our CNN models to generate a fixed-length representation regardless of image size and scale [6]. Furthermore, SPP-net could extract characteristic graphs by

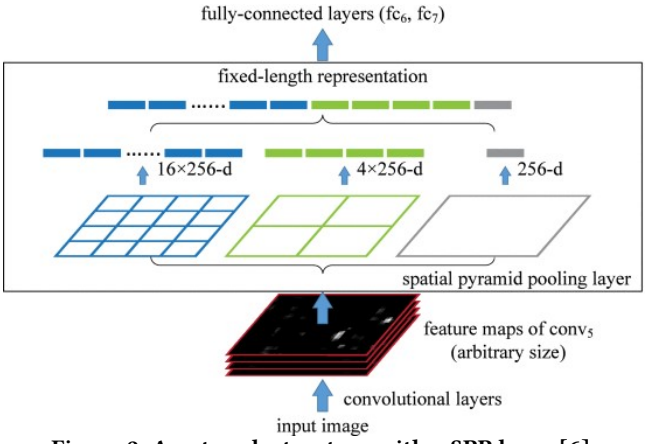

Figure 2: A network structure with a SPP layer [6].

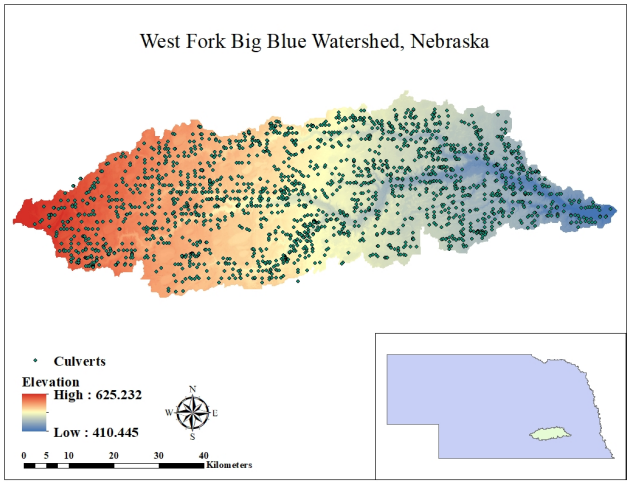

Figure 3: Topography and Locations of West Fork Big Blue Watershed, Nebraska.

pooling layers with different sizes, which enhances the ability to capture the feature detail and improve the detection accuracy [30].

2.2 SPP-Net

Existing CNNs require input images with a fixed size. For image with an arbitrary size, to fit it to the fixed size, image cropping or warping are commonly used, which may lead to content loss or distortion and reduce the recognition accuracy. A new network structure, SPP-Net, was developed to eliminate this requirement by using a spatial pyramid pooling strategy (Figure 2). It can transform the feature map with any size into a fixed-size feature vector. By dividing images from finer to coarser levels, the SPP-net could provide multi-scale information, which boosts the accuracy of CNN architectures and makes the network more flexible and robust [12]. Zhang et al. [36] proposed a detection method for defect insulators based on YOLO and SPP-Net, whose detection accuracy reached 89%. Wang et al. [30] developed a data-driven method for vehicle detection based on tiny-YOLOv3, which is improved by SPP-net. Compared with tiny-YOLOv3, the average accuracy of the improved algorithm is 7.12% higher, which has reached 91.03%. Wu [35] proved that SPP-net has better performance than YOLO in static pedestrian detection situations. Thus, in this study, SPP-Net was added into CNN architecture to improve model performance potentially.

3 DATA SET AND PREPROCESSING

3.1 Study Area and Dataset

The study will be conducted at West Fork Big Blue Watershed, Nebraska (Figure 3). It lies on a gently undulating loess plain with descending elevation from its west to its east. The landscape in this watershed is dominated by intensive agriculture. The dense road networks and depressional wetlands lead to a poorly developed drainage system. The sources of 4-band Digital Orthophotos used in this study are from the USGS National Agriculture Imagery Program (NAIP). The NAIP under the USDA Farm Service Agency (FSA) acquired color infrared aerial orthophotos at a resolution of 1-meter ground sample distance. 2022 drainage crossing locations were digitized manually to prepare the training datasets.

3.2 Preprocessing

Since the data size of Orthophotos is over 10 GB, to increase the computing efficiency, four-band samples with size of 100-by-100 pixels were clipped by using a square bounding box with a size of 100-by-100 meters where the drainage crossing location is at the center (Figure 4).

4 NEURAL ARCHITECTURE SEARCH

Deep learning models have gained extensive usage across various tasks, such as computer vision [38] and augmented reality/virtual reality [26]. Nevertheless, designing an effective structure for these models demands substantial time and effort. Recently, the architecture of neural network models has been recognized as valuable intellectual property [31]. A well-designed deep learning model structure cannot only enhance researchers' task accuracy but also significantly reduce the time and cost associated with training and testing.

However, achieving optimal performance with deep learning models through manual tuning is challenging. This is due to the immense search space inherent in deep learning model design [39, 40]. Consequently, researchers have introduced neural architecture search, which aims to automatically discover optimal neural network architecture configurations that often outperform models designed by humans. This advancement empowers researchers to design their models more efficiently, thereby achieving notable cost savings in the training process.

4.1 Neural Network Intelligence

In this study, we have selected the Neural Network Intelligence framework, Retiarii [17], as our preferred toolkit. Despite its lightweight nature, this toolkit boasts substantial capabilities in assisting model developers with automatic searches within the neural network domain. We opted for the Retiarii framework for the following reasons. Firstly, Retiarii seamlessly integrates with established deep learning training frameworks such as TensorFlow and PyTorch,

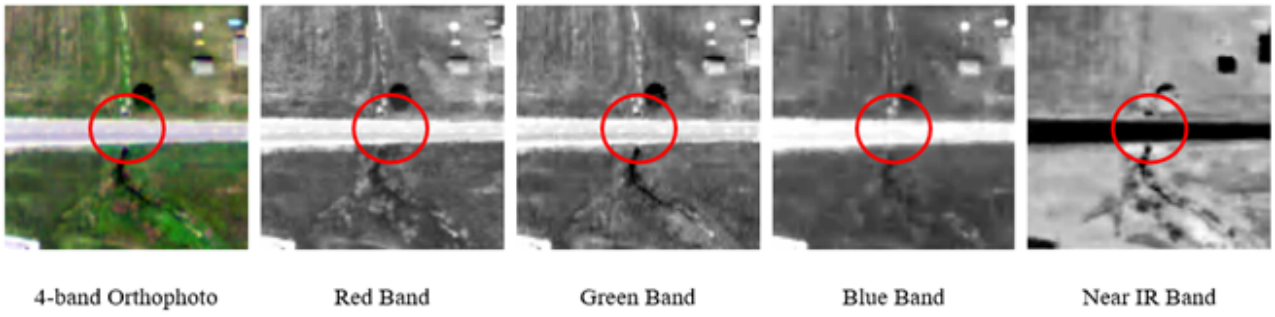

Figure 4: Examples of 4-band Orthophoto Samples. (The Circles Point to the Location of Drainage Crossing.)

delivering a versatile platform for our research needs. Additionally, the toolkit features a Just-In-Time (JIT) engine that manages model instantiation, oversees model training, compiles data for the exploration strategy's utilization, and facilitates decision-making in alignment with the gathered information. Lastly, it provides a powerful Mutator, empowering model developers to define the neural network model's search space and exploration strategies according to their requirements. Nonetheless, it's important to note that Retiarii currently supports NAS exclusively for single GPU setups. For NAS on multi-GPU systems, which has been addressed in studies like Weingram et al.[32] and Li et al.[15], we consider this as a potential area for future research and development.

4.2 Model Search Setup

As described in Section 2.2, SPP-Net comprises three primary components: feature engineering, the SPP Layer, and fully-connected layers. The process begins with multiple convolution layers and max pooling layers extracting essential features from raw images. Subsequently, the SPP layer divides the input feature map into three distinct regions, pooling the features within each region to generate a fixed-size output, irrespective of the input image's dimensions. Finally, the output of SPP-Net is directed to the fully connected layers for classification and bounding box regression. We explore the following search spaces for all three components:

- **Feature engineering**: We define the search space for the filter size of the first convolutional layer as ranging from 1 to 9 (1, 3, 5, 7, 9).
- **SPP layer**: We experiment with five different filter sizes for the first SPP (Spatial Pyramid Pooling) layer, spanning from 1 to 5 (1, 2, 3, 4, 5).
- **Fully-connected layers**: We customize the feature size for two fully-connected layers within the following ranges: 128, 256, 512, 1024, 2048, 4096, and 8192.

In our model search strategy, we employ a multi-trial strategy [17] wherein a model evaluator assesses the performance of each sampled model. This approach necessitates an exploration strategy to sample models from predefined model space. For our exploration strategy, we opted for the random search strategy, which involves randomly selecting an architecture with each iteration. For the model evaluator, we used FunctionalEvaluator, which is the default evaluator provided by the Retiarii framework.

5 INFERENCE EFFICIENCY IMPROVEMENT

5.1 Customized Demands on Inference Efficiency

The incorporation of neural architecture search yielded candidate models with high prediction accuracy. However, the SPP-Net based models specialize in taking variable-sized inputs which posits a huge load on the inference process, resulting in very high inference latency for large image sizes. Furthermore, our requirement to process a large number of images requires inference efficiency optimization of the candidate models.

5.2 Inter-Operator Scheduler for Inference Efficiency Optimization and Measurement

Inference efficiency can be improved via intra-operator parallelism (e.g. data parallelism) and the more novel approach of inter-operator parallelism. Inter-operator parallelism relies on branched structures within the neural architecture (e.g. Inception cell) within which branches can be executed simultaneously. The parallel execution yields an increase in performance over the general sequential approach taken by traditional deep learning frameworks (e.g. Pytorch and TensorFlow). We incorporate both of these methodologies by introducing data parallelism in the form of the standard practice of batching, and using Inter-Operator Scheduler (IOS) [3] for inter-operator parallelism. IOS uses blocks (which can be nested structures) to describe a branched substructure of the neural network, where the input and output should be convergent. Each block is optimized for inter-operator parallelism by dividing the execution into multiple stages which are sequential in nature. Each stage contains the execution of multiple groups in parallel, and each group is made up of one or more operators of a single branch which are executed sequentially. The group executions of a stage are synchronized before proceeding to the next stage. IOS uses a dynamic programming algorithm to determine the best division of a block into groups and stages and generate an optimal execution schedule. Our candidate models contain branched adaptive pooling layers (SPP layer) which can potentially yield an enhancement in inference efficiency and therefore, we test IOS to determine the maximum possible performance gain.

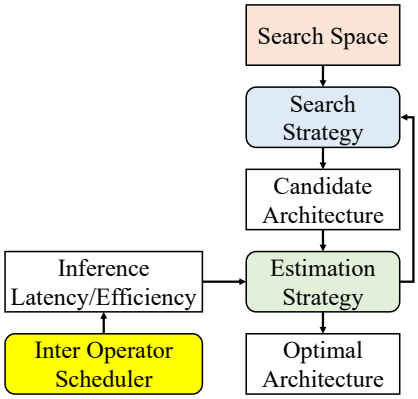

Figure 5: Neural architecture search for accuracy-constrained efficiency optimization.

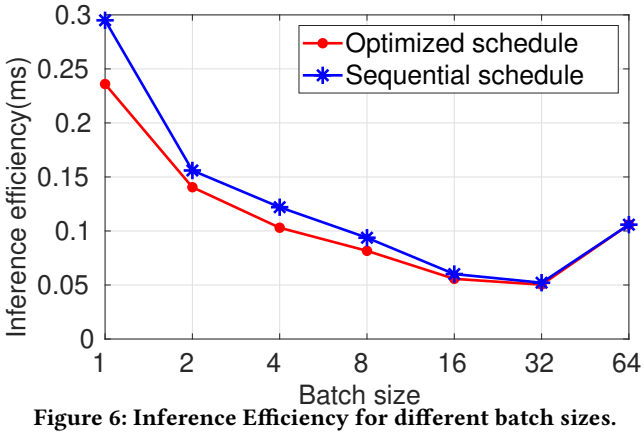

Figure 6: Inference Efficiency for different batch sizes.

5.3 Resource-Aware NAS by Incorporating the Inter-Operator Scheduler

Conventionally, neural architecture search seeks to optimize the prediction accuracy of a model. However, the nature of our task requires us to optimize the inference efficiency as well. Therefore, we incorporated the dual optimization problem into our neural architecture design process. The pipeline first generated candidate models with threshold accuracy which were then benchmarked via IOS to select the most efficient design. Figure 5 illustrates this process.

5.4 Inference Efficiency Optimization under the Prediction Accuracy Constraint

We present resource-aware NAS as a dual optimization problem where the objective function seeks to maximize both inference efficiency (i.e., throughput) and prediction accuracy. We formalize the problems as: **given** a neural architecture space N, **maximize** $a(n)$ and **maximize** $e(n), n \in N$, where a and e denotes the prediction accuracy and inference efficiency, respectively. We convert the dual optimization problem into a single target optimization problem by transforming one of the objectives to a constraint. Thus, the new formulation is to **maximize** $e(n), n \in N$, **subject to** $a(n) > A$, where $e(n)$ is the updated objective function for inference efficiency and A denotes a threshold accuracy constant.

6 EVALUATION

6.1 Experiment Setup

We trained all SPP-Net candidate models on one NVIDIA RTX A5500 GPU. The batch size for input images was set at 20. Our dataset was divided into training and testing sets with an 80/20 ratio. We employed the stochastic gradient descent (SGD) optimizer, with a learning rate of 0.005, weight decay set to 0.0005, and momentum at 0.9.

To evaluate the model performance, we used the average precision (AP) metric. AP is a commonly used measure for assessing object detection and instance segmentation algorithms. It gauges the precision-recall trade-off of a model across varying confidence thresholds. The equation for the average precision (AP) is given Equation 1:

$$AP = \frac{1}{n}\sum_{i=1}^{n}(Recall_i - Recall_{i-1}) \times Precision_i \quad (1)$$

6.2 Results for Different SPP-Net Models

Using the NNI, we generated three candidate SPP-Net models. Their respective hyper-parameter configurations are provided in Table 1. The initial SPP-Net achieved an average precision of 95%. However, through the hyper-parameter optimization facilitated by NNI, we managed to enhance the performance further. The resulting model, SPP-Net #3, exhibited an average precision of 97.40%.

6.3 Inference Latency for Different SPP-Net Models

We selected the candidate models generated by NNI. Then, we used IOS to optimize their execution schedule and recorded the inference latency of the sequential schedule and the optimized schedule for the batch size of 1. Table 2 shows the inference latency of the different candidate models. SPP-Net#2 outperforms the other models and was therefore selected as the final model.

6.4 Inference Efficiency for Different Batch Sizes

We define inference efficiency as the inference latency of an execution divided by the batch size of that execution. We used SPP-Net #2 as the base model and generated optimized schedules for batch sizes 1, 2, 4, 8, 16, 32 and 64. Figure 6 visualizes the performance enhancement for the sequential and the optimized schedules. The model shows a diminishing gain in performance up to batch size 32 due to hardware constraints, which we select as the optimal batch size.

7 PROFILING-BASED ANALYSIS

To facilitate the performance tuning of GPU applications, GPU vendors (Nvidia and AMD) and neural network system stacks (TensorFlow) have introduced execution profilers [37]. As an example, Nvidia offers Nsight System [8] to aid users in monitoring and optimizing their CUDA programs. Nsight System provides profiling capabilities in three primary respects: tracking CUDA memory

Model	Hyper-parameters Settings	Average Precision
Original SPP-Net	$C_{64,3,1} - P_{2,2} - C_{128,3,1} - P_{2,2} - C_{256,3,1} - P_{2,2} - SPP_{4,2,1} - F_{1024}$	95.00%
SPP-Net # 1	$C_{64,5,1} - P_{2,2} - C_{128,3,1} - P_{2,2} - C_{256,3,1} - P_{2,2} - SPP_{4,2,1} - F_{1024}$	96.10%
SPP-Net # 2	$C_{64,3,1} - P_{2,2} - C_{128,3,1} - P_{2,2} - C_{256,3,1} - P_{2,2} - SPP_{5,2,1} - F_{4096}$	96.70%
SPP-Net # 3	$C_{64,3,1} - P_{2,2} - C_{128,3,1} - P_{2,2} - C_{256,3,1} - P_{2,2} - SPP_{5,2,1} - F_{2048}$	97.40%

Table 1: Result for different SPP-Net structures. (C=`Convolution` and the subscripts of C stand for the size of the filter, number of filters, and stride. F=`Fully-connected` and the number of neurons in F are shown in subscript. P=`Pooling`, and its subscripts represent filter size and stride. SPP=`SPP layer`, and its subscripts represent filter size. The activation function is omitted due to space limitations.)

Model	Sequential Inference Latency	Optimized Inference Latency
Original SPP-Net	0.512 ms	0.268 ms
SPP-Net # 1	0.419 ms	0.379 ms
SPP-Net # 2	0.295 ms	0.236 ms
SPP-Net # 3	0.562 ms	0.427 ms

Table 2: Inference latency for the candidate models.

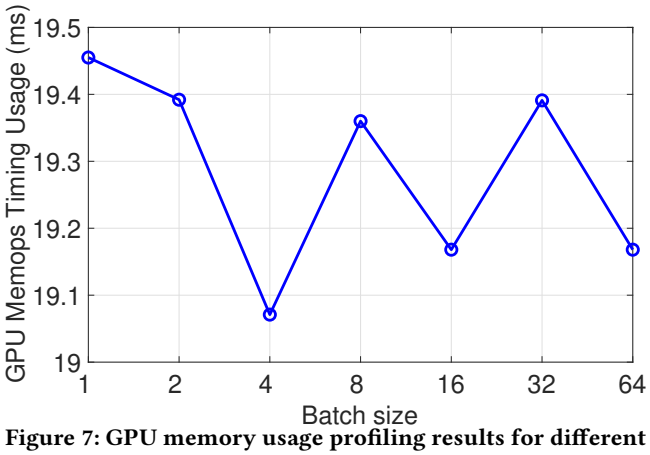

Figure 7: GPU memory usage profiling results for different batch sizes.

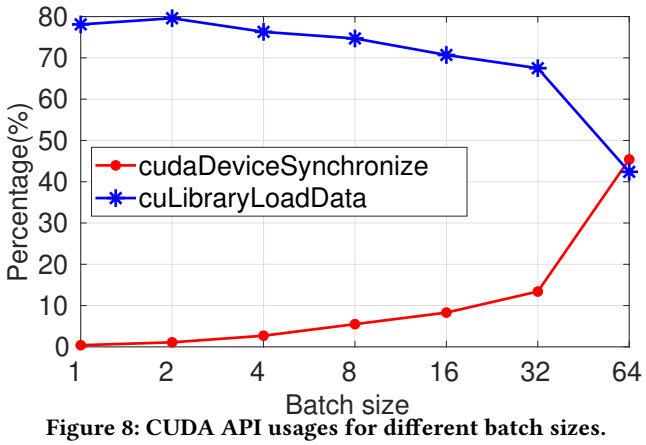

Figure 8: CUDA API usages for different batch sizes.

utilization, monitoring CUDA API usage, and performing in-depth analysis of CUDA kernel profiling.

In our study, we utilize the Nvidia Nsight System to profile the CNN inference in the IOS. This profiling helps us delve deeply into the inference execution to identify performance bottlenecks. Ultimately, we explore strategies for mitigating these bottlenecks.

7.1 CUDA Memory Utilization

During CNN inference, multiple images are processed in the model to achieve maximum throughput. In this study, we tested batch sizes ranging from 1 to 64 (1, 2, 4, 8, 16, 32, and 64). The Nsight profiling system was employed to profile each inference task. The GPU under test was the NVIDIA RTX A5500 GPU, boasting 10240 CUDA cores and 24 GB of graphics memory.

As depicted in Figure 7, the profiling results display the GPU memory timing usage for various batch sizes. The GPU Memops Timing Usage defines the speed at which data can be read from or written to the memory on a GPU. This parameter plays a crucial role in evaluating the overall performance of a GPU. Notably, as the batch size reaches 16, the GPU memory usage timing decreases and stabilizes at 19168 ns. Our findings reveal that GPU memory does not constrain the inference timing, as the available GPU memory surpasses the requirements for inference images. Even when 64 images are processed on the GPU, their memory usage remains considerably lower than the 24 GB capacity.

7.2 CUDA API Utilization

In the preceding subsection, we established that GPU memory does not serve as a constraint on inference timing. In this section, we conduct a comprehensive profile of all CUDA API usage for different batch sizes. Here, we only show the two primary CUDA APIs under scrutiny are *cuLibraryLoadData* and *cudaDeviceSynchronize*.

The *cuLibraryLoadData* API is a standard CUDA function responsible for loading libraries and data specific to a given code. Conversely, the *cudaDeviceSynchronize* API ensures that all previously executed CUDA calls are completed, guaranteeing that the device has concluded its tasks before proceeding with subsequent CPU operations. It acts as a synchronization barrier, ensuring all CUDA operations are finalized before CPU execution continues. This function addresses the potential asynchronous behavior between the CPU and GPU. Due to this asynchrony, the CPU might attempt to access GPU results not to be computed. The *cudaDeviceSynchronize* function addresses such issues.

Our observations indicate a significant increase in the utilization of CUDA APIs as batch sizes expand. As depicted in Figure 8, for a batch size of 1, approximately 80% of the time is consumed by the *cuLibraryLoadData* API, while merely 0.4% is attributed to the

Batch Size	Matrix Multiplication (%)	Pooling (%)	Conv (%)
1	41.6	14.1	7.7
2	34.8	14.4	9.7
4	39.9	13.5	9.5
8	34.8	13.7	10
16	18.1	17.1	16.6
32	15.7	14.7	13.4
64	7.4	8.6	77.2

Table 3: GPU kernel profiling for different batch sizes.

cudaDeviceSynchronize API. Conversely, as the batch size escalates to 64, the usage of the *cudaDeviceSynchronize* API surpasses that of the *cuLibraryLoadData* API, accounting for 45.40%.

As the batch size increases, more CUDA streams are initiated to process the augmented data. However, given the finite nature of GPU resources, the *cudaDeviceSynchronize* API is increasingly employed to ensure synchronization across these multiple CUDA streams. Despite the consistent memory utilization depicted in Figure 7, our observation leads us to hypothesize that the bottleneck could be attributed to limited PCIe bandwidth between the CPU and GPU during CNN inference. This conjecture gains traction as synchronization overhead becomes more pronounced. This phenomenon arises from the augmented data being queued for transmission to GPU kernels and subsequently back to CPU memory. The process of synchronizing this higher volume of data becomes more resource-intensive, contributing to the observed bottleneck.

7.3 CUDA kernel Profiling Analysis

The GPU kernel profiling outcomes are concisely presented in Table 3. This table outlines three primary operators: Matrix Multiplication, Pooling, and Convolution. These operators correspond to three distinct layer types: fully-connected layers, pooling layers, and convolution layers, respectively.

With SPP-Net primarily composed of convolutional layers, a clear pattern emerges: as the batch size grows, convolutional computations progressively outstrip other operations in terms of their dominance within the execution timeline. This trend is distinctly evident in the data outlined in Table 3. As the batch size increases, the proportion of time allocated to Matrix Multiplication continues to dwindle while the Convolution operation gains increasing prominence. In contrast, the Pooling operation displays a more stable behavior across varying batch sizes. Notably, upon reaching a batch size of 64, convolutional operations seize the lion's share of processing time, constituting a remarkable 77.2% of the total. This stark contrast with lower batch sizes suggests that this behavior could be attributed to the highly parallel design intrinsic to GPU kernels.

However, in scenarios where only the inference task is performed for a single image, the distribution of processing time varies. Here, a substantial portion of time, 41.6%, is utilized for Matrix Multiplication, while convolutional operations consume only 7.7% of the total time. Given that GPUs are designed to achieve high throughput and possess exceptionally high data parallelism capabilities, they are well-suited for processing large batch sizes.

8 RELATED WORK

8.1 Drainage Crossing Object Detection

Region-based CNN (RCNN) method combines region proposals with CNNs. Its evolution, Fast R-CNN, achieves near real-time rates using very deep networks [25], but exposes region proposal computation as a bottleneck. To overcome this shortage, the latest incarnation, faster region-based convolutional neural networks (faster R-CNN) introduces novel region proposal networks (RPNs) that share full-image convolutional features with object detection networks, thus making region proposal computations nearly cost-free [5]. As one of the potential tools for drainage crossing detection, Li et al. has applied a faster R-CNN to West Fork Big Blue Watershed in NE based on 1-meter resolution DEM samples with a size 800 by 800 meters. Pre-trained Resnet-50 was used as the backend convolutional neural network for this model. The SGD optimizer was used with learning rate of 0.001, a decay factor of 0.005, and a momentum of 0.9. A confidence threshold of 0.7 was selected to filter out low confidence detection. The results show that the accuracy of the model is 0.882 and the intersection over union (IOU) of predicted bounding boxes is 0.668.

8.2 Neural Architecture Search Toolkits

Two primary types of NAS toolkits are prevalent, namely DeepHyper [28] and NNI [19]. The first tool provides hyperparameter tuning and architecture optimization of deep learning models for research problem-solving in various domain sciences on supercomputers [2, 4, 18]. In the second track, NNI has been introduced as a toolkit for exploring optimal model structures using a single GPU [17]. Due to its lightweight nature and exceptional performance, we have selected NNI as our preferred toolkit for NAS in this study.

8.3 Inference Efficiency Improvement

The predominant inference efficiency optimization technologies are Rammer [17], Nimble [11], IOS [3], and HIOS [10]. Rammer enhances performance by utilizing both inter and intra-operator optimization and generates an optimized static parallel schedule during the compilation of the data flow graph which serves to reduce the scheduling overhead. Nimble offers enhancement by utilizing ahead-of-time scheduling which generates the optimal schedule before GPU kernel execution, hence, saving overhead. IOS uses a dynamic programming algorithm to test all the schedules in its schedule search space and returns the optimal schedule. HIOS is a hierarchical inter-operator scheduler for inference latency reduction by exploiting inter-GPU and intra-GPU operator parallelization on multi-GPU platforms. While Rammer and Nimble pre-generate the schedule, saving overhead time, IOS returns the more efficient schedule and since our task requires the best possible schedules, even at the computational cost of generating the schedules, we selected IOS in a single GPU for our project.

9 CONCLUSION

In this study, we present the utilization of SPP-Net for the assessment of hydrologic connectivity. Particularly, we discover that employing convolutional neural networks leads to a significant

enhancement in accuracy when detecting drainage crossings. Moreover, we showcase the effectiveness of resource-aware NAS in fine-tuning the hyperparameters of SPP-Net, resulting in substantial improvements in inference efficiency. Furthermore, we conduct in-depth profiling of our detection models on GPU systems and analyze the performance bottlenecks specific to single GPU setups.

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.

REFERENCES

[1] Fernando Aristizabal, Lauren E. Grimley, Jerad Bales, Danielle Tijerina, Trey Flowers, and Edward P Clark. 2018. National Water Center Innovators Program Summer Institute Report 2018. *Consortium of Universities for the Advancement of Hydrologic Science, Inc.* Technical Report No. 15 (2018).

[2] Prasanna Balaprakash, Romain Egele, Misha Salim, Stefan Wild, Venkatram Vishwanath, Fangfang Xia, Tom Brettin, and Rick Stevens. 2019. Scalable Reinforcement-Learning-Based Neural Architecture Search for Cancer Deep Learning Research. In *Proc. of ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC)*. 1–33.

[3] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. 2021. IOS: Inter-Operator Scheduler for CNN Acceleration. In *Conf. on Machine Learning and Systems (MLSys) (MLSys)*. Virtual, 1–14.

[4] Romain Egele, Prasanna Balaprakash, Isabelle Guyon, Venkatram Vishwanath, Fangfang Xia, Rick Stevens, and Zhengying Liu. 2021. AgEBO-Tabular: Joint Neural Architecture and Hyperparameter Search with Autotuned Data-Parallel Training for Tabular Data. In *Proc. of ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC)*. 1–14.

[5] Ross Girshick. 2015. Fast r-cnn. In *Proc. of IEEE Intl. Conf. on Computer Vision (ICCV)*. 1440–1448.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 37, 9 (2015), 1904–1916.

[7] 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.

[8] Kumar Iyer and Jeffrey Kiel. 2016. GPU Debugging and Profiling with NVIDIA Parallel Nsight. *Game Development Tools* (2016), 303–324.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems* 25 (2012).

[10] 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.

[11] Woosuk Kwon, Gyeong-In Yu, Eunji Jeong, and Byung-Gon Chun. 2020. Nimble: Lightweight and Parallel GPU Task Scheduling for Deep Learning. In *Conf. on Neural Information Processing Systems (NeurIPS)*. Virtual Event, 1–12.

[12] Lingling Li, Zhengyan Yang, Licheng Jiao, Fang Liu, and Xu Liu. 2019. High-Resolution SAR Change Detection based on ROI and SPP Net. *IEEE Access* 7 (2019), 177009–177022.

[13] 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.

[14] Wenwen Li, Bin Zhou, Chia-Yu Hsu, Yixing Li, and Fengbo Ren. 2017. Recognizing Terrain Features on Terrestrial Surface Using a Deep Learning Model: An Example with Crater Detection. In *Proceedings of the 1st Workshop on Artificial Intelligence and Deep Learning for Geographic Knowledge Discovery*. 33–36.

[15] 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.

[16] John B Lindsay and K Dhun. 2015. Modelling Surface Drainage Patterns in Altered Landscapes Using LiDAR. *Intl. Journal of Geographical Information Science* 29, 3 (2015), 397–411.

[17] 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.

[18] Romit Maulik, Romain Egele, Bethany Lusch, and Prasanna Balaprakash. 2020. Recurrent Neural Network Architecture Search for Geophysical Emulation. In *Proc. of ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC)*. 1–14.

[19] Microsoft. 2021. *Neural Network Intelligence*. https://github.com/microsoft/nni

[20] Sandra K Poppenga, Bruce B Worstell, Jason M Stoker, and Susan K Greenlee. 2010. *Using Selective Drainage Methods to Extract Continuous Surface Flow from 1-Meter Lidar-Derived Digital Elevation Data*. Technical Report. U.S. Geological Survey.

[21] Wolfgang Schwanghart, Geoff Groom, Nikolaus J Kuhn, and Goswin Heckrath. 2013. Flow Network Derivation from a High Resolution DEM in a Low Relief, Agrarian Landscape. *Earth Surface Processes and Landforms* 38, 13 (2013), 1576–1586.

[22] Mairead Shore, PNC Murphy, Philip Jordan, P-E Mellander, M Kelly-Quinn, M Cushen, S Mechan, O Shine, and AR Melland. 2013. Evaluation of a Surface Hydrological Connectivity Index in Agricultural Catchments. *Environmental modelling & software* 47 (2013), 7–15.

[23] 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.

[24] 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.

[25] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

[26] Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh, and Jiasi Chen. 2023. Going through the Motions: AR/VR Keylogging from User Head Motions. In *Proc. of USENIX Security Symposium (Security)*. Anaheim, CA, 159–174.

[27] Giulia Sofia, Giancarlo Dalla Fontana, and Paolo Tarolli. 2014. High-Resolution Topography and Anthropogenic Feature Extraction: Testing Geomorphometric Parameters in Floodplains. *Hydrological Processes* 28, 4 (2014), 2046–2061.

[28] DeepHype Team. 2018. DeepHyper: A Python Package for Scalable Neural Architecture and Hyperparameter Search. https://github.com/deephyper/deephyper

[29] Jiahu Wang, Li Li, Zhenchun Hao, and Jonathan J Gourley. 2011. Stream Guiding Algorithm for Deriving Flow Direction from DEM and Location of Main Streams. *IAHS-AISH Publication* 346 (2011), 198–205.

[30] Xiaolan Wang, Shuo Wang, Jiaqi Cao, and Yansong Wang. 2020. Data-Driven Based Tiny-YOLOv3 Method for Front Vehicle Detection Inducing SPP-net. *IEEE Access* 8 (2020), 110227–110236.

[31] 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 Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN)*. Valencia, Spain, 125–137.

[32] 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.

[33] 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.

[34] 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.

[35] Jiatu Wu. 2018. Complexity and Accuracy Analysis of Common Artificial Neural Networks on Pedestrian Detection. In *MATEC Web of Conferences*, Vol. 232. EDP Sciences, 01003.

[36] Xibo Zhang, Yan Zhang, Miao Hu, and Xiaoming Ju. 2020. Insulator Defect Detection based on YOLO and SPP-Net. In *Intl. Conf. on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. 403–407.

[37] Yicheng Zhang. 2021. *Stealing Deep Learning Model Secret through Remote FPGA Side-channel Analysis*. University of California, Irvine.

[38] Yicheng Zhang, Carter Slocum, Jiasi Chen, and Nael Abu-Ghazaleh. 2023. It's all in your Head(set): Side-Channel Attacks on AR/VR Systems. In *Proc. of USENIX Security Symposium (Security)*. Anaheim, CA, 3979–3996.

[39] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. 2021. Stealing Neural Network Structure through Remote FPGA Side-Channel Analysis. In *Proc. of ACM/SIGDA Intl. Symposium on Field-Programmable Gate Arrays (FPGA)*. Virtual Event, USA, 225.

[40] 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. Inf. Forensics Secur.* 16 (2021), 4377–4388.

[41] Xiran Zhou, Wenwen Li, and Samantha T Arundel. 2019. A Spatio-Contextual Probabilistic Model for Extracting Linear Features in Hilly Terrains from High-Resolution DEM Data. *Intl. Journal of Geographical Information Science* 33, 4 (2019), 666–686.

A ARTIFACT APPENDIX

A.1 Abstract

Our purpose for providing this artifact is to make our results reproducible. We have three main results:

- **Prediction Accuracy**: We used neural architecture search (NAS) to explore our model space for models with best accuracy results.
- **Inference Efficiency**: We utilized inter-operator scheduler (IOS) [3] to determine best schedules for the NAS-generated models and choose the optimal inference batch size.
- **GPU Performance Profiling**: We utilized *nsys* to profile and analyse GPU performance metrics when we choose different batch sizes.

A.2 Hardware dependencies

Dell Precision 5820 Tower Workstation with the NVLink Nvidia RTX A5500 GPU

A.3 Software dependencies

- CUDA 12.2.14
- cuDNN 8.9.5
- Python 3.11
- numpy 1.24.3
- torchvision 0.14.1+cu117
- matplotlib 3.7.1
- PyTorch 2.0
- NNI 2.0
- IOS
- CMake >=3.10

A.4 Installation

Firstly, clone the GitHub repository and then install the required packages.

```
git clone https://github.com/SHUs-Lab/SHDA23YZ.git
pip install -r requirements.txt
```

A.5 Evaluation

Neural Architecture Search. First, download the data.zip file provided in the GitHub repository and unzip it. Next, execute object_detection.py to initiate the training of SPPNet. Additionally, there is an option to run sppnet_search.py to conduct a neural architecture search. For details on the NAS search space, refer to Section 4.2.

IOS. The file IOS_Model.py contains the SPP-Net model in IOS code. Execute the file to get the optimized schedule for the model. To test for different models in the model space, edit the file as follows:

- Change hyperparameters based on the candidate model in lines 9-14
- Change the batch sizes in lines 33, 39, 41

For more details on IOS scheduling refer to Section 5.2

GPU Performance Profiling. To get the GPU profiling information, use the command:

```
nsys profile --stats=true --force-overwrite true -o
[report file name] python [IOS_Model].py
```