Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

journal homepage: www.elsevier.com/locate/isprsjprs

# Space-to-speed architecture supporting acceleration on VHR image processing

Shenlu Jiang[a], Yuliya Tarabalka[b,c], Wei Yao[d,f], Zhonghua Hong[e,*], Guofu Feng[e]

[a] Faculty of Information Technology, Macau University of Science and Technology, Macao Special Administrative Region of China
[b] Titane Team, National Institute for Research in Computer Science and Automation (Inria), Sophia Antipolis, France
[c] Luxcarta Technology, Mouans-Sartoux, France
[d] Otto Poon Charitable Foundation Smart Cities Research Institute, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
[e] College of Information Technology, Shanghai Ocean University, Shanghai, China
[f] Department of Land Surveying and Geo-Informatics, The Hong Kong Polytechnic University, Hung Hom, Hong Kong

## ARTICLE INFO

*Keywords:*
Space-to-speed architecture
Building segmentation
Deep neural networks (DNNs)
Very high-resolution (VHR) aerial images

## ABSTRACT

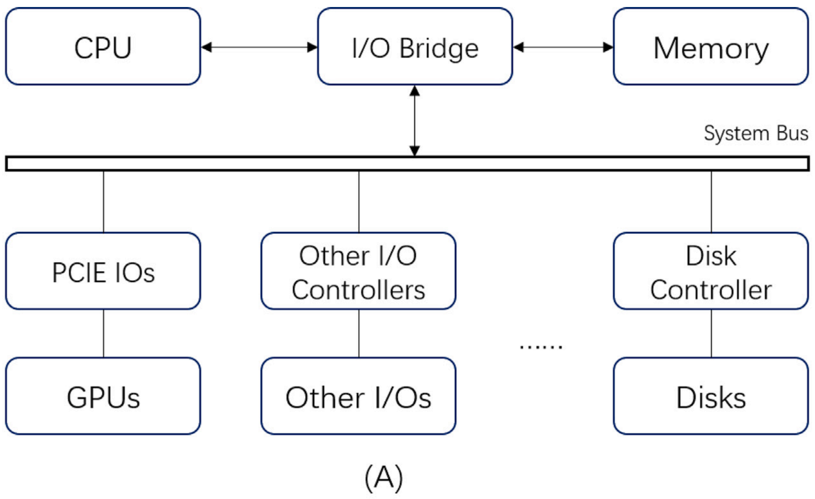
One of the major focuses in the remote sensing community is the rapid processing of deep neural networks (DNNs) on very high resolution (VHR) aerial images. Few studies have investigated the acceleration of training and prediction by optimizing the architecture of the DNN system rather than designing a lightweight DNN. Parallel processing using multiple graphics processing units (GPUs) increases VHR image processing performance. It drives extremely large and frequent data transfers (input/output(I/O)) from random access memory (RAM) to GPU memory. As a result, the system bus congestion causes the system to hang, resulting in long latency in training/predicting. In this paper, we evaluate the causes of long latency and propose a space-to-speed (S2S) DNN system to overcome the aforementioned challenges. A three-level memory system aiming to reduce data transfer during system operation is presented. Distribution optimization with parallel processing was used to accelerate the training. Training optimizations on VHR images (such as hot-zone searching and image/ground truth queues for data saving) were used to train the VHR images efficiently. Inference optimization was performed to speed up prediction in the release mode. To verify the efficiency of the proposed system, we used aerial image labeling from the Institut National de Recherche en Informatique et en Automatique (INRIA) and benchmarks from the Massachusetts Institute of Technology Aerial Imagery for Roof Segmentation (MITAIRS) to test the system performance and accuracy. Without the loss of accuracy, the S2S system improved prediction speed on the testing dataset by eight GPUs in a normal setting in both the INRIA dataset (from 534 to 72 s) and the MITAIRS dataset (818 to 120 s). With the prediction in half-float (using float-16 data), an 8-GPU parallel processing increased the speed to 38 s in the INRIA dataset and 83 s in the MITAIRS dataset. In a pressure test, our proposed system operated on 18,000 images with a size of 5000 × 5000 from 18.2 to 1.8 h with the prediction in full-float (using float-32 data) and 43 min with the prediction in half-float, increasing the speed by a factor of 9.78 and 25.3, respectively, when compared to system runs without optimization.

## 1. Introduction

Deep learning has been a major trend in the remote sensing community, with numerous applications such as object detection (Cheng et al., 2020; Feng et al., 2020; Fu et al., 2020), scene understanding (Chaib et al., 2017; Li et al., 2017; Rahnemoonfar et al., 2021), and semantic segmentation (Mi and Chen, 2020; Sun et al., 2020; Yuan et al., 2021). Semantic segmentation is a type of scene understanding that uses labels in each pixel of an image to categorize it. It can be used in remote sensing for traffic management (Zhang et al., 2019) and land use detection (Sun et al., 2020; Wang et al., 2020). Several semantic segmentation methods have been proposed, and there exists two major types of semantic segmentation: fully convolutional DNNs (Huang et al., 2017; Jégou et al., 2017; Ronneberger et al., 2015) and encoder–decoder DNNs (Chen et al., 2018; Liu et al., 2019). However, most methods are time-consuming when performing remote sensing tasks, and they cannot meet the demands for fast processing

* Corresponding author.
*E-mail addresses:* shenlujiang@must.edu.mo (S. Jiang), ytarabalka@luxcarta.com (Y. Tarabalka), wei.hn.yao@polyu.edu.hk (W. Yao), zhhong@shou.edu.cn (Z. Hong), gffeng@shou.edu.cn (G. Feng).

https://doi.org/10.1016/j.isprsjprs.2023.02.010
Received 11 January 2022; Received in revised form 17 January 2023; Accepted 19 February 2023
Available online 8 March 2023



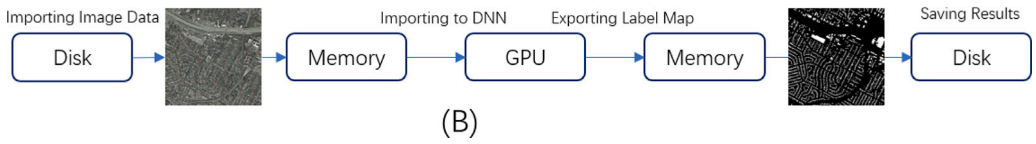


**Fig. 1.** System description of the most common computer architecture. In A, the system architecture is shown. In B, the general flow map of building segmentation is shown. **I/O means input/output**.

in very large regions. Lightweight architecture for real-time semantic segmentation has been proposed in some studies (Yu et al., 2018; Zhao et al., 2018). Although these methods can achieve rapid processing speed on normal-sized images, they are difficult to apply to very high resolution (VHR) images. In general, the above methods can reduce the number of nodes and output map size, resulting in an inevitable loss of accuracy.

Aside from DNN architecture limitations, another major drawback of VHR image processing is the long latency period caused by data transferals, as well as the brute-force strategy in training and prediction. In this paper, semantic segmentation was selected for a comprehensive evaluation. In particular, building/roof segmentation was employed because several datasets (Chen et al., 2019; Maggiori et al., 2017) in VHR images have been open-sourced, allowing for the rapid training and testing of our model. Building segmentation focuses on optimizing the DNN architecture (Bischke et al., 2019; Kurdi and Awrangjeb, 2020; Shi et al., 2020). Previously, very limited effort was put into optimizing the training and interference architecture. This major limitation is detected in the system architecture rather than the DNN. First, we briefly introduce the computer system and demonstrate how the semantic segmentation method operated. A computer is divided into two major parts: the north and south bridges, which communicate via the system bus. As shown in Fig. 1, external components (such as GPUs and disks) are located in the south bridge, whereas internal components (such as the central processing unit [CPU] and memory) are located in the north bridge.

To begin, we tracked the steps involved in predicting an image and output its label map to show any limitations. Fig. 1A depicts the architecture of a typical DNN system, with the north bridge region housing the CPU and random access memory (RAM), both of which are high-speed transferring areas. Unlike RAM, data stored in read-only memory (ROM) cannot be electronically modified after the memory device is manufactured. The south bridge region includes peripheral component interconnect express (PCIE) facilities, GPUs, and disks, which are low-speed transferring regions. The steps for DNN prediction (shown in Fig. 1B) are as follows:

(1) The CPU orders the disk to transfer the allocated data to the memory.

(2) The disk transfers data via the system bus to the memory at the allocated address.

(3) The memory transfers the decoded data to the GPU via the system bus for processing.

(4) The GPU transfers the results to RAM via the system bus.

(5) The results are encoded and saved in the disk via the system bus.

We found that the data were transferred four times via the system bus in a single step. The current DNN applied to VHR images produced congestion in the system bus. We conceptualize a road with a dual path traffic flow and only a single lane, where the maximum speed of the auxiliary lane is only 1/10,000 of the minimum speed of the main road, even when the main road is blocked by millions of vehicles. The disk functions as a low-speed auxiliary lane or solid state drive (SSD) with write/read (input/output, or I/O) speeds ranging from 300–500 mb/s. We found three major issues with the above architecture:

(1) The disk ran significantly slower than the capacity of the system bus, resulting in frequent thread hanging during DNN processing.

(2) Frequent writing and reading of the label map were random, resulting in a significant decrease in system speed;

(3) Because the system bus was used four times, the reading and writing functions occurred simultaneously, with different devices on the computer competing for the system bus.

More frequent and larger data transfers are required, particularly for parallel processing involving multiple GPUs. Seal et al. (2020) reported that the memory cost and complexity of the DNN model increase as a function of $n^2$, where n is the input image size. Although a larger input image produces better results because image splitting and inaccurate boundary object detection are avoided, the GPU RAM in the design does not meet the requirements for images with resolutions greater than 2048 × 2048. Moreover, the majority of current methods predict images without any optimization, i.e., they predict an image simply via a forward transfer, similar to a debug mode, leading to high resource costs and long latency during prediction.

We evaluated the factors slowing down the processing speed and proposed a space-to-speed (S2S) system architecture to process a VHR image (5000 × 5000) in less than 1 s (s). The five major causes of slow processing speed are as follows:

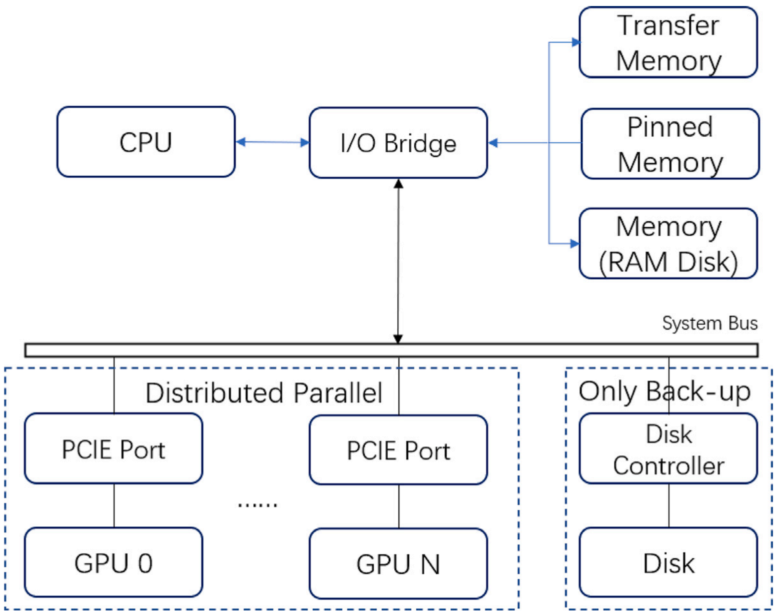


**Fig. 2.** Architecture of space-to-speed system.

(1) Inputting high-resolution images into a DNN significantly increases the memory usage of a GPU with additional nodes in the DNN;

(2) Frequent data transfer through the ROM, RAM, and GPU/CPU requires a long loading time (over 1 GB/s in multiple GPU parallel processing), thus reducing speed;

(3) Parallel synchronization of multi-core processing results in asynchronous outputs and hangs during processing;

(4) During model training and VHR image prediction, very few optimizations are conducted. The current method operates in a sequence mode without multi-threading, wasting computing resources on the desktop and resulting in long latency during model operation;

(5) Previous methods had issues with imbalanced background noise and long pre-processing times (e.g., splitting VHR images to save small blocks), or the combining of small blocks to form a complete VHR image was insufficient.

Because of the time spent on encoding/decoding VHR images, the frequent I/O on original images and output labels, and pre-processing on oversized VHR images due to their large size, these protocols are difficult to deploy for this task. The proposed S2S system includes the following optimizations. First, the maximum GPU memory usage rate is compared to the input resolution to determine the most reasonable input size to ensure a large receptive field. Second, a three-level memory architecture is proposed to increase operating RAM capacity and create a high-speed RAM disk path, reducing the frequency of data transfer via the system bus and thus decreasing processing latency. Third, distributed GPU allocation technology with accumulating loss reduction is proposed to resolve the system hang caused by synchronous lock in different GPU steps during training. The training/prediction optimization is used to maximally allocate computing resources during DNN operation. In addition to architecture optimizations, we enhanced the method for region proposal during training and inference. Common methods include image splitting or randomization. These typically encounter issues with imbalance selection on samples because the background labels occupy the largest part on the VHR image, but common methods make it difficult to maintain the balance inputting on different samples. The sample imbalance on VHR is solved by hot-zone search during training. Finally, optimizations, such as reducing data type complexity, combining blocks as a concat from the rectified linear unit (ReLU) bias and convolution, and decreasing unnecessary operations and outputs, are conducted otherwise, queue inputting and VHR sampling on RAM are proposed to ensure the prediction is in the release mode to increase its speed on VHR image.

To demonstrate the efficacy of our proposed S2S architecture, we conducted the following experiments: (1) The GPU memory and image input size relationship was evaluated to select the most reasonable VHR image input size. (2) The speeds of common disks (e.g., SSDs and hard disk drives [HDD]) and RAM disks were evaluated under different sequence I/O and random I/O settings. (3) The efficiency of the proposed S2S system architecture was tested with and without the designed components and parallel processing on datasets from the Institut National de Recherche en Informatique et en Automatique (INRIA) and the Massachusetts Institute of Technology Aerial Imagery for Roof Segmentation (MITAIRS). (4) Finally, a pressure test was performed to compare the proposed system with the standard system and to evaluate its performance for long-term prediction. The results showed that our proposed methods could maintain the same accuracy while increasing the speed by more than 10 times when using multiple GPUs. When using eight GPUs, the proposed system only required 1.8 h to process 18,000 VHR images with a size of 5000 × 5000, which was 10 times faster than it would be without the S2S system (18.2 h). When using the float-16 data type, the speed increased to 43 min, which was approximately 20 times faster than that without the S2S system. The results illustrated that the system could overcome the data transfer limitation in the face of data traffic jams on the system bus.

The major contributions of this paper can be summarized as follows:

(1) The S2S system promises to reduce the number of data transfers via the system from four to two times in every step of training and prediction, releasing system bus bandwidth resources and enabling maximum speed processing using multi-GPU processing of VHR images.

(2) Training optimizations are proposed to speed up training by using multi-CPU resources to coordinate the GPU for optimally allocating the computing resources. To avoid imbalance training, the hot-zone searching proposal is used during training to separate the foreground region from the substantial background on the VHR images.

(3) Prediction optimizations are proposed to accelerate inference speed, filter the trivial components of DNN in prediction, speed up processing by a factor of 10 without accuracy loss, and use the half-float (float-16) data type to speed up inference by a factor of 25.3 with only a 1% accuracy loss.

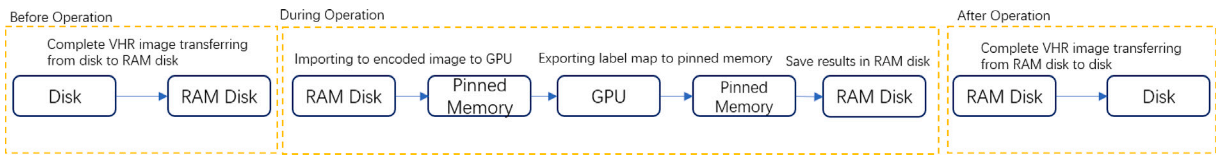

Before Operation
Complete VHR image transferring from disk to RAM disk
Disk
RAM Disk
During Operation
Importing to encoded image to GPU
RAM Disk
Pinned Memory
Exporting label map to pinned memory
GPU
Pinned Memory
Save results in RAM disk
RAM Disk
After Operation
Complete VHR image transferring from RAM disk to disk
RAM Disk
Disk

**Fig. 3.** Proposed predicting/training procedure using DNN. .

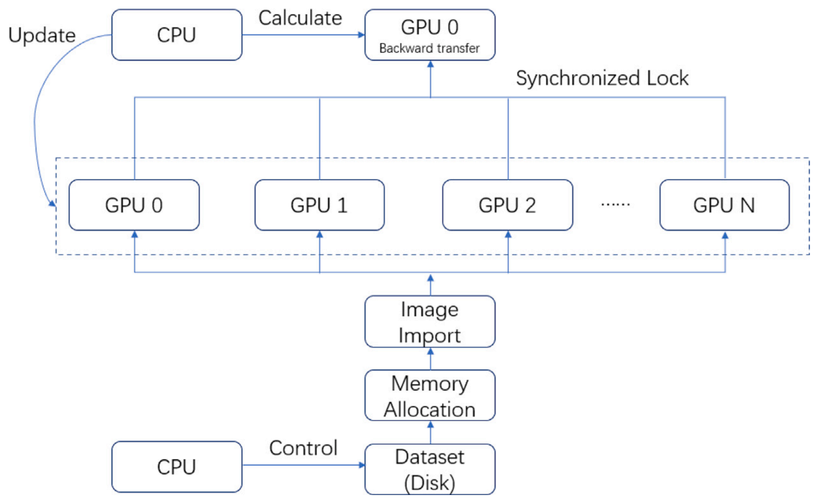


**Fig. 4.** Detailed flow map of parallel processing with synchronized lock.

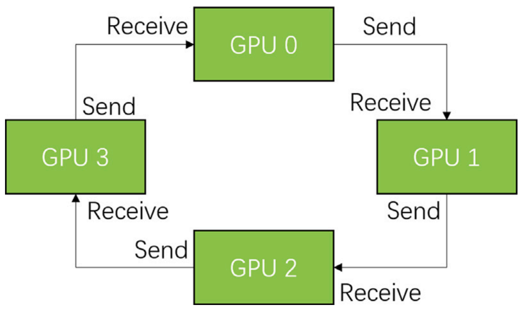


**Fig. 5.** General flow map of parallel processing with **synchronized lock**.

## 2. Methods

### *2.1. Low latency of data transfer*

Based on semantic segmentation of VHR satellite images, frequent and long data transfers from the ROM disk to the RAM are required. However, because satellite images are generally very data intensive, frequent decoding of such images and processing of the CPU are time-consuming. The major limitations have been discussed in the Introduction. To solve these limitations, we optimized the system architecture of the DNN using lower frequency data transfers via the system bus at a higher transfer speed.

The semantic segmentation of VHR satellite images necessitates frequent and lengthy data transfers from the ROM disk to the RAM. However, because satellite images are typically highly data-intensive, frequent decoding and CPU processing are time-consuming. The major limitations were discussed in the Introduction. To overcome these limitations, we optimized the system architecture of the DNN using lower frequency data transfers via the system bus at a higher transfer speed.

As shown in Fig. 2, our proposed S2S system divides RAM into three parts: transfer memory, which exchanges temporary data in the system, pinned memory, which pre-allocates the address for DNN training and prediction in different threads, and the RAM disk, which serves as a high-speed data-saving region.

The architectural concept is adapted to accelerate processing by employing task-specific tools. Sequence reading/writing with a disk is significantly faster than random I/O therefore, the original data of the VHR images are pre-transferred to the RAM disk. The CPU reads images directly from the RAM disk, decoding them and writing them back onto the RAM disk simultaneously. During the preparation for DNN training or testing, the pinned memory pre-allocates the required resources (e.g., GPU computing and loss updating) with a fixed address, reducing the time required for memory allocation. This strategy avoids the out-of-memory problem, which is caused by a delay in data release due to the fixed address of the active memory. Memory transfer for the general system operates in the same way as the transfer for the essential operation system (OS) (see Fig. 3).

In comparison, with the system depicted in Fig. 1(B), the original system only inputs the image, decodes it, exports it to the GPU, outputs it to the memory, and saves it onto the ROM disk again. The architecture must cross four times via the system bus. This will not cause a delay for single GPU processing; however, with parallel processing, the GPU captures more resources, requiring greater data transferring capabilities. In contrast, our method copies the entire dataset from the disk to the RAM disk before operation. The system then imports data from the RAM disk to the pinned memory, transfers the data to the GPUs, redelivers the obtained results to the pinned memory, and saves them back on the RAM disk. Finally, the complete data transfer from the RAM disk to the disk is repeated. The proposed memory system employs the existing operational mechanism, as well as the sequence I/O importing/exporting large-sized data to the RAM disk, which is a major driver of random I/O for further processing. The adequate I/O ability ensures that the GPUs can be optimally utilized. In contrast to the original mechanism, the S2S system transfers data only twice via the system bus (from pinned memory to GPUs then back to pinned memory) during training and testing in each step, significantly releasing the bandwidth during operation. In terms of its technology, the S2S system only occupies an address from RAM, which does not require extra memory resources to operate.

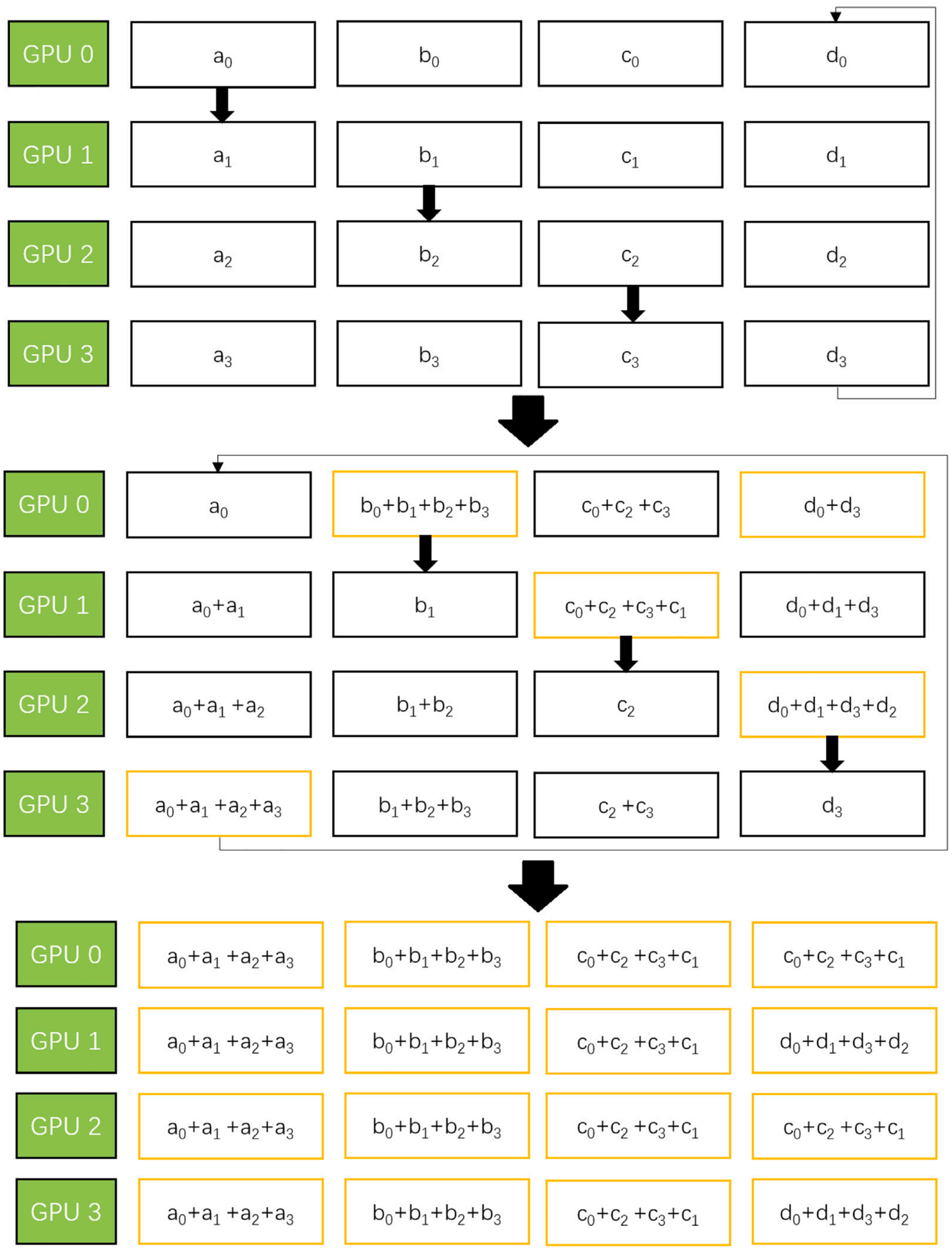

GPU 0
GPU 1
GPU 2
GPU 3
$a_0$ $b_0$ $c_0$ $d_0$
$a_1$ $b_1$ $c_1$ $d_1$
$a_2$ $b_2$ $c_2$ $d_2$
$a_3$ $b_3$ $c_3$ $d_3$
$a_0$ $b_0+b_1+b_2+b_3$ $c_0+c_2+c_3$ $d_0+d_3$
$a_0+a_1$ $b_1$ $c_0+c_2+c_3+c_1$ $d_0+d_1+d_3$
$a_0+a_1+a_2$ $b_1+b_2$ $c_2$ $d_0+d_1+d_3+d_2$
$a_0+a_1+a_2+a_3$ $b_1+b_2+b_3$ $c_2+c_3$ $d_3$
$a_0+a_1+a_2+a_3$ $b_0+b_1+b_2+b_3$ $c_0+c_2+c_3+c_1$ $c_0+c_2+c_3+c_1$
$a_0+a_1+a_2+a_3$ $b_0+b_1+b_2+b_3$ $c_0+c_2+c_3+c_1$ $d_0+d_1+d_3+d_2$
$a_0+a_1+a_2+a_3$ $b_0+b_1+b_2+b_3$ $c_0+c_2+c_3+c_1$ $c_0+c_2+c_3+c_1$
$a_0+a_1+a_2+a_3$ $b_0+b_1+b_2+b_3$ $c_0+c_2+c_3+c_1$ $d_0+d_1+d_3+d_2$

**Fig. 6.** Detail of ring all-reduce of distributed system.

### 2.2. *Multi-GPU training*

Multi-GPU parallel processing has recently emerged as a key technology for processing DNNs with large batch sizes to reduce training and testing times. Based on the semantic segmentation of satellite images (VHR), the parallel processing of large batch sizes can reduce the processing time for prediction using VHR images.

As shown in Fig. 4, the system reads the image and label data from the disk to the RAM during each step of the training. The images are then exported to the GPUs via forward transfer, and the optimizer calculates the information loss caused by this step. Next, the loss values in different GPUs are combined to update the model with the summarized loss (as shown in Fig. 5). In the ordinary training model, the GPU results are reduced in comparison to the other GPUs and then backward transferred with the loss calculated based on the average value from the other GPUs. However, a major limitation is that the current methods also utilize a synchronized lock on different GPUs at each step to maintain the global update on the information loss. This strategy requires additional time as each GPU must wait for the unfinished GPUs.

Because of the large data transfer requirement, the processing speed can be affected due to the training of the semantic segmentation on satellite images. To solve this problem, we utilized a distributed parallel training strategy. There are two major differences between this and the original method: (1) the proposed system generates multiple N+1 threads rather than just one; and (2) synchronization is processed by a ring update rather than updating at every step. We assume that at each start, the GPUs are independently operated by a thread. In each step, a GPU receives the status from the previous GPU and transmits it to the

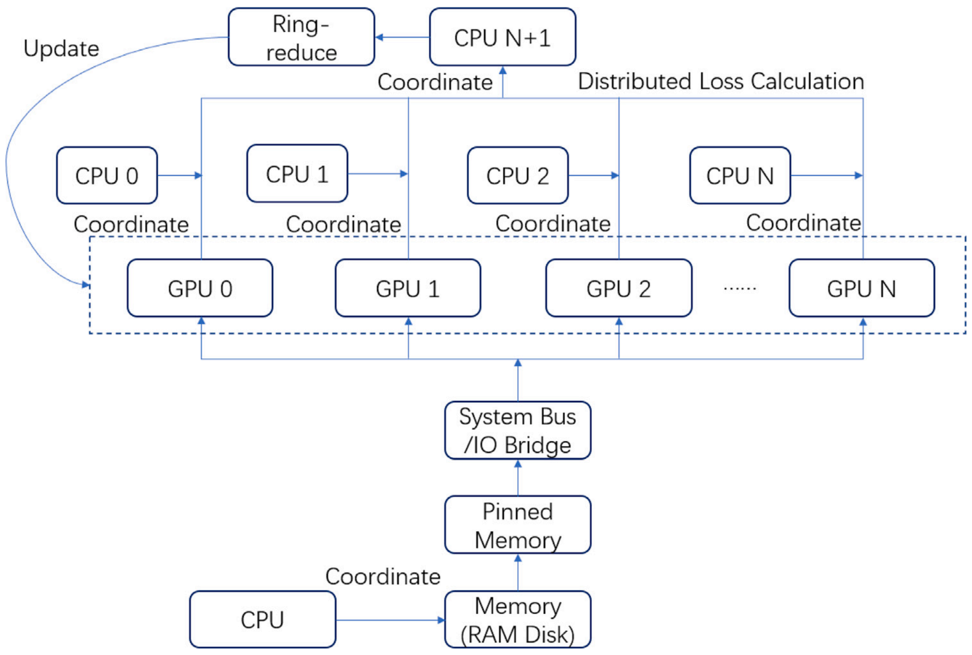


**Fig. 7.** Detail operation map of proposed training in distribution setting.

subsequent GPU. After $N$ times within a cycle, the GPU information is sent to all GPUs.

Fig. 6 depicts the cycle in greater detail. In each step, the status of a GPU is sent to the next GPU, which receives the status from the previous GPU concurrently. Assuming there are four GPUs, after four cycles, each GPU receives the complete information of all GPUs. The information is then transferred in the next cycle. In contrast to the original system, the GPUs do not need to wait for a synchronous lock, which significantly increases the information update response speed.

The detailed framework is shown in Fig. 7. Our proposed S2S system can reduce the transferring frequency between the north and south bridges by using the RAM disk for fast processing and pinned memory to reduce the CPU for system resource allocation. CPUs are coordinated with multi-thread processing, and GPUs are operated in distribution computing mode. However, one major limitation of using this strategy is bus congestion caused by parallel communication and extra thread resources, i.e., disk resources cannot keep up with processing speed. The I/O test is evaluated in greater depth in the Appendix (**Disk I/O Test**).

## 2.3. Training optimizations on VHR **images**

In addition to data transfer, VHR image training optimizations are limited. As shown in Fig. 8A, there are two major trends. Some methods split the images into small blocks before saving the data to a disk for future training. This method is ROM-intensive because the small blocks require more space than one large image. It also encounters the issue of sample imbalance during training. Most images in VHR image processing have a lot of background but only a few building labels. Splitting uses all pixels for training, which results in over-fitting to the background because the background occupies the most space. Another method, similar to the one described in Fig. 8B, was suggested. For every step, the image is loaded onto the RAM disk from the ROM disk, and then a random proposal is used for the cropped image samples. This method also suffers from background over-fitting because the region proposal is completely random. Because the background occupies the most space on VHR images, they are mostly distributed on the background during sampling. Apart from the issue of over-fitting, decoding the VHR image from the disk takes time, slowing training because every image must be decoded before every step longer than 10 s. These methods employ the one-by-one strategy, which only allows for the input of one image in each step and has limited efficiency.

To solve the aforementioned issues, we proposed an optimized training procedure for VHR images, the framework of which is depicted in Fig. 9. The VHR image queue was operated at each step to decode the VHR images from the RAM disk to the transferring memory. Instead of a one-by-one strategy, we input multiple images from the RAM disk simultaneously (using multi-thread decoding). After one image is proposed, the next image is loaded from the back thread to fill the queue. This strategy utilized the CPU resource efficiently to decode the VHR images, reducing the waiting list during training. As illustrated in the U-Net section in the Appendix (**U-Net**), the GPU is unable to train the VHR image directly therefore, region proposal is required to input suitable data sizes for training. When the VHR image queue was ready, we ran the hot-zone searching step to search for areas with foreground targets. The proposal is shown in Fig. 10.

We re-sized the ground truth image to 1/100 using the close-nearest strategy; we then used the grouping strategy to search for and propose hot zones on the downsized image. We randomly proposed the images in the hot zones, which were then input into the training image queues for both the ground truth image and image. After the hot-zone is proposed, the method continues to propose other regions for background selection to a fixed training step. The GPUs load the image queues for distributed training at the same time on region proposals (see Figs. 11 and 13).

RAM and CPU resources are fully utilized throughout the proposed strategy, which greatly reduces the waiting time on GPU training sample input. Moreover, reasonable sample proposals help the trained model avoid over-fitting to the background. Note that the hot-zone searching schedule temporarily works only for foreground/background tasks.

## 2.4. Predicting in “Release” mode

In addition to training optimizations, we propose prediction optimizations. DNNs for prediction use less memory than DNNs in training, allowing for a larger batch size on each GPU. This results in a busy system bus during data transferring. Current prediction optimization strategies are very limited, resulting in slow prediction procedures. Therefore, we propose a novel strategy for predicting VHR images

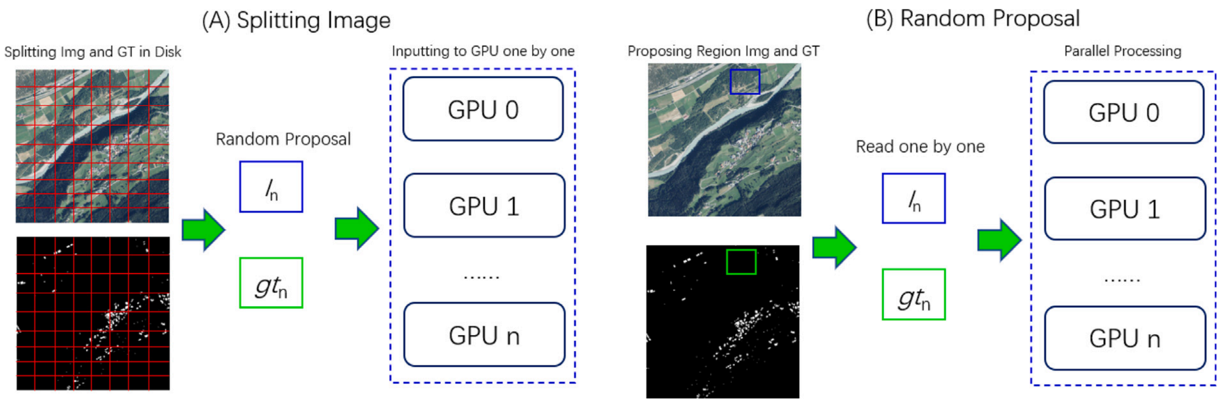


**Fig. 8.** Architecture of most common proposal method; The left panel displays random proposal, whereas the right panel displays splitting image with small blocks.

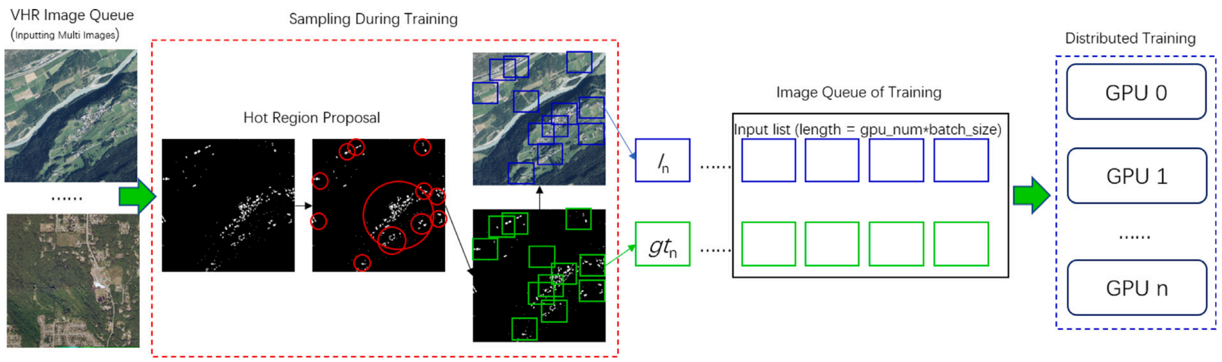


**Fig. 9.** Flow map of training optimization.

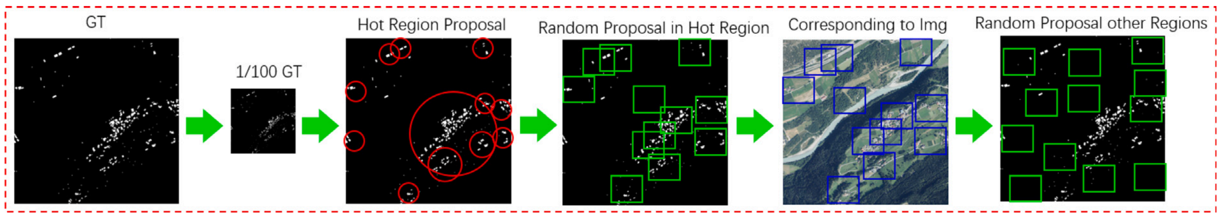


**Fig. 10.** Flow map of hot-zone searching.

in parallel GPUs. As shown in Fig. 12, the original VHR images are imported from the RAM disk to build a VHR image queue. Each image is divided into several blocks, each with a unique ID. The block is then imported into the predicting image queue. The GPUs read images from the queue and write the predicted label map to the predicted label queue. Based on the IDs, the label image copies directly back to the original VHR image in memory. Finally, after a VHR image is filled, it will be saved to the RAM disk. The major distinction between predicting and training modes is that the predicting mode includes a reward step and scans each block of VHR images. Similar to the training procedure, the VHR image queue continuously decodes images with multi-threads to avoid waiting for predictions.

Prediction eliminates the need for frequent image saving from ROM to disk with small blocks, instead requiring only the saving of a complete VHR label map to the RAM disk. Because sequence writing is faster than random writing, saving, and transferring a large image is quicker than transferring its divided blocks (see Tables 9 and 10). The reasonable architecture produces a more accurate prediction model than the original architecture.

Most methods are designed to predict images in debug mode, i.e., they only forward transfer the trained model, with no optimizations. However, several components are unnecessary and require additional computing resources during prediction, such as ReLu and batch normalization. In the proposed system, the data type is one of the most time-consuming settings. Most methods use float-32 to forward transfer, like in the training. During training, a higher predicted fraction number can provide the nodes with more powerful representation ability. However, once the DNN is covered, the data type complexity becomes marginally effectively, particularly after modifying to float-16. Therefore, using float-32 is time-consuming and results in little improvement in testing. We test the half-precise or one-quarter-precise data type in the DNN in comparison with the original float-32 data type. Aside from the data type, the DNN itself is not optimized. We combine a fixed combination, such as Conv 3 * 3 + Bias + ReLU (3 × 3 convolutional filter followed by one Bias layer and one ReLU layer), with a single component, which reduces repeated data transfer.

## 3. Evaluation

### 3.1. Experimental setup

The evaluation of the performance and efficiency of the proposed S2S system is discussed in this section. We first introduce the implementation details of the platform. Next, the evaluation of the I/O speed and performance of a general ROM disk versus a RAM disk is discussed

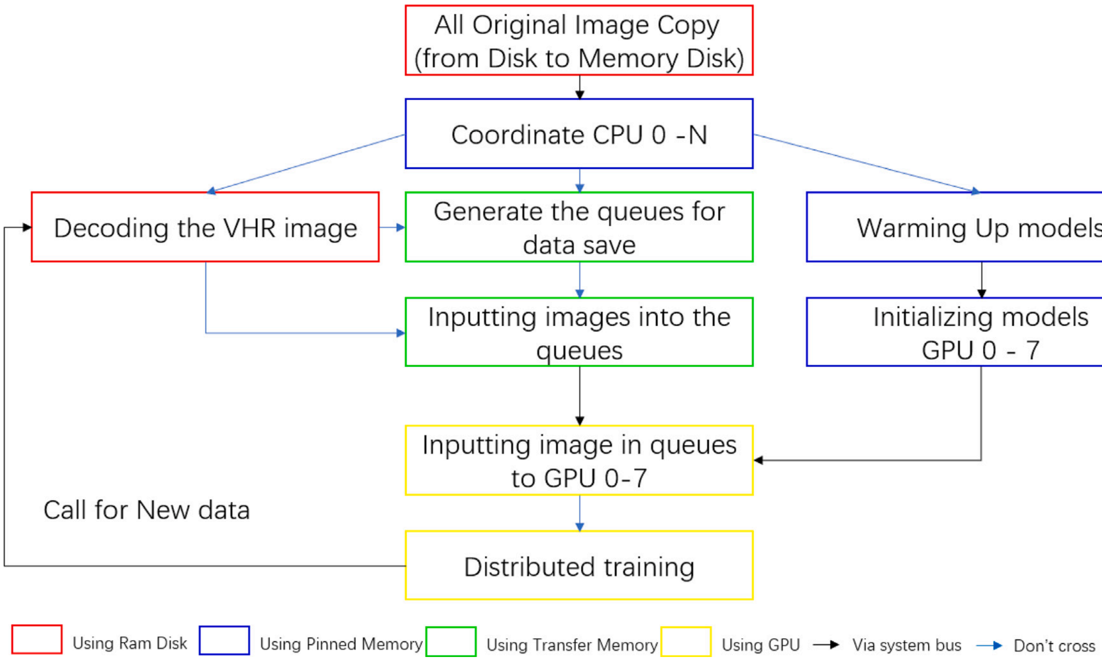


**Fig. 11.** Cost map of computer resources during training.

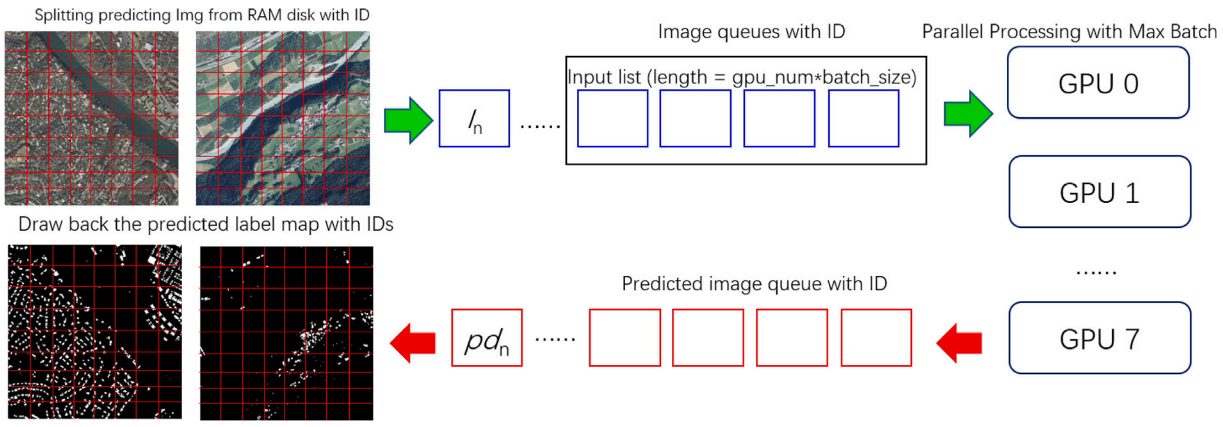

Splitting predicting Img from RAM disk with ID
Image queues with ID
Parallel Processing with Max Batch
Input list (length = gpu_num*batch_size)
$I_n$
GPU 0
GPU 1
GPU 7
Draw back the predicted label map with IDs
Predicted image queue with ID
$pd_n$

**Fig. 12.** Flow map of predicting optimization.

in various settings. Subsequently, the training speed evaluation with and without distributed deployment and of the S2S system is described. Following that, the multi-GPU test is processed in an INRIA aerial building segmentation test. Finally, a pressure test is described to evaluate the efficiency of the VHR image building segmentation over a long processing period.

### 3.1.1. *Implementation and training details*

To run the proposed system in an experiment, we used a desktop computer with an AMD Ryzen 9 5950X CPU (16score, 32 tensors), 128-GB DDR 4 Memory (four paths), and eight Titan RTX. We allocated 32 GB of RAM disk space. The platform was used for both the training and testing of the models, including those in the system. To evaluate the system, we employed the INRIA Air building segmentation dataset of (Maggiori et al., 2017), which contains 540 VHR images with a resolution is 5000 × 5000, of which covers 810 km$^2$ (405 km$^2$ for training and 405 km$^2$ for testing) in 0.3m/pixel, and MIT Roof Segmentation dataset (Chen et al., 2019) covering 457 km$^2$of orthorectified aerial images with over 220,000 buildings in 0.75 m/pixel. Based on the assessment presented in the Appendix (**Image Size versus Memory in Training and Testing**), the memory usage in the GPUs approaches the maximum value (24 GB) when using 1024 × 1024 resolution; therefore, we selected 1024 × 1024 as the input resolution. Subsequently, eight GPUs were used in tandem to train the DNN. The ADAM (adaptive moment estimation) optimizer with 0.00025 initial learning rate was employed to train the dataset. The loss function was defined by the cross-entropy between the ground truth and the segmentation masks.

### 3.1.2. *Training detail*

To evaluate the efficiency of the S2S and distributed systems during training, we tracked 9 rounds of training and compared them with the original parallel training settings.

The results in Table 1 illustrate that with the S2S architecture, training optimization, and distribution deployment, the speed approached its maximum, i.e., 2.1 s/step, in a simple step. However, there was no accuracy loss or delay in the coverage (all methods had coverage of eight epochs). With the single S2S system, the training speed in a single step was also faster than the parallel processing deployment at 3.4 s/step. However, when only distribution deployment was used, the speed dropped to 0.3 s. As a result, the total training time was reduced from 12.3 h to 5.0 h for INRIA and from 40.8 h to 13.5 h for MITAIRS. Moreover, the TO contributes to the DP, S2S, and distribution

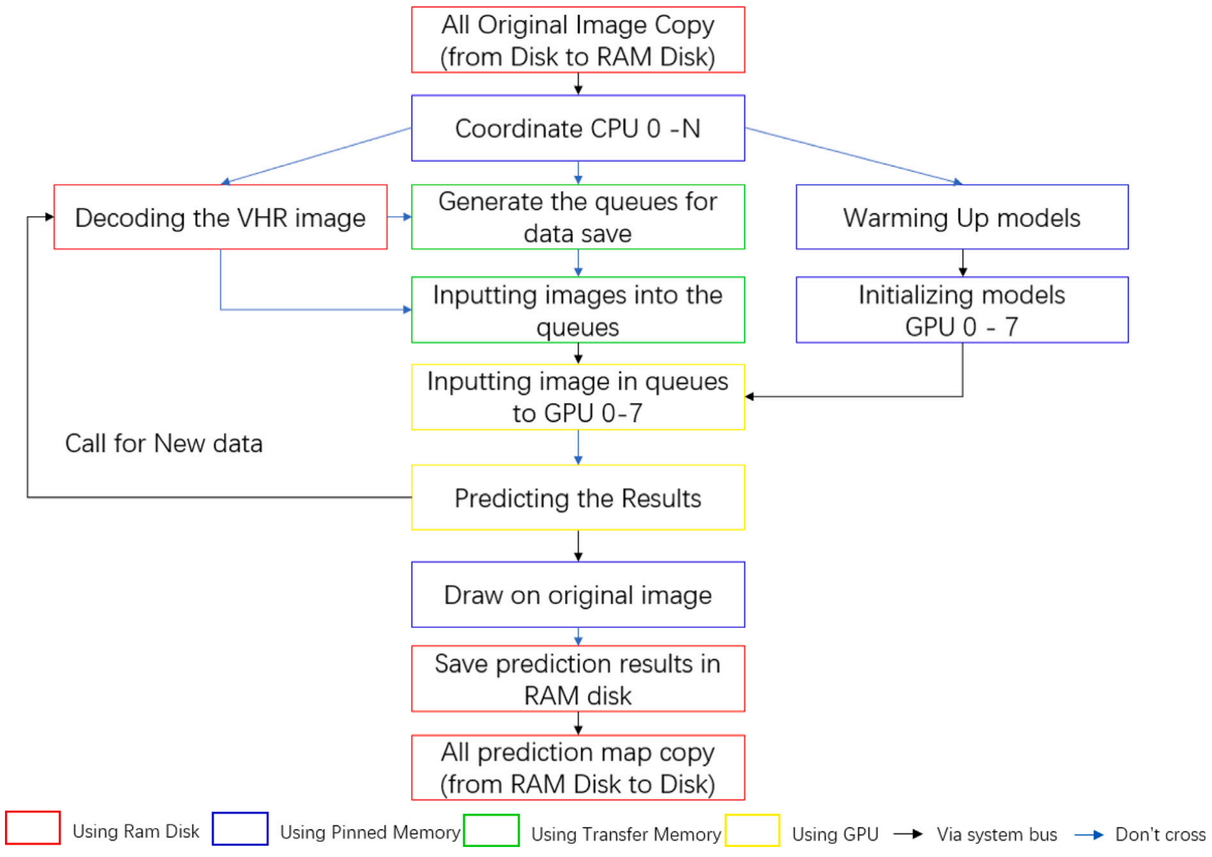

All Original Image Copy (from Disk to RAM Disk)
Coordinate CPU 0 -N
Decoding the VHR image
Generate the queues for data save
Warming Up models
Inputting images into the queues
Initializing models GPU 0 - 7
Inputting image in queues to GPU 0-7
Call for New data
Predicting the Results
Draw on original image
Save prediction results in RAM disk
All prediction map copy (from RAM Disk to Disk)
Using Ram Disk
Using Pinned Memory
Using Transfer Memory
Using GPU
Via system bus
Don't cross

**Fig. 13.** Cost map of computer resources during prediction.

**Table 1**
Speed and accuracy of training in different settings.

| Item | Accuracy INRIA (%) | Accuracy MIT (%) | Epoch | Time/Step | Total time INRIA | Total time MIT |
|---|---|---|---|---|---|---|
| DP | 67.43 | 87.21 | 8 | 4.8 s | 12.3 h | 40.8 h |
| DP with TO | 72.46 | 95.30 | 8 | 4.1 s | 11.3 h | 37.5 h |
| S2S | 67.49 | 86.21 | 8 | 3.4 s | 9.6 h | 34.5 h |
| S2S with TO | 72.46 | 95.18 | 8 | 3.0 s | 8.6 h | 30.2 h |
| Distribution | 68.32 | 87.35 | 8 | 4.3 s | 10.4 h | 24.6 h |
| Distribution with TO | 72.44 | 95.32 | 8 | 3.4 s | 8.4 h | 20.8 h |
| S2S+Distribution | 67.28 | 87.41 | 8 | 3.0 s | 7.2 h | 17.6 h |
| S2S+Distribution+50 epochs | 68.05 | 86.38 | 50 | 3.0 s | 45.2 h | 110.5 h |
| S2S+Distribution with TO | 72.51 | 95.50 | 8 | 2.1 s | 5.0 h | 13.5 h |

TO means training optimization (shown on Fig. 7); DP means original training architecture (shown on Fig. 5).

deployment with a speed improvement by one-fifth as TO assigns the computer resources more reasonably to reduce the hang time on training. It also has 5% accuracy in INRIA and 8% accuracy in MIT because more reasonable region proposal causes DNN coverage to be faster than that without optimizations. To prove that more training epochs do not influence the final accuracy, we also employ a training “S2S + Distribution + 50 epochs”. The results show that the accuracy remains at 68.05% and 86.38% in INRIA and MITAIRS datasets, respectively, with only minor improvements. The system bus congestion was the main cause of the improved performance of the S2S system. Because multiple GPUs could occupy the system bus in the system I/O, the bandwidth on the desktop was consumed, and frequent write/reads using the ROM disk resulted in long latency. The speed of random I/O on the desktop is insufficient to provide the necessary GPU processing capacity, resulting in low speeds in the original distributed system compared to the S2S system. We found that a high transfer speed and a low amount of data transfer via the system bus increased the training speed significantly. The Appendix (**Disk I/O Test**) has discussed the influence from disk type and proved the RAM disk can effectively speed up the data I/O even GPU is maximally occupied. In the following section, we present the evaluation of the factors influencing the I/O speed and accuracy.

### *3.2. Memory and speed cost*

As the RAM disk advances during full occupation of the GPUs, the exact enhancements of accuracy and efficiency during prediction also required evaluation. Because prediction is less memory consuming than training, the batch size can be enlarged to its maximum.

According to the Appendix (**Image Size versus Memory in Training and Testing**), the inputting image size larger than 2048 × 2048 makes the memory cost larger than maximum on GPU RAM (24 GB), blindly increasing the inputting image size leads the failure on system operation. Based on the data summarized in Table 2, we selected four batch/GPUs in the original setting (no optimization), five batch/GPUs in the float-32 format, seven batch/GPUs in the float-16 format, and 11 batch/GPUs in the Int-8 format. Subsequently, the accuracy changes on inference optimizations and speed variations using different numbers of GPUs were evaluated. Furthermore, we assessed the accuracies of the float-32, float-16, and Int-8 data types. The speed of predicting

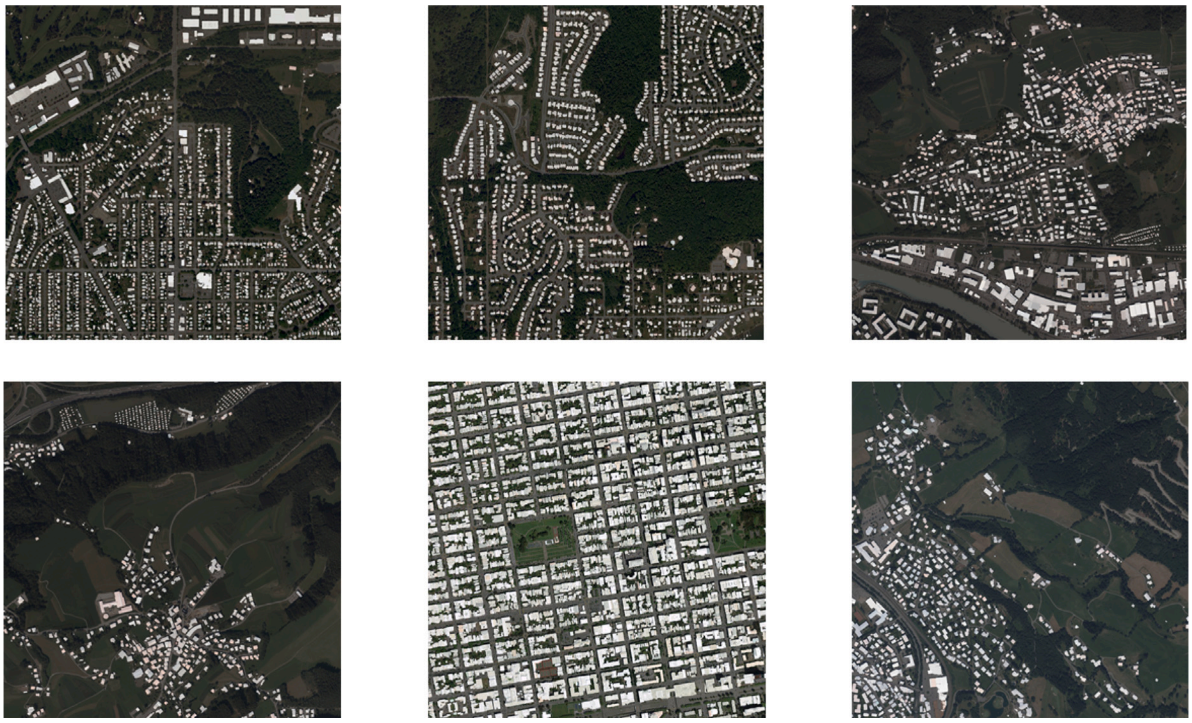


**Fig. 14.** Visualization of results using U-Net in INRIA dataset. We set the alpha index (60%) on label map and make it mask on original image for visualization.

**Table 2**
Memory cost for related items.

| Items | Memory cost |
|---|---|
| U-Net No Optimization | 6.1 GB |
| U-Net Float 32 | 4.7 GB |
| U-Net Float 16 | 3.5 GB |
| U-Net Int 8 | 2.4 GB |
| Per Image in RAM (INRIA average) | 136 MB |
| Per Label Map in RAM (INRIA average) | 128 MB |
| Per Image in RAM (MIT average) | 636 MB |
| Per Label Map in RAM (MIT average) | 592 MB |
| Per Image in RAM Disk (INRIA average) | 43 MB |
| Per Label Map in RAM Disk (INRIA average) | 151 KB |
| Per Image in RAM Disk (MIT average) | 119 MB |
| Per Label Map in RAM Disk (MIT average) | 483 KB |

the INRIA dataset using different number of GPUs were evaluated, as were the speeds using float-32, float-16, and Int-8 data formats with eight GPUs. The above settings were tested using different ROM/RAM disks. Note that the before operation and after operation times were also counted in the experiment; it required 1.5 s to copy the INRIA dataset to RAM, 0.6 s to write the INRIA results, 1.6 s to copy the MITAIRS dataset, and 1.0 s to write the MIT test dataset. Aside from GPU memory costs, we also summarized the memory costs. The RAM memory requirements for RGB images and label maps are similar: 136 and 128 mb per image/map on average in the INRIA dataset, respectively. Since image size in the MITAIRS dataset is larger than that the in INRIA dataset, the RAM memory costs are higher, at 636 and 592 MB, respectively. The amount of storage for the images is less than that in RAM. The storage costs of images in the INRIA and MITAIRS datasets are 43 and 119 MB per image, respectively. Label maps have storage costs of 151 and 483 KB, respectively. We allocated 32 GB for RAM disk because the total size of the INRIA and MITAIRS datasets is around 24 GB, and 8 GB is sufficient for saving label maps.

In Tables 3 and 4, the PO indicates that the DNN is used for prediction. The results show that the accuracy does not change when using different disks. The original version, as well as the optimized float-32 data type, achieve a mean intersection over union (MIoU) accuracy of 72.5%. When using the float-16 format, the MIoU accuracy decreases by 0.2% in the INRIA dataset, and by 95.50% in the MITAIRS dataset with a 1% accuracy loss. MIoU accuracy decreases to 67.9% and 89.20% when Int-8 is used. The ROM disks are unable to accelerate multi-GPU parallel processing when inference optimization settings are enabled or disabled. The speed cannot increase with the increase in GPUs. The program uses hybrid hard disk (HHD) and runs in 1142 s and 1095 s in INRIA and MIT in the original parallel processing settings, respectively, and in 1090 s/ using the inference optimizations. The SSD (Sata3, NVME, and PCIE) achieves comparable speeds. The speed (eight GPUs) is approximately 540 s for INRIA and 830 s for MITAIRS, with inference optimizations of approximately 490 and 820 s for float-32, 450 and 790 s for float-16, and 440 and 853 s for Int-8, respectively. However, the speed with eight GPUs (540 s) is similar to that with 2/4 GPUs (580/520 s in INRIA and 1000/860 s in MITAIRS). The results show that system bus congestion occurs during operation and limits prediction speed as the number of GPU increases.

In contrast, RAM disks with different path numbers have demonstrated that even with a single path, the influence of system bus content is significantly less than that of ROM disks. When using the RAM disk, the speed using four GPUs increases to approximately 240 s in INRIA and 292 in MIT; when using eight GPU, it increases to approximately 110/207 s. When using inference optimizations and float-16, it accelerates to 74/122 and 38/83 s, respectively. This is significantly faster than that of the ROM disk. Due to the reduced transfer volume on the system bus, the S2S system can ensure high-speed I/O even with several GPUs in the bandwidth content when the south bridge is busy. The results show that the S2S system can effectively enhance the multi-GPU parallel processing of the DNN on the VHR images. With prediction optimization, the VHR image prediction speed on increased significantly (3.2 times quicker with limited accuracy loss).

Figs. 14 and 15 depict the visualization of INRIA and MITAIRS datasets. Significantly, the proposed method with training optimization has no accuracy loss and successfully segments buildings of various sizes with sharp boundaries. More detailed images can be found in the Appendix (**Link for Original Images**).

**Table 3**
Accuracy and speed of proposed architecture in predicting in INRIA dataset. The first column means the type of storage device. The 2–4 columns mean the MIoU accuracies. The speed means the time process the whole dataset.

| Type | Acc INRIA | Float 32 (PO) | Float 16 (PO) | Int 8 (PO) | Speed (1GPU) | Speed (2GPU) | Speed (4GPU) | Speed (8GPU) | Speed 32(PO) | Speed 16(PO) | Speed 8(PO) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HHD | 72.5% | 72.5% | 72.3% | 67.9% | 1325 s | 968 s | 1069 s | 1142 s | 863 s | 792 s | 750 s |
| Sata 3 | 72.5% | 72.5% | 72.3% | 67.9% | 846 s | 582 s | 519 s | 542 s | 495 s | 454 s | 438 s |
| NVME | 72.5% | 72.5% | 72.3% | 67.9% | 852 s | 585 s | 524 s | 540 s | 496 s | 448 s | 441 s |
| PCIE | 72.5% | 72.5% | 72.3% | 67.9% | 860 s | 583 s | 519 s | 534 s | 492 s | 450 s | 437 s |
| RAM 1 | 72.5% | 72.5% | 72.3% | 67.9% | 842 s | 440 s | 244 s | 111 s | 74 s | 38 s | 25 s |
| RAM 2 | 72.5% | 72.5% | 72.3% | 67.9% | 840 s | 418 s | 242 s | 109 s | 74 s | 38 s | 27 s |
| RAM 4 | 72.5% | 72.5% | 72.3% | 67.9% | 840 s | 418 s | 242 s | 109 s | 72 s | 38 s | 27 s |

**Table 4**
Accuracy and speed of proposed architecture in predicting in MIT dataset. The first column means the type of storage device. The 2–4 columns mean the MIoU accuracies. The speed means the time process the whole dataset.

| Type | Acc MIT | Float 32(PO) | Float 16(PO) | Int 8(PO) | Speed (1GPU) | Speed (2GPU) | Speed (4GPU) | Speed (8GPU) | Speed 32(PO) | Speed 16(PO) | Speed 8(PO) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HHD | 95.50% | 95.50% | 94.18% | 89.20% | 1582 s | 1285 s | 1070 s | 1095 s | 1090 s | 850 s | 853 s |
| Sata 3 | 95.50% | 95.50% | 94.18% | 89.20% | 1542 s | 1095 s | 864 s | 841 s | 820 s | 794 s | 844 s |
| NVME | 95.50% | 95.50% | 94.18% | 89.20% | 1498 s | 990 s | 831 s | 825 s | 805 s | 775 s | 822 s |
| PCIE | 95.50% | 95.50% | 94.18% | 89.20% | 1478 s | 1025 s | 850 s | 818 s | 791 s | 780 s | 818 s |
| RAM 1 | 95.50% | 95.50% | 94.18% | 89.20% | 1182 s | 540 s | 294 s | 209 s | 128 s | 87 s | 64 s |
| RAM 2 | 95.50% | 95.50% | 94.18% | 89.20% | 1192 s | 518 s | 298 s | 220 s | 120 s | 88 s | 62 s |
| RAM 4 | 95.50% | 95.50% | 94.18% | 89.20% | 1180 s | 497 s | 287 s | 207 s | 122 s | 83 s | 61 s |

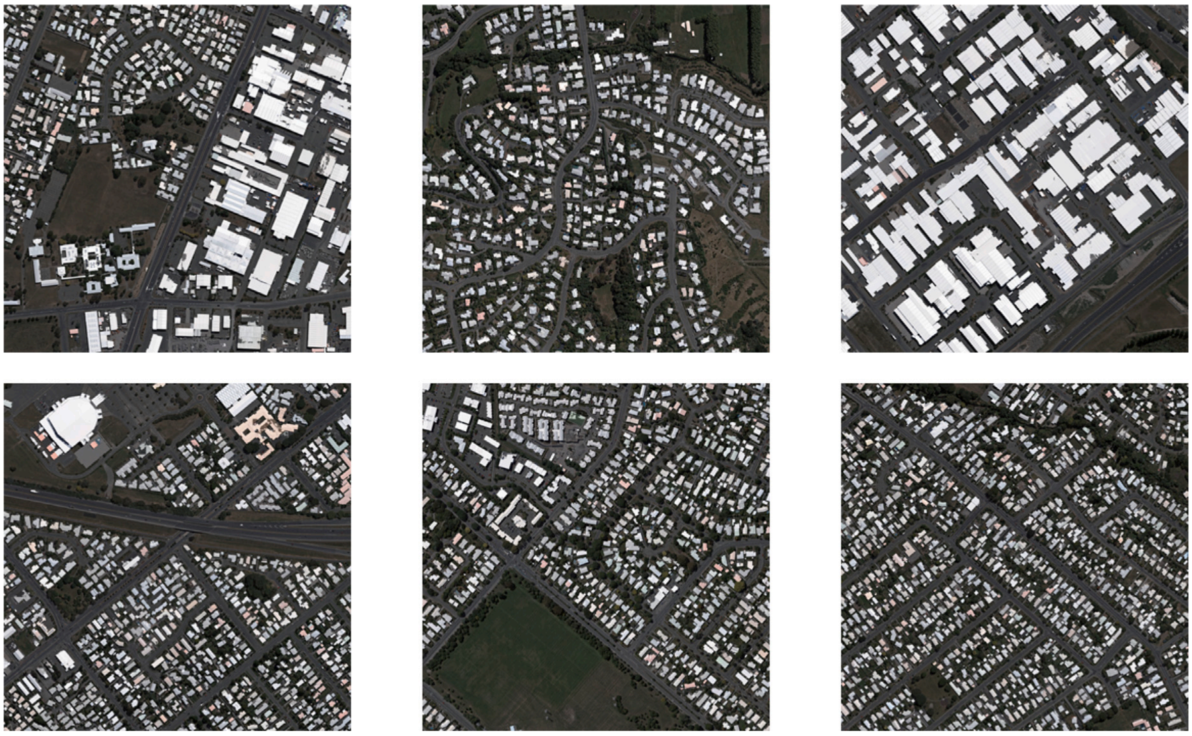


**Fig. 15.** Visualization of results using U-Net in MIT dataset. We set the alpha index (60%) on label map and make it mask on original image for visualization.

### 3.3. Comparison with real time DNNs

The comparison of U-NET DNNs and real-time semantic segmentation neural networks is presented in this section. We compared three DNNs namely Bise-Net (Yu et al., 2018), **Deeplab_V3** (Chen et al., 2018) **and FCNet** (Riaz et al., 2017). **The efficiency evaluation is shown as below:** (see Table 5)

Because real-time DNNs for semantic segmentation typically occupy a small number of neural nodes, they have less encoding and decoding ability than other semantic segmentation DNNs. Therefore, their accuracies are significantly lower than that of U-Net (72.5%), at 58.6% for Bise-Net, 62.3% for Deeplab_V3, and 54.8% for FCNet. Their processing speed when using multi-GPU and inference optimizations follows the same trend as U-Net, i.e., only using the proposed architecture with RAM disk can maximize the computing resources, and the processing speed grows by a factor of eight for all DNNs when using eight GPUs. However, the maximum speed (82 s for Bise-Net, 64 s for DeeplabV3, and 57 s for FCNet) is not significantly faster than U-Net (109 s). Therefore, the results show that the S2S system can speed up real-time semantic segmentation without accuracy loss.

### 3.4. Evaluation on different inputting image size

In addition to using 1024 × 1024 as the DNN input image size, we compare the differences in accuracy when using various input image sizes.

**Table 5**
Accuracies of real-time DNNs in INRIA dataset. The items with HDD and PCIE applied on real-time DNNs operate without S2S system.

| Type | Acc F32 | Acc F16 | Acc Int 8 | 1 GPU | 2 GPU | 4 GPU | 8 GPU |
|---|---|---|---|---|---|---|---|
| Bise-Net HDD | 58.6 | 56.3 | 47.5 | 823 s | 658 s | 567 s | 558 s |
| Bise-Net PCIE | 58.6 | 56.3 | 47.5 | 641 s | 548 s | 325 s | 328 s |
| Bise-Net RAM 4 | 58.6 | 56.3 | 47.5 | 630 s | 256 s | 134 s | 82 s |
| DeeplabV3 HDD | 62.3 | 61.5 | 54.3 | 658 s | 522 s | 481 s | 480 s |
| DeeplabV3 PCIE | 62.3 | 61.5 | 54.3 | 552 s | 321 s | 301 s | 298 s |
| DeeplabV3 RAM 4 | 62.3 | 61.5 | 54.3 | 482 s | 242 s | 118 s | 64 s |
| FCNet HDD | 54.8 | 52.5 | 45.6 | 582 s | 482 s | 475 s | 466 s |
| FCNet PCIE | 54.8 | 52.5 | 45.6 | 450 s | 324 s | 318 s | 310 s |
| FCNet RAM 4 | 54.8 | 52.5 | 45.6 | 368 s | 185 s | 105 s | 57 s |
| U-Net HDD | 72.5 | 72.3 | 67.9 | 1325 s | 968 s | 1069 s | 1142 s |
| U-Net PCIE | 72.5 | 72.3 | 67.9 | 860 s | 583 s | 519 s | 534 s |
| U-Net RAM 4 | 72.5 | 72.3 | 67.9 | 840 s | 418 s | 242 s | 109 s |

**Table 6**
Accuracy vs input image size in INRIA (s means second, speed means how many time to process the whole dataset).

| Inputting image size | Speed in PCIE | Speed in RAM 4 | Accuracy |
|---|---|---|---|
| 64 × 64 | 1032 s | 18 s | 43.3 |
| 128 × 128 | 1018 s | 26 s | 51.8 |
| 256 × 256 | 542 s | 43 s | 64.5 |
| 512 × 512 | 502 s | 72 s | 69.2 |
| 1024 × 1024 | 540 s | 109 s | 72.5 |

**Table 7**
Total time of pressure test in different settings.

| Type | HDD | SSD | 1 path | 2 path | 4 path |
|---|---|---|---|---|---|
| Float 32 | 43.2 h | 18.2 h | 2.5 h | 2.3 h | 1.8 h |
| Float 16 | 35.8 h | 15.9 h | 59 min | 48 min | 43 m |

The results are displayed in Table 6. Although DNNs with small input image sizes have faster processing speeds, their accuracy significantly decreases to 43.3% at 64 × 64, 51.8% at 128 × 128, 64.5% at 256 × 256, and 69.2% at 512 × 512. The processing speed of a PCIE disk is much slower than that of a RAM disk, especially with small input image sizes (1032 s vs 18 s at 64 × 64 and 1018 s vs 26 s at 128 × 128, respectively). The low speed is due to the small split size, which increases the frequency of I/O. However, the RAM processing speed shows significant enhancement. Because the RAM disk can perform over 2 GB/s for random I/O even with the maximum GPU operation in 8 GPUs, these results illustrate that the S2S architecture is capable of serving for a larger number of GPU to parallel processing. The low speed when using 1024 × 1024 is primarily due to the increased complexity of DNN as $n^2$increases along with the image size. In terms of accuracy, low accuracies are primarily caused by frequent board cross when using small input image sizes. Therefore, with fewer DNN parameters, the processing speed for small input image sizes is significantly lower than that for 1024 × 1024, but the accuracies have a significant influence. However, because accuracy should be considered, we recommend using a 1024 × 1024 input image size to maintain the balance between accuracy and speed.

### *3.5. Pressure test*

Next, we compare long-term prediction using the S2S system and the ROM disk. We run the INRIA dataset test 1000 times, with 18,000 VHR images with a size of 5000 × 5000 to determine the speed.

Notably, the S2S system performed in approximately 1.8 h with the float-32 data type and 43 min with the float-16 data type, which is over 10 and 20 times faster than the HDD and SSD, respectively. The results show that the S2S system is stable over long-term data transfer. Because the INRIA dataset has a resolution of 0.3 m/pixel, the pressure test shows that it can process a 40,500 km$^2$ area in just 43 min using float-16. The same region is processed in days (15.9 h using a PCIE disk) using conventional methods the processing time was drastically reduced in the present study. This would change the urban information update from a weekly to hourly scale. The size of VHR images increased significantly along with the increase in resolution on satellite photos. Assume, for instance, that one image has a resolution of 0.3 m/pixel and another has a resolution of 0.05 m/pixel. Although the second VHR image has better clarity than the first, it requires 36 times as many pixels to depict the same-sized region in reality (0.3/0.05)2. As satellite technology advances, the resolution of VHR images will undoubtedly rise in the future. Therefore, architecture that can manage larger data volumes and higher VHR image resolutions, such as parallel processing, will become increasingly necessary.

## 4. Conclusion

The proposed S2S system can accelerate DNN operation in VHR images and evaluates the efficiency using building segmentation. The I/O test evaluates the limitation of the ROM disk using frequent random I/Os, and the RAM disk used in the S2S system processes over 2000 mb/s, which is 20 times faster than the RAM disk in the original system. The distributed GPU setting of the S2S system enables multi-GPU training from 12 to 5 h, and the accuracies become similar. Our results prove that the proposed system accelerates the prediction time by a factor of 11 (from 450 s to 38 s) in the INRIA dataset and by a factor of 9.4 (from 780 s to 83 s) in the MITAIRS dataset using half the data precision.

The pressure test demonstrates that the proposed S2S system can be maintained for high-intensity long-term tasks, reducing operation time from 18.2 to 1.8 h for the float-32 data type and 43 m for the float-16 data type. The overall evaluation proves that the S2S system provides a rapid DNN operation environment without accuracy loss. Because the system optimizes the OS without changing the neural network, it can be easily converted into other DNNs, and we plan to extend our system to other tasks in the future.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant 41871325.

## Appendix

**Link for Original Images:**
Link:https://pan.baidu.com/s/1FBytuHf_jM-w7MkzvuzCEg

Code: mlha
**U-Net**

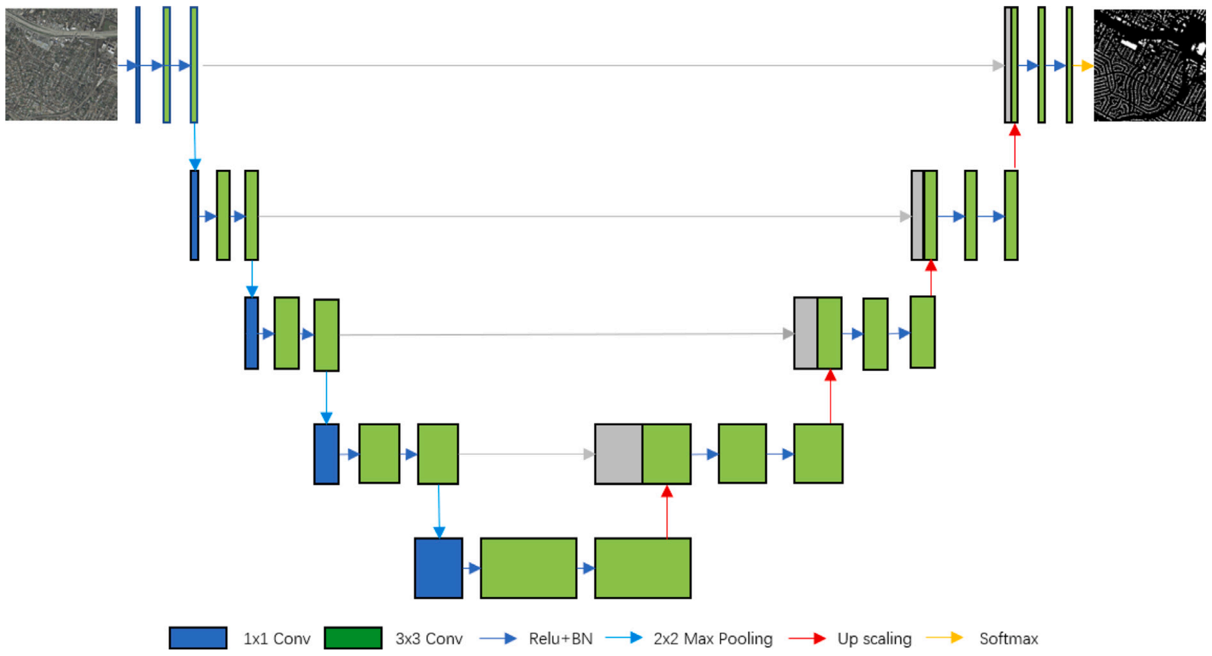


**Fig. 16.** The architecture of U-Net.

In this study, we employ U-Net as a fundamental DNN to evaluate the performance and efficiency of the proposed S2S system. U-Net was originally designed for cell segmentation in biomedical images and has been demonstrated to effectively learn with limited image samples and perform well in binary classification detection. The classical U-Net architecture is illustrated in Fig. 16. The DNN operates a down-sampling encoding phase, followed by an up-sampling decoding phase. The path of the encoder extracts features from the input images to capture context information from the spatial features. The decoder path groups and analyzes the features obtained from the encoder. The encoder commences with two chained $3 \times 3 \times 64$ filters. Subsequently, the feature map is down-sampled by a $2 \times 2$ max pooling operation with a stride of 2. The sequence is repeated four times to down-sample the size to 1/16 to capture high-level features. At each scale, the number of convolutional filters is doubled. After the DNN reaches the bottom, the feature map is up-scaled using a sequence of two $3 \times 3$ convolutional layers, followed by a $2 \times 2$ transposed convolution operation. The outputs during the encoder path, which are at the same size and level, are concatenated with an up-scale decoder of the same scale (skipping connection). The combination is repeated four times to recover the original input size. The final segmentation is achieved by a $1 \times 1$ convolution whose filter is the same as for the output classification. All convolution layers utilize the ReLU activation function and batch normalization. The last convolution layer uses a sigmoid activation function.

In this section, we present the assessment of the structure of a U-Net model based on its relationship with the memory requirements for segmentation. We assume that there exist L levels in the down/up-scale paths, (i.e., $l = \{0, 1, 2 \ldots \ldots L-1\}$) and the number of convolutional layers is $n_c$. The number of filters at level $l$ is defined as $n_f^l$. In the proposed U-Net, filter size K, padding size P, stride step s, and number of convolutional layers $n_c$ remain unchanged. Moreover, a $2 \times 2$ max pooling is employed to connect one encoder level to another, and a $2 \times 2$ up-convolutional layer connects one decoder level to the next.

In the U-Net architecture, the number of filters doubles as the level becomes deeper. Therefore, the number of filters at each level can be defined as $n_f^l = 2^{l-1} n_f$. $I_l$ denotes the feature size during the encoding path at level $l$. The corresponding layer for the decoder path is defined as $J_l$. The memory requirements of the neural network for a $1024 \times 1024$ sample are listed in Table 7.

**Image Size versus Memory in Training and Testing**

In this architecture, the memory cost is very high because the number of nodes increases as $n^2$ as the width/height of the images increases.

Owing to the property of satellite images, semantic segmentation in satellite image processes has a much higher resolution than general image segmentation. Generally, the GPU memory cost increases as $n^2$ with the increase in resolution. The relationship between the proposed U-Net and the GPU memory cost is shown in Fig. 17 (see Table 8).

As shown in Fig. 17, training U-Net for building segmentation consumes a very large GPU memory for storing the nodes of the DNN, particularly when the image resolution is high. The GPU memory cost in training U-Net is larger than that in testing. This is because of the consistency in the training procedure. The training procedure contains both backward and forward transfer, an optimizer (e.g., Stochastic gradient descent and adaptive moment estimation (ADAM)), and data type (generally, float 32) in the node. However, testing the DNN uses only forward transfer. Therefore, the GPU memory during training is significantly larger than that during testing. Not only in training U-Net (58.2 GB) but also in testing (18.2 GB), the cost of RAM cannot be processed by a modern server when the resolution is higher than 2048 $\times$ 2048. Both down-sampling images with large strides and reducing the model depth can reduce the memory requirement; however, the change will also decrease the accuracy of the resulting model because of the smaller receptive field.

**Disk I/O Test**

In this section, we discuss the assessment of the read and writing ability of the ROM and RAM disk. We select four evaluation factors: sequence writing/reading and random writing/reading. Sequence I/O refers to large-piece file transferring (e.g., a VHR image over 200 mb). Random I/O refers to the transfer of a small-piece file (e.g., a series of label files whose sizes are less than 20 kb). Alternatively, during the prediction operation, the south bridge (PCIE ports and ROM disk ports) is heavily occupied, which causes the scramble system bus during prediction, particularly when using multiple GPUs. Therefore, we test two scenarios: (1) test of the ROM/RAM disks without other operation. (2) test of the ROM/RAM disks when the GPUs are fully operated. As

**Table 8**
Parameter of U-Net with 1024 × 1024 × 3 as input.

| Layer | Parameters | Memory |
|---|---|---|
| Input [1024 × 1024 × 3] | 0 | 3.14 M |
| Conv3_64 [1024 × 1024 × 64] | 1728 [(3 × 3 × 3) × 64] | 67.1 M |
| Conv3_64 [1024 × 1024 × 64] | 36864 [(3 × 3 × 64) × 64] | 67.1 M |
| Pooling2 [512 × 512 × 64] | 0 | 16.77 M |
| Conv3_128 [512 × 512 × 128] | 73728 [(3 × 3 × 64) × 128] | 33.55 M |
| Conv3_128 [512 × 512 × 128] | 147456 [(3 × 3 × 128) × 128] | 33.55 M |
| Pooling2 [256 × 256 × 128] | 0 | 8.38 M |
| Conv3_256 [256 × 256 × 256] | 295912 [(3 × 3 × 128) × 256] | 16.77 M |
| Conv3_256 [256 × 256 × 256] | 589824 [(3 × 3 × 256) × 256] | 16.77 M |
| Pooling2 [128 × 128 × 256] | 0 | 4.19 M |
| Conv3_512 [128 × 128 × 512] | 1179648 [(3 × 3 × 256) × 512] | 8.38 M |
| Conv3_512 [128 × 128 × 512] | 2359296 [(3 × 3 × 512) × 512] | 8.38 M |
| Pooling2 [64 × 64 × 512] | 0 | 2.09 M |
| Conv3_1024 [64 × 64 × 1024] | 4718592 [(3 × 3 × 512) × 1024] | 4.19 M |
| Conv3_1024 [64 × 64 × 1024] | 9437184 [(3 × 3 × 1024) × 1024] | 8.38 M |
| TransConv2 [128 × 128 × 512] | 4718592 [(3 × 3 × 1024) × 512] | 8.38 M |
| Concat [128 × 128 × 512] × 2 | 0 | 16.77 M |
| Conv3_512 [128 × 128 × 512] | 4718592 [(3 × 3 × 1024) × 512] | 8.38 M |
| Conv3_512 [128 × 128 × 512] | 2359296 [(3 × 3 × 512) × 512] | 8.38 M |
| TransConv2 [256 × 256 × 256] | 1179648 [(3 × 3 × 512) × 256] | 16.77 M |
| Concat [256 × 256 × 256] × 2 | 0 | 33.55 M |
| Conv3_256 [256 × 256 × 256] | 1179648 [(3 × 3 × 512) × 256] | 16.77 M |
| Conv3_256 [256 × 256 × 256] | 589824 [(3 × 3 × 256) × 256] | 16.77 M |
| TransConv2 [512 × 512 × 128] | 295912 [(3 × 3 × 256) × 128] | 33.55 M |
| Concat [512 × 512 × 128] × 2 | 0 | 67.10 M |
| Conv3_128 [512 × 512 × 128] | 295912 [(3 × 3 × 256) × 128] | 33.55 M |
| Conv3_128 [512 × 512 × 128] | 147456 [(3 × 3 × 128) × 128] | 33.55 M |
| TransConv2 [1024 × 1024 × 64] | 73728 [(3 × 3 × 128) × 64] | 67.10 M |
| Concat [1024 × 1024 × 64] × 2 | 0 | 134.3 M |
| Conv3-64 [1024 × 1024 × 64] | 73728 [(3 × 3 × 128) × 64] | 67.10 M |
| Conv1-Output [1024 × 1024 × c] | 64 c [(1 × 1 × 64) × c] | 1.04 M × C |

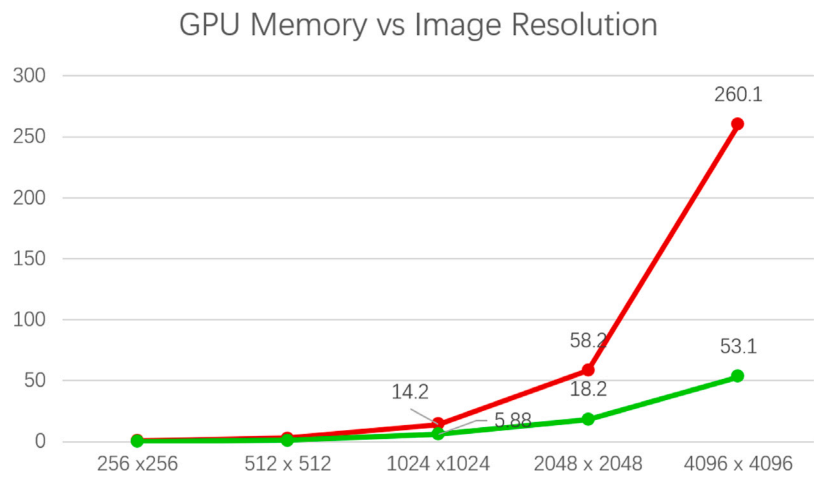


**Fig. 17.** GPU memory consumed by U-Net for different resolutions.Red line represents training, and green line represents testing. The $x$ axis is image size of inputting DNN, the $y$ axis is the memory cost (GB). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

hardware, we use a Wild Data HHD (Hard Disk Drive) disk, Samsung SSD for Sata 3, NVME (Non-volatile Memory Express), and PCIE for evaluation. For the RAM selection, we select a four-path main board with four 32-GB DDR4 4000 memory. To adapt to different settings, the RAM is blocked in a selected path, and the frequency of the RAM is set in the BIOS.

Based on the results summarized in Table 9, when testing without pressure from the GPUs, the HHD ROM disk (7200/s) writes and reads at very low speeds in the sequence I/O at 184.35 mb/s and 190.42 mb/s, random I/O at 18.42 mb/s and 12.28 mb/s. The SSD operates significantly faster than the HHD, particularly the PCIE SSD, which can operate with approximately 3GB/s in sequence I/O, and 600 mb/s in random I/O. The RAM disk, even under the lowest setting (1 path, 2400 MHz), outperforms the SSD disk, whose sequence I/Os are 5032.28 mb/s and 9685.23 MB/s, respectively, random I/Os are 668.65 mb/s and 857.80 mb/s, respectively. With path and frequency increase, the I/O ability significantly increases to sequence I/Os of 21425.32 and 32418.43 mb/s, respectively, and random I/Os of 2073.45 and 2283.93 mb/s, respectively, when using a four-path 4000 Hz memory.

Based on the results in Table 10, under GPU full resource operation, the GPUs, HHD, and SSD perform significantly lower than without the pressure from the GPUs, which can only sequence I/O of approximately 350 mb/s and random I/O approximately 50 mb/s. For the RAM Disk, there is a very limited loss of the speed, and it remains 21135.90mb/s and 31948.80mb/s as the sequence I/O and 2025.60mb/s and 2248.71 mb/s in random I/O.

These results are expected. Regardless of the HDD or SSD using the NVME, Sata 3, or PCIE ports, they occupy the external importer in the south bridge. In a daily operation, it is not significantly influenced when a single GPU is operating. However, when multiple GPUs are

**Table 9**
Speed of I/O test without GPU operation.

| Device\Test Item (mb/s) | Seq writing | Seq reading | Random writing | Random reading |
|---|---|---|---|---|
| HHD (7200 /s) | 184.35 | 190.42 | 12.28 | 18.42 |
| SSD (Sata 3) | 550.42 | 668.11 | 125.83 | 146.67 |
| SSD (NVME) | 2152.42 | 1967.88 | 348.70 | 405.37 |
| SSD (PCIE) | 3124.12 | 2871.60 | 660.70 | 693.48 |
| RAM Disk 1 path 2400 | 9685.23 | 5032.28 | 857.80 | 668.65 |
| RAM Disk 1 path 3200 | 11432.80 | 5848.15 | 1043.17 | 785.20 |
| RAM Disk 1 path 4000 | 12678.79 | 7527.51 | 1123.52 | 846.30 |
| RAM Disk 2 path 2400 | 14113.70 | 10324.28 | 953.81 | 827.88 |
| RAM Disk 2 path 3200 | 16532.80 | 11848.15 | 1168.73 | 860.50 |
| RAM Disk 2 path 4000 | 18762.79 | 13041.28 | 1303.87 | 979.74 |
| RAM Disk 4 Path 2400 | 19025.91 | 14368.32 | 1570.35 | 1254.69 |
| RAM Disk 4 path 3200 | 24562.49 | 17732.75 | 2049.61 | 1896.81 |
| RAM Disk 4 path 4000 | 32418.43 | 21425.32 | 2283.93 | 2073.45 |

**Table 10**
Speed of I/O test with max GPU operation.

| Device\Test Item (mb/s) | Seq writing | Seq reading | Random writing | Random reading |
|---|---|---|---|---|
| HHD (7200 /s) | 82.54 | 76.27 | 9.87 | 11.63 |
| SSD (Sata 3) | 380.48 | 378.94 | 74.90 | 72.48 |
| SSD (NVME) | 349.72 | 325.51 | 48.42 | 54.77 |
| SSD (PCIE) | 351.50 | 263.54 | 62.49 | 63.64 |
| RAM Disk 1 path 2400 | 8432.53 | 4818.72 | 805.43 | 565.31 |
| RAM Disk 1 path 3200 | 10218.75 | 5631.84 | 962.86 | 607.62 |
| RAM Disk 1 path 4000 | 11571.09 | 6249.48 | 1015.30 | 703.75 |
| RAM Disk 2 path 2400 | 13217.27 | 9849.50 | 948.14 | 708.75 |
| RAM Disk 2 path 3200 | 15378.24 | 11284.26 | 1093.89 | 829.48 |
| RAM Disk 2 path 4000 | 17645.38 | 12584.67 | 1283.70 | 950.40 |
| RAM Disk 4 Path 2400 | 18421.55 | 14120.39 | 1526.47 | 1216.48 |
| RAM Disk 4 path 3200 | 24127.57 | 17273.20 | 2016.95 | 1811.59 |
| RAM Disk 4 path 4000 | 31948.80 | 21135.90 | 2248.71 | 2025.60 |

operational and frequent data I/O are transferring, system contention is inevitable because the GPUs also require importing the predicted image data and exporting the labeled map to save on the disk. The RAM disk performs significantly better than the ROM disks. The major reason is that the RAM disk is established in the RAM memory, which only has two data transfers via the system bus, and the data is directly written in the RAM disk located in the north bridge. Therefore, the bus content is avoided.

## References

Bischke, B., Helber, P., Folz, J., Borth, D., Dengel, A., 2019. Multi-task learning for segmentation of building footprints with deep neural networks. In: 2019 IEEE International Conference on Image Processing (ICIP). IEEE, pp. 1480–1484.

Chaib, S., Liu, H., Gu, Y., Yao, H., 2017. Deep feature fusion for VHR remote sensing scene classification. IEEE Trans. Geosci. Remote Sens. 55 (8), 4775–4784.

Chen, Liang-Chieh, Y., Zhu, G., Papandreou, F., Schroff, H., Adam, 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 801–818.

Chen, Q., Wang, L., Wu, Y., Wu, G., Guo, Z., Waslander, S.L., 2019. Aerial imagery for roof segmentation: A large-scale dataset towards automatic mapping of buildings. ISPRS J. Photogramm. Remote Sens. 147, 42–55.

Cheng, G., Si, Y., Hong, H., Yao, X., Guo, L., 2020. Cross-scale feature fusion for object detection in optical remote sensing images. IEEE Geosci. Remote Sens. Lett. 18 (3), 431–435.

Feng, X., Han, J., Yao, X., Cheng, G., 2020. Progressive contextual instance refinement for weakly supervised object detection in remote sensing images. IEEE Trans. Geosci. Remote Sens. 58 (11), 8002–8012.

Fu, K., Chang, Z., Zhang, Y., Xu, G., Zhang, K., Sun, X., 2020. Rotation-aware and multi-scale convolutional neural network for object detection in remote sensing images. ISPRS J. Photogramm. Remote Sens. 161, 294–308.

Huang, G., Liu, Z., Maaten, L.V.D., Weinberger, K.Q., 2017. Densely connected convolutional cetworks. In: Proc. in CVPR. pp. 4700–4708.

Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., Bengio, Y., 2017. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 11–19.

Kurdi, T., Awrangjeb, F., 2020. Automatic evaluation and improvement of roof segments for modelling missing details using Lidar data. Int. J. Remote Sens. 41 (12), 4702–4725.

Li, L., Ota, K., Dong, M., Borjigin, W., 2017. Eyes in the dark: Distributed scene understanding for disaster management. IEEE Trans. Parallel Distrib. Syst. 28, 3458–3471.

Liu, C., Chen, L.C., Schroff, F., Adam, H., Hua, W., Yuille, A.L., Fei-Fei, L., 2019. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 82–92.

Maggiori, E., Tarabalka, Y., Charpiat, G., Alliez, P., 2017. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In: 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). IEEE, pp. 3226–3229.

Mi, L., Chen, Z., 2020. Superpixel-enhanced deep neural forest for remote sensing image semantic segmentation. ISPRS J. Photogramm. Remote Sens. 159, 140–152.

Rahnemoonfar, M., Chowdhury, T., Sarkar, A., Varshney, D., Yari, M., Murphy, R.R., 2021. Floodnet: A high resolution aerial imagery dataset for post flood scene understanding. IEEE Access 9, 89644–89654.

Riaz, A., Asad, M., Al-Arif, S.M., Alonso, E., Dima, D., Corr, P., Slabaugh, G., 2017. Fcnet: a convolutional neural network for calculating functional connectivity from functional mri. In: International Workshop on Connectomics in Neuroimaging. Springer, pp. 70–78.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, pp. 234–241.

Shi, Y., Li, Q., Zhu, X.X., 2020. Building segmentation through a gated graph convolutional neural network with deep structured feature embedding. ISPRS J. Photogramm. Remote Sens. 159, 184–197.

Sun, X., Shi, A., Huang, H., Mayer, H., 2020. BASˆ4Net: Boundary-aware semi-supervised semantic segmentation network for very high resolution remote sensing images. IEEE J. Selected Top. Appl. Earth Observations Remote Sens. 13, 5398–5413.

Wang, S.W., Gebru, B.M., Lamchin, M., Kayastha, R.B., Lee, W.-K., 2020. Land use and land cover change detection and prediction in the Kathmandu district of Nepal using remote sensing and GIS. Sustainability 12, 3925.

Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N., 2018. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 325–341.

Yuan, X., Shi, J., Gu, L., 2021. A review of deep learning methods for semantic segmentation of remote sensing imagery. Expert Syst. Appl. 169, 114417.

Zhang, X., Rajan, D., Story, B., 2019. Concrete crack detection using context-aware deep semantic segmentation network. Comput.-Aided Civ. Infrastruct. Eng. 34 (11), 951–971.

Zhao, H., Qi, X., Shen, X., Shi, J., Jia, J., 2018. Icnet for real-time semantic segmentation on high-resolution images. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 405–420.