electronics

MDPI

Article

Semantic Segmentation Network Slimming and Edge Deployment for Real-Time Forest Fire or Flood Monitoring Systems Using Unmanned Aerial Vehicles

Youn Joo Lee [1], Ho Gi Jung [2] and Jae Kyu Suhr [1,*]

[1] Department of Intelligent Mechatronics Engineering, Sejong University, Seoul 05006, Republic of Korea; yjlee21@sejong.ac.kr
[2] Department of Electronic Engineering, Korea National University of Transportation, Chungju 27469, Republic of Korea; hogijung@ut.ac.kr
* Correspondence: jksuhr@sejong.ac.kr; Tel.: +82-2-3408-3481

Abstract: In recent years, there has been a significant increase in the demand for unmanned aerial vehicle (UAV)-based monitoring systems to ensure proper emergency response during natural disasters such as wildfires, hurricanes, floods, and earthquakes. This paper proposes a real-time UAV monitoring system for responding to forest fires or floods. The proposed system consists of a hardware part and a software part. The hardware configuration is an embedded camera board mounted on the UAV, a Qualcomm QCS610 SoC with cores suitable for running deep learning-based algorithms. The software configuration is a deep learning-based semantic segmentation model for detecting fires or floods. To execute the model in real time on edge devices with limited resources, we used a network slimming technique which generates a lightweight model with reduced model size, number of parameters, and computational complexity. The performance of the proposed system was evaluated on the FLAME dataset consisting of forest fire images and the FloodNet dataset consisting of flood images. The experimental results showed that the mIoU of slimmed DeepLabV3+ for FLAME is 88.29%, and the inference speed is 10.92 fps. For FloodNet, the mIoU of the slimmed DeepLabV3+ is 94.15%, and the inference speed is 13.26 fps. These experimental results confirm that the proposed system is appropriate for accurate, low-power, real-time monitoring of forest fires and floods using UAVs.

Keywords: unmanned aerial vehicle; real-time monitoring; forest fire detection; flood detection; network slimming; embedded devices; semantic segmentation; Qualcomm SoC

Citation: Lee, Y.J.; Jung, H.G.; Suhr, J.K. Semantic Segmentation Network Slimming and Edge Deployment for Real-Time Forest Fire or Flood Monitoring Systems Using Unmanned Aerial Vehicles. *Electronics* **2023**, *12*, 4795. https://doi.org/10.3390/electronics12234795

Academic Editor: Palden Lama

Received: 17 October 2023
Revised: 16 November 2023
Accepted: 24 November 2023
Published: 27 November 2023

1. Introduction

Recently, due to climate change, natural disasters such as fires, earthquakes, and floods have occurred frequently, and their intensity is gradually increasing. These natural disasters have caused substantial economic losses and threatened human life, animals, and plants [1]. Early detection of forest fires or floods for proper emergency response is essential to reduce the damage from natural disasters. Traditional approaches for detecting forest fires or floods include setting up surveillance cameras or sending human patrols to the areas where fires or floods have occurred. These approaches are time-consuming, require considerable labor, and could be less effective in identifying forest fire or flood situations [2]. On the other hand, unmanned aerial vehicles (UAVs) equipped with cameras can be beneficial in discovering forest fires or floods in real time via on-board processing of aerial images. For this reason, the demand for forest fire or flood monitoring systems using UAVs is rapidly increasing.

A UAV system for wildfire or flood monitoring consists of hardware and software parts: a suitable processor to enable the real-time operation of artificial intelligence (AI) algorithms and a deep learning-based semantic segmentation algorithm to enable scene

understanding of aerial images. First, processors that can be used for AI algorithms include CPU (Central Processing Unit), GPU (Graphic Processing Unit), NPU (Neural Processing Unit), FPGA (Field-Programmable Gate Arrays), and DSP (Digital Signal Processor).

A CPU is a general-purpose processor specialized in serial processing which sequentially interprets and processes commands. This serial processing method makes it difficult to quickly process large-scale operations used for deep neural networks (DNNs). On the other hand, a GPU, which consists of thousands of cores, is a processor specialized to quickly process large-scale data in parallel and is significantly more suitable for tasks that iteratively perform the large amount of multiply–accumulate operations necessary to execute DNNs. However, GPU systems such as servers are too expensive and not portable, consume high power, and generate substantial heat. These shortcomings are significant obstacles to implementing AI technology in edge devices.

An NPU, as a microprocessor specialized in the acceleration of DNNs, consumes low power and improves resource utilization for executing DNNs compared to GPUs and CPUs. FPGAs provide the ability to reconfigure the hardware to meet specific use case requirements, and their power consumption is low. These factors make FPGAs especially useful for implementing deep learning algorithms in embedded systems. However, they are expensive and require considerable expertise to program the circuit. Lastly, a DSP is an auxiliary processor specialized in processing numerical operations at high speed and repeatedly performing multiply–accumulate operations [3], so it is appropriate for accelerating the inference of DNNs in edge devices. Usually, a DSP is mounted on a system-on-chip (SoC) along with a CPU and a GPU [4].

In this paper, we use an embedded camera device with Qualcomm's QCS610 SoC as the hardware of the UAV monitoring system. The QCS610 SoC is beneficial in terms of low power, miniaturization, and commercialization and the Hexagon 685 DSP equipped in this SoC is a specialized core to accelerate DNNs [5].

As software in a UAV monitoring system, DNN-based semantic segmentation networks are considerably helpful in understanding images of wildfires or floods. However, despite their high performance, these networks require significant computational complexity and inference time, so lightweight DNN models with fewer parameters and lower computational complexity are needed for edge computing [6].

There are two ways to optimize DNN models: designing a lightweight network, and removing redundancy of the deep network, such as network slimming [7]. The former is a relatively simple method but has the inconvenience of having to redesign the architecture of a lightweight network. The latter creates a lightweight network by removing unnecessary filters (channels) or layers from deep networks with a proven high performance. This approach makes it easy to use the architecture of the existing network without designing a new one and can retain a high performance that is very close to that of the existing network. This paper uses a network slimming technique to optimize a DNN-based semantic segmentation network and then execute a lightweight model on a DSP.

This paper focuses on generating a lightweight semantic segmentation model and porting it to Qualcomm's QCS610 chip to propose a UAV system for real-time monitoring of forest fires or floods. The main contribution of this paper is to reveal that network slimming, semantic segmentation, and edge deployment are practically combined and implemented on off-the-shelf hardware. To the best of our knowledge, in the field of disaster monitoring, this paper provides the first study that shows that the combination of all three elements can effectively operate on an off-the-shelf edge device. The experimental results clearly demonstrate the feasibility of the proposed system on drones. In the experiments, DeepLabV3 and DeepLabV3+ were used as semantic segmentation networks, and their performance was evaluated on the FLAME and FloodNet datasets. In experimental results, in the case of forest fires, the mIoU of the slimmed DeepLabV3 and V3+ was 83.32% and 88.29%, respectively, and compared to the baseline models, there was almost no performance degradation even if 90% of all channels of each model were removed. The inference speed was 22.03 fps and 10.92 fps, respectively; the model size was 5.7 MB

and 15.8 MB, and the GFLOPs was 4.73 and 38.21, respectively. In the case of floods, the mIoU of the slimmed DeepLabV3 and V3+ was 93.69% and 94.15%, respectively. The inference speed was 23.58 fps and 13.26 fps, respectively; the model size was 5.8 MB and 11.8 MB, and the GFLOPs was 4.26 and 26.01, respectively. The power consumption of slimmed DeepLabV3+ was 0.034 mWh/inference for FLAME and 0.025 mWh/inference for FloodNet. These experimental results demonstrate that the slimmed network can accurately detect forest fires or floods in real time and at low power, even in embedded devices, without performance degradation.

2. Related Works

This section describes studies related to real-time semantic segmentation for UAV-based forest fire or flood monitoring systems and semantic segmentation network optimization for edge computing. The proposed system is about implementing a real-time monitoring system that detects floods or forest fires by applying semantic segmentation to images using edge devices. Therefore, this section summarizes related research on detecting floods or forest fires based on real-time semantic segmentation, detecting fires using object detection or classification algorithms, and finally optimizing DNNs to implement on edge devices.

2.1. Semantic Segmentation for Flood Detection

Safavi et al. [8] analyzed the suitability of state-of-the-art (SOTA) semantic segmentation models for identifying disasters such as floods from images for the purpose of UAV-based flood detection. To this end, they compared the performance of real-time and non-real-time segmentation models on aerial imagery. The experiments were performed on NVIDIA GeForce RTX 3090 GPU for training and evaluation, and the performance was evaluated in terms of computational capacity (MAC), parameter size, inference time, and accuracy. The experimental results showed that the real-time model using the U-Net architecture with MobileNetV3 as a backbone of the encoder outperformed other models. In [9], they compared the performance of existing real-time semantic segmentation models for UAV-based flood detection. In the experiments, they measured the performance of the models in terms of MAC, number of parameters, inference speed, and segmentation accuracy on the RTX 3090 GPU. Hernández et al. [10] proposed a GPU-based edge computing pipeline for a real-time flood detection system based on UAVs. They used three devices from the NVIDIA Jetson family as portable GPUs and a combination model of three segmentation network architectures and three backbone models for the semantic segmentation network. The experimental results showed that the PSPNet using ResNet152 as a backbone was the most suitable for a real-time flood detection system in terms of model size, inference speed, and accuracy. Some studies applied a Fully Convolutional Network (FCN) among semantic segmentation networks to assess the extent of flooding based on UAV systems [11,12].

2.2. Semantic Segmentation for Fire Detection

Guan et al. [2] proposed a novel instance segmentation method based on mask scoring R-CNN for early wildfire detection and segmentation for a UAV monitoring system. The proposed method consists of image classification, fire region detection, and flame segmentation. They used U-Net architecture for semantic segmentation and ResNet as the backbone of the encoder. The experiments were conducted on NVIDIA RTX 3070 and showed excellent performance compared to existing SOTA methods. However, they needed to analyze the suitability of the proposed network for embedding in portable devices. Ghali et al. [13] presented a comprehensive review of deep learning models related to the three steps required to detect wildfires based on UAV systems, classification, detection, and segmentation, and introduced public fire datasets for these tasks. In the experiments, the comparison with traditional machine learning methods confirms the superiority of deep learning models, but they do not handle network optimization for real-time detection.

Mengna et al. [14] proposed a real-time semantic segmentation for forest fire detection. The proposed method is an improved version of DeepLabV3+, which does not use atrous convolution to improve the inference speed of the encoder and uses MobileNetV3, a lightweight network. Also, to compensate for the performance degradation, they proposed a structure to add more feature information to the decoder. The network was conducted on RTX 2080 Ti GPUs, and its performance resulted in better accuracy and speed than the existing Deeplabv3+.

2.3. Object Detection or Image Classification for Fire Detection

Wang et al. [15] showed the potential of realizing forest fire detection based on YOLOv4 on edge devices. Their experiments conducted YOLOv4 with MobileNetV3 as a backbone on the NVIDIA Jetson Xavier NX platform and showed that forest fires were detected in real time. Xiong et al. [16] proposed a UAV-based forest fire detection system. They used YOLOv3 to detect the flame and smoke of wildfires and conducted the fire detection algorithm on the FPGA-SoC. Tahir et al. [17] proposed a YOLOv5-based deep learning model for a UAV-based wildfire detection system. These approaches, which use only object detection algorithms, have difficulty obtaining detailed information such as the shape, size, and location of the fire. To solve this problem, approaches that combine object detection with segmentation techniques have been proposed [18,19]. Mseddi et al. [18] proposed a novel structure that combines YOLOv5 and U-Net architecture to improve the accuracy of forest fire detection. Based on YOLOv5, they found the fire as a bounding box in the image and applied a U-Net architecture to the cropped image generated by the bounding box to obtain a semantic segmentation result. Cao et al. [19] proposed a YOLO-SF algorithm, which combines the YOLOv7-Tiny object detection algorithm with an instance segmentation technique, to improve the accuracy of fire detection. Finally, here is an example of applying the classification algorithm for fire detection. Almeida et al. [20] proposed a novel lightweight CNN model for real-time wildfire detection through surveillance camera and UAV-based wildfire monitoring, and the proposed model performs the task of classifying whether the input image contains smoke or flames.

2.4. Semantic Segmentation Network Optimization

Most research on real-time forest fire or flood segmentation for UAV monitoring systems has mainly used real-time semantic segmentation networks that combine a U-Net architecture with a lightweight backbone. This section describes studies focused on optimizing semantic segmentation networks for edge computing. Liu et al. [21] proposed a lightweight architecture for performing a real-time semantic segmentation task on UAV images and an attention module for efficient feature extraction. Rosas-Arias et al. [22] proposed a new lightweight network with two modules to implement real-time semantic segmentation for embedded systems: one to extract contextual information in the encoder and one to refine context and spatial information. There are also studies [23,24] that apply a channel pruning technique, a well-known model optimization method in classification, to semantic segmentation networks. However, these works were not proposed for flood or wildfire detection but for real-time semantic segmentation tasks on general images. He et al. [23] offered a modified channel pruning method suitable for semantic segmentation networks. It removed unnecessary channels by adding contextual information to the existing network slimming method [25]. The proposed method was applied to PSPNet, ICNet, and SegNet to demonstrate the optimization effectiveness. Chen et al. [24] proposed a new approach to select unnecessary channels by considering classification loss and segmentation loss when applying channel pruning to semantic segmentation methods. The effectiveness of the proposed method was evaluated on DeepLabV3, PSPNet, and BiSeNet.

3. System Configuration

As shown in Figure 1, the proposed system consists of acquiring aerial images with a camera mounted on a UAV, running a semantic segmentation network stored in the

memory of the embedded camera device on a DSP to detect forest fires or floods, and finally notifying the ground station of the detected results.

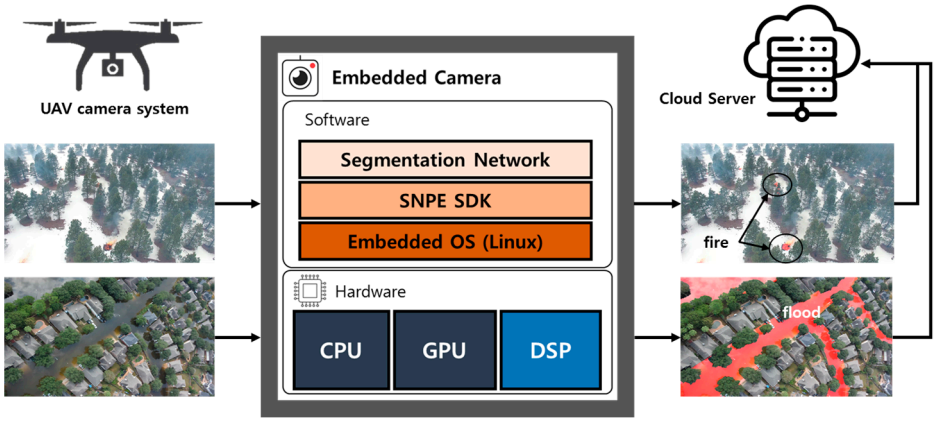

Figure 1. System configuration.

Semantic segmentation is a popular technique for detecting forest fires or floods because it results in a pixel-wise classification of the image. In the case of forest fires, each pixel is classified as fire or not, and in the case of flood, each pixel is classified as water or not. This makes it easy to analyze the location, shape, and size of fire or flood in the image. Some studies have used object detection networks to detect fires and forest fires [15–17,26]. However, although object detection can identify the location of the fire, as shown in Figure 2, it is difficult to analyze the size and shape of the fire accurately, so the detected area must be segmented to analyze the results. Flood detection is difficult to determine with a bounding box because the shape of the flood is amorphous, as shown in Figure 2. Therefore, it is more effective to detect forest fires or floods using semantic segmentation.

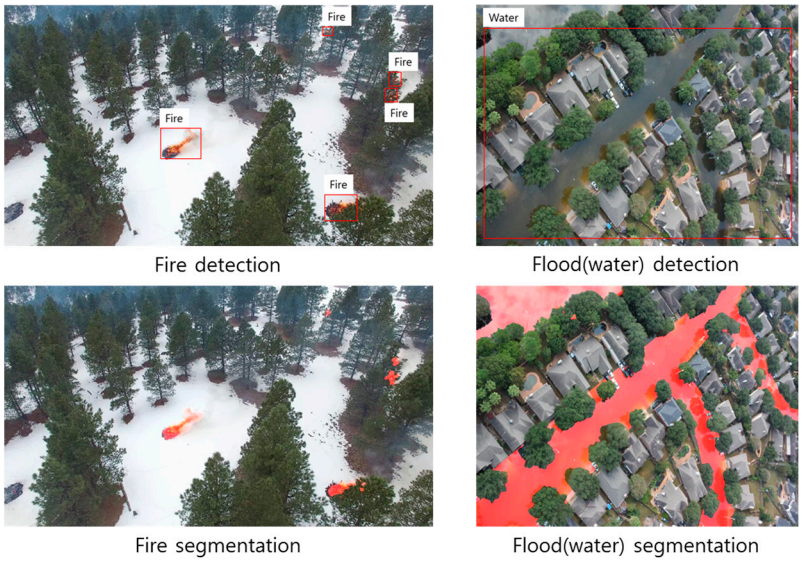

Figure 2. Examples of object detection and semantic segmentation to identify forest fires or flooding in aerial images.

3.1. Hardware Configuration

A Qualcomm QCS610 chip contains three cores: the Kyro 460 CPU, Adreno 612 GPU, and Hexagon 685 DSP. Figure 3 shows the block diagram of the chip and the embedded board developed by WITHROBOT Inc. Its size is 5 cm wide and 3.8 cm long.

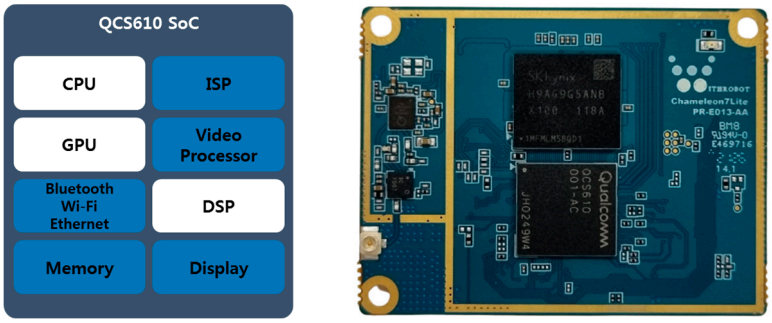

Figure 3. QCS610 block diagram (**left**) and QCS610 embedded board developed by WITHROBOT Inc. (Seoul, Republic of Korea) (**right**).

This paper uses a Hexagon 685 DSP to run a semantic segmentation network on the embedded board. DSPs have an architecture that allows multiple mathematical calculations to be performed at once, and they have the unique ability to perform high-speed multiplication. Dedicated DSPs are also usually more power-efficient, which makes them more suitable for portable devices such as mobile phones with limited power consumption [3]. Qualcomm's Hexagon 685 DSP is a co-processor designed for artificial intelligence and machine learning, with a specialized architecture for AI algorithms [5]. Qualcomm provides the Snapdragon Neural Processing Engine Software Development Kit (SNPE SDK) to reduce the time and effort it takes to optimize the performance of deep neural networks trained on devices with Qualcomm AI products [27]. It is a software framework designed to help quickly deploy and run AI models on-device to execute tasks on each core (CPU, GPU, DSP) that runs Qualcomm's SoC. SNPE SDK includes a conversion tool that moves existing neural networks based on Cafe/Cafe2 and TensorFlow to Qualcomm's SoC runtime environment and debug tools and performance optimization tools for each core.

3.2. Software Configuration

Executing deep learning-based AI models on edge devices typically requires three steps, as shown in Figure 4. Train a DNN model on the neural network framework of the GPU system on the desktop and convert the trained model to a format suitable for the embedded environment by utilizing the device conversion tool. Finally, import the converted model into the embedded device and run it through the inference API.

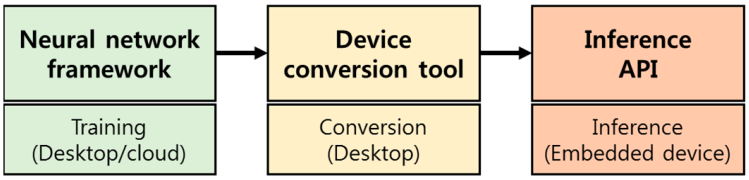

Figure 4. Workflow for deploying a trained DNN model to an embedded device.

Figure 5 shows the process followed in this paper. First, we create a lightweight semantic segmentation network through network slimming in the TensorFlow framework. Network slimming is a type of network lightweight, which is a way to optimize a network to have a suitable size and computation for resource-constrained edge devices. Network slimming consists of sparsity training, channel pruning, and fine-tuning, as shown in Figure 5, and is described in detail in Section 4. Next, we utilize the inference API (SNPE SDK) provided by Qualcomm to convert the slimmed network into a Deep Learning Container (DLC) file, a format suitable for the QCS 610 embedded board environment. This DLC file stores model parameters in floating point format and runs only on processors capable of 32-bit floating-point operations. However, DSPs only support 8-bit integer operations, so parameter quantization is required. Quantization methods include Post Training Quantization (PTQ) and Quantization Aware Training (QAT) [28]. PTQ is a method

that quantizes parameters after network training. QAT is a method that simultaneously finds the optimal parameters and quantizes them by simulating the effect of quantization in the inference during training. This paper applies the PTQ provided by the SNPE tool to convert the DLC model with 32-bit floating-point parameters to the one with 8-bit integer parameters. Finally, the quantized DLC file is imported into the embedded board, and its inference is executed on the board.

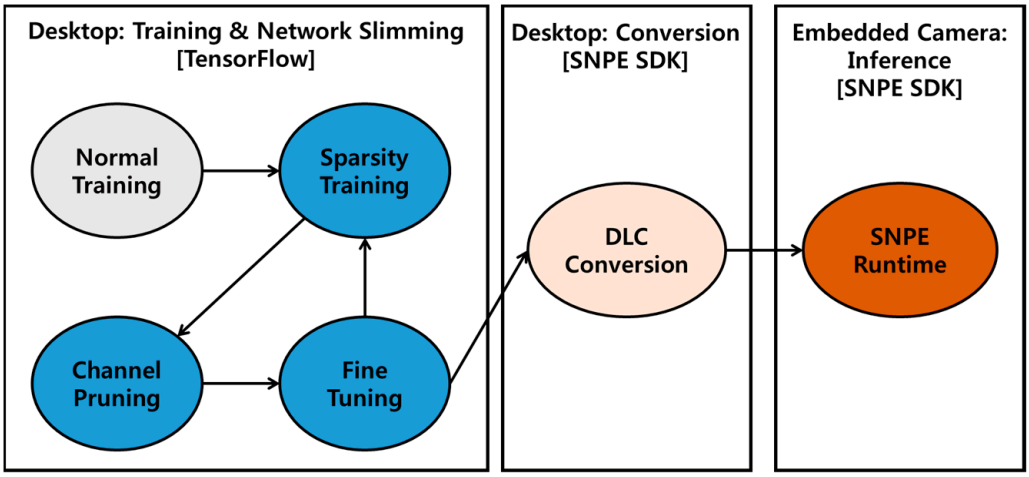

Figure 5. Workflow for slimming a semantic segmentation network and deploying the slimmed network to an embedded camera board with a Qualcomm QCS610 SoC.

4. Network Slimming

Network slimming [25] is an effective way to reduce the size, parameters, and computation of deep learning models, eliminating unnecessary channels (or filters) while leaving only the important ones. In this method, it is crucial to decide which channels to keep and which to remove. To do this, Liu et al. [25] used the scale parameter γ of the Batch Normalization (BN) layer following the Convolutional layer as the scaling factor of the channels, trained them to be sparse, and then determined the importance of the channels. Recently developed CNN models typically include a BN layer, which is well known to improve convergence speed and generalization performance. The BN layer normalizes the input feature map based on mini-batch statistics and then performs a linear transformation to obtain the output feature map, as shown in Equation (1).

$$z_0 = \gamma \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + c}} + \beta \tag{1}$$

In Equation (1), z_i is the input value, z_0 is the output value, and μ_B and σ_B are the mean and standard deviation of the input values obtained from the small batch, respectively. γ and β are the learning parameters related to scale and shift, respectively. Here, γ corresponds to one channel and affects the magnitude of the output of that channel, which helps identify the importance of the channel.

The sparsity training stage is performed after normal training to determine the channels that are essential to achieve the target task of the network. In this stage, the network is trained by adding an L1 regularization for the scaling factor to the initial loss function of the network, as shown in Equation (2), in order to make as many scaling factor values as possible zero.

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_{\gamma \in \Gamma} |\gamma| \tag{2}$$

In Equation (2), (x,y) is the input and target of the training data, and W is the weights. The first part of the equation is the loss function of the segmentation network, the second part is the penalty for eliciting sparsity of the scaling factors, and λ is the sparsity parameter that balances the two parts.

Next, we apply channel pruning to the sparsity trained network. Once the target pruning ratio (R) is determined, the number of channels to be removed (N) is calculated as the product of the pruning ratio (R) and the total number of channels (K) of the network. The global threshold is then selected as the (KxR)th value after sorting the scaling factor values for all channels in descending order, and unnecessary channels are pruned based on this threshold. When pruning channels using a global threshold, the number of channels removed and location (index) of the channels are different for each layer. In the case of a network with a sequential structure, there is no problem in pruning the network even if the number of channels removed or their positions vary from layer to layer. However, it causes a problem in the networks with residual blocks due to add operations of skip connections. In order to apply channel pruning to add operations in this network, the number of pruned channels between the two convolutional layers connected to the add layer must match, and the positions of the channels being pruned must be the same. There is a simple method that is commonly used to solve this problem [29]. First, find the index set of channels to be retained in each convolutional layer that needs to be matched. Next, apply OR, AND, or Head-First (HF) mode between the different index sets to be matched. The OR and AND modes prune channels of the related Convolutional layers equally using the results of applying logical sum and logical product operations to multiple index sets, respectively. HF mode prunes channels of all remaining Convolutional layers equally by utilizing the index set of the Convolutional layer immediately before the residual block. The residual blocks of ResNet used in this paper include the identity and convolution blocks. Figure 6 illustrates how to apply channel pruning in the identity and convolution blocks, where $\mathbf{S}_i$ denotes the set of indices of the necessary channels to keep in the i-th Convolutional layer. As shown in Figure 6, in the case of the identity block, we need to match the channel index ($\mathbf{S}_0$) of the Convolutional layer (conv2D_0) that enters the residual block with the channel index ($\mathbf{S}_3$) of the Convolutional layer (conv2D_3) placed immediately before the Add layer. Similarly, in the convolution block, channel pruning is performed by matching the channel index of the two Convolutional layers (conv2D_3 and conv2D_4) placed just before the Add layer.

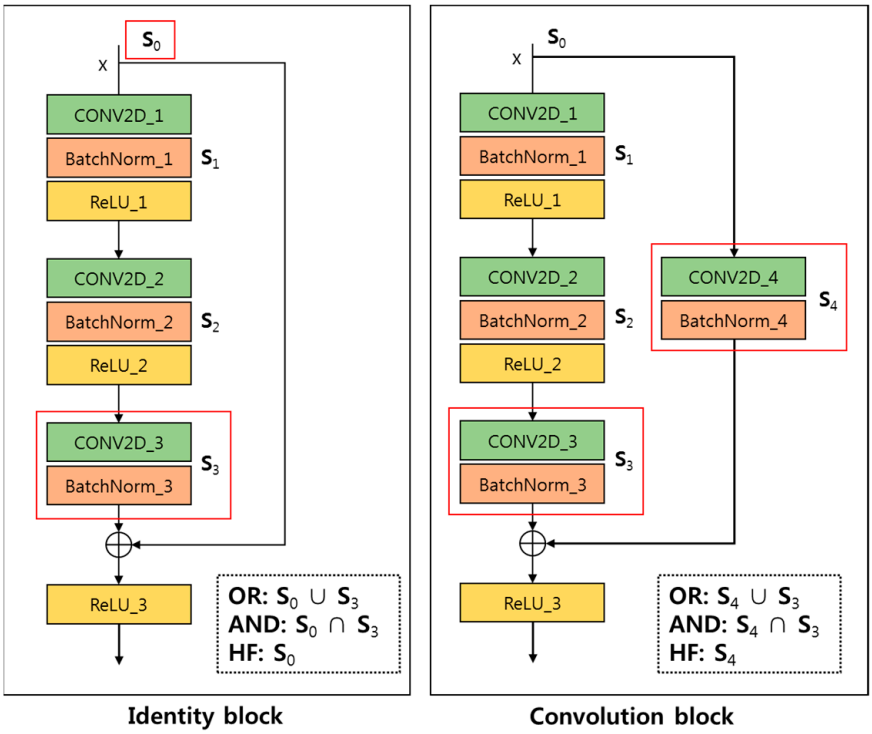

Figure 6. Example descriptions of the three modes for applying channel pruning to residual blocks in ResNet. The red boxes indicate layers that need to match the index of the channel to be pruned.

Finally, the network is retrained to fine-tune its parameters after channel pruning. The performance of a pruned network will degrade very little when the number of channels removed is tiny but will degrade significantly as the number of channels removed increases. Therefore, after pruning, the model's parameters must be trained again to restore the performance degraded by channel pruning.

5. Experimental Results

5.1. Datasets

In the experiments, we used two benchmark datasets collected by UAVs such as drones: FloodNet [30] for floods and FLAME [31] for wildfires. FloodNet [30] consists of aerial imagery collected by DJI Mavic Pro quadcopters, a small UAV platform, after Hurricane Harvey in 2017 near Texas and Louisiana. This dataset reflects actual flood situations and includes 2343 high-resolution (4000 × 3000) images taken at a low altitude of 200 feet above the ground. Ground-truth images are labeled pixel-wise for nine different classes: building-flooded, building-non-flooded, road-flooded, road-non-flooded, water, tree, vehicle, pool, and grass. In this paper, the goal is to detect floods, not to evaluate the damage caused by floods, so we reset to two classes: water and background. 'building-flooded', 'road-flooded', and 'water' belong to the water class, and the rest of the classes belong to the background class. Also, we excluded 476 incorrectly annotated images and conducted the experiments with the remaining 1867 images. The dataset split was 1146 for training, 351 for validation, and 370 for test. FLAME [31] is a dataset collected for early detection of forest fires. This dataset consists of aerial imagery captured by a camera mounted on a drone after stacking burning piles at a predetermined location to simulate an earlier forest fire. Among the various datasets in FLAME, we experimented with a dataset consisting of 2003 still images acquired with a general camera and annotated images labeled pixel-wise as fire and background. The image resolution is 3480 × 2160, and the dataset split was 1503 for training, 200 for validation, and 300 for test. Figure 7 shows examples of aerial images and labeled images from FloodNet and FLAME.

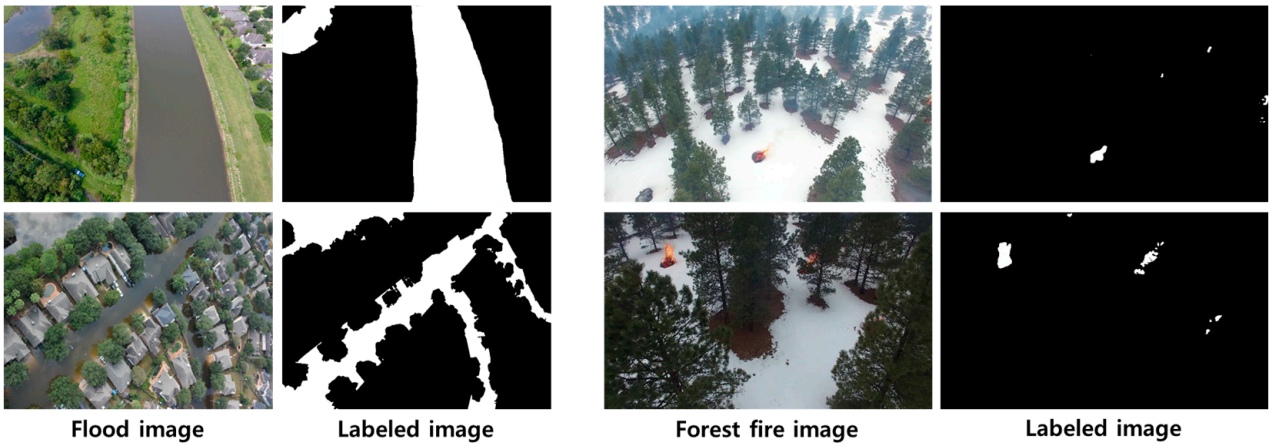

Figure 7. Examples of aerial imagery and labeled images from the FloodNet (**left**) and FLAME datasets (**right**).

5.2. Experiment Details

We conducted the experiments using two systems with the following specifications: NVIDIA TITAN RTX with 24 GB on-board memory using a Tensorflow 2.10 and Ubuntu OS installed on an Intel Core i9-10900X CPU with 64 GB RAM, and Hexagon 685 DSP using SNPE 2.12 installed on Qualcomm QCS610 SoC. Our experiments used DeepLabV3 and DeepLabV3+ as semantic segmentation network models. In the sparsity training stage, the accuracy and sparsity of the model were measured by changing the sparsity parameter (λ) in 10-fold increments from 0.001 to 100.0. The sparsity model was determined as in [25], where the sparsity parameter is maximized among the cases with negligible performance

degradation. In the channel pruning stage, the accuracy of the models was obtained by varying the pruning ratio from 10% to 90%. In addition, OR, AND, and HF modes were applied to the residual blocks. After fine-tuning, the best-performing model was inferred on the DSP of the QCS 610 chip, and then its performance was evaluated. The performance was mainly measured in mean-over-union accuracy (mIoU) and frame per second (fps). We also reported the number of parameters, size of the model, computational complexity in GFLOPs, and power consumption on the embedded board. The same training setting was used for all models, where the learning rate is 0.001, the batch size is 8, and the epoch is 50. Data augmentation consists of horizontal flip, vertical flip, rotation, and zoom, and its rate is 0.8. We trained with 512 × 512 crop size for the FloodNet and FLAME datasets. For all experiments, the inference speed was calculated as the number of inferences obtained when running the semantic segmentation network in high-performance mode on the DSP for one minute. The power consumption was evaluated on TensorFlow 2.3 and SNPE 1.66.

5.3. Semantic Segmentation Results for FLAME Dataset

This section summarizes the performance evaluation results of DeepLabV3 and DeepLabV3+ on the FLAME dataset. Table 1 shows the performance of the baseline model and the sparsity trained model for both networks. For the sparsity trained model, the sparsity parameter was chosen as described in Section 5.2 and set to 1.0 for both networks. Figure 8 shows histograms of the channel's scaling factors for DeepLabV3 and DeepLabV3+. This figure indicates the sparsity of the channel before and after sparsity training. These histograms display that after sparsity training, the channel scaling factors are clustered around 0.0, which indicates adequate sparsity training results.

Table 1. Performance of DeepLabV3 and DeepLabV3+ before and after sparsity training on FLAME.

Networks	**DeepLabV3**		**DeepLabV3+**	
Models	**Baseline**	**Sparsity** *	**Baseline**	**Sparsity**
mIoU (%)	84.59	84.32	88.12	88.13

* Sparsity means sparsity trained model.

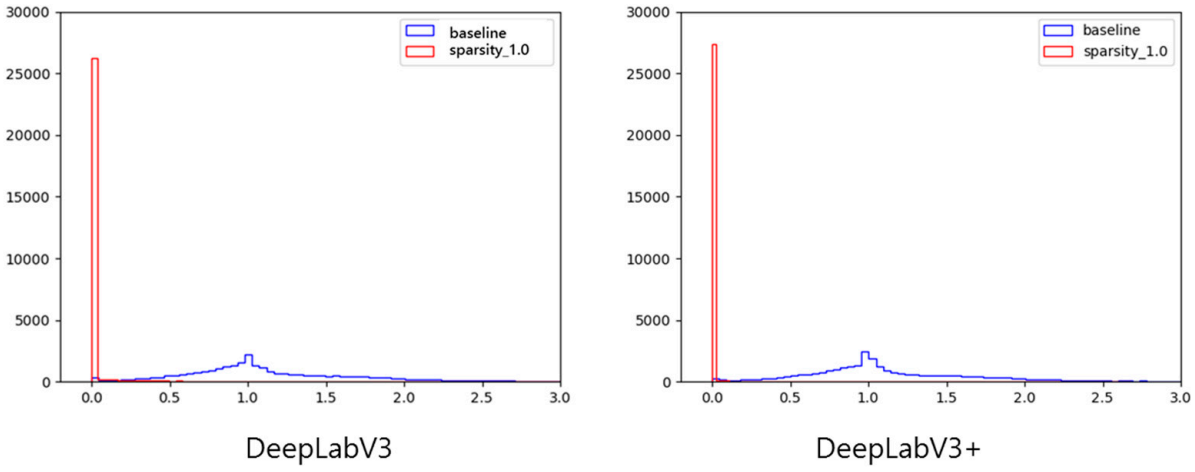

Figure 8. Histograms of channel scaling factors for DeepLabV3 and DeepLabV3+ on FLAME.

Table 2 shows the results of applying channel pruning to the sparsity trained model. The first column of the table shows the pruning ratio, where baseline means the model obtained from normal training with a pruning ratio of 0%. The pruning ratio means the actual pruning ratio, calculated as the number of channels removed compared to the total number of channels in the network. Comparing the accuracy performance of the two models, the mIoU of DeepLabV3+, the most updated version of the DeepLab family, is about 3~4 percentage points larger than the mIoU of DeepLabV3. Comparing the before

and after channel pruning, we can see that there is almost no performance degradation for both models. These results reveal that the redundancy of the two models in performing the semantic segmentation task on FLAME is relatively high and that unnecessary channels were appropriately removed by the sparsity training and channel pruning processes.

Table 2. Performances of DeepLabV3 and DeepLabV3+ according to the pruning ratio on FLAME (mIoU: %).

Networks	**DeepLabV3**			**DeepLabV3+**		
Ratio (%)	**OR**	**AND**	**HF**	**OR**	**AND**	**HF**
Baseline	84.59	84.59	84.59	88.12	88.12	88.12
10	84.66	84.43	84.56	87.99	88.11	88.13
20	84.51	84.70	84.65	88.16	87.94	87.91
30	84.40	84.53	84.48	88.00	88.13	87.91
40	84.64	84.49	84.51	88.16	88.11	88.13
50	84.43	84.58	84.47	88.24	88.13	88.06
60	84.65	84.33	84.41	88.09	88.16	88.18
70	84.55	84.23	84.38	88.17	88.02	87.84
80	84.65	84.41	84.51	87.91	88.15	88.05
90	**84.55**	**84.44**	**84.54**	**88.25**	**88.12**	**87.95**

Table 3 shows the inference results of running DLC files for the baseline and 90% pruned models on the QCS610 DSP. DeepLabV3 has the highest performance in AND mode, and DeepLabV3+ has the highest performance in HF mode. In the case of DeepLabV3, the performance on DSP is degraded by about 1~3 percentage points compared to the performance on Desktop GPU, and in the case of DeepLabV3+, there is almost no performance degradation. In both cases of DeepLabV3 and V3+, the inference speed was increased by more than 15 times, and model size and GFLOPs were significantly reduced compared to the baseline. DeepLabV3+ has an encoder–decoder architecture to heighten performances concerning object boundary recovery and segmentation accuracy. Because of this architecture, DeepLabV3+ has a slightly larger model size and GFLOPs and a lower inference speed but better accuracy than DeepLabV3.

Figure 9 shows the semantic segmentation results obtained by the DSP. The columns in the figure show the input image, the ground-truth image labeled as fire, the image inferred by DeepLabV3, and the image inferred by DeepLabV3+. For both models, we can see that both large and small fires are segmented well compared to the ground truth.

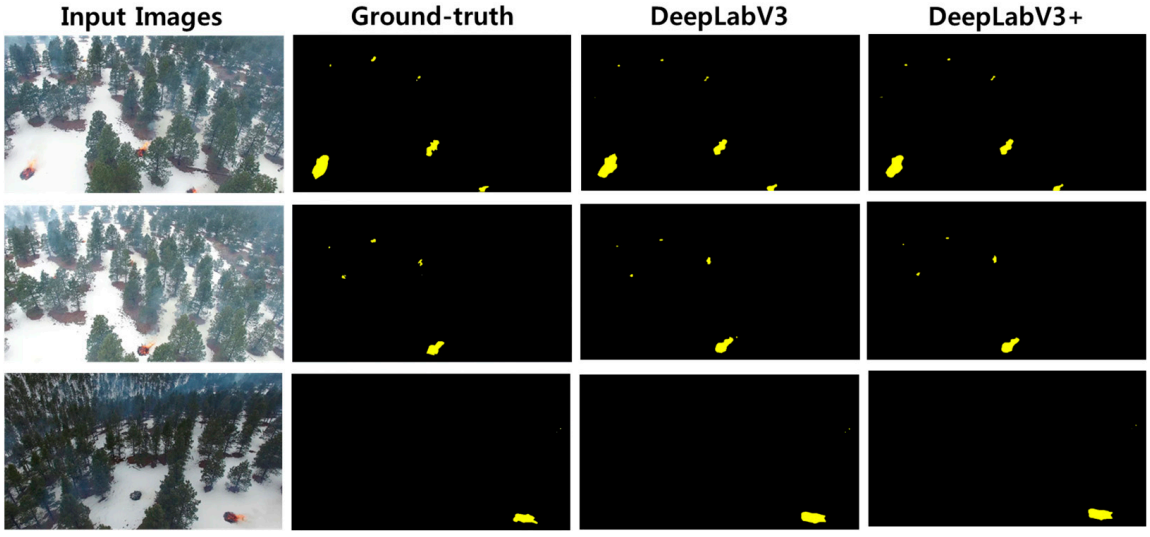

Figure 9. Qualitative results of 90% pruned DeepLabV3 and 90% pruned DeepLabV3+ for FLAME on QCS610 DSP.

Table 3. Semantic segmentation results of DeepLabV3 and DeepLabV3+ for FLAME on QCS610 DSP.

	DeepLabV3				DeepLabV3+			
	mIoU	**Speed** [1]	**Size** [2]	**GFLOPs**	**mIoU**	**Speed** [1]	**Size** [2]	**GFLOPs**
Baseline	81.65	0.64	470.2	101.76	87.74	0.61	485.9	144.46
OR	82.66	23.98	2.8	2.39	88.06	9.54	16.6	42.05
AND	83.32	22.03	7.4	5.60	88.10	11.54	16.1	34.89
HF	82.28	22.96	5.7	4.73	88.29	10.92	15.8	38.21

[1] Speed means inference speed (fps). [2] Size means model size (MB).

5.4. Semantic Segmentation Results for FloodNet Dataset

This section summarizes the performance evaluation results of DeepLabV3 and DeepLabV3+ on the FloodNet dataset. Table 4 shows the performance of the baseline model and the sparsity trained model for both networks. For the sparsity trained model, the sparsity parameter was chosen as described in Section 5.2 and set to 100.0 and 10.0 for both networks, respectively. Figure 10 shows histograms of the channel's scaling factors for DeepLabV3 and DeepLabV3+. These histograms display that after sparsity training, the channel scaling factors are clustered around 0.0, which indicates adequate sparsity training results.

Table 4. Performance of DeepLabV3 and DeepLabV3+ before and after sparsity training on FloodNet.

Networks	**DeepLabV3**		**DeepLabV3+**	
Models	**Baseline**	**Sparsity** *	**Baseline**	**Sparsity**
mIoU(%)	93.23	93.23	94.08	94.45

* Sparsity means sparsity trained model.

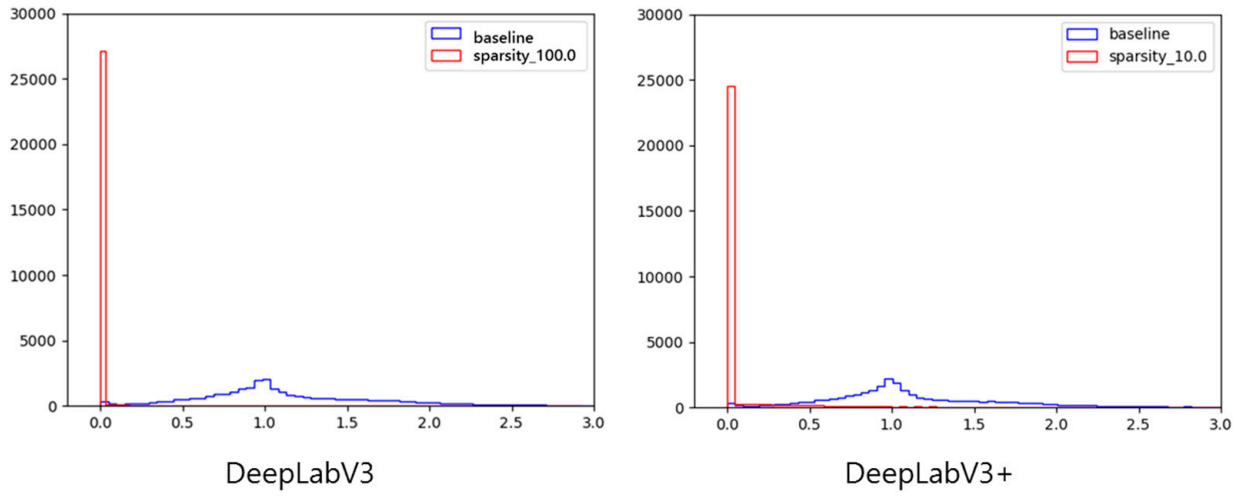

Figure 10. Histograms of channel scaling factors for DeepLabV3 and DeepLabV3+ on FloodNet.

Table 5 shows the results of applying channel pruning to the sparsity trained model. Comparing the accuracy performance of the two models, the mIoU of DeepLabV3+ is about one percentage point larger than the mIoU of DeepLabV3. Comparing the before and after channel pruning, we can see that there is almost no performance degradation for both models. Similar to FLAME, the results of FloodNet also reveal that the redundancy of the two models in performing the semantic segmentation task is relatively high and that the sparsity training and channel pruning processes properly removed unnecessary channels.

Table 6 shows the results of running the baseline and 90% pruned models on the QCS610 DSP. DeepLabV3 has the highest performance in OR mode, and DeepLabV3+ has the highest performance in HF mode. For both networks, there is little performance

degradation on the DSP compared to the performance on the Desktop GPU. As with FLAME, for both cases of DeepLabV3 and V3+, the inference speed was improved by more than 35 times, while model size and GFLOPs were also greatly reduced compared to the baseline. These experimental results show that the proposed system, which ports the slimmed network to embedded devices, can accurately detect forest fires or floods in real time with negligible performance degradation.

Table 5. Performances of DeepLabV3 and DeepLabV3+ according to the pruning ratio on FloodNet (mIoU: %).

Networks	**DeepLabV3**			**DeepLabV3+**		
Ratio (%)	**OR**	**AND**	**HF**	**OR**	**AND**	**HF**
Baseline	93.23	93.23	93.23	94.08	94.08	94.08
10	93.86	93.87	93.98	94.56	94.61	94.67
20	94.01	93.98	93.89	94.60	94.59	94.66
30	93.98	93.92	93.93	94.71	94.61	94.56
40	93.97	93.76	93.85	94.63	94.54	94.51
50	93.92	93.66	93.75	94.56	94.57	94.40
60	93.97	93.61	93.74	94.55	94.53	94.50
70	93.91	93.10	93.83	94.69	94.54	94.63
80	93.85	92.93	93.57	94.48	94.47	94.64
90	**93.93**	**92.53**	**93.50**	**92.09**	**93.74**	**94.25**

Table 6. Semantic segmentation results of DeepLabV3 and DeepLabV3+ for FloodNet on QCS610 DSP.

	DeepLabV3				**DeepLabV3+**			
	mIoU	**Speed** [1]	**Size** [2]	**GFLOPs**	**mIoU**	**Speed** [1]	**Size** [2]	**GFLOPs**
Baseline	93.92	0.64	470.2	101.76	94.05	0.61	485.9	144.46
OR	93.69	23.58	3.3	2.62	91.97	18.78	2.5	2.21
AND	91.83	22.10	9.1	4.82	93.74	13.36	15.2	27.60
HF	93.23	23.41	5.8	4.26	94.15	13.26	11.8	26.01

[1] Speed means inference speed (fps). [2] Size means model size (MB).

Figure 11 shows the semantic segmentation results obtained by the DSP. The columns in the figure show the input image, the ground-truth image labeled as water, the image inferred by DeepLabV3, and the image inferred by DeepLabV3+. Both networks can segment the flooded area in the image relatively accurately. In particular, we can see that the results are closer to the ground truth when the water areas have simple shapes, such as in the second, fourth, and fifth input images. On the other hand, if the flooded area is more complex, such as in the first and third images, where flooding occurs in a cluster of houses, DeepLabV3+ segments the flood boundaries more accurately than DeepLabV3.

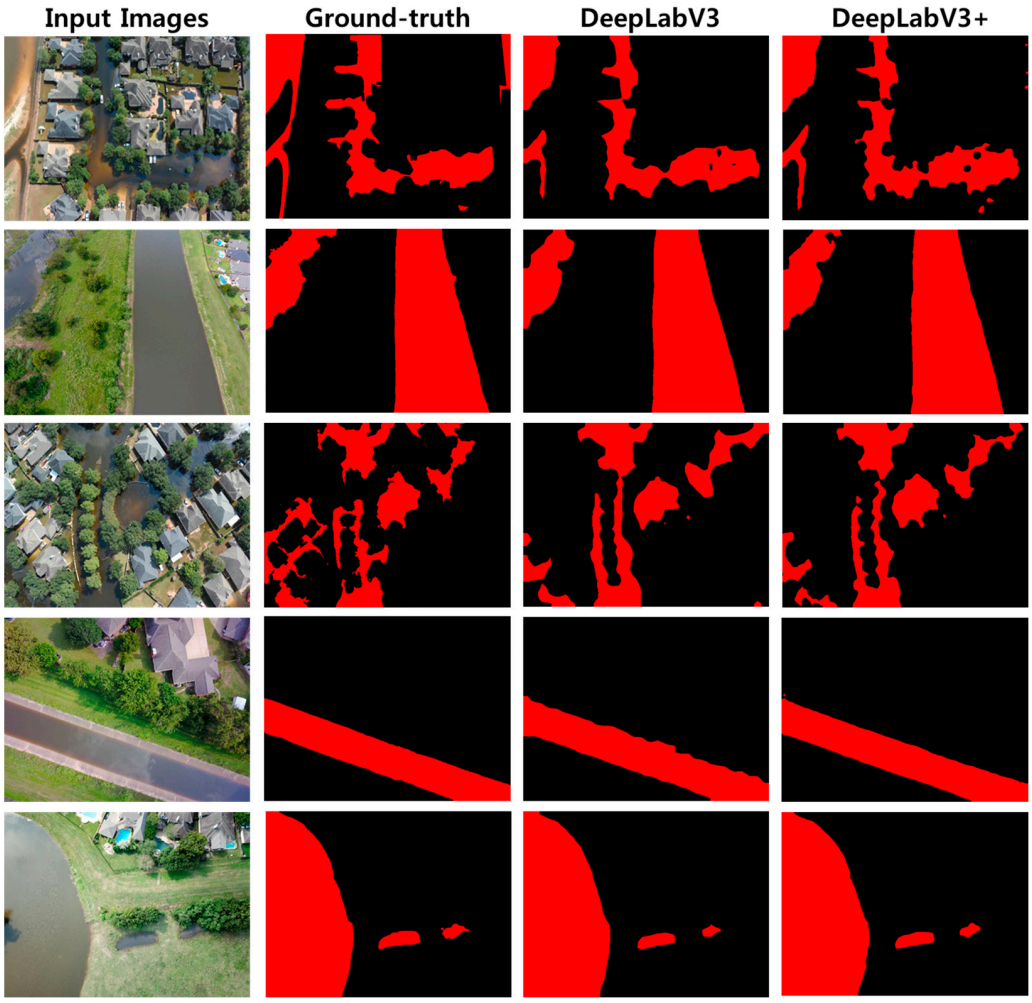

Figure 11. Qualitative results of 90% pruned DeepLabV3 and 90% pruned DeepLabV3+ for FloodNet on QCS610 DSP.

5.5. Results for Power Consumption

To measure the power consumption when the DNN model is running on the QCS610, we can use the Qualcomm Snapdragon Profiler. The QCS610 SoC has nine operating modes, and the inference speed and power consumption change depending on the mode. So, users can select the appropriate operating mode according to the situation. Figure 12 shows the power consumption graph versus time stamps when slimmed DeepLabV3+ trained on FloodNet is executed for nine different modes of operation. In this figure, the black line represents sleep mode, with a duration of 5 s. The red line shows the power consumption when running slimmed DeepLabV3+ for each of the nine operating modes to process one image. From left to right, the operating modes are 'low balanced', 'balanced', 'default', 'high performance', 'sustain high performance', 'burst', 'low power saver', 'power saver', and 'high power saver.' As shown in Figure 12, comparing the operation time before and after network slimming, we can find that the time taken for the slimmed DeepLabV3+ network to operate is significantly reduced regardless of the operation mode. We can also see that the power consumption is reduced dramatically in the slimmed DeepLabV3+ model compared to the baseline model. Default mode sacrifices inference speed and reduces power consumption among the operating modes. This mode is the same as the balanced mode. 'Low power saver', 'power saver', and 'high power saver' modes can save more power than 'balanced' mode, but at the cost of reduced performance in inference speed. 'High performance', 'sustain high performance', and 'burst' are modes that increase inference speed but increase power consumption.

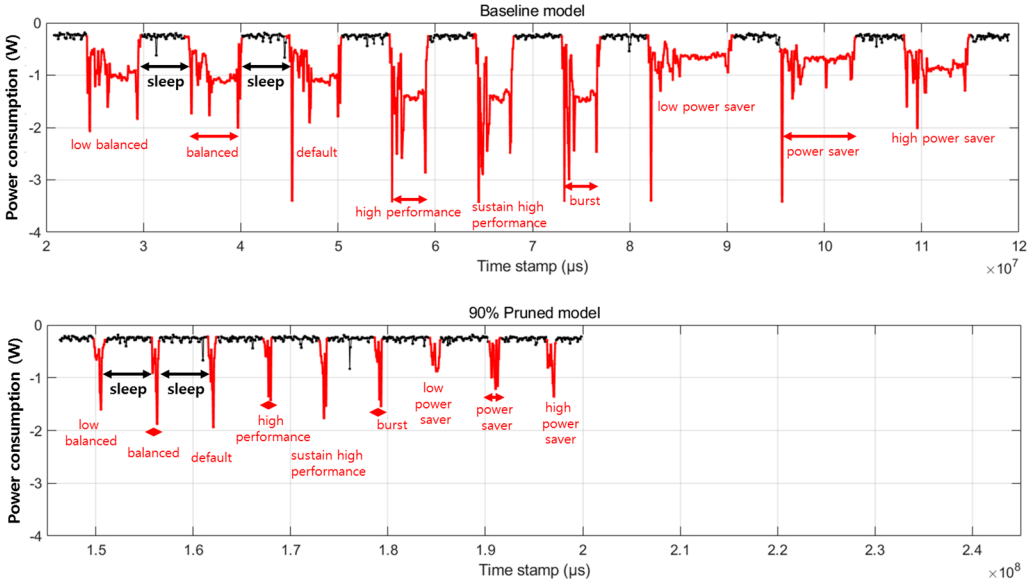

Figure 12. Results of measuring the power consumption of Slimmed DeepLabV3+ on FloodNet using Snapdragon profiler.

Tables 7 and 8 show the inference speed and power consumption per inference for four representative operating modes: 'burst', 'balanced', 'power saver', and 'high performance'. Power consumption can be thought of as the area of the graph in Figure 12 for each mode. Since this is an integration of power over the time axis, we used 'Wh' as the unit and 'mWh' due to the small scale. In Tables 7 and 8, we can find that the inference speed of the slimmed DeepLabV3+ was about 10 times faster than that of the baseline regardless of operating modes. We also found that the slimmed network consumes about 1/25 less power than the baseline when processing the same number of images.

Table 7. Inference speed and power consumption of slimmed DeepLabV3+ with 90% pruning ratio according to operating modes in QCS610 DSP for FLAME.

Operating Mode	Baseline		Slimmed	
	Speed [1]	Power [2]	Speed	Power
burst	0.76	1.238	7.43	0.048
power saver	0.34	2.453	5.91	0.040
balanced	0.62	1.559	9.90	0.045
high performance	0.76	1.163	6.88	0.034

[1] Speed means inference speed (fps). [2] Power means measured power consumption (mWh) during the execution of one frame on DSP.

Table 8. Inference speed and power consumption of slimmed DeepLabV3+ with 90% pruning ratio according to operating modes in QCS610 DSP for FloodNet.

Operating Mode	Baseline		Slimmed	
	Speed [1]	Power [2]	Speed	Power
burst	0.77	1.179	6.65	0.022
power saver	0.34	2.467	8.12	0.044
balanced	0.62	1.731	13.48	0.026
high performance	0.76	1.189	11.73	0.025

[1] Speed means inference speed (fps). [2] Power means measured power consumption (mWh) during the execution of one frame on DSP.

5.6. Discussion

As shown in Table 3, in terms of accuracy performance, DeepLabV3+ outperforms DeepLabV3 by approximately 5–6 percentage points in mIoU for forest fire detection. This means that DeepLabV3+, which is robust to object boundary recovery, is more effective for segmenting forest fires with small flame sizes as shown in Figure 9. On the other hand, in terms of edge-computing performance, DeepLabV3 seems to be more effective than DeepLabV3+. The inference speed of DeepLabV3 is about two times faster than DeepLabV3+, and the GFLOPs of DeepLabV3 are about six times smaller than DeepLabV3+. For flood detection, DeepLabV3+ has a higher mIoU than Deeplab3 by about one percentage point, as shown in Table 6. On the other hand, the inference speed of Deeplab3 is about 1.8 times faster than Deeplab3+, and the GFLOPs of DeepLabV3 are about nine times smaller than DeepLabV3+. Since there is a trade-off between accuracy and speed, DeepLabV3+ may be more effective for accuracy-oriented applications, and Deeplab3 may be more effective for speed-oriented applications. In summary, since the purpose of this paper is to implement a forest fire or flood monitoring system on the edge device of a drone, it is more effective to apply DeepLabV3, which is superior in terms of computation and inference speed.

The power consumption results obtained from the experiments show the availability of the proposed system when deployed on a real drone. For example, when the flood detection system based on the slimmed network is operated in 'high performance' mode, it consumes 0.025 mWh of energy to infer one image, as shown in Table 8. If the system processes ten images per second for an hour, it consumes $10 \times 60 \times 60 \times 0.025$ mWh = 900 mWh = 0.9 Wh. Assuming that a lithium polymer battery, commonly used in drones, has a power (energy) of 19.2 Wh, the proposed system uses only about 4.7% of the drone's battery. On the other hand, if a baseline semantic segmentation network is used, it consumes 1.189 mWh of energy to infer a single image and processes 0.76 frames per second, so it takes $0.76 \times 60 \times 60 \times 1.189$ mWh = 3253.104 mWh = 3.253104 Wh of energy to run the system for one hour. In this case, it is not only difficult to perform in real time but also uses about 17% of the battery's power. In conclusion, the proposed system with the slimmed network enables real-time semantic segmentation and can significantly benefit the flight time of drones.

6. Conclusions and Future Work

This paper proposes a monitoring system for real-time detection of floods and forest fires using UAVs such as drones. The proposed system consists of an embedded board equipped with a Qualcomm QCS610 SoC and deep learning-based semantic segmentation models (DeepLabV3 and DeeLabV3+) for detecting wildfires or floods. However, since these models are large in scale and have many parameters, it is difficult to operate them in real time on embedded devices. To solve this problem, this paper applied channel pruning-based network slimming to generate slimmed DeepLabV3 and V3+. The experimental results showed that for FLAME, slimmed DeepLabV3+ has the mIoU accuracy of 88.29% and an inference speed of 10.92 fps. For FloodNet, the mIoU accuracy of slimmed DeepLabV3+ was 94.15%, and the inference speed was 13.26 fps. The slimmed networks ported to DSPs showed little performance degradation compared to the baseline, but the inference speed was about 20 times faster, and the model size and computation (GFLOPs) were reduced by about 90%. The experimental results showed that slimmed networks could accurately detect forest fires and floods in real time at low power with little performance degradation on embedded devices. As a result, the proposed system is suitable for implementing a UAV-based real-time monitoring system for detecting forest fires and floods.

In future work, we plan to apply the proposed system to video data captured during day and night and in adverse weather conditions, utilizing the embedded camera device used in this paper. In addition, we will compare various semantic segmentation networks to find a more suitable one for flood or forest fire detection.

Author Contributions: Conceptualization, Y.J.L., H.G.J. and J.K.S.; methodology, Y.J.L., H.G.J. and J.K.S.; software, Y.J.L.; validation, Y.J.L., H.G.J. and J.K.S.; formal analysis, Y.J.L., H.G.J. and J.K.S.; investigation, Y.J.L.; resources, Y.J.L.; data curation, Y.J.L.; writing—original draft preparation, Y.J.L.; writing—review and editing, H.G.J. and J.K.S.; visualization, Y.J.L.; supervision, H.G.J. and J.K.S.; project administration, J.K.S.; funding acquisition, J.K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2020R1A6A1A03038540).

Data Availability Statement: The data presented in this study are openly available in reference numbers [30,31].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mohapatra, A.; Trinh, T. Early Wildfire Detection Technologies in Practice-A Review. *Sustainability* **2022**, *14*, 12270. [CrossRef]
2. Guan, Z.; Miao, X.; Mu, Y.; Sun, Q.; Ye, Q.; Gao, D. Forest Fire Segmentation from Aerial Imagery Data Using an Improved Instance Segmentation Model. *Remote Sens.* **2022**, *14*, 3159. [CrossRef]
3. What Is Digital Signal Processor: Working & Its Applications. Available online: https://www.elprocus.com/digital-signal-processor/ (accessed on 13 October 2023).
4. Springer, T.; Eiroa-Lledo, E.; Stevens, E.; Linstead, E.J. On-Device Deep Learning Inference for System-on-chip (SoC) Architectures. *Electronics* **2021**, *10*, 689. [CrossRef]
5. Qualcomm Hexagon 685 DSP Is a Boon for Machine Learning. Available online: https://www.xda-developers.com/qualcomm-snapdragon-845-hexagon-685-dsp/ (accessed on 13 October 2023).
6. Hatcher, W.G.; Yu, W. A Survey of Deep Learning: Platforms, Applications and Emerging Research Trends. *IEEE Access* **2018**, *6*, 24411–24432. [CrossRef]
7. Deng, B.L.; Li, G.; Han, S.; Shi, L.; Xie, Y. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* **2020**, *18*, 485–532. [CrossRef]
8. Safavi, F.; Chowdhury, T.; Rahnemoonfar, M. Comparative Study Between Real-Time and Non-Real-Time Segmentation Models on Flooding Events. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 4199–4207.
9. Safavi, F.; Rahnemoonfar, M. Comparative Study of Real-Time Semantic Segmentation Networks in Aerial Images During Flooding Events. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2023**, *16*, 15–31. [CrossRef]
10. Hernández, H.; Cecilia, J.M.; Cano, J.; Calafate, C.T. Flood Detection Using Real-Time Image Segmentation from Unmanned Aerial Vehicles on Edge-Computing Platform. *Remote Sens.* **2022**, *14*, 223. [CrossRef]
11. Gebrehiwot, A.; Hashemi-Beni, L.; Thompson, G.; Kordjamshidi, P.; Langan, T.E. Deep Convolutional Neural Network for Flood Extent Mapping Using Unmanned Aerial Vehicles Data. *Sensors* **2019**, *19*, 1486. [CrossRef] [PubMed]
12. Barkhade, V.; Mahakarkar, S.; Agrawal, R.; Dhule, C.; Morris, N.C. Flood Extent Mapping with Unmanned Aerial Vehicles Data using Deep Convolutional Neural Network. In Proceedings of the 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS), Coimbatore, India, 14–16 June 2023; pp. 466–471.
13. Ghali, R.; Akhloufi, M.A. Deep Learning Approaches for Wildland Fires Remote Sensing: Classification, Detection, and Segmentation. *Remote Sens.* **2023**, *15*, 1821. [CrossRef]
14. Li, M.; Zhang, Y.; Mu, L.; Xin, J.; Yu, Z.; Jiao, S.; Liu, H.; Xie, G.; Yingmin, Y. A Real-time Fire Segmentation Method Based on A Deep Learning Approach. *IFAC-PapersOnLine* **2022**, *55*, 145–150. [CrossRef]
15. Wang, S.; Chen, T.; Lv, X.; Zhao, J.; Zou, X.; Zhao, X.; Xiao, M.; Wei, H. Forest Fire Detection Based on Lightweight Yolo. In Proceedings of the 2021 33rd Chinese Control and Decision Conference (CCDC), Kunming, China, 22–24 May 2021; pp. 1560–1565.
16. Xiong, C.; Yu, A.; Rong, L.; Huang, J.; Wang, B.; Liu, H. Fire detection system based on unmanned aerial vehicle. In Proceedings of the 2021 IEEE International Conference on Emergency Science and Information Technology (ICESIT), Chongqing, China, 22–24 November 2021; pp. 302–306.
17. Tahir, H.U.A.; Waqar, A.; Khalid, S.; Usman, S.M. Wildfire detection in aerial images using deep learning. In Proceedings of the 2022 2nd International Conference on Digital Futures and Transformative Technologies (ICoDT2), Rawalpindi, Pakistan, 24–26 May 2022; pp. 1–7.
18. Mseddi, W.S.; Ghali, R.; Jmal, M.; Attia, R. Fire Detection and Segmentation using YOLOv5 and U-NET. In Proceedings of the 2021 29th European Signal Processing Conference (EUSIPCO), Dublin, Ireland, 23–27 August 2021; pp. 741–745.
19. Cao, X.; Su, Y.; Geng, X.; Wang, Y. YOLO-SF: YOLO for Fire Segmentation Detection. *IEEE Access* **2023**, *11*, 111079–111092. [CrossRef]
20. Almeida, J.S.; Huang, C.; Nogueira, F.G.; Bhatia, S.; de Albuquerque, V.H.C. EdgeFireSmoke: A Novel Lightweight CNN Model for Real-Time Video Fire–Smoke Detection. *IEEE Trans. Ind. Inform.* **2022**, *18*, 7889–7898. [CrossRef]

21. Liu, S.; Cheng, J.; Liang, L.; Bai, H.; Dang, W. Light-Weight Semantic Segmentation Network for UAV Remote Sensing Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 8287–8296. [CrossRef]
22. Rosas-Arias, L.; Benitez-Garcia, G.; Portillo-Portillo, J.; Olivares-Mercado, J.; Sanchez-Perez, G.; Yanai, K. FASSD-Net: Fast and Accurate Real-Time Semantic Segmentation for Embedded Systems. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 14349–14360. [CrossRef]
23. He, W.; Wu, M.; Liang, M.; Lam, S.-K. CAP: Context-Aware Pruning for Semantic Segmentation. In Proceedings of the 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, 3–8 January 2021; pp. 959–968.
24. Chen, X.; Zhang, Y.; Wang, Y. MTP: Multi-Task Pruning for Efficient Semantic Segmentation Networks. In Proceedings of the 2022 IEEE International Conference on Multimedia and Expo (ICME), Taipei, Taiwan, 18–22 July 2022; pp. 1–6.
25. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning Efficient Convolutional Networks through Network Slimming. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 2755–2763.
26. Wang, T.; Cao, R.; Wang, L. FE-YOLO: An Efficient and Lightweight Feature-Enhanced Fire Detection Method. In Proceedings of the 2022 3rd International Conference on Electronics, Communications and Information Technology (CECIT), Sanya, China, 23–25 December 2022; pp. 253–258.
27. Qualcomm Neural Processing SDK. Available online: https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk (accessed on 13 October 2023).
28. Nagel, M.; Fournarakis, M.; Amjad, R.A.; Bondarenko, Y.; van Baalen, M.; Blankevoort, T. A White Paper on Neural Network Quantization. *arXiv* **2021**, arXiv:2106.08295.
29. Choi, K.; Wi, S.M.; Jung, H.G.; Suhr, J.K. Simplification of Deep Neural Network-Based Object Detector for Real-Time Edge Computing. *Sensors* **2023**, *23*, 3777. [CrossRef] [PubMed]
30. Rahnemoonfar, M.; Chowdhury, T.; Sarkar, A.; Varshney, D.; Yari, M.; Murphy, R.R. FloodNet: A High Resolution Aerial Imagery Dataset for Post Flood Scene Understanding. *IEEE Access* **2021**, *9*, 89644–89654. [CrossRef]
31. Shamsoshoara, A.; Afghah, F.; Razi, A.; Zheng, L.; Fulé, P.Z.; Blasch, E. Aerial Imagery Pile Burn Detection Using Deep Learning: The FLAME Dataset. *Comput. Netw.* **2021**, *193*, 108001. [CrossRef]