



OPEN HashEye: a real-time on-drone high-resolution tiny object detection via spatial pruning

Hyeonji Hong¹, Nakyeong Lee¹, Kwangwoo Jang², Gague Kim², Yangjae Jeong² & Chanyoung Oh¹✉

While recent deep learning-based object detection has achieved great success in various fields, it remains challenging to find tiny objects in aerial imagery on-the-fly using mobile devices. Since mobile platforms such as drones operate with limited onboard computing power, handling high-resolution images to find tiny objects with compute-intensive deep learning-based applications often fails to meet their real-time constraints. To mitigate this problem, we propose HashEye, a novel framework that enables fast on-drone tiny object detection by efficiently suppressing spatial redundancy in aerial imagery. HashEye utilizes a lightweight hashing algorithm to rapidly scan image patches; patches exhibiting high hash collision frequencies are identified as background and suppressed. Subsequently, the remaining salient patches are dynamically rearranged into a hardware-friendly dense image for efficient inference. Experimental results on two real-world datasets demonstrate that HashEye achieves up to a $5.25\times$ speedup compared to the baseline, maintaining detection capability.

Unmanned Aerial Vehicles (UAVs) and drones have established themselves as essential platforms for a wide range of mission-critical applications, such as reconnaissance, surveillance, and disaster rescue¹. The core of such missions lies in the real-time analysis of collected video streams to enable immediate decision-making. In particular, due to the inherent nature of drones operating at high altitudes or monitoring distant targets, objects of interest occupy only a tiny portion of the image frame as shown in Fig. 1. Therefore, processing high-resolution imagery beyond FHD (1920×1080), such as 4K (3840×2160) or higher, is imperative to identify these tiny objects effectively.

However, achieving real-time processing of high-resolution video streams in resource-constrained onboard environments is extremely challenging. This is not merely a performance issue but directly correlates with mission failure. The following two scenarios illustrate why both high resolution and real-time processing must be simultaneously guaranteed.

First, in case of Urban Air Mobility (UAM), a vehicle flying at hundreds of km/h must detect hostile objects ahead at an early stage for safe operation. A hostile vehicle, which is hundreds of meters away, would occupy only a few pixels even in 4K imagery; thus, down-sampling the image for fast detection leads to the loss of critical visual features, making detection impossible (Fig. 1a). Conversely, taking as input 4K images directly into a deep learning model on a resource-constrained onboard mission computer causes a surge in inference latency, making it infeasible to take action for collision avoidance.

Second, continuous frame processing is critical in early wildfire detection scenarios. Initial embers in dense forests are often occluded by trees and observed only intermittently (Fig. 1b). If frame skipping techniques are applied to reduce the computational burden of 4K video, there is a high risk of missing the critical frame which contains the ember. Therefore, high-resolution object detection must be performed in real-time for every frame to prevent mission failure.

Despite these requirements, current mobile deep learning systems face severe limitations due to hardware constraints. Edge devices suffer from significant latency ranging from hundreds of milliseconds to several seconds when running existing object detection models like YOLO² on 4K inputs. Existing approaches to address high resolution images are also not suited for mobile environments. For example, slicing-based techniques such as SAHI³ improve detection performance for small objects, but they linearly increase the number of inference calls by dividing a 4K image into dozens of slices. Even though it achieves high detection rate, latency is also increased, making real-time processing infeasible.

¹Department of Software, Kongju National University, Chungnam 31080, South Korea. ²Air Mobility Research Division, Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea. ✉email: cyoh@kongju.ac.kr

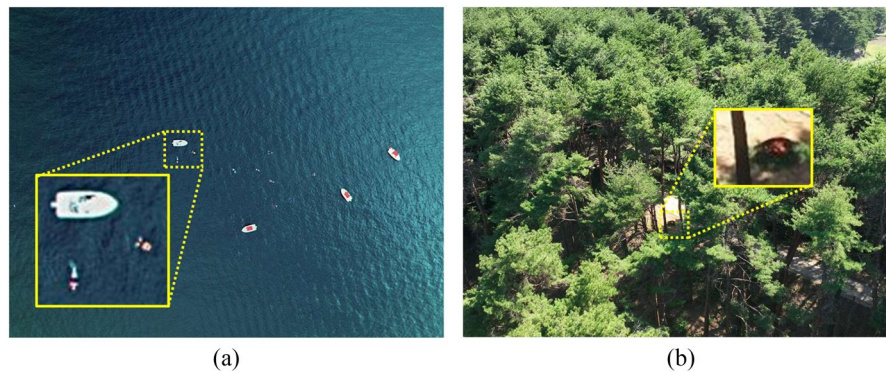


Fig. 1. An example of drone-captured image. Simple-textured background occupies most portion of the image and objects of interest occupy only a tiny portion of it.

Meanwhile, simply masking background regions as zero does not lead to practical acceleration. Modern GPU accelerators are optimized for General Matrix Multiply (GEMM) operations on dense grid structures. Since simple masking does not destroy this grid topology, the hardware still performs multiplication operations even on zero-filled background regions, wasting computational resources. If conditional branching is introduced to skip zero regions, it induces severe non-coalesced memory access problem and warp divergence in the GPU's SIMD architecture, resulting in a huge degradation of overall processing speed. In addition, special sparse algebra libraries such as cuSPARSE cannot be a solution as they focus on much higher sparsity in computational science domain.

We aim to systematically address this problem by exploiting the inherent characteristics of drone imagery. High-resolution aerial images exhibit significant spatial redundancy, predominantly occupied by background regions such as vast skies, oceans, and forests where no objects of interest exist. Our goal is to drastically reduce the computational load while preserving the benefits of high-resolution input by pruning and reconstructing these background regions prior to the deep learning model execution. Crucially, this is done in a manner that preserves the parallel processing structure of the GPU (e.g., maintaining a dense grid topology), thereby avoiding hardware inefficiencies.

In this paper, we propose HashEye, a novel framework that accelerates high-resolution tiny object detection in edge drone environments by efficiently suppressing spatial redundancy. Instead of relying on computationally intensive vision models, HashEye utilizes a data-independent and low-cost hashing algorithm to rapidly scan patches of the input image. It instantly filters out patches with a high probability of being background based on hash collision frequency, and selectively aggregates salient patches potentially containing objects to reconstruct a dense image. As this reconstructed image is directly compatible with existing object detection models such as YOLO, HashEye achieves immediate acceleration without requiring any modifications to the model architecture.

In summary, this paper makes the following contributions:

- We identify the necessity of on-board frame-by-frame processing for high-resolution aerial imagery to guarantee mission success in dynamic environments, challenging the conventional frame-skipping or down-sampling approaches.
- We design and implement a framework that integrates an SimHash-based lightweight pre-processing technique with a dynamic image reconstruction technique, significantly reducing end-to-end latency while maintaining detection accuracy.
- We validate the feasibility of performing real-time on-board analysis of 4K video streams on resource-constrained drones, as shown in our experiments on a real embedded device.

Related work

Recent trends in aerial and video analysis

Conventional object detection models, such as YOLO², demand huge computational costs when processing high-resolution images as they are. Moreover, naive down-sampling leads to the loss of critical features for tiny objects, resulting in detection failures. To accurately detect tiny objects, slicing-based techniques, such as SAHI³, have been proposed to enhance accuracy by dividing a high-resolution image into multiple small patches for inference. Another direction is considering both coarse- and fine-grained features^{4–6} to address the multi-scale challenges inherent in drone-view imagery. Recent advancements in aerial and video analysis have increasingly focused on sophisticated representation learning and architectural efficiency⁷. For instance, several studies have leveraged dual-space representation learning to better capture hierarchical structures and global contexts in video violence detection^{8,9}, person re-identification¹⁰, and unsupervised anomaly detection^{11,12}. Zhang et al.¹³ proposed a lightweight distillation network to aerial person detection for edge deployment.

While these methods primarily focus on optimizing latent representations or model architectures, HashEye distinguishes itself by addressing the problem at the input level. By utilizing SimHash-based spatial pruning and dense image rearrangement, HashEye significantly reduces computational redundancy before it reaches the model, offering a complementary approach to the aforementioned architectural optimizations.

Redundancy pruning techniques

Recent works in Vision Transformers^{14,15} have demonstrated the efficacy of token pruning for acceleration. However, such Transformer-based methods are computationally expensive, even though they show promising detection capability, making it difficult to deploy in real-time drone systems. There also have been continuous efforts to reduce computation of CNN-based methods by skipping less informative region in videos or high-resolution images. SBN¹⁶ optimized sparse CNN by gathering and scattering the dense parts of input. This approach is suitable for handling LiDAR-based inputs. DeepMon¹⁷ caches computation of patches in continuous video frames and reuse them in later frames if the similarity between patches is high. DeepReuse¹⁸ implemented much general computation reuse. It evaluates similarities among patches within an image as well as across independent images, and builds a representative patch among the similar patches. Then, only those representative patches involve the following computation passes. The results of computation are reused by all other patches.

While one might consider generating sparse images via simple masking and leveraging sparse algebra libraries (e.g., cuSPARSE), this approach induces control flow divergence on the GPU's SIMD architecture, leading to performance degradation. Similarly, Sparse Convolution¹⁹ techniques are primarily optimized for 3D point cloud processing, making them highly complex to implement for 2D images. In contrast, HashEye rearranges salient patches into a dense image, thereby maximizing the parallel processing efficiency of the GPU.

Dynamic routing techniques^{20–22}, such as BlockDrop²⁰ and SkipNet²¹, determine skipping at the layer level but do not directly address spatial redundancy.

Deep learning on mobile devices

Various techniques have been investigated to accelerate deep learning inference on mobile and edge devices. Offloading and distributed processing frameworks^{23,24}, such as Neurosurgeon²⁵, offload compute-intensive layers to the cloud; however, in high-mobility environments like drones, guaranteeing real-time performance is challenging due to network bandwidth instability.

μ Layer²⁶ accelerates CNN applications by utilizing the CPU and GPU of a mobile SoC in parallel. On the other hand, Several works^{27,28} accelerate CNN inference on mobile GPUs by exploiting weight or activation sparsity. Meanwhile, layer fusion²⁹ or persistent kernel³⁰ approaches attempted to achieve fast CNN execution by effectively eliminating kernel launch overhead.

HashEye operates complementarily with these existing lightweight techniques by eliminating spatial redundancy in the input data at the early stage of inference in a model-agnostic manner.

HashEye: overview

In this section, we define the system design considerations for realizing high-resolution tiny object detection in mobile drone environments and present the overall architecture of HashEye designed to satisfy these requirements.

Design considerations

We have identified three essential requirements and constraints that a drone-based real-time object detection system must satisfy.

On-board processing without cloud offloading

Since commercial cellular networks are primarily designed for terrestrial communication and optimized for down-link traffic, transmitting 4K video streams in real-time on drones, operating at high altitude, causes severe network bottlenecks, degrading the overall system response time. In such environments, cloud offloading via networks is unreliable due to the limited bandwidth and long round-trip times (RTT). Consequently, all computations must be performed independently within the on-board mission computer.

Real-time constraints for aerial mobility

For platforms operating at high altitudes and high velocities, such as UAM and surveillance drones, a latency of even hundreds of milliseconds can lead to catastrophic consequences, such as collisions or target loss. Therefore, the system must deliver inference results in real-time for incoming frames to ensure safe and successful operation.

Entire-frame analysis without frame skipping

Down-sampling or resizing is commonly used in general object detection. However, it leads to the loss of information for tiny objects that occupy only a few pixels in 4K imagery. Moreover, existing methods that selectively process only representative frames (a.k.a. frame skipping) pose a risk of target loss. Scenarios like early wildfire detection, where the initial embers are extremely small and occluded by the background such as trees, reveal this issue. Such a scarce target object may appear in just one frame out of entire video streams, necessitating observation of entire frames without skipping. Therefore, the system must perform a search on entire frames while preserving the original resolution.

System architecture

Incorporating the aforementioned design considerations, we propose HashEye, a framework that enables real-time on-drone object detection for 4K video streams via spatial pruning. Fig. 2 illustrates an overview of HashEye framework.

HashEye first partitions the input image into fixed-size patches and applies a lightweight SimHash algorithm to each patch. Based on the generated hash signatures, patches with a high probability of being background, which is indicated by a high frequency of hash collisions, are instantly filtered out. It is worth noting that the

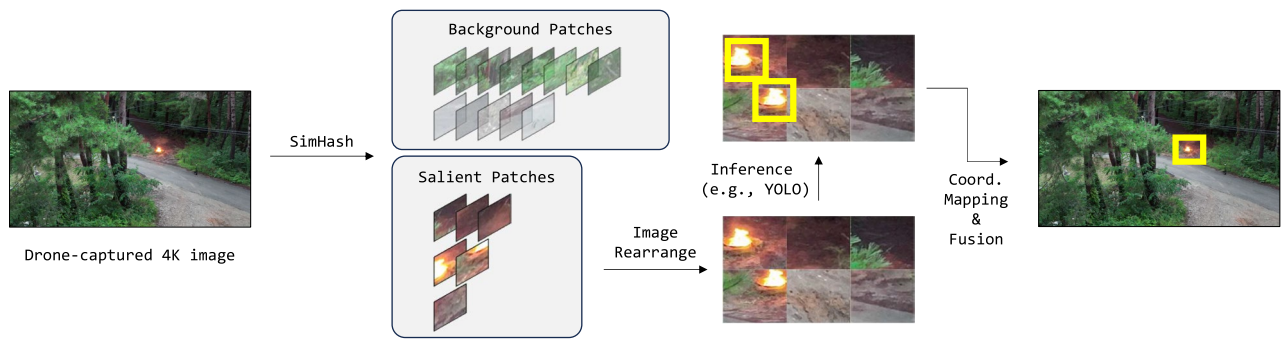


Fig. 2. Overview of HashEye.

primary goal of this stage is not to precisely identify patches containing objects, but to rapidly prune background regions that are highly unlikely to contain objects of interest.

Once the salient patches are identified, HashEye reconstructs a new dense image composed exclusively of these patches. This reconstruction is necessary because existing deep learning models are not suited for processing sparse inputs. Thus, we reorganize the data into a GPU-friendly dense format to maximize computational efficiency. To mitigate the boundary effect where objects might be truncated at patch edges, each patch is copied in a dilated form, including its surrounding context. The reconstructed image contains significantly fewer pixels than the original 4K frame, thereby dramatically accelerating the inference speed.

Upon reconstruction, object detection is performed using standard models such as YOLO. HashEye is a model-agnostic framework, compatible with any off-the-shelf object detection architecture. Since the input resolution is preserved without down-sampling, the model retains the capability to detect tiny objects. Finally, because the detected coordinates correspond to the reconstructed image, they are inversely mapped to the original 4K coordinate system to yield the final detection results. Note that the bounding boxes on the same object are integrated.

Implementation

In this section, we describe the implementation details of HashEye's core mechanisms. Instead of exhaustively processing the massive data of 4K imagery, HashEye prunes redundant background regions quickly via SimHash-based techniques and reconstructs the data into a format optimized for deep learning models.

Spatial redundancy suppression

To eliminate spatial redundancy, the system must rapidly determine the visual similarity between image patches. While sophisticated visual comparison techniques may offer higher accuracy, they are computationally expensive for real-time processing. In this paper, we adopt SimHash-based Locality Sensitive Hashing (LSH). LSH maps data points that are close in a high-dimensional space to the same hash bucket with high probability.

Given an input image I , we partition it into a set of $N \times N$ patches, denoted as $P = \{p_1, p_2, \dots, p_M\}$. Each patch p_i is flattened into a D -dimensional vector $v_i \in \mathbb{R}^D$, where $D = N \times N \times C$ and C is the channel count. LSH transforms v_i into an L -bit hash signature using a projection matrix $R \in \mathbb{R}^{L \times D}$, which consists of L random hyperplanes. The hash function $H(v_i)$ is defined as follows:

$$H(v_i) = \text{sgn}(R \cdot v_i)$$

where $\text{sgn}(\cdot)$ is a sign function that returns 1 if the input is non-negative and 0 otherwise. The resulting L -bit vector serves as a fingerprint that compactly represents the visual features of patch p_i . Patches with high cosine similarity are highly likely to share the same hash value.

In drone imagery, background regions such as forests, oceans, and skies exhibit repetitive textures. Consequently, these background patches tend to collide intensively within specific hash buckets in the LSH space. HashEye leverages this characteristic to identify background regions.

We calculate the frequency $f(b_k)$, which is the number of patches mapped to each hash bucket b_k . Based on our premise that the background occupies a dominant portion of the total patches, the top buckets with high frequencies are highly likely to correspond to the background. To determine this quantitatively, we sort the bucket list by frequency and employ the Kneedle algorithm³¹. This algorithm generates a difference curve by calculating the difference between the normalized histogram and the line $y = x$. Using the local maximum of this curve (y_{lmax}) and a sensitivity hyperparameter S , it calculates a threshold T_{lmax} as shown below. The point where the difference curve first drops below this threshold is identified as the knee point in this work.

$$T_{lmax} = y_{lmax} - S \cdot \frac{1}{2^L - 1}$$

where 2^L is the number of buckets. This process acts as a filter that retains only the salient patches potentially containing objects. A larger S value indicates lower sensitivity, while a smaller value implies higher sensitivity.

It is crucial to note that the primary objective of spatial redundancy pruning at this stage is not to precisely identify patches containing objects of interest, but to rapidly suppress patches that are highly likely to be background. Therefore, the hyperparameter must be conservatively set to minimize false negatives.

Dynamic image rearrangement

The filtered salient patches are sparsely distributed across the original image. Since GPUs are inefficient at processing such sparse inputs, a rearrangement process is essential to reorganize them into a format conducive to efficient processing. During the spatial pruning phase, patches are uniformly partitioned independent of the image content. Consequently, objects of interest may not be centered within a patch and often lie on the boundaries. Simply copying the $N \times N$ salient patches can make objects fragmented, which is a critical factor leading to severe degradation in detection performance.

This issue is resolved through a dilated copy strategy. When copying a salient patch p_i to the canvas, we extract an expanded region of size $K \times K$ ($K > N$), rather than the original $N \times N$ size, to include surrounding spatial context. The expanded size K is determined based on the maximum expected size of the target objects. The reconstruction process is executed via a parallel memory copy kernel on the GPU, which maps the source coordinates of salient patches to sequential coordinates in the reconstructed image I_{re} . The resulting I_{re} consists of significantly fewer pixels than the original 4K frame. Note that HashEye preserves complete object information as the original pixels in the salient patch are neither down-sampled nor resized.

Although dilated copying may introduce minor redundancy where adjacent salient patches overlap, since K is likely to be much larger than N , this overhead is negligible compared to the massive reduction achieved by suppressing background redundancy. While it is conceivable to group densely clustered salient patches and dynamically adapt the dilation factor K for copying, this approach necessitates the sequential calculation of destination coordinates on the canvas. This dependency makes parallel memory copy operations unavailable, resulting in huge rearrangement overhead³². On the other hand, employing a fixed K enables the determination of location in canvas via a single simple atomic operation.

HashEye provides following two reconstruction strategies.

Mosaic image construction

The first approach is the Mosaic strategy, which arranges selected patches into a grid format on a large 2D canvas to generate a single packed image. We pack the dilated $K \times K$ salient patches into a pre-defined canvas of size $W_{canvas} \times H_{canvas}$. Since each patch is copied to the canvas concurrently by the parallel memory copy kernel, we manage the target position indices atomically to prevent data races. The coordinates (x_{mosaic}, y_{mosaic}) occupied by each patch on the canvas are maintained in a lookup table, which is subsequently used to restore the original coordinates.

Batch image construction

The second approach stacks salient patches along the batch dimension to create a 4D tensor of shape (M', C, K, K) , where M' is the number of salient patches. Unlike the Mosaic strategy, this method eliminates the need for 2D coordinate calculations or 2D offset computations during memory copying, resulting in lower pre-processing overhead. However, since the number of salient patches M' varies dynamically for each input frame, this approach requires an inference engine that supports dynamic batch sizes.

Inference and post-processing

HashEye achieves high-speed processing by feeding only the dense information, of which spatial redundancy has been effectively suppressed in the pre-processing stage, into the inference engine. The inference workflow has two distinct paths depending on the adopted image reconstruction strategy.

Inference strategies

The mosaic-based reconstructed image, I_{re}^{mosaic} , serves as input for conventional object detection models such as YOLO. However, the artificial stitching of patches introduces high-frequency artifacts at patch boundaries (as shown in Fig. 2), resulting in discontinuities in the data distribution. This sharp shift plays one of the important factors degrading the detection accuracy of models trained solely on original imagery. To address this, we construct a specialized training dataset by transforming original images to mimic the topology of I_{re}^{mosaic} and fine-tune the model, so as to preserve robustness against boundary artifacts.

On the other hand, the batch-constructed 4D tensor, I_{re}^{batch} , is utilized as input for image classification models like MobileNet³³. The individual patches that survived at the spatial pruning stage are small ($K \times K$) and highly likely to contain a single object. Exploiting this characteristic, employing a simple classification model is computationally more efficient than using a detection model that simultaneously predicts both class and location. In addition, as we do not stitch the salient patches, there are no high-frequency artifacts. In this scheme, the patch indices inherently represent the object locations, eliminating the need for explicit bounding box regression.

Coordinate mapping and fusion

Since the bounding box coordinates (x', y') obtained from inference are relative to the reconstructed image, an inverse mapping process is required to translate them back to the original 4K coordinate system (x, y) . We restore the global coordinates by referencing the source index metadata recorded during the reconstruction phase. For batch-based inference, the classified patch itself is treated as the bounding box. Finally, we address the issue of duplicate detections arising from patch partitioning and dilated copying. An object on patch boundaries may be detected as fragmented pieces, or the same object may be redundantly detected across adjacent patches due to

the dilated copy. Conventional Non-Maximum Suppression (NMS) is not suited for this scenario as it discards overlapping boxes, failing to integrate the fragmented information of a single object. To resolve this, we adopt the Weighted Boxes Fusion (WBF) algorithm³⁴ for post-processing. We treat adjacent boxes with an Intersection over Union (IoU) greater than zero as detections of the same object, grouping them together. We then merge their coordinates to derive the tightest bounding box that encompasses the entire object region. This approach effectively reconstructs fragmented object information at patch boundaries and enhances detection reliability.

Evaluation

In this section, we evaluate the proposed HashEye framework on a real embedded platform and perform a comparative analysis with state-of-the-art object detection techniques for high-resolution tiny object detection.

Environmental setup

All experiments were conducted on the NVIDIA Jetson Orin NX (8GB), a platform widely adopted as an on-board mission computer for drones. We configured the power profile to the 15W mode. For the inference engine, we utilized TensorRT 10.3.0 and CUDA 12.6.

We applied CUDA-based parallelization to each module constituting the pipeline and optimized the kernels with our best using NVIDIA Nsight. For tasks that are not suited for GPU execution, we implemented a fallback mechanism to the CPU. Crucially, to minimize the latency overhead during Device-to-Host (D2H) and Host-to-Device (H2D) data transfers, we implemented zero-copy communication leveraging Unified Memory.

Regarding the backbone models, we employed YOLOv8² for the HashEye (mosaic) configuration and MobileNetv3³³ for the HashEye (batch) configuration. It is worth noting that HashEye is designed as a model-agnostic framework, allowing these underlying detection models to be easily replaced depending on the specific application requirements.

Dataset and model training

For the experiments, we utilized the SeaDroneSee³⁵ dataset to simulate maritime drone surveillance scenarios and the K-WF (Dataset titled 'Collection of wildfire prevention data' from The Open AI Dataset Project (AI-Hub, S. Korea); we call this dataset K-WF for brevity. All data information can be accessed through 'AI-Hub (<https://www.aihub.or.kr>)' dataset for early wildfire prevention scenario. We utilized the subset of the dataset consisting of video sequences: 24 in SeaDroneSee and 1 in K-WF. The LSH projection matrix of HashEye is generated and fixed based on the initial frame, and is subsequently reused for the remaining frames in the sequence. All input images were resized to a 4K resolution (3840×2160). Since we aim to find tiny objects in high-resolution images, images that do not contain small object (Small object based on COCO dataset.) at all are not used in any experiment including model training stage. Note that, in any frame where at least one small object was present, all other objects including large ones were fully retained and included in the evaluation.

All model training for object detection and classification in this study was conducted on an NVIDIA RTX 4090 environment. For each dataset, we constructed a total of four model configurations: (1) a detection model based on 1280×1280 input, (2) a detection model based on 640×640 input, (3) a mosaic-based detection model, and (4) a batch-based classification model. The 1280×1280 resolution is adopted as it is the maximum input size feasible for training under the GPU memory constraints of the NVIDIA RTX 4090 with a batch size of 1; this model is utilized for experiments on the baseline YOLO model for 4K inputs.

For the detection models (1) and (2), we constructed new datasets by resizing the original images to 1280×1280 and 640×640 , respectively, and fine-tuned them for 5 epochs using the default hyperparameter settings of YOLOv8. For the detection and classification models corresponding to (3) and (4), the original images were partitioned into patches of size 40×40 . We generated labels such that an object is classified as present only if its bounding box area constitutes at least 5% of the patch area. The mosaic-based detection model utilized inputs reconstructed 520×520 mosaic image, and was trained for 300 epochs using the YOLOv8 auto tuner. For the batch-based classification model, we modified the stride of the first convolution layer from 2 to 1 to minimize spatial loss in the initial stages and trained it for 20 epochs.

Overall result

The value of N was set to 8, while D and K were configured to 192 and 40, respectively. The number of hash function (i.e. L) depends on scenarios. For patches identified as salient, we perform a dilated copy with a target size of 40×40 . It is crucial to set N to a sufficiently small value; if N is large, even visually similar patches tend to map to distinct hash buckets. The confidence threshold was set to 0.3 for all experiments. The Intersection over Union (IoU) threshold used for evaluating detection accuracy, including mAP and Recall, was set to 10%. In the context of tiny object detection, the primary goal is to establish object presence (detection capability), rather than achieving high localization precision (accurately bounding the object's size and extent). The inherent difficulty of precisely bounding objects that span only a few pixels necessitates the use of a low IoU threshold, as higher thresholds would unfairly penalize successful detections that fail perfect alignment, obscuring the model's mission-critical performance. However, we provide commonly used metric (mAP@50) as well in "Detailed analysis" so as to facilitate a direct comparison with prior work.

Tables 1 and 2 presents the average inference latency and detection accuracy (recall and mAP). For comparison, we employed the original YOLOv8 as a baseline, along with SAHI³ and QueryDet⁶. SAHI performs inference by slicing the image to detect small objects, while QueryDet utilizes a sparse query-based mechanism to identify sparse tiny objects.

Experimental results demonstrate that the HashEye ($L = 3$, Mosaic) configuration achieves an ultra-low latency of 57 ms, delivering a $5.25\times$ speedup compared to YOLO on K-WF dataset. This performance corresponds to a throughput of approximately 17 FPS for 4K inputs, validating that near-real-time processing is

Method	Latency (ms)	Speedup	Recall	mAR @100	mAP @10
YOLOv8 (4K)	280	–	90.43	75.44	73.07
YOLOv8 (640)	28	10.00×	72.47	60.43	59.97
SAHI ³	270	1.03×	75.65	79.86	71.42
QueryDet ⁶	589	0.47×	75.91	80.54	48.44
HashEye ($L = 11$, Batch)	95	2.94×	65.10	59.54	65.09
HashEye ($L = 5$, Mosaic)	61	4.59×	76.55	73.62	54.37
HashEye ($L = 11$, Mosaic)	75	3.73×	88.84	87.24	65.99

Table 1. Performance comparison on SeaDroneSee dataset.

Method	Latency (ms)	Speedup	Recall	mAR @100	mAP @10
YOLOv8 (4K)	300	–	98.23	98.24	98.37
YOLOv8 (640)	28	10.71×	87.05	97.06	94.39
SAHI ³	282	1.06×	1.00	100	92.56
QueryDet ⁶	1662	0.18×	97.06	97.06	89.46
HashEye ($L = 7$, Batch)	388	0.77×	98.23	98.24	86.31
HashEye ($L = 3$, Mosaic)	57	5.25×	94.11	94.12	81.52
HashEye ($L = 7$, Mosaic)	174	1.72×	99.41	99.41	94.96

Table 2. Performance comparison on K-WF dataset.

guaranteed even on resource-constrained edge devices. However, it sacrifices some accuracy showing mAP of 81.52. HashEye (mosaic) tends to result in a bit of lower mAP compared to baseline YOLO. Even if HashEye has lower mAP, its recall scores are comparable to or higher than that of YOLO. For mission-critical domains such as drone-based SAR (Search and Rescue) or early wildfire detection, Recall is considered the primary performance metric; a high Recall score ensures that the drone does not miss a target. In order words, it is much crucial to detect true positives (TPs) correctly. As shown in Table 1, HashEye maintains comparable recall score (88.84) compared to the baseline (90.43). Even when evaluated based on mAR (mean Average Recall), HashEye (87.24) significantly outperforms the baseline (75.44). This result demonstrates the efficacy of HashEye in mission-critical domains. In addition, the lower mAP is not a result of a failure to detect objects. We will discuss it in later (“Detection capability” and “Analysis of false positives”).

SAHI showed marginal speed improvements due to the overhead induced by slicing. While QueryDet achieved the highest accuracy, its latency exceeds 1 second, which is infeasible to deploy for real-time drone missions.

Figure 3 shows visualized detection results for YOLOv8 and HashEye. The green and red labels indicate true positives and false positives, respectively. We can validate the HashEye’s detection capability, in which we obtain high true positives and low false negatives. On the other hand, YOLO with down-sampled input image of typical 640×640 size sometimes misses several objects. It is worth noting that this conventional down-sampling is already efficient for larger objects; it has only 28 ms of latency as shown in Tables 1 and 2. HashEye is specifically tailored for scenarios where high resolution is non-negotiable so as to detect tiny objects that lose critical visual cues when down-sampling is applied.

Detailed analysis

Impact of hash length (L)

The hash length (L) of the LSH algorithm is the key parameter determining the aggressiveness of the spatial pruning. As shown in Table 1, the $L = 5$ configuration exhibits the highest speed (61 ms) but a relatively low recall score of 73.62. Conversely, strengthening the filtering criteria by setting $L = 11$ increases the processing time to 75 ms but recovers the recall score to 87.24, outperforming the baseline (75.44). This suggests that users can flexibly manage the trade-off between speed priority and accuracy priority based on mission requirements.

Table 3 presents the number of clusters pruned by the LSH and Kneedle algorithms. Across both datasets, two clusters were consistently identified as background, regardless of the hash length (L) value. These two clusters account for the great part of the total patches. Consequently, HashEye achieves a pruning rate of up to 97.54% of the total patches. The object loss rate, where objects are completely missing within a sequence due to being mistakenly clustered into the background, is extremely small, remaining below 1.76% at most. These results underline the remarkably high spatial redundancy present in drone-captured imagery and strongly validate the efficacy of HashEye’s SimHash-based spatial pruning mechanism.

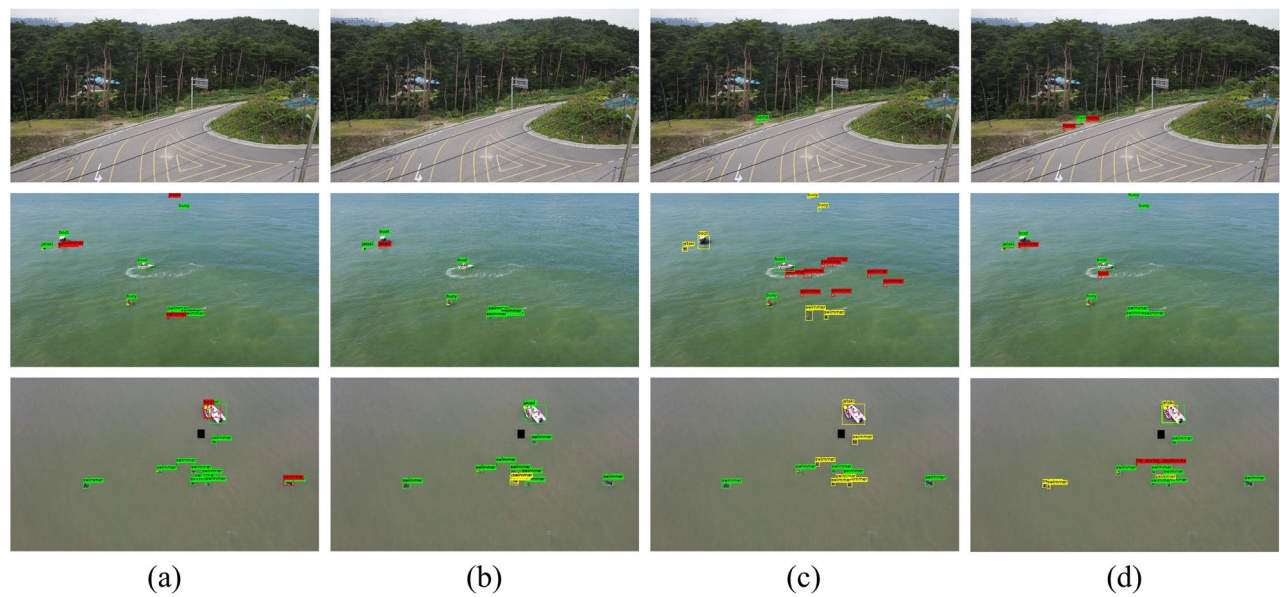


Fig. 3. Visualized detection results for each method: **(a)** YOLOv8 (4K), **(b)** YOLOv8 (640 × 640), **(c)** HashEye (batch), and **(d)** HashEye (mosaic), (Green: TP; Red: FP; Yellow: Correct detections with low IoU; misclassified as FP). Although HashEye exhibits some false alarms due to false positives, it demonstrates high true positive and low false negative performance, validating its applicability to real-world drone operations.

Dataset	L	# BG	Pruned	Object loss
		Clusters	Patches	Rate
SeaDroneSee	5	1.91	97.54%	1.29%
SeaDroneSee	11	2	96.44%	0.45%
K-WF	3	2	95.80%	1.76%
K-WF	7	2	80.59%	0.00%

Table 3. The number of background clusters, and ratio of pruned patches with corresponding object loss rate.

Method	TP (correct)	FP (false alarm)	FN (miss)
YOLOv8 (4K)	5872	1134	621
YOLOv8 (640)	4706	246	1787
HashEye (L = 5)	4971	3381	1522
HashEye (L = 11)	5769	5589	724

Table 4. Detailed detection analysis (TP/FP/FN) on the SeaDroneSee dataset.

Method	TP (correct)	FP (false alarm)	FN (miss)
YOLOv8 (4K)	167	9	3
YOLOv8 (640)	148	2	22
HashEye (L = 3)	160	200	10
HashEye (L = 7)	169	139	1

Table 5. Detailed detection analysis (TP/FP/FN) on the K-WF dataset.

Detection capability

We analyzed the actual detection capability rather than the simple accuracy metric (mAP@10) to understand why the mAP of HashEye (mosaic) is comparatively low. Tables 4 and 5 present the detailed True Positive (TP), False Positive (FP), and False Negative (FN) metrics for the two datasets, respectively.

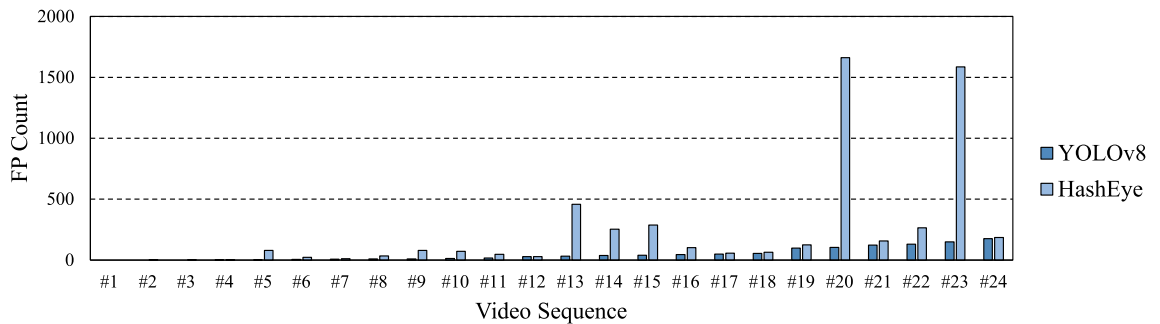


Fig. 4. The number of false positives (FPs) for each video sequence in the SeaDroneSee dataset.

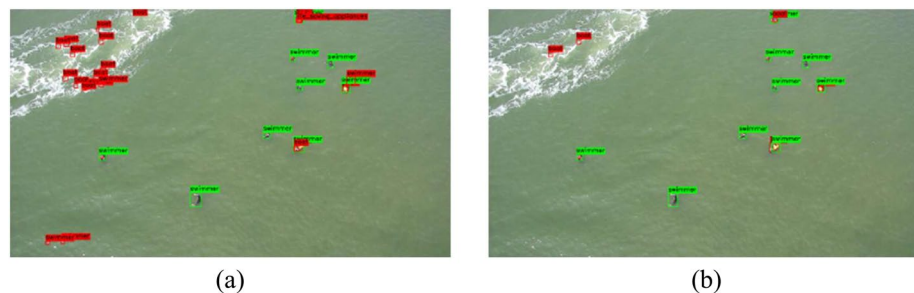


Fig. 5. Visualization of detection results according to confidence thresholds: (a) threshold = 0.3, (b) threshold = 0.75. As the threshold increases, the number of TPs is maintained while FPs decrease significantly.

The high count of FP, corresponding to false alarms where an object is detected where none exists, is the primary factor causing loss of the mAP. This high FP rate stems from the mosaic image structure itself. Since the reconstructed mosaic image is composed entirely of local patches, the detection model is confined to interpreting local features, losing the opportunity to leverage global feature interpretation. Consequently, patches containing elements visually similar to flames, such as streetlights or sunsets, are frequently misclassified as the target object for K-WF.

In contrast, the TP count (correctly detecting an existing object) and the FN count (failing to detect an existing object) show performance nearly analogous to the baseline. In drone-based surveillance and searching missions, reliably ensuring the existence of the object (High TP and Low FN) is significantly emphasized rather than false alarming. While frequent false alarms can be possibly mitigated through novel post-processing techniques or human intervention, missing a target directly results in mission failure. Therefore, this characteristic of HashEye is sufficiently acceptable within real-world operational environments.

Although the HashEye framework prioritizes object presence over pixel-perfect bounding box alignment, we provide results with the IoU threshold set to 50% to facilitate comparisons with previous studies. Based on the SeaDroneSee dataset, the baseline Recall@50 (mAP@50) for YOLOv8 (4K) and YOLOv8 (640) are 67.35 (62.77) and 50.39 (46.89), respectively; in comparison, HashEye achieves 54.63 (34.86) when $L = 5$ and 71.77 (48.26) when $L = 11$. While HashEye tends to exhibit lower mAP due to a relatively high number of FPs, it demonstrates superior Recall compared to the baseline by maintaining TPs while performing inference at high speeds.

Analysis of false positives

While the primary objective of HashEye is to maximize recall for mission-critical tasks, the relatively high FP count necessitates a deeper analysis to identify its root causes; we performed a per-video sequence analysis. Figure 4 illustrates the number of FPs for each sequence in the SeaDroneSee dataset, comparing HashEye with the YOLOv8 baseline. FPs are not uniformly distributed but rather concentrated within specific video sequences. Two video sequences (i.e., #20 and #23) account for 3249 false positives, representing 58% of the total FPs. Even within these two sequences, FPs do not occur indiscriminately; rather, they are primarily caused by environmental artifacts such as breaking waves, sun glint, and sea lines. Specifically, 99.67% of the FPs in sequence #20 occurred at the sea line, as well as 95.26% of the FPs in sequence #23 were caused by breaking waves.

As discussed in “Detection capability”, the cause of these FPs is the loss of interpretability for global features, which arises from the nature of the images being partitioned into local patches. However, the majority of FPs tend to exhibit relatively low confidence scores. As noted in “Overall result”, all experiments in this paper were conducted with a confidence threshold set to 0.3. Figure 5 illustrates the results when the confidence threshold is increased from 0.3 to 0.75: it can be observed that TPs do not decrease, while FPs are significantly reduced.

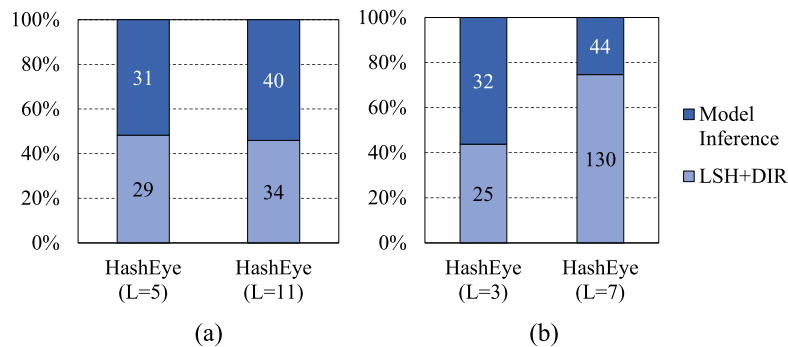


Fig. 6. Latency breakdown of the HashEye pipeline, comparing overhead against pure model inference time. Annotated values indicate execution time (ms): (a) SeaDroneSee, (b) K-WF.

Thus, the FP problem generated by HashEye is not insurmountable but can be sufficiently mitigated by minimal engineering effort depending on the specific scenario or context.

Overhead analysis

Figure 6 presents a latency breakdown of the HashEye pipeline; it compares the execution time of LSH and Dynamic Image Rearrangement (DIR) over pure model inference time. On average, these overheads account for 53.21% of the total end-to-end latency. It is crucial to note that all performance metrics evaluated in this paper are based on the complete end-to-end latency, which includes entire overheads as well as model inference time. Thus, while the overhead ratio appears substantial relative to the total latency, it proves that even with the added overhead, the reduction in computationally expensive model inference more than compensates for the preprocessing time, yielding a definitive net gain in end-to-end latency.

Through these efficient pre-processing mechanisms, we successfully shifted the computational burden from the computationally expensive neural network to the lightweight LSH and DIR modules.

Limitations and future work

Although this study validates HashEye as an effective solution for high-resolution tiny object detection on drone environments, several limitations remain.

Object scale variations

The current design of HashEye is predicated on the assumption that the size of the target object falls within the patch size (limited to 40×40 pixels in the experiments). While this assumption is valid in typical high-altitude drone surveillance scenarios, limitations arise when the drone operates closer to the ground (Low-altitude flight) or needs to detect large objects, such as ships. In case an object's size exceed the expanded patch size ($K \times K$), its visual features are fragmented across multiple patches. To address this, future work will introduce an adaptive patching mechanism that dynamically adjusts the patch size (N).

Sensitivity to background heterogeneity

The performance of HashEye's spatial redundancy pruning is maximized when the background texture is uniform, leading to frequent LSH collisions (e.g., sea, forest, sky). However, in complex urban areas or terrains with significant texture variations, the distribution of hash buckets tends to flatten. This increases the risk of background patches being mistakenly identified as salient patches and consequently not filtered. This directly results in reduced compression efficiency and slower inference speed. To ensure robustness in diverse environments, Learnable LSH mechanism can be adopted. This mechanism will adaptively learn or update the LSH projection matrix, R , based on the prevalent texture distribution of the flight environment, instead of using random hyperplanes.

Spatio-temporal redundancy pruning

The most critical direction for future research involves addressing temporal redundancy as well as spatial redundancy. Drone-captured video exhibits high correlation between consecutive frames. While HashEye currently processes each frame independently, leveraging cached hash maps or detection results from previous frames would allow us to aggressively skip computations for background regions that remain unchanged or areas where objects were previously absent.

Conclusion

In this paper, we proposed HashEye, a framework designed to realize real-time object detection of 4K high-resolution imagery on resource-constrained drone environments. For mission-critical applications such as maritime Search and Rescue or early wildfire detection, conventional approaches relying on down-sampling or cloud offloading fail to provide viable solutions due to the loss of fine-grained details essential for tiny objects and inherent communication latency.

To address this dilemma, HashEye introduces a novel pipeline that reduces spatial redundancy in the input data using a lightweight SimHash-based algorithm prior to heavy neural network inference. By pruning background patches and rearrange sparse salient patches into GPU-friendly dense images, HashEye enables efficient inference for subsequent detection models.

In our experiments on the NVIDIA Jetson Orin NX platform, which is frequently used as drone mission computer, we demonstrate that HashEye achieves up to $5.25\times$ speedup compared to YOLOv8. Furthermore, we verify that it preserves the detection capability for tiny objects by fully exploiting the 4K resolution, exhibiting even higher Recall score than the baseline. These results validate the feasibility of performing real-time, frame-by-frame high-resolution analysis without frame skipping on edge devices, rather than relying on high-performance servers.

Data availability

The datasets generated and/or analysed during the current study are available from <https://seadronesee.cs.uni-tuebingen.de/> (SeaDroneSee) and <https://www.aihub.or.kr/> (K-WF).

Received: 19 March 2026; Accepted: 30 April 2026

Published online: 06 May 2026

References

- Zhu, P. et al. Detection and tracking meet drones challenge. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2021).
- Jocher, G., Chaurasia, A. & Qiu, J. Yolo by ultralytics (2023). Version 8.0.0.
- Akyon, F. C., Altinuc, S. O. & Temizel, A. Slicing aided hyper inference and fine-tuning for small object detection. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)* (2022).
- Leng, J. et al. Pareto refocusing for drone-view object detection. *IEEE Trans. Circuits Syst. Video Technol.* **33**, 1320–1334 (2022).
- Zhang, X., Feng, Y., Zhao, Z., Ren, Y. & Mei, S. Rethinking coarse-to-fine fully convolutional transformers for salient maritime ship detection in optical remote sensing imagery. *J. Remote Sens.* **6**, 1038 (2026).
- Yang, C., Huang, Z. & Wang, N. Querydet: Cascaded sparse query for accelerating high-resolution small object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022).
- Zhang, X., Feng, Y., Wang, N., Lu, G. & Mei, S. Aerial person detection for search and rescue: Survey and benchmarks. *J. Remote Sens.* **5**, 0474 (2025).
- Leng, J. et al. Beyond euclidean: Dual-space representation learning for weakly supervised video violence detection. *Adv. Neural. Inf. Process. Syst.* **37**, 17373–17397 (2024).
- Leng, J. et al. Piercingeye: Dual-space video violence detection with hyperbolic vision-language guidance. *IEEE Trans. Pattern Anal. Mach. Intell.* (2025).
- Leng, J. et al. Dual-space video person re-identification. *Int. J. Comput. Vision* **133**, 3667–3688 (2025).
- Leng, J. et al. Dual-space normalizing flow for unsupervised video anomaly detection. *IEEE Trans. Image Process.* (2025).
- Mo, M. et al. A2seek: Towards reasoning-centric benchmark for aerial anomaly understanding. arXiv preprint [arXiv:2505.21962](https://arxiv.org/abs/2505.21962) (2025).
- Zhang, X. et al. Robust aerial person detection with lightweight distillation network for edge deployment. *IEEE Trans. Geosci. Remote Sens.* **62**, 1–16 (2024).
- Bolya, D. et al. Token merging: Your vit but faster. arXiv preprint [arXiv:2210.09461](https://arxiv.org/abs/2210.09461) (2022).
- Chen, X. et al. Sparsevit: Revisiting activation sparsity for efficient high-resolution vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2061–2070 (2023).
- Ren, M., Pokrovsky, A., Yang, B. & Urtasun, R. Sbnets: Sparse blocks network for fast inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
- Huynh, L. N., Lee, Y. & Balan, R. K. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2017).
- Ning, L. & Shen, X. Deep reuse: Streamline cnn inference on the fly via coarse-grained computation reuse. In *Proceedings of the ACM International Conference on Supercomputing*, 438–448 (2019).
- Park, J. et al. Efficient sparse convolution on gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2019).
- Wu, Z. et al. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
- Wang, X., Yu, F., Dou, Z.-Y., Darrell, T. & Gonzalez, J. E. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018).
- Li, F., Li, G., He, X. & Cheng, J. Dynamic dual gating neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5330–5339 (2021).
- Teerapittayanon, S., McDanel, B. & Kung, H.-T. Distributed deep neural networks over the cloud, the edge and end devices. In *Proceedings of the 37th International Conference on Distributed Computing Systems (ICDCS)* (2017).
- Laskaridis, S., Venieris, S. I., Almeida, M., Leontiadis, I. & Lane, N. D. Spinn: Synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom)* (2020).
- Kang, Y. et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2017).
- Kim, Y., Kim, J., Chae, D., Kim, D. & Kim, J. μ layer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization. In *Proceedings of the Fourteenth EuroSys Conference*, vol. 2019, 1–15 (2019).
- Oh, C., So, J., Kim, S. & Yi, Y. Exploiting activation sparsity for fast CNN inference on mobile gpus. *ACM Trans. Embedded Comput. Syst. (TECS)* **20**, 1–25 (2021).
- Niu, W. et al. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 907–922 (2020).
- Li, D., Wang, X. & Kong, D. Deeprebirth: Accelerating deep neural network execution on mobile devices. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2018).
- Oh, C., Park, G., Kim, S., Kim, D. & Yi, Y. Towards real-time CNN inference from a video stream on a mobile GPU (WIP paper). In *The 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, 136–140 (2020).
- Satopaa, V., Albrecht, J., Irwin, D. & Raghavan, B. Finding a “kneedle” in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, 166–171 (IEEE, 2011).

32. Hong, H. & Oh, C. Image rearrangement for fast on-drone tiny object detection. In *2025 International Conference on Mobile, Military, Maritime IT Convergence (ICMIC)*, 294–297. <https://doi.org/10.1109/ICMIC66299.2025.11257814> (2025).
33. Howard, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
34. Uijlings, J. R., Van De Sande, K. E., Gevers, T. & Smeulders, A. W. Selective search for object recognition. *Int. J. Comput. Vision* **104**, 154–171 (2013).
35. Varga, L. A., Kiefer, B., Messmer, M. & Zell, A. Seadronesee: A maritime benchmark for detecting humans in open water. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2260–2270 (2022).

Author contributions

H.H. and C.O. wrote the main manuscript text. H.H. and N.L. conducted the experiments. K.J, G.K, Y.J. conducted conceptualization and data acquisition. All authors reviewed the manuscript.

Funding

This study was carried out with the support of R&D Program for Forest Science Technology (Project No. RS-2025-25441817) provided by Korea Forest Service (Korea Forestry Promotion Institute). This work was also supported by the research grant of Kongju National University in 2024.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to C.O.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2026