Low-Bandwidth Remote Sensing of Rare Events

Shilpa Anna George

CMU-CS-23-104

March 2023

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mahadev Satyanarayanan, Chair
Deva Ramanan
Ameet Talwalkar
Padmanabhan Pillai (Intel Labs)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under award numbers 1518865 and 2106862, the Defense Advanced Research Projects Agency under award number HR001117C0051, Facebook under award number RPS15PO70000055890, Lockheed Martin Corp. under award number MRA19001RPS006, and the Naval Surface Warfare Center under award number N001742310001. This work was done in the CMU Living Edge Lab, which is supported by Intel, ARM, Vodafone, Deutsche Telekom, CableLabs, Crown Castle, InterDigital, Seagate, Microsoft, the VMware University Research Fund, and the Conklin Kistler family fund. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Edge Computing, Remote Sensing, Active Learning, Live Learning, Video Analytics, Training Data Creation

To my family.

Abstract

Remote Sensing enables knowledge discovery from live data collected by unmanned probes. Planetary exploration, drone surveillance, and underwater sensing are three examples of domains in which remote sensing plays a central role. Near real-time knowledge acquisition of a rare target during such missions is challenging due to three extremes: *low bandwidth, novelty of target, and class imbalance*. We call the learning that happens in these extreme conditions as *Live Learning*. This is a new capability at the intersection of edge computing and machine learning. It aims to learn a model for a rare target from unlabeled data captured on distributed probes that are only reachable over a low-bandwidth network.

The main contribution of this thesis is the design, implementation, and evaluation of *Hawk*, an interactive model-agnostic live learning system that enables the discovery of rare novel phenomena from a stream of extremely skewed unlabeled visual data capture on weakly-connected remote sensing probes. Hawk is designed to optimize the use of two critical resources: (a) the network bandwidth from the remote source to the human expert, and (b) the expert's labeling bandwidth. Live Learning embodies a new semi-supervised learning algorithm to train models on-the-fly to discover instances of a target from very few initial labeled data. We show the effectiveness of Hawk by performing extensive validation on three very demanding publicly-available datasets from the domains mentioned above. Each of these datasets was released within the past few years, and has been used in recent ML research publications in its domain.

Our experiments show that even at bandwidths as low as 12 kbps and a base rate of 0.1%, a team of 7 probes is able to use Hawk to discover up to 87% of the event instances that could have been discovered using a brute-force model. Such a model is created from advance knowledge, transmission and labeling of all mission data. Our results show 1.5X–2X improvement in recall when Live Learning in Hawk is combined with recent Few Shot Learning algorithms such as SnaTCHer. Our results also show how the use of Diversity Sampling can further improve recall in Hawk.

Acknowledgments

I am deeply indebted to everyone who supported and encouraged me throughout my Ph.D. studies at Carnegie Mellon. Foremost, I would like to express my deepest gratitude to my mentor and my advisor, Prof. Mahadev Satyanarayanan (Satya), for his support and patience in guiding me through this journey. It has been a great honor to work with him and learn from his wealth of knowledge in designing and building good mobile and edge systems. His insights, knowledge, and expertise in the field have guided me in every step of my research journey. He has been a role model and a source of wisdom and support for me during many challenging times during my studies. Additionally, I would like to extend my sincere thanks to Padmanabhan (Babu) Pillai, whose vast knowledge in computer systems and machine learning has been invaluable to me. I have learned a lot from my discussions with Babu and I truly appreciate his patience and kindness in helping me during paper submissions. In addition, I would like to thank my thesis committee members, Deva Ramanan, and Ameet Talwalkar. Their insightful feedback and guidance on machine learning and computer vision has helped me refine my work.

I have been fortunate to work with talented and diligent people in Satya's research group. This endeavor would not have been possible without the help, support, and, close collaborative work with them. I look back very fondly on the memories I had with them both as a researcher and an individual.I would like to especially thank Deborah Kelly, Mihir Bala, Jim Blakley, Zhuo Chen, Jason Choi, Qifei Dong, Tom Eiszler, Ziqiang Feng, Jan Harkes, Roger Iyengar, Chanh Nguyen, Eric Sturzinger, Haithem Turki, and Junjue Wang. I also want to thank Sara Golembiewski for her prompt assistance for group administrative matters.

My PhD research has also benefited significantly from the larger research community. I was very fortunate to have collaborated with many extraordinary academic researchers and industry professionals. In particular, thank you, Kevin Christensen, Anupam Das, Greg Ganger, Wei Gao, Pulkit Grover, Roberta Klatzky, Tan Li, Adi Masputra, Rolf Schuster, Dan Siewiorek, Nihar B. Shah, Asim Smailagic, Kaiwen Wang and Canbo (Albert) Ye. During my time in Pittsburgh, I had the great fortune to make a wonderful group of friends both within and outside of CMU. Their faith in my abilities even when I did not, has been a source of strength and comfort for me. I am forever thankful for, Shamin Ajay, Nirav Atre, Toni Berning, Rachel Friend, Alex Hujdus, Prachi Gokhale, Ankush Jain, Pallavi Koppol, Gracious Shavers, Rui Sun, Adithya Pratapa, Sai Sandeep, Jonathan, and Anika Cordle.

Finally, I would like to thank my family. Their love, encouragement, and understanding have been instrumental in helping me navigate the ups and downs of this challenging journey. During my PhD, I had the blessing of welcoming two little boys into my life, Joshua and Bentley, they have brought immeasurable joy in my life. I dedicate this work to my family and my Lord Almighty, without Whom I could not have completed this degree, and I am eternally grateful for everything they have done in my life.

Contents

List of Figures

List of Tables

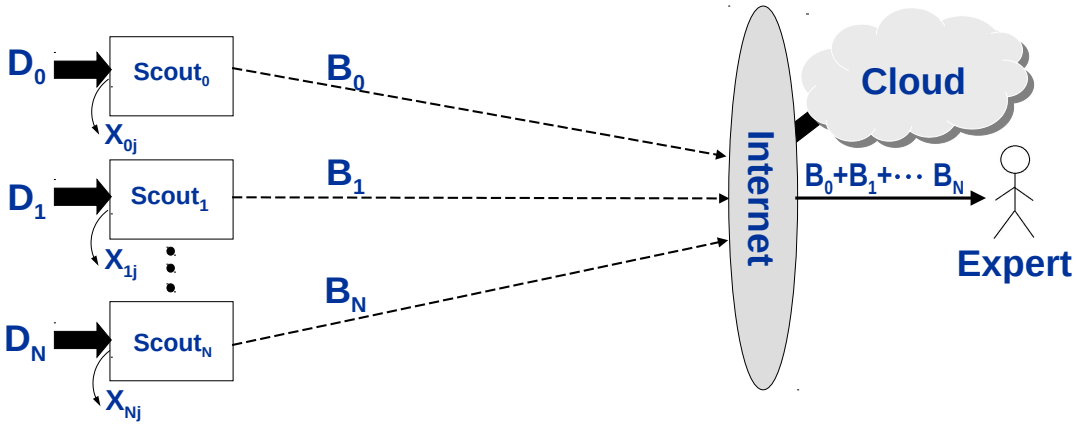
Chapter 1

Introduction

Remote sensing refers to the process of acquiring information about a phenomenon or object from a distance. It involves detecting and monitoring the physical characteristics of a remote object using emitted and reflected radiation. Cameras mounted on unmanned autonomous probes such as aerial drones (UAVs), satellites, and underwater vehicles, collect data to be analyzed by the mission personnel or researchers on the ground. Remote sensing has a wide range of applications in many domains; some examples include, use of UAVs in wildfire surveillance, satellite imagery for weather forecast, and more.

Historically, aerial photography was the primary form of remote sensing and has since been in use for more than 160 years [5]. Aerial photography is a faster and cheaper way of producing maps as compared to ground surveys. It was widely used in World War I as a reconnaissance tool, cameras mounted on aircraft were used to record enemy positions and movements. By World War II there was tremendous growth in the field of remote sensing, with the development of radar, sonar, and infrared detection systems. The "space race" in the 1950s paved the way for satellite-based remote sensing. Since then, the development of new acquisition platforms and the availability of cheaper and smaller sensors have brought significant advances in the field of remote sensing.

Remotely sensed data, such as satellite and aerial imagery, is beneficial for a number of use-cases, such as urban planning, natural resource management, and military operations. The remote sensing platforms are continuously generating a huge amount of data that needs to be analyzed to extract meaningful information for the targeted application. The generated data is transmitted to the ground receiving station either during the mission or post-mission. There are mainly two ways of receiving the data collected by the remote platforms. First, the generated data is stored in non-volatile memory and then off-board processing is performed when the remote probe lands after the mission. This acquisition method though simple, is limited in its application because it does not allow visualization or near real-time response during the mission. The second approach is to transmit compressed data back to mission personnel via a wireless network. The near real-time transmission of remote sensed data helps mission personnel to visualize and make time-critical decisions according to it. This is especially useful in applications such as surveillance, and search and rescue (SAR) missions. However, the network bandwidth sets a limit on the resolution and the transmission rate of the captured data. *How can we overcome this difficulty of transmitting useful data back to mission personnel during a time-critical mission in a network*



Labeling rate $B_i << D_i$

D_i = data rate into scout$_i$

X_{ij} = labeled data sharing rate between scouts i and j

Figure 1.1: Bandwidth-challenged Remote Sensing

bandwidth-challenged environment? In this dissertation we explore how to perform on-board processing to intelligently select relevant data for transmission.

Figure 1.1, illustrates the problem setting considered in this dissertation. A swarm of robotic probes called "scouts" are deployed to collect data from a remote area. A scout is an unmanned autonomous system equipped with ample sensing, compute, memory and storage capability. The team of scouts works collaboratively to discover and transmit instances of rare novel phenomena. These scouts have weak network connectivity to the Internet. They are equipped with rich image sensors that continuously capture unlabeled visual data. One such time-critical collaborative usecase is the use of UAVs during SAR missions, the authors in [10] used machine learning (ML) to locate victims from images captured from UAVs of avalanche debris. Another remote sensing example is the use of multiple drones by a team led by Reckling to locate and monitor the rare plant species, *Geum radiatum*, that is known to occur only on hard-to-access, high-elevation bluffs in the Blue Ridge Mountains in North Carolina [55]. For both these usecases, each scout (Scout$_i$) captures high-resolution data at a very high acquisition data rate given by D_i. However, the available communication bandwidth (B_i) is many orders less than the input data rate, i.e, $B_i << D_i$. Thus, only a tiny fraction of the captured data is sent to the mission personnel for inspection over the network. Can a computer system help in discovering the few positive instances containing the target from the huge volume of captured data?

The novelty of the target class precludes the existence of an off-the-shelf ML model. If an accurate ML model for the target class exists, then only data with instances of the target class is transmitted, ensuring that precious bandwidth is not wasted. The rarity of the target class also violates common machine learning (ML) assumptions about balanced classes. In this scenario, learning is difficult mainly because of three extremes: low bandwidth, novelty of target, and class imbalance. We refer to semi-supervised learning that takes place under these conditions as *live learning*. We go over these challenges in more detail in Sections 1.1 to 1.3.

1.1 Low network bandwidth

Often, remote sensing missions are conducted in regions that are inaccessible and may have poor backhaul bandwidth to the Internet. Scouts are equipped with rich sensors such as cameras or sonars to collect samples from the region. Over the years, technical advancements have been made in developing miniature camera sensors that capture videos and images at 4K and 8K resolution. While we target visual data in our work, its principles are applicable to any kind of data. Visual data is especially challenging because it involves very high data rates.

The camera sensors on the scout capture raw data at a very high data rate. Even with efficient encoding, continuous transmission of data from a single 4K camera at 30FPS would demand over 30 megabits per second (Mbps) of bandwidth. On the other hand, the scouts have poor network bandwidth to the mission personnel. For example, the backhaul bandwidth from an interplanetary space scout is only a few tens of kbps [46]. As another example, a military drone may be bandwidth-challenged because of the need for stealth and jam resistance in communication. This forces the use of low-signature wide-band spread-spectrum frequency-hopping techniques that constrain bandwidth to a few tens of kbps [4]. As a third example, the physics of acoustic communication in underwater sensing limits bandwidth to a few tens of kbps [22].

The quality of sensors on the scouts are likely to improve in the future. As scouts are equipped with multiple video cameras, multi-spectral sensing, and other adjuncts; input data rates higher than what is mentioned are likely in the future. In contrast, worst-case backhaul bandwidth is unlikely to improve dramatically. Hence, there exists a severe mismatch between input and output data rates on a scout. Today, this mismatch ratio can easily reach $10^3 - 10^4$ or more. Thus, both the absolute value of this bandwidth, and its extreme mismatch relative to the incoming data rate on a scout severely limit the amount of data that can be transmitted back to the mission personnel.

1.2 Target Novelty

The earliest works in deep learning and its training algorithm, backpropagation, can be traced back to the 1960s[30]. The remarkable progress in deep learning can be attributed to the introduction of graphics processing units (GPUs) and the availability of big labeled datasets such as 1M labeled images from ImageNet[59]. Today, Deep Neural Networks (DNN) are widely used in solving many important visual applications across multiple domains such as retail, medical imaging, autonomous military systems, and so on. The use of DNN technique ameliorates the burden of needing expertise in the area for hand-crafting features.

The DNN models are trained in a fully supervised manner using a large amount of labeled training data. For example, ImageNet has around a million labeled images and each class has about 700 to 1000 labeled examples. In many real-world deployments, DNN models pre-trained on a large collection of labeled data having pre-determined classes are used to infer and classify incoming data.

As discussed earlier, remote sensing helps in the acquisition of new knowledge, which are not known a priori before the start of the mission. For example, sightings of a new enemy combat vehicle initiates the need to develop a DNN model capable of detecting the new threat. Thus,

there is a need to customize the model on the newly available data. To improve model accuracy for new targets from different viewpoints, we use a learning technique called transfer learning. Transfer learning modifies the parameters of an existing DNN model to train a new task on a smaller dataset. Transfer learning significantly improves model accuracy without incurring the cost of training on a large training set.

To train an accurate model we need 100 to 1000s of labeled examples. However, for a novel target class there might only be a handful of labeled examples available to the mission personnel at the start of the mission. During the mission, there may be more positive instances of the target class hidden among the unlabeled raw images captured by the sensors on the scout. The poor backhaul bandwidth to the Internet inhibits the transmission of all the data to the mission personnel for examination and labeling. A small fraction of data needs to be intelligently selected for transmission using the knowledge acquired during the mission. Thus, there is a need to create a system that integrates learning and labeling and can perform live learning by creating self-improving models for a novel target from the data that is captured on the scouts.

1.3 Low Base Rate

A major challenge in creating models for a novel target is the low rate of occurrence of these interesting events from a large collection of unlabeled data. Interesting events by definition are rare. The rarity of an event is expressed by its *base rate*[41], which is defined as the probability of encountering a target in the acquired data. The task of finding positive instances of a target from the voluminous amount of data captured by the scouts is very demanding.

In machine learning (ML), a low base rate maps to extreme class imbalance. A classification dataset having skewed data distribution across classes is called imbalanced. The degree of imbalance is considered to be extreme, when the proportion of the minority class which is equivalent to the base rate of the target class is less than 1% of the entire data collection [29].

As mentioned earlier, the poor communication bandwidth precludes the system from transmitting all the data to the mission personnel for examination. The precious bandwidth can be saved by transmitting only positive instances to the mission personnel. If an ideal model was present in the scout, this would be possible. However, as mentioned earlier, training an accurate model using transfer learning requires hundreds of labeled examples of the target class. For a novel target, there may not be that many examples available. If, however, the whole point of the mission is to collect more positives needed to train an accurate model, we have a chicken-or-egg problem. Thus, there is a need for a selective transmission mechanism on the scouts that is capable of intelligently choosing images for transmission. The transmitted images are labeled by the personnel on the ground and then added to the labeled set to train models capable of recognizing instances of the target class from the captured images.

1.4 Object Recognition with Deep Learning

Computer vision is a field of artificial intelligence that enables computers to interpret and understand visual information from images, videos and other inputs. The remarkable success in the

field of computer vision over the past decade can be attributed to the application of deep learning models to machine perception tasks.

Classic computer vision models used hand-crafted feature descriptors from pixel data, such as color histograms, textures, and shapes. Engineering these features needed expertise in the field to precisely tunes these features and creating robust accurate models was very challenging.

In 2012, a large deep neural network called AlexNet showed excellent performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[33]. This marked the beginning of the wide adoption and development of deep neural network (DNN) models. Unlike previous algorithms, these deep models avoid the need to manually create features. Instead, these DNN models "learn" how to extract features from an image and infer its contents. Deep learning brought great successes in the field of image recognition, and face recognition algorithms achieving above human-level performance.

Image recognition is an application of computer vision that seeks to recognize objects in an image. It includes a set of sub-tasks such as image classification and object detection. In image classification, the DNN model processes an image and outputs the classification label of that image. Image classification applications are used in many areas, such as medical imaging, object identification in satellite images, traffic control systems, and more. Object detection, combines image classification with object localization to answer the question, "What objects are where?"

With the advent of deep learning, there has been rapid progress in creating deeper and faster DNN models. Accuracy and inference speed are two competing criteria in DNN model development. In general, to get higher accuracy, we need to use a "deeper" model. But a deeper model has many more compute parameters that make it slower to train and infer.

DNN models are typically trained in a supervised manner, where the model learns the parameter values from the labeled data provided. The process of creating such labeled data to train DNN models needs tedious human work. Today, companies recruit workers to accelerate this process using crowd-sourcing tools such as Amazon Mechanical Turk, Appen, and Upwork. This essentially leverages human-level parallelism offered by many crowd workers. Some of the popular datasets that have millions of labeled data that are used for model training are, ImageNet, Google Open Images, and MS COCO.

The visual analysis of remote sensing imagery require expertise in the field. For example, to derive meaningful climate-related insights from satellite imagery the labeler needs knowledge about meteorology. There may also be data access restrictions that limit the use of crowd workers for labeling. Thus, it is valuable to minimize labeling effort and enable a small number of domain-experts to build accurate ML models from very few labeled examples.

1.5 Thesis Statement

In this dissertation, I address the problem of discovering data of rare novel phenomena under conditions of poor network connectivity. We claim that:

It is feasible and effective to create a distributed system that integrates selective transmission, human labeling, and model training to perform low base rate active learning in an edge computing setting. Such a system can be valuable in collecting training data of a novel phenomenon in extreme bandwidth-challenged environments. Such a system can

transparently perform machine learning in the background and automatically select tactical parameters to match the network bandwidths, data distribution, and computational loads on robotic probes.

1.5.1 Why is this thesis statement important?

In remote sensing, the constraints of low network bandwidth, extreme class imbalance, and novelty, hampers the creation of deep learning models for the discovery of a rare novel phenomenon. In many learning scenarios, only a handful of labeled examples of a target class may be available for training an ML model. New positive instances need to be discovered from millions of unlabeled images captured by the sensors on remote sensing platforms. Only a system that integrates labeling, machine learning, and bandwidth-aware adaptation can solve these challenges. The thesis, if validated, will address the challenges mentioned in Sections 1.1 to 1.3 and enable the creation of model-agnostic bandwidth-adaptive learning system for the discovery of rare new phenomena on unmanned missions.

1.5.2 Why does it not follow trivially from what is already known?

This challenge is unaddressed by existing deep learning algorithms. A full description is provided in Chapter 2. In brief, the closest work that address the challenges mentioned is the decentralized approach of Federated learning. However, federated learning, assumes that data is already pre-labeled. Since scouts are unmanned, obtaining labels is only possible by transmitting data to a distant human. Two most relevant works, Snorkel[53] and Eureka[20], have demonstrated methods of reducing labeling effort to create training sets, but they expect some programming or machine learning knowledge from its users. Snorkel requires users to provide weak-supervision in the form of labeling functions, and it relies on the meta-data associated with each item. Though Eureka does not necessitate the presence of meta-data, it expects users to explicitly create and supply new classifiers. Other works in this space include no-code or low-code AI tools released by companies such as Google and Microsoft that leverage automated machine learning (AutoML) to enable users to train and deploy DNN models. These platforms, however, require users to provide a significant amount of labeled data to train highly accurate classifiers. Given these challenges, it would be valuable to have a system that automates the formulation and execution of machine learning in the background and let the domain expert focus solely on labeling.

1.6 Thesis Validation

We validate this thesis in this dissertation by building Hawk, an interactive model agnostic live learning system that enables discovery of rare novel phenomena from a stream of unlabeled visual data captured on weakly-connected remote sensing platforms. The key performance indicator (KPI) is the number of positive instances discovered during the mission.

The main contributions of the dissertation are as follows:

1. We introduce the concept of *live learning*, a new capability at the intersection of edge computing and machine learning. No prior system performs live learning. We recognize that in the presence of extreme imbalance and low bandwidth, even a weak model learned on-the-fly is helpful in enriching the live sensor stream by pruning away irrelevant data.
2. We present the system Hawk which operates in a bandwidth-constrained environment, maximizing the use of two most critical resources: (a) the network bandwidth from the remote source to the human expert, and (b) the expert's labeling bandwidth.
3. We propose a semi-supervised learning algorithm to rapidly train models on-the-fly to discover positives instances of a target from very few labeled data.
4. We present extensive experimental results on multiple contemporary datasets to validate the design and implementation of Hawk.

1.7 Thesis Overview

The rest of this dissertation is organized as follows:

- Chapter 2 introduces recent work in machine learning that are relevant to live learning. We also show how the unique attributes of live learning differs from these works.
- Chapter 4 describes the system architecture and provides quantitative and qualitative analysis of design choices in Hawk.
- Chapter 3 provides detail about the learning algorithm in Hawk. We also formalize the semi-supervised learning used in Hawk, which selects a random subset of data to be pseudo-labeled and used for model training. We also provide descriptions of the parameters used for Hawk learning.
- Chapter 5 describes the system implementation and the application interfaces of Hawk.
- Chapter 6 evaluates the effectiveness of Hawk on three challenging datasets. We examine the effect on Hawk performance based on the site of model learning. We also compare the performance of Hawk learning to an ideal system and inspect the headroom for improvement.
- Chapter 7, presents two techniques to improve the recall of Hawk. The two techniques explored are, (a) diversity sampling, and (b) few-shot learning. We examine and evaluate the improvement in recall when integrating these algorithms to Hawk.

Chapter 2

Background and Related Work

Hawk is a live-learning system that trains ML models on-the-fly to enrich the result streams from remote platforms by pruning away irrelevant unlabeled data. Live learning sits at the intersection of edge computing (Section 2.1) and machine learning. Table 2.1 illustrates the unique attributes of live learning. As seen from the table, though live learning shares some attributes, it is fundamentally different from works in active learning, distributed learning, and federated learning as discussed in Sections 2.2 to 2.4.

2.1 Edge Computing

Mobile devices are everywhere, rapidly growing in number and applications. According to a report from CISCO[16], the number of mobile devices will grow from 8.8 billion in 2018 to 13.1 billion by 2023. The prevalence of mobile devices such as smartphones has been instrumental in fulfilling the vision of "information at my fingertips at any time and place", which was only a dream in the mid 1990s[61]. The mobility of these devices comes at the cost of limited compute capability. Applications such as scene understanding and video processing have high computational demands which cannot be met by the limited resources available on these mobile devices alone. One solution is to augment these mobile devices' capabilities via offloading computation to a more resource-rich computing infrastructure.

This was first demonstrated by Brian Noble et al.[47], in 1997, by implementing speech recognition on a resource-limited mobile device by offloading computation to a nearby server. The term *cyber foraging* was coined by Satyanarayanan [62], to signify the augmentation of computational and storage capabilities of resource-limited mobile devices by leveraging one or more resourceful servers. Today, cloud computing and edge computing serve as the optimal sites for compute offload.

The modern edge computing landscape can be represented as a tiered model as described by Satya et al[63]. Figure 2.1, illustrates the three tiers in edge computing, where each tier are separated by distinct design constraints. Tier-1 represents "the cloud", where elasticity and permanence are the dominant features. With the emergence of Cloud computing in the mid-2000s, the cloud has become the infrastructure to offload computation from a Tier-3 mobile device. The main feature of Tier-3 devices, such as smartphones, VR/AR headsets, and drones,

	Live	Distributed	Federated	Active
Labeling is within scope of problem	✓			✓
Data is live (streaming)	✓			
Low bandwidth connectivity (Kbps)	✓		✓	
Low base rate (rare target)	✓			
Privacy is a major consideration			✓	
Training occurs outside cloud	✓	✓	✓	

Table 2.1: Unique Attributes of Live Learning

is its mobility and sensing capabilities. These devices are continuously generating data that require to be processed to extract important information from them. Because of its mobility Tier-3 devices are always more resource-constrained than Tier-2 and Tier-1 devices. For example, many services, such as language translation, and video processing, require more compute power than what is available on Tier-3 devices. Offloading to Tier-1 is not always ideal. In 2010, Li et al. [37] report that the average round trip time (RTT) from 260 global vantage points to their optimal Amazon EC2 instances is 74 ms. For applications that need crisp-response, where the end-to-end delay needs to be less than a few tens of milliseconds, the cloud as an offloading site is not adequate.

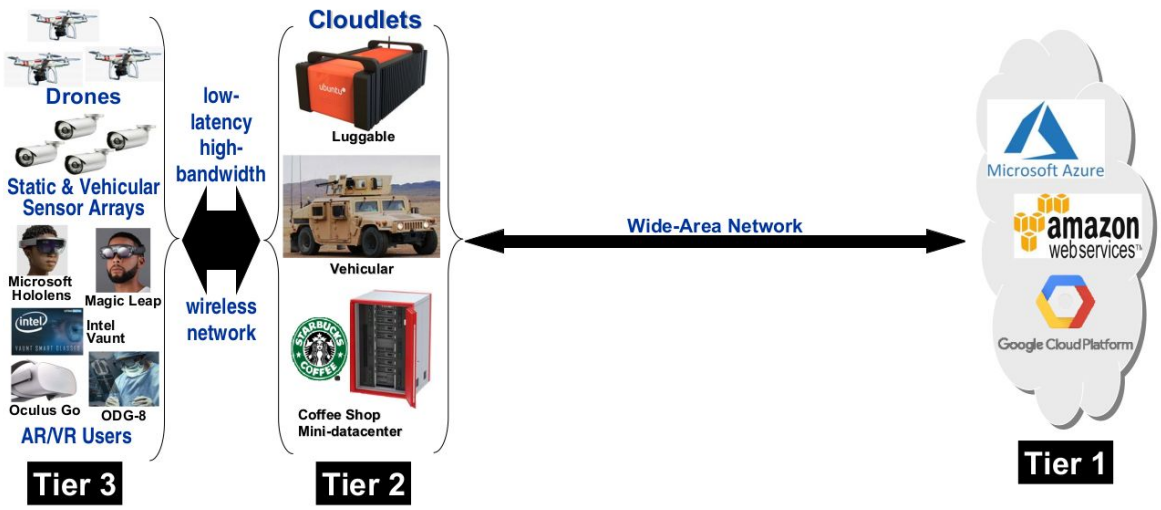
To overcome these challenges, edge computing (Tier-2) is proposed which creates the illusion of bringing Tier-1 "closer". The network proximity to Tier-3 is the main purpose of Tier-2. Servers in Tier-2 are organized into small, dispersed data centers called cloudlets, which enable Tier-3 devices to offload compute-intensive operations at very low latency. Edge computing enables latency-critical and bandwidth-hungry applications, such as, autonomous driving, by enabling offloading of compute to servers in proximity to Tier-3.

In Hawk, a scout is equipped with sensors for data collection (Tier-3) and it has compute and storage comparable to a high-end desktop. ML training and inference are possible on such a platform (Tier-2). Thus, a scout can be viewed as a platform where Tier-3 is colocated with Tier-2.

2.2 Active Learning

In machine learning, the optimal selection of data for labeling is known as active learning [64]. Active learning algorithms aim at selecting the most useful samples from the unlabeled dataset and transmit to the human annotator for labeling. The goal is to reduce the number of items labeled while maintaining performance.

The approaches in active learning can be divided mainly into "Stream-based" and "Pool-based" selective sampling. In stream-based active learning[45, 67], the algorithm makes an independent judgment whether a sample needs to be queried for its labels. The algorithm never revisits a previously discarded sample and the accuracy is usually worse than pool-based tech-



Source Satya et al.[63]

Figure 2.1: Three-Tiered Model of Modern Computing

niques.

In pool-based sampling [43, 81], the algorithm examines the entire dataset before selecting the samples to query. The time taken for this preview is ignored, even if it takes many hours. Only data presented for labeling incurs a cost; as long as no data is presented to the user for labeling, examining data is viewed as zero-cost. This, however, is not true in the case of a near real-time low-bandwidth system like Hawk; the preview is most definitely not "free."

Active learning algorithms have historically been explored in a cloud setting, where access to data is cheap and easy. In contrast, data is dispersed and connectivity is poor in the setting we consider. When time is of the essence, as in the case of live learning, it is counter-productive to wait a long time just to obtain an optimal labeling sequence. The entire decision process has to be more online in nature.

Classic wisdom in active learning suggests that it is best to sample data in a way that provides maximal information gain. One widely-used sampling criteria is *maximum-entropy*, where the learner selects data examples about which it is most confused [64]. Interestingly, recent work has suggested that such methods may not be optimal in the deep learning setting, where the exposure of difficult examples early in the learning process may confuse non-convex learners [11]. Also, the literature targets data whose distribution across class and non-class is balanced, whereas a low base rate implies a highly skewed distribution. Thus, in this work we develop an active learning technique to work in an imbalanced data setting by asymmetric handling of negatives and positives. The selection mechanism is tunable to match the low-bandwidth setting.

2.3 Distributed Learning

The increasing demand for learning from large amounts of data has given rise to new challenges regarding efficiency and scalability of learning algorithms with respect to computational and memory resources. A possible solution to this large-scale learning problem is distributing the learning across multiple worker nodes as a way of scaling out to improve performance. Thus, distributed machine learning refers to scaling out the training phase of machine learning by leveraging cross-machine parallelism and storage capacity.

As the recent survey by Verbraeken et al [77] states: "... the long runtime of training the models steers solution designers towards using distributed systems for an increase of parallelization and total amount of I/O bandwidth, as the training data required for sophisticated applications can easily be in the order of terabytes [13]. In other cases, a centralized solution is not even an option when data are inherently distributed or too big to store on single machines."

There are two main types of distributed machine learning: data-parallel and model-parallel distributed learning. [52]. In Data-parallelism, the training data is partitioned to the worker nodes in the computer cluster. The same model is replicated across the worker nodes, and each worker applies the learning algorithm to its own subset of the data. The worker nodes communicate to synchronize the model parameters or gradients, to ensure that a consistent model is trained. In model-parallelism, the model is split into different parts that can run parallelly in different nodes, and the same copies of the data sets are available on the nodes. The final model trained is the aggregate of all the model parts on the worker nodes.

Conventional distributed algorithms are designed for controlled environments, such as data centers, where the training data is distributed equally among machines or worker nodes, and high-throughput networks are available. The training nodes may be located within a single data center, or across multiple well-connected data centers. In contrast to live learning, where labeling is integral to the problem specification, distributed learning assumes that data has already been labeled. Exactly how that happens is outside the scope of the problem. Distributed learning is typically done on clusters of computers in data centers, where network connectivity is 10 Gbps or higher. This mindset is well captured by Langer's 2019 work [34], that states "... we focus our attention on analyzing how existing distributed deep learning systems perform in typical network environments with limited communication bandwidth, such as Gigabit Ethernet." That Gigabit Ethernet is viewed as "limited communication bandwidth" says it all!

2.4 Federated Learning

Federated Learning is a machine learning technique that trains a centralized model while training data remains distributed over multiple decentralized edge devices or servers. Privacy and compliance with data movement restrictions (such as HIPAA in the US [18] and GDPR in the EU [74]) are the two key motivations for federated learning. Li et al [38] describe federated learning as "... training statistical models over remote devices or siloed data centers, such as mobile phones or hospitals, while keeping data localized." In contrast, privacy and regulatory concerns in data movement are not significant considerations in live learning.

Labeling is outside the scope of federated learning. Any necessary labels are assumed to

have already been acquired prior to the start of federated learning. Exactly how that happens is unspecified. In contrast, labeling is an integral part of the live learning problem specification. Similar to distributed learning, works in federated learning also assume the data is distributed among devices in a balanced fashion.

An area of overlap between live learning and federated learning are the bandwidths involved. Training on data across smartphones with 4G or 3G connectivity is an often-cited use case of federated learning. The lower end of this range (3G) is typically a few hundred Kbps to a few Mbps. In contrast, live learning is applicable even to much lower bandwidths (down to 12 Kbps in the experiments reported here).

2.5 Prior Work in Training Set Creation By Experts

A training set is a set of examples each annotated with ground truth labels pertaining to a specific task. Training data is a key ingredient in deep learning. The quantity, quality, and diversity of a training set impact the accuracy of trained models. Many publicly available datasets are created by crowd-sourcing the task of image labeling. However, due to privacy concerns and the lack of expertise of specialized domains the task of labeling is the burden of domain experts. Previous works, Eureka and Snorkel aimed at reducing the labeling efforts of domain-experts in collecting training data.

Snorkel [53] expected weak-supervision in the labeling function and can be used only if there is textual meta-data associated with the items. To quote the paper: "Snorkel enables users to generate large volumes of training data by writing labeling functions, which are simple functions that express heuristics and other weak supervision strategies." This approach requires domain experts to write code, raising the barrier to access of the system.

Eureka [20] is a distributed system whose goal is to enable domain-experts to collect training data for a rare phenomenon from visual data. Eureka supports DNN-based feature extraction, and SVM-based filters. Eureka does not integrate training and learning. It requires the user to explicitly create new classifiers and trigger the next iteration of labeling. Eureka uses thresholds as a selective mechanism to transmit items for labeling. This needs knowledge of the machine learning algorithm to ensure the number of transmitted items do not overwhelm the network bandwidth.

Chapter 3

Learning in Hawk

In this chapter, we elaborate on the learning in Hawk. Hawk uses *Active* and *Semi-Supervised* learning to intelligently select items for labeling and training new improved models. The learning in Hawk can be divided into three parts, (a) labeling, (b) model training, and (c) model inferencing. We go over each of them in more detail in the following sections.

This chapter is organized as follows. First, in Section 3.0, we provide a brief overview about the working of Hawk. In Section 3.1, we go over the two types of labeling in Hawk, namely (a) human-provided labels and (b) system obtained pseudo-labels; and how they jointly help improve the quality of learning. Next, in Section 3.3, we describe the model training process and present the various hyperparameters used during training in Section 3.3. Then, we describe the different phases of a Hawk model in Section 3.4. In Section 3.5, we provide details on how a new learning algorithm can be added to Hawk. Finally, in Section 3.6, we briefly introduce a web-based toolkit for generating a Hawk mission configuration.

3.0 Overview

Hawk enables the discovery of novel target instances from a stream of unlabeled data captured by a team of scouts. There is an extreme imbalance in the data distribution across classes, in this work we consider scenarios where the base rate of the target class is 0.1% or less. The scouts have very poor backhaul bandwidth to the Internet typically tens of kbps. Unbeknownst to the expert, model learning happens in the background. Hawk intelligently chooses items to transmit to the expert for labeling; the aggregated labels are then used to intermittently train and deploy improved models during the mission. The labels used for training may be human-labeled data or data pseudo-labeled by the system (Section 3.2).

Hawk performs rapid creation of models for rare new targets from data captured at scouts. Hawk infers the incoming data using the trained model and selects a small fraction of images to query the human-expert for labels. The labels from the expert are shared among the scouts. The accumulated human-labeled data and some pseudo-labeled data are used in training new models. A new model trained with additional labeled training data has a better accuracy compared to a previous model. An improved model helps discard easy negatives and discover more positives from the incoming data stream. Thus, recursively replacing the model in deployment with an

improved model helps improve the quality of results transmitted to the expert. This enables efficient utilization of the backhaul bandwidth by transmitting better results to the expert.

As mentioned in Section 4.4, Hawk may train models on the cloud or on the scouts. The site of training determines the image resolution of the results transmitted to the expert. Full-resolution tiles are required for model training. Thus, when the training happens on the scouts, we can optimize downlink bandwidth usage by transmitting thumbnails of size 256x256 pixels to the expert. However, in case of training on cloud, full-resolution tiles are transmitted. The tile size depends on the choice of DNN model architecture, for example, we use tile size of 256x256 pixels for ResNet50 whereas we use much larger tile size of 600x600 pixels for YOLO.

Labels provided by the expert are transmitted back to the scout as they are generated. The uplink bandwidth demand for this is very small. No image data has to be transmitted; only numerical values indicating tile identity, labels and bounding boxes. The expert is expected to provide accurate labels for the transmitted results. In our work, we do not deal with bias in the labels or imperfect human labelers. This has been studied in detail in works such as [12] [66]. Their solution can be integrated in a future version of Hawk.

3.1 Labeling in Hawk

Supervised machine learning is an approach that uses labeled datasets to train algorithms to classify data or predict outcomes accurately. In supervised learning, the algorithm "learns" from the labeled training dataset by iteratively making predictions on the data and adjusting for the correct output. It is called supervised learning because the process of a model learning from a labeled training dataset is analogous to a teacher supervising the learning process. Supervised learning models tend to be more accurate than unsupervised learning models, and the majority of practical ML applications use supervised training. One main disadvantage of supervised learning is the upfront cost of obtaining labels for the data.

The raw incoming image streams in scouts do not have labels associated with them. The poor network connectivity to the Internet prohibits sending of all the data to the domain expert for labeling. However, the acquisition of unlabeled data is relatively cheap compared to the cost of labeling the data. In such scenarios, *semi-supervised learning*(SSL) which uses unlabeled data in conjunction with a small amount of labeled data, is used to improve the learning accuracy [80]. Semi-supervised learning lies in the intersection of unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data). We do SSL by using a modified form of pseudo-labeling [35] to obtain labels for the negatives that are available locally on each scout. Hawk uses a combination of human-labeled and pseudo-labeled data to train the models. We provide more detail in Section 3.2.

The poor backhaul bandwidth limits the number of items Hawk can query the expert for labels. Thus, Hawk uses active learning strategies to select optimal items and train the models during a mission. As mentioned in Section 2.2, active learning techniques select the most useful samples from the unlabeled data stream and transmit to the human annotator for labeling. The selective transmission strategy in Hawk selects a small subset of tiles to present to the domain expert for labeling. The expert is not expected to possess machine learning or programming knowledge. The expert examines the results transmitted from the scouts and provides accurate

labels using the Hawk UI (Section 5.3). In the following section, we describe the active learning and SSL in Hawk in more detail.

3.2 SSL and Active Learning in Hawk

Pseudo-labeling [35] is a popular semi-supervised learning technique, it assigns approximate labels to some of the unlabeled data based on the available labeled data. Here, we first train a model using the training data and then we use that model to generate pseudo labels for the unlabeled dataset. In classical pseudo-labeling, only unlabeled-samples with high confidence scores are added to the training labels (pseudo-labels); this reduces the density of data points at the decision boundary and can be viewed as a form of entropy minimization. Finally, both the original labels and pseudo-labels are combined to learn a new model. In Hawk, we use a modified form of pseudo-labeling to train a more robust model because of two main reasons; (a) poor accuracy of initial models, and (b) extreme-class imbalance of the unlabeled pool. The earlier models trained in Hawk are not well calibrated and may misclassify items with very high confidence scores. Because of the rarity of targets, any wrongly labeled positive can degrade the quality of learning and affect the model evolution in Hawk.

To address the challenges of low bandwidth, extreme class imbalance, and target novelty, Hawk makes three changes to classic pseudo-labeling. Asymmetric handling of negatives and positives is the essence of these changes:

- First, no data item is added to a training set as a positive unless confirmed by human labeling.
- Second, most data that is pseudo-labeled as negative is unlikely to be transmitted.
- Third, a fresh random subset of available pseudo-labeled negatives is used in the training set of each model.

The first rule is motivated by extreme class imbalance. When there are very few TPs, even a single misclassified positive can have devastating impact in training. It is therefore worth spending limited bandwidth to ensure that every single positive added to the training set is verified as a TP.

The second rule is motivated by low bandwidth, and the desire to use this scarce resource for high value. As mention in Section 4.5, relative to the current model, a data item may be a positive ($\mathcal{P}$), a hard negative ($\mathcal{HN}$) or an easy negative ($\mathcal{EN}$). Hawk prioritizes the transmission of $\mathcal{HN}$s over $\mathcal{EN}$s because their chances of misclassification are higher, and therefore obtaining a human label is more valuable. Hawk transmits $\mathcal{EN}$s only when bandwidth is under-utilized because of too few $\mathcal{P}$s and $\mathcal{HN}$s.

The third rule is motivated by the danger of confirmation bias in SSL. A TP that is mislabeled as an $\mathcal{EN}$ in an early training set can wreak havoc on the entire evolution of models. There is no way to totally avoid this danger — it is inherent to pseudo-labeling. However, the risk can be reduced by choosing a random selection of $\mathcal{EN}$s afresh for the training of each new model. Because of the very small number of TPs and the extreme class imbalance, the chances that a misclassified TP will be selected again from the huge volume of $\mathcal{EN}$s is low. A specific model may be affected, but it is unlikely that this will harm the evolution of models over a longer term. $\mathcal{HN}$s pose less of a danger because they are prioritized over $\mathcal{EN}$s in selective transmission for

human labeling, and are therefore likely to eventually receive a human label. The use of $\mathcal{ENs}$ in training serves two purposes. First, it reduces the tendency for model drift to occur in training. Second, it tunes the model better to the specific attributes of the incoming data.

Formally, let $x \in \mathbb{R}^d$ be an unlabeled data item having d dimensions and let $y \in \{0,1\}^C$ be the corresponding one-hot label vector for C classes, then we aim to learn a model, $M(.;\theta_G) : \mathbb{R}^d \rightarrow \{0,1\}^C$, where θ_G are the parameters of the model M of generation Gen-G and $M(x;\theta_G) = [p_c]_{c=0}^C$ are the softmax probabilities ($p_c \in [0,1]$) produced by the model.

The data items, D_G are the unlabeled data processed on the scouts after the deployment of model Gen-G, but before the training of a new generation $G+1$. The items in D_G, can either belong to the human-verified set D_{hG} or the unlabeled set D_{uG}. The number of items in the set D_{hG} (or D_uG) is given by N_{hG} (or N_{uG}). Then, we can represent D_{hG} and D_{uG} as follows:

$$D_{hG} = \{(x_i, y_i)\}_{i=0}^{N_{hG}}, \qquad D_{uG} = \{x_i\}_{i=0}^{N_{uG}},$$

where x_i is a data item and y_i is the corresponding one-hot label vector.

Suppose the ground-truth labels of unlabeled set D_{uG} were known, then we can denote the number of data items of class c as N_c, i.e., $\sum_{c=0}^{C} N_c = N_{uG}$. The degree of imbalance in the unlabeled dataset may be given as $\gamma = \frac{N_{uG}}{min\ N_c}$.

Algorithm 1 Pseudo-Labeling Negatives in Hawk

Input: Model M_G, Data stream D_G

1: **for** x_i `in` D_G **do**
2: $\quad [p_{ci}]_{c=0}^2 \leftarrow M(x_i)$
3: **end for**
4: $\{x_0, \ldots, x_{|D_G|}\} \leftarrow$ **Sort**$(p_{1i}), \forall i \in |D_G|$
5: $D_{u'} = \{x_0, \ldots, x_M\}$, where $M < \delta |D_G|$
▷ Set of low-scoring items
6: $D_{\tilde{u}} = \{x'_0, \ldots, x'_{N_0}\} \leftarrow$ **RandomSelection**$(D_{u'}, N_0)$
7: $\tilde{y}_{0i} = 1$ for i $\in D_{\tilde{u}}$
▷ Pseudo-labeling data as negatives

Under conditions of high imbalance ($\gamma \gg 1$), Kim et al [31] show that naively applying pseudo-labels can be detrimental to model training. In Hawk, as stated before, we consider an extremely imbalanced binary classification scenario, where $\gamma \geq 1000$ and C is 2. Respecting Kim et al's guidance, we only pseudo-label a random subset of low-scoring $\mathcal{ENs}$, as given in Algorithm 1. We first select a fraction δ of low-scoring unlabeled data items in D_{uG} (where the probability of x_i belonging to the target class (class 1) is small, i.e., $y_{1i} < 0.5$).

Adding all the pseudo-labeled negatives to the training set may negatively impact the quality of the trained model. Instead, we balance the data across classes in the training set as recommended by Li et al [39] and Mahajan et al [42]. We balance the number of items in positive and negative classes such that, $N_0 + N_{h0} = \beta N_{h1}$. The sampling ratio β is a parameter, from our experiments we set $\beta = 50$.

The pseudo-labeled negatives are then added to the training set along with human-labeled items. We train and optimize the model parameters using cross entropy loss. In the presence of

class imbalance, $\mathcal{EN}$s being more frequent in the training set can cause instability during training. This can be mitigated using a weighted variant of Cross Entropy loss such as [17, 40]. We use the class-balanced variant of Focal Loss [40] as follows:

$$L(p) = \begin{cases} -\frac{1}{\beta}(1-p)^2 \log(p) & \text{if } c = 1 \\ -(1-\frac{1}{\beta})p^2 \log(1-p) & \text{otherwise,} \end{cases}$$

where β is the sampling ratio and p is the probability estimate given by the model.

Hawk uses a TopK selective transmission protocol to implement active learning. Data items that are inferenced by the current model on a scout are added to a priority queue and sorted by their confidence score. The highest-scoring items are likely to be $\mathcal{P}$s. Below them are likely to be $\mathcal{HN}$s. Further below are the $\mathcal{EN}$s. The value of "K" in TopK is chosen to be bandwidth adaptive, so as to fill (but not overfill) the end-to-end pipeline from incoming data to a distant human labeler. When bandwidth is scarce, very few $\mathcal{HN}$s and $\mathcal{EN}$s are transmitted. When bandwidth is plentiful, there is less need to be frugal. At all bandwidths, TPs are precious and the protocol aims to find and confirm every one of them via human labeling. The learning in Hawk is biased to improving recall, the selective transmission policy aims to transmit most of the positives at the expense of wasting some bandwidth on FPs. In Section 4.7, we discussed the strategy of revisiting discards in the hope of finding misclassified positives.

3.3 Model Training in Hawk

Hawk recursively trains new improved models as the training set improves as more number of positives and negatives are labeled by the expert. Training of a new model is triggered in Hawk when the number of labeled data exceeds a certain threshold. To allow flexibility, Hawk supports declarative specification of the training strategy used on a mission. The experts can control the training process by tuning the hyperparameters. In Section 3.3.1, we describe some hyperparameters used during model training. Hawk expects no programming or machine learning skills of domain experts.

3.3.1 Hawk Hyperparameters

Hyperparameters are variables whose values control the training process and influence the quality of the trained model. These are configurable variables and the values are set before the start of the mission. We briefly describe some of these hyperparameters and how they influence training. In Hawk, we provide default values for these hyperparameters which are sufficient for satisfactory operation. The labeler may choose to tune these values before the start of the mission by modifying the configuration file. Table 3.1 gives a list of hyperparameters used in Hawk along with their default values. We explain these hyperparameters in the paragraphs below.

Learning rate

The learning rate is a hyperparameter that affects the rate at which the DNN model weights are updated with respect to the error estimate. If the value is too small, it results in a long

Parameter name	Description	Default Values
Learning rate (LR)	Rate of model learning	0.01
LR warm-up epochs	Initial number of warm-up epochs	10
Learning rate schedule	Pre-defined LR schedule	Cosine
Momentum	Percentage of gradient retained	0.9
Weight decay	L2 regularizer	1e-4
Training Epochs	Number of training epochs	10
Optimizer	Name of optimization algorithm	SGD
Label Smoothing	Degree of smoothing	0.1
EMA momentum	Update momentum	0.6
Batch-size	Number of images in a batch	64
Image-size	Resolution of image feed	256
Layers to unfreeze	Number of model layers to train	3
Model architecture	Name of DNN model architecture	resnet50

Table 3.1: Hyperparameters in Hawk

training process due to the small gradient descent steps. However, if the value is too large, the gradient descent could overshoot the minima and we may learn sub-optimal model weights. Thus, choosing a good value of learning rate is vital as it controls how quickly the DNN model can converge to local minima and yield good accuracy. In Hawk, as we are training from a pre-trained DNN model, we set the learning rate to a small value (0.01).

When we train the initial (Gen-0) model, we use the first few epochs to "warm-up" the learning rate. If we use a high learning rate at the start of training, the model could suffer from "early-overfitting" by learning highly differentiable features from the labeled bootstrapping data. To prevent this, we apply a linear warm-up of learning rate by start with a learning rate much smaller than the initial learning rate and then gradually increase our learning rate. We perform learning rate warm-up during the first 10 epochs of training the Gen0 model.

A learning rate schedule adjusts the learning rate during training based on a pre-defined schedule. In the early epochs of training the learning rate is set to be high, then as the training progresses and the model accuracy improves, the learning rate is reduced. In Hawk we support two types of learning rate schedules: a) Step Decay and b) Cosine Decay. In step decay, the schedule drops the learning rate by a factor after every few epochs. The default schedule in Hawk is cosine decay, where the learning rate is decayed as a cosine function. Figure 3.1, shows the learning rate across epochs in Hawk when the cosine decay schedule is used. For the first 10 warm-up epochs, the learning rate is linearly increased and then the learning rate is smoothly decreased.

Optimizer

Optimizers are optimation algorithms that find the value of the parameters such as weights and learning rate to minimize the loss function. These algorithms influence the accuracy of the model and the speed of training. In Hawk, there are three optimizers available: SGD (default), RMSProp, and Adam. They are briefly described below.

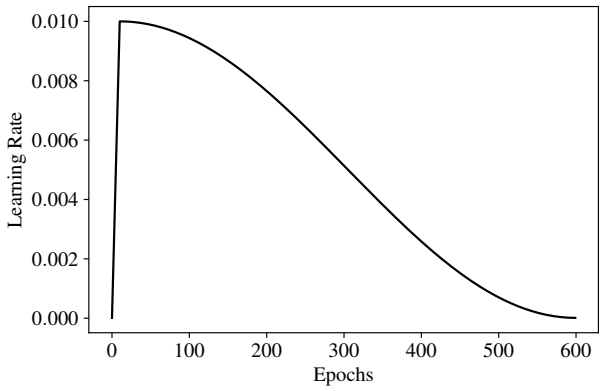


Figure 3.1: Cosine Learning Rate Schedule

MiniBatch Stochastic Gradient Descent (SGD):

This is one of the most widely used optimizer that uses the derivative of the loss function to reduce the loss function and find the minima. It iteratively reduces the loss function by updating parameters in the direction along the steepest descent. The dataset is divided into small batches and after processing each batch, the model weights are updates.

Root Mean Squared Propagation (RMSProp):

RMSProp is a gradient based optimization which uses an adaptive learning rate instead of treating the learning rate as a hyperparameter. It ensures that successive mini-batches have similar gradient updates by normalizing the gradient using a moving average of squared gradients. This helps balance the size of updates and helps mitigate the problem of exploding and vanishing gradients.

Adaptive Moment Estimation (Adam):

It is an optimization algorithm which is a combination of RMSProp and SGD with momentum. It stores both the moving average of the past gradients (momentum) and the moving average of the past squared gradients (RMSProp). The authors[32] empirically show that an Adam optimizer gives much higher performance results than the other optimizers.

Momentum

Momentum, as the name signifies enables the gradient updates to have inertia in the direction of the update. Therefore, instead of only relying on the gradient of the current step, the momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. This accelerates learning and helps overcome the oscillations of noisy gradients.

Weight decay

Weight decay is a regularization technique to prevent the model from overfitting. It is done by adding a penalty term to the loss function. This helps keep the weights small and avoid the problem of the gradients exploding. In Hawk, we use 1e-4 as the default value of weight decay.

Training Epochs

The number of epochs determines the duration of model training. In the presence of a validation dataset, the model is trained until convergence, training for any more epochs will result in accuracy drop. This is done by using an early-stopping scheduler which keeps track of the validation loss and stops the training process if there is no improvement after a given number of epochs.

For a novel rare target, collecting such a substantially large validation dataset (number of positives = 100) may be tedious. In Hawk, if a validation dataset is not available to calculate an estimate of model accuracy, we use a fixed number of epochs. This may result in a suboptimal model, which may be sufficient during live learning. The number of epochs is a configurable hyperparameter in Hawk. In the default configuration, the initial (Gen-0) model is trained for 15 epochs or until converge if a validation dataset is available. Online models are trained for 10 to 15 epochs. In the beginning of a mission, we train the online models for 10 epochs. When the number of labeled positives exceeds 100, the models are trained for 15 epochs. When configuring the training strategy, the labeler can provide the number of training epochs relative to the number of labeled positives.

3.3.2 Label Smoothing

We use label smoothing regularization to stop the model from becoming overconfident and hence reduce overfitting. This technique was introduced to train Inception-v2[69]. Using one-hot encoded labels in cross-entropy minimization encourages the largest possible logit gaps to be fed into the softmax function. This along with bounded gradient will make the model less generalizable and too confident about its predictions. In label smoothing we replace one-hot encoded label y with:

$$y' = (1 - \epsilon)y + \epsilon/K$$

where K is the number of classes and ϵ is a hyperparameter that determines the degree of smoothing. Thus to avoid overfitting, in our experiments we use ϵ = 0.1, the same value as on the paper[69].

3.3.3 Exponential Moving Averaging

Performing exponential moving average (EMA) on the model weights are done to increased accuracy and have more stable models. In Hawk, the averaging happens when a new model is trained, and helps in having a smoothed version of the model. The EMA model weight is updated as follows:

$$M_{N+1} = (1 - \mu).M_N + \mu.M_{N+1,update}$$

where μ is the update momentum (default is 0.6), M_N is the averaged model weight and $M_{N,update}$ is the model update for the $N + 1$ model generation.

3.4 Model Inference in Hawk

Hawk is build to support a wide range of object classification and detection models and its modular design allows easy integration of new models. The ever-evolving research in machine learning is developing new models that are smaller, faster, and more efficient. In this section, we go over the different phases of a model in Hawk learning and describe how a new model can be added to the backend.

3.4.1 Phases of Hawk Model

During a Hawk mission, the model evolves through multiple generations such as "Gen-0", "Gen-1", and so on. Based on the time of creation, training, and functionality, a Hawk model can belong to one of the following three phases: (a) Initial phase, or "Gen-0" model is trained from a handful of labeled positives before a mission starts (b) Online phase, the models ("Gen-1", "Gen-2", ...) are trained quickly during the mission from human-labeled and pseudo-labeled data, and (c) Post-mission phase, the models are used after the mission to infer on new data

From Section 3.4.2 to Section 3.4.4, we describe these phases in more details, their functionality and how they are trained. In the next section, we go over different hyperparameters that are used in the Hawk's training process.

3.4.2 Initial Hawk Model

Before the mission begins, Hawk trains the initial model, Gen-0, from the initial bootstrapping data. This model may be trained on the cloud or on the scouts. When the mission is configured, the labeler may upload the initial bootstrapping examples or she may upload an initial model trained on the cloud. If only examples are uploaded then Hawk trains the initial model on the scouts for 30 epochs or until convergence, if a validation dataset is provided. Hawk samples some of the data captured on the scouts and pseudo-labels them as negatives for the initial training. As the base rate is less than 0.1%, the probability of finding a positive among the sampled data is minuscule. Training on the cloud until convergence yields better accuracy models as the training process can be monitored and controlled by the labeler. In this case, the initial model is uploaded to the scouts during the configuration phase.

As explained in Section 3.1, the number of bootstrapping examples are too few to train a DNN from scratch, in most cases less than 20 positives. In Hawk, we use a model pre-trained on Imagenet that is available on `torchvision.models` subpackage as the base model from which we train the initial Hawk model. Before the training begins, the labeler also provides the configuration file containing the values of hyperparameters listed in Table 3.1. For satisfactory

performance it is recommended that the initial model has an AUC of 0.10 or above on a validation dataset having a similar base rate as the incoming data stream.

3.4.3 Online Models

As the mission progresses, Hawk learns new models from the data labeled by the expert. When training on the scout, the number of epochs are kept short due to the limited compute resources available at the scouts. Since, the main purpose is selective transmission, suboptimal improvement is acceptable as a trade-off for simplicity. Training and inference happens concurrently on Hawk. Therefore, resources used for training a new model limit the inference rate of the current Hawk model. During configuration, the expert can specify the number of training epochs relative to the number of positive labels. By default, Hawk trains the online models for 10 epochs and increases the number of epochs to 15 when the number of labeled positives is above 100 examples. The training of online models takes about 3 minutes on average. A new model generation, Gen-$N+1$ is fine-tuned using the weights of Gen-N model. After training, we perform an EMA averaging of the newly trained model with the current model. This ensures that Hawk learning is not affected by noisy training data. The default value of the EMA momentum is 0.6.

3.4.4 Post-mission Models

Once the mission completes, the Hawk models can be evaluated using a held-out test data. The labeler may either transport all the models to the cloud or she may send control messages to evaluate the models on the scouts. The call returns the metrics of the model relative to the test data. The metrics include model AUC, precision, recall, and f1 scores. The labeler may ship the desired model version back to home and deploy the model for inferencing outside of Hawk.

3.4.5 Hawk model interface

Hawk models are trained on-the-fly during the mission and are used for inferencing incoming datastream. A Hawk mission can operate with or without a test set against which the accuracy of a model is evaluated. A learning algorithm in Hawk contains a 'trainer' which is responsible for training new models. It keeps track of the model versions and the labeled data used during training. The trainer yields a model that is used for inference. A trainer and model in Hawk implement the following methods:

- `trainer.get_version()`: Calls to this method provides the model version of the current Hawk model.
- `trainer.train_model()`: The labeled directory is provided as input to this method. It sets the hyperparameters as per the configuration file and trains a new model generation.
- `model.load_model()`: This method is provided with the path of a trained model. The model weights are loaded to GPU before deploying it in Hawk.
- `model.infer()`: This method takes tiles as input and outputs the class probability along with bounding boxes, in the case of detection.

- `model.evaluate_model()`: Can be used only if a test dataset is available. It is used to evaluate a model against the test dataset. It takes the path to the test dataset as input and outputs metrics such as AUC, precision, and so on.

3.5 Adding a new learning algorithm in Hawk

Hawk learning can be extended by adding new learning algorithms. The user can create a sub-package of the new learning algorithm in the directory `hawk/trainer`. The new trainer inherits its methods from the parent class `ModelTrainer`. The user may modify the `train_model()` method according to the learning algorithm. A configuration YAML file containing the key-value pairs of the hyperparameters should also be included in the package. The output of the `train_model()` method should be an instance of the class `Model`. As mentioned in the previous Section 3.4.5, this class implements methods such as `infer()` and `evaluate_model()`. The user may modify these methods as necessary.

3.6 Configuration Wizard

As mentioned in Section 3.3.1, learning in Hawk uses various hyperparameters the values of which may vary depending on the mission scenario. To assist the domain-expert in choosing the appropriate values for these parameters we have created a web-based configuration toolkit or configuration "wizard" which can generate a Hawk configuration file for the specific mission. The configuration wizard asks the user a series of questions, such as:

- *How many examples are available in the bootstrapping set?*
- *Are the captures scenes cluttered?*
- *Do the images have low-contrast?*

Based on the answers provided by the user, the wizards suggests the most appropriate learning algorithm along with hyperparameter values and generates a mission configuration file. Then, at the start of the mission, the expert provides the generated configuration file to Hawk which learns new models according to the parameters specified.

3.7 Discussion

In this chapter, we discuss the need to develop a new semi-supervised technique for the live learning setting. Because of the rarity of positives in the data, any wrongly labeled positive can degrade the quality of learning and affect the model evolution in Hawk. Thus, we use a modified pseudo-labeling technique to deal with the extreme class imbalance in the data. We also provide details on the hyperparameters used during model training and the different phases of a Hawk model. In the next chapter, we present the architecture of Hawk and provide qualitative analysis in support of various design choices.

Chapter 4

Hawk Design and Architecture

In Chapter 1, we presented the challenges in collecting training data for a rare novel target from weakly connected distributed remote data-sources. To address those challenges, we designed and built the system *Hawk*, a bandwidth-frugal, live learning system that trains machine learning (ML) models on-the-fly. Hawk aims at reducing the volume of data transmitted for discovering instances of a rare target from unlabeled data captured at remote sources. The system has a self-improving selective transmission mechanism that chooses the best items to present to the labeler. The system tightly integrates machine learning, selective transmission, labeling and bandwidth-aware adaptation.

Unlabeled data for labeling and learning are continuously captured by sensors on remote platforms called scouts. As mentioned in Chapter 1, we assume a scout to have compute and storage comparable to a high-end desktop today, and both inferencing and training of DNN models are possible on such a platform. Some scouts may be static, while others are mobile, such as autonomous drones. A scout uses its on-board compute capability to process incoming data, and to selectively transmit a small subset of the data that it deems to be "interesting" relative to the phenomena being explored. However, data that is not immediately perceived as "interesting" is not lost, but retained on local storage. Scouts have ample storage resources to allow for retention periods on the order of a few days to a few weeks. This is easily achieved today by available storage solutions [1]. This allows for reexamination of archived data to find missed positives, when new knowledge is available.

Data collection, inferencing, selective transmission, labeling, and training happen concurrently in Hawk. The complexity of Hawk learning is hidden from the labeler. She is only aware that, over time, more of the data sent to her for labeling are positives and fewer are trivial negatives. Hawk expects no programming or machine learning skills of domain experts. Accurate domain-specific labeling is all that is expected.

This chapter is organized as follows. We first discuss the working of Hawk and provide an end-to-end data pipeline in Section 4.1. In Sections 4.3 to 4.9, we describe the different design decisions in Hawk. We provide quantitative and qualitative analysis of the design choices against three calibration tasks given in Section 4.2.

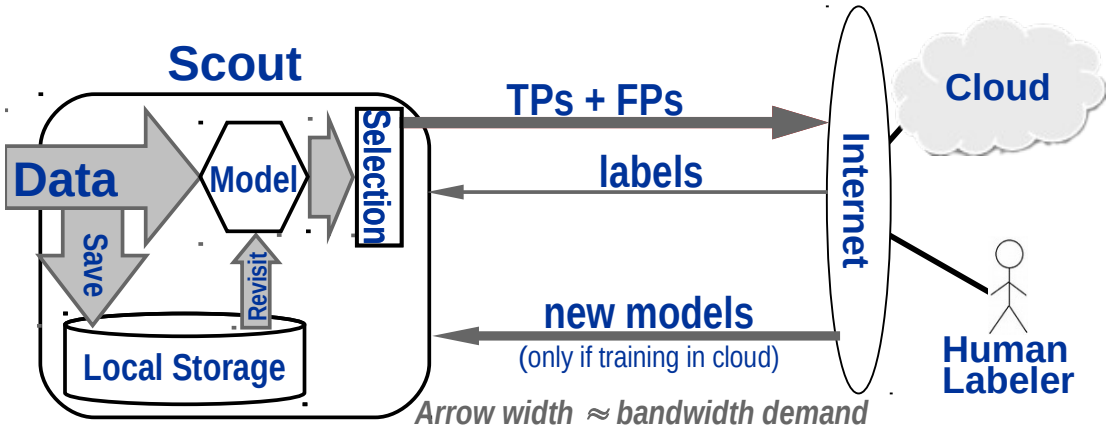


Figure 4.1: End-to-End Data Flow in Hawk

4.1 End to End Data Flow

In this Section, we provide a high-level overview of the Hawk architecture. Figure 4.1 illustrates the end-to-end flow of data in Hawk. In the figure, the width of the arrows is proportional to the volume of data flow. As shown in the figure, incoming video streams from the sensors to the scout are both written to the local storage and processed. The processing at the scout consists of three steps. First, the video stream is decoded to individual image frames. Each of these high-resolution image frames is then broken up into smaller tiles (Section 4.3). Finally, the tiles are inferenced in small batches using the current model.

Inferencing is done in small batches for three main reasons. First, batching is typically needed to make efficient use of Graphics Processing Units (GPUs). Neural networks are embarrassingly parallel algorithms, where computations of each node are generally independent of other nodes [6]. GPUs excel in parallel programming and helps accelerate model training and inference by several orders of magnitude. Processing data in batches help effectively utilize the thousands of processing cores available in a GPU and reduce the time spent in data stalls. Second, certain selective transmission strategies, such as "TopK" and "Maximum Entropy", are only meaningful on batches. More about this can be found in Section 4.5. Third, the reactive time in live learning is lowered by generating results in small batches rather than aggregating results from the entire data.

A small subset of the tiles inferenced by the model are selected for transmission (Section 4.5). The poor backhaul network bandwidth prevents transmitting all the tiles to the labeler. Ideally, we want to transmit only tiles that contain instances of the target class. This would have been possible if an accurate model was already available on the scout. The occasional false positives (FPs) will only waste a small amount of band- width. However, no such pre-trained model exists for a novel target and the purpose of the mission is to collect more true positives (TPs) needed to train an accurate model. Thus, we have a chicken-or-egg problem.

Early in the mission, when the model is weak, the selective transmission will be suboptimal. Many FPs will be transmitted for labeling, and some TPs will not. Later, as the model improves, these errors are reduced. At all times, the inherent low base rate of the target class

Figure 4.2: Common Yellowthroat

Figure 4.3: MQ-9 Reaper Drone

means that most incoming data is not transmitted. Hawk has a tunable data sourcing policy (Section 4.7) that guides how it balances between processing new incoming data and re-processing previously-discarded data that is still available on scout storage. The selective transmission process is agnostic to data-source; all that matters is that selected items receive a high score from inferencing using the current model. The selective transmission algorithm is adaptive with respect to end-to-end runtime bandwidth. Its goal is to fill (but not overfill) the end-to-end pipeline and thus optimize the use of scarce bandwidth.

Transmitted items are queued for labeling by one or more human experts. All data received for labeling is also available for training in the cloud. We assume labels provided by experts are accurate. Labels are transmitted back to the scout as they are generated. The uplink bandwidth demand for this is very small. No tile data has to be transmitted; only numeric values indicating tile identity, labels, and bounding boxes.

4.2 Calibration Tasks

To provide quantitative and qualitative insights for Hawk's design decisions, we created three pilot tasks that are representative of Hawk missions. These tasks are derived from publicly available datasets. Most publicly available labeled datasets are balanced, they have an equal number of examples across classes. In Hawk, we assume the target class is rare, having a base rate of 0.1% or lower. The tasks considered in this section are unrelated to the datasets that are used to validate the system in Section 6.2. Hence, the danger of overfitting is completely avoided.

4.2.1 YFCC-derived Task

The *YFCC-derived task* aims to collect TPs of a rare bird species in a bird-rich environment. The dataset for this task starts from the publicly-available YFCC100M dataset of nearly 10^8 unlabeled images [72]. From this dataset, we selected 10^6 images that pass a pre-trained YOLOv3 [56] filter for the class "bird" at low threshold. The resulting dataset is diverse, yet strongly biased towards birds. From the publicly available, labeled iNaturalist dataset [76], we augment our bird-rich dataset with 10^3 images of the bird "Common Yellowthroat" (Figure 4.2). Since this bird does not appear in the YFCC dataset, we are able to precisely achieve a base rate of 0.1% which meets our definition of "rare phenomenon."

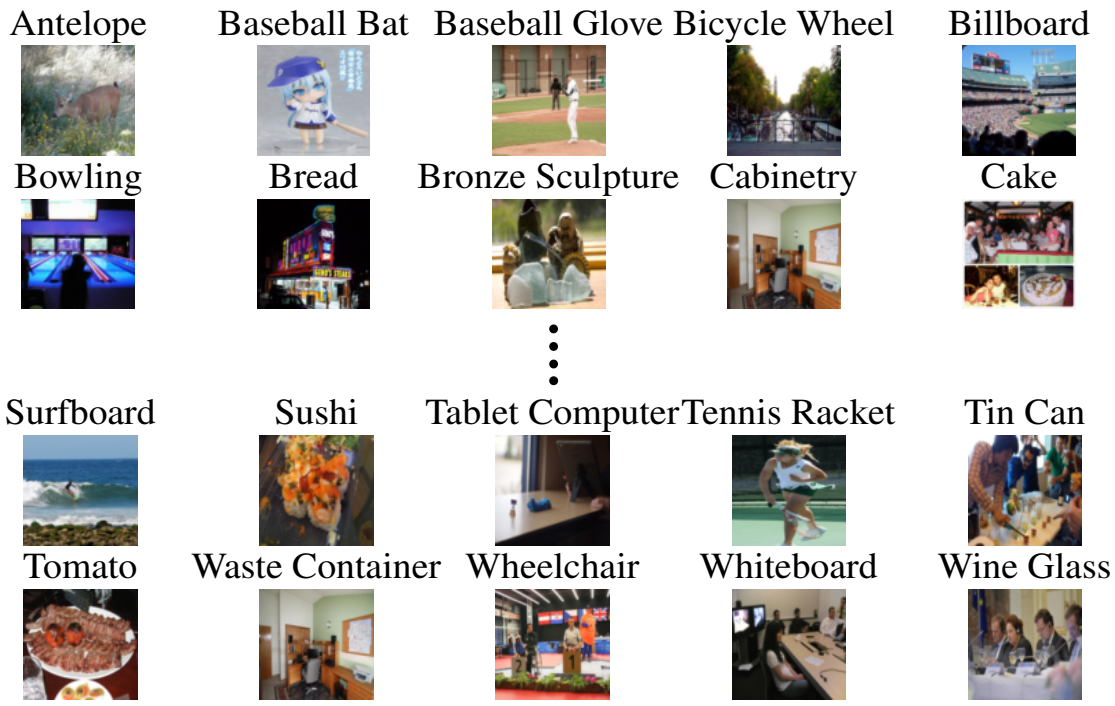


Figure 4.4: Sample of Target Classes Chosen from Open Images

4.2.2 DVIDS-derived Task

The *DVIDS-derived task* aims to collect TPs of a new aerial threat in an aircraft-rich environment. The dataset for this task starts from two million military-related labeled images available from the Defense Visual Information Distribution Service (DVIDS) [3]. From this dataset, we constructed a dataset of 680,000 total images with a heavy bias towards military aircraft. We then added 680 images of the target class, MQ-9 Reaper Drone (Figure 4.3), to achieve a base rate of 0.1%. This is a challenging task because the target class looks very similar to many other military aircraft in the dataset.

4.2.3 Open_Images-derived Task

The Open_Images-derived dataset uses the entirety of 9.2 million images in the Google Open Images Dataset V6, labeled with bounding boxes for 600 different object classes. Of these classes, we select 100 targets, a subset of which are shown in Figure 4.4. This dataset is used test the generalization of Hawk design. In each experimental run, one of these 100 classes is used as the target, and a random dataset of 500,000 images is constructed: 500 contain the target class, the rest are randomly sampled from all images without the target. Experimental results are averages over three runs for each of the 100 target classes.

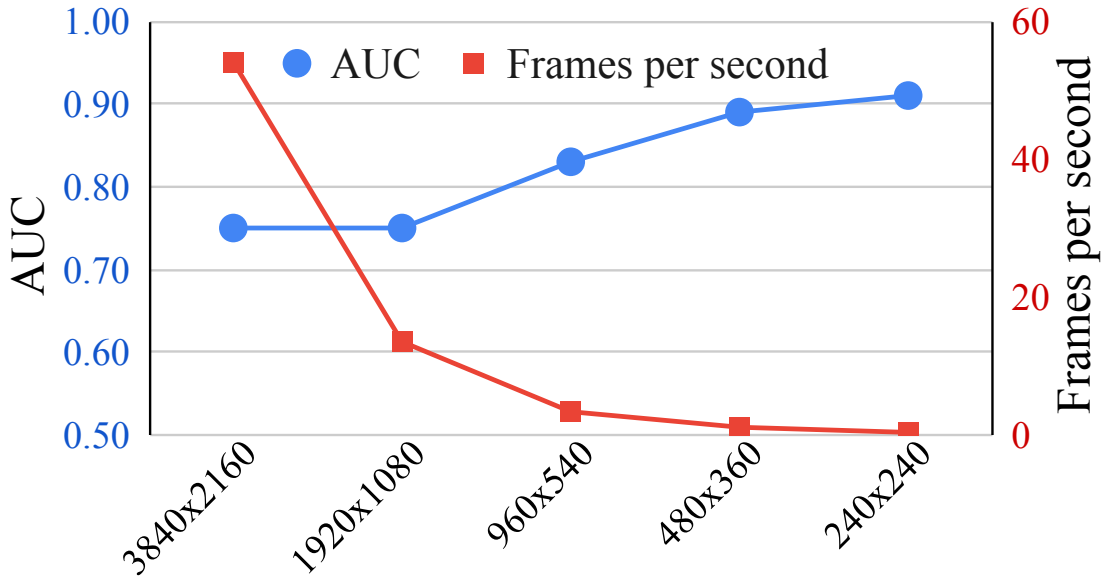


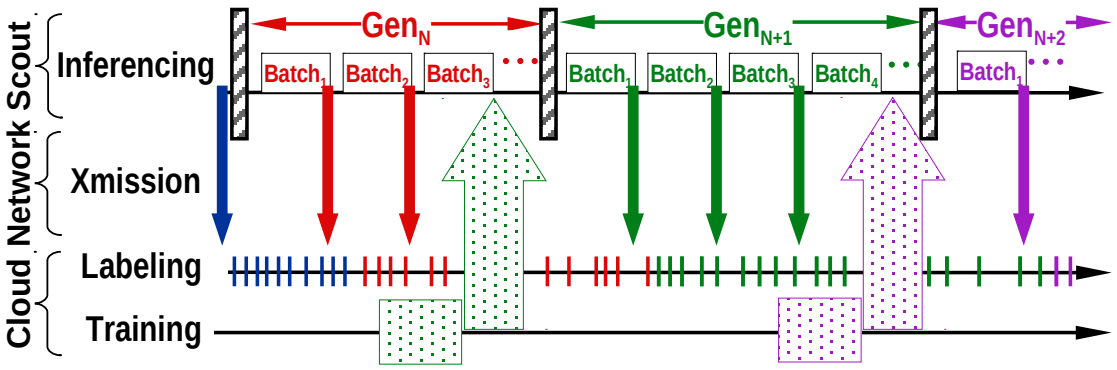
Figure 4.5: Tile Size Trade-off on 4K Okutama Dataset. AUC (area under the precision-recall curve) is a standard metric of model accuracy.

4.3 Tiling of High Resolution Input Data

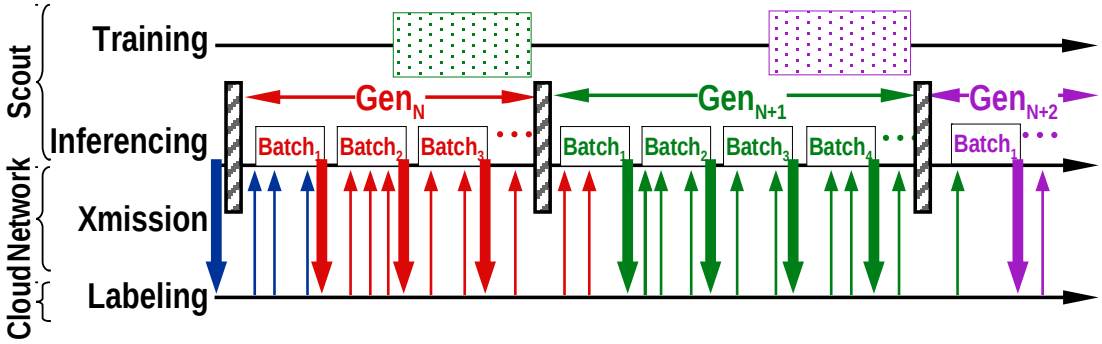
Before inferencing, a model typically transforms input data to a fixed low resolution (e.g., 256x256 for ResNet, and 600x600 for YOLO). Unfortunately, downsampling negatively impacts the accuracy of inferencing on high-resolution input data. For a 4K video frame, downsampling to 256x256 pixels shrinks a small object (20x20 pixels) down to barely a single pixel. It is inherently difficult to reliably detect such a small object [26]. This problem will only worsen as camera resolution increases to 8K and beyond.

To address this problem, Hawk splits high-resolution frames into smaller tiles. Tile size is a configuration parameter of Hawk, and is set to match the downsampling of the model. The same tile size is used both for training and for inferencing. The price paid for tiling is increased computational demand, both in training and in inferencing. For example, splitting a 4K frame into 128 tiles results in that many more inferences to perform within a finite time budget determined by the frame sampling rate.

We illustrate the trade-off in Figure 4.5, using the 4K Okutama dataset [9]. For this experiment, we use ResNet50 as the ML model. As stated earlier, irrespective of the input frame, the model down-scales the input to a fixed resolution of 256x256. The blue curve shows the gain in model AUC, when the input frame is split into smaller tiles. When whole 4K frames are given as input to the model we only get a model AUC of 0.74, in contrast when the frames are tiled to 256x256 resolution we observe an increase in recall and hence an improvement in model AUC to 0.91. However, when a frame is split into tiles of 256x256, the model has to process 128 tiles on average resulting in a decrease in throughput as given by the red curve. The throughput drops from 55 FPS when a single frame is inferenced to less than 1 FPS, because of the increase in the number of input tiles the model has to inference. Accuracy improves as tiles become smaller, but the sustainable frame rate drops [78].



(a) Training in the Cloud



(b) Training on scout

Time flows horizontally in these figures, which are not drawn to scale. If multiple scouts are participating, their transmissions are merged into a single stream for labeling.

Figure 4.6: Timelines at scout and Cloud

4.4 Training Site

Standard machine learning approaches require centralizing the training data on one machine or in a datacenter. Also, training in the cloud benefits from the unlimited hardware resources available there. In Hawk, the data is collected by distributed remote platforms. However, all the TPs required for training are available at the cloud for the construction of a training set, since they are transmitted for labeling (Figure 4.1). The negatives for the training set can be assembled from all FPs that were transmitted, augmented by negatives from cloud archives. By definition, since the phenomenon being studied is new and rare, there are few TPs in the archives — almost everything there is negative. Figure 4.6(a) and (b) shows the timeline of events in the Hawk pipeline when training is done in the cloud and scouts respectively.

Time flows from left to right in this figure. In both timelines, a sequence of improving models (Gen_{N-1}, Gen_N, Gen_{N+1}, ...) inferences data at the scout. A small, high-scoring subset of the data is then selected (Section 4.5) for transmission and labeling. Concurrent with selecting results to present for labeling, Hawk also trains a new and improved classifier with a training set that includes recently-labeled data. Training of a new model is triggered when the number of new

	bzip2			rsync		
Model Instance	Input (MiB)	Output (MiB)	Compressed (%)	File size (MB)	Data sent (MB)	Reduction (%)
a	89.95	85.56	4.88%	94.32	88.84	5.81%
b	89.95	85.57	4.87%	94.32	88.87	5.78%
c	89.95	85.56	4.88%	94.32	88.84	5.81%

Table 4.1: Efficacy bzip2 and rsync on DNNs (ResNet-50). Compression level was consistent across different trained instances, so only 3 examples shown here.

Model	Input (A) (MB)	Output (B) (MB)	Compression Ratio (A:B)
ResNet-50	97.69	38.61	2.53
YOLOv5-small	27.60	11.95	2.31
EfficientNet-B4	74.26	27.71	2.68

Table 4.2: DeepIoT Model Compression

positives labeled has increased by a certain percentage. This percentage is a tunable parameter of Hawk. When training is complete, the new model replaces the current model.

As the figures show, the human labeler is oblivious to models evolving over time. She is only dimly aware that the quality of processing seems to be improving. Poor backhaul bandwidth to the Internet, combined with the cognitive delay of an expert in labeling, shift the labeling timeline to the right relative to the inference timeline. Hence, even after Hawk has switched to a new classifier, the human labeler might be still be labeling results from a previous model for some time.

When the training happens on the cloud, the new model has to be transmitted from the cloud to the scout. Deep learning models for image classification tend to be large. For example, in our experiments, the ResNet-50 model has a size of 95 MB and the YOLOv5-small model has a size of 14 MB. At 12 kbps, transmitting these models would take over 17 hours and over 2.5 hours respectively. Only with aggressive compression can transmitting models that are tens of MBs in size over kbps networks be practical. Classic compression and deduplication techniques tend to be ineffective on DNNs. Table 4.1 shows, for example, how ineffective `bzip2` and `rsync` are on a sampling of the models used in our experiments. A much more promising approach is the seminal work of Yao et al [82] on DeepIoT, which reports size reductions of 90–99% on sequential model architectures such as LeNet and VGG. DeepIoT obtains a global view of deep model parameter redundancies and learns to compress DNN model structures into smaller dense matrices without compromising much on the model performance. It does this by finding the minimum number of non-redundant hidden elements in the deep model. Unfortunately, DeepIoT is less effective in DNNs having more complex architectures such as ResNet-50 and YOLO that are used today on image datasets. Our measurements (Table 4.2) only show factors of two to three reduction in size. Using DeepIoT, a ResNet-50 model of size 97.69 MB can be compressed by 2.53X to obtain a model of size 38.61 MB. In the case of YOLOv5-small and EfficientNet-B4, the compression ratio is 2.27X and 2.68X respectively. While substantial, this is not adequate

Model	Original (A) (MB)	LAYER-1		LAYER-2		LAYER-3	
		Output(B) (MB)	Compression Ratio (A:B)	Output(B) (MB)	Compression Ratio (A:B)	Output(B) (MB)	Compression Ratio (A:B)
ResNet-50	97.69	7.81	12.50	24.84	3.93	41.86	2.33
YOLOv5-small	27.60	5.39	5.12	7.64	3.61	8.77	3.15
EfficientNet-B4	74.26	6.84	10.86	9.92	7.49	21.55	3.45

Weights of earlier layers are frozen and do not change during training

Table 4.3: Model Compression: Freezing Layers

for extreme low bandwidths. Later in Section 6.4.4, we explore the impact of 100X and 10X model compression if they were attainable on DNNs such as ResNet-50, YOLOv5-small, and EfficientNet-B4.

On the other hand, training on the scout is severely limited by the hardware resources available, and by the need to share these limited resources with concurrent inferencing of incoming data. However, it has the benefit of avoiding the transmission of large models over an extremely low-bandwidth network. The lower the bandwidth, the more valuable this benefit. Hence, Hawk uses an adaptive training strategy. When bandwidth is high, training in the cloud is an obvious win. At very low bandwidth, however, training at the scout is the better approach. If an early decision is made to train on the scout, a further bandwidth optimization is possible. Only thumbnails are needed for labeling, while full-resolution data is needed for cloud-based training. Even though the training is now much slower than cloud-based training, it is more than compensated by avoiding model transmission. The cross-over point is dependent on the model compression achievable. In Section 6.4.4, we explore these trade-offs in Hawk.

A more sophisticated compression technique would be to train only the last few layers of the model, and freeze (or fix weights) the initial layers. The initial few layers learn generic features of images, such as colors, edges or corners of objects. By keeping their weights fixed we retain the knowledge learned during pre-training and reduce the time required for transfer learning on new data. Since, only the weights of a few layers are trained, we save the network bandwidth required for transmitting the model. Table 4.3, gives the compression ratio for different models when only the last layers are trained and transmitted during cloud training.

4.5 Selective Transmission

The poor backhaul bandwidth limits the number of images that can be transmitted to the expert for labeling. Hawk has to be selective in its transmission of images that needs to be labeled by the expert. As explained in Section 4.1, Hawk inferencing is done in batches on a combination of live and reexamined data. It then filters the output through a selection policy. Hawk trains models recursively. The current model thus embodies the learning that has occurred since the start of the mission. A newly-trained model is deployed as soon as possible. This helps in the efficient utilization of network bandwidth by reducing the volume of trivial FPs being transmitted.

In stream-based active learning, when a data item is encountered the algorithm makes a decision whether to query the human. The decision is made independent of other items in the data stream. This strategy may fill up the network bandwidth if the algorithm is not selective.

On other hand, if the algorithm is too conservative it may starve the labeler or waste available network bandwidth. To counter this, Hawk processes data in small batches before it selects items to query the labeler. As mentioned before in Section 4.1, inferencing in batches ensure efficient use of GPUs and some strategies for selective transmission are only meaningful on batches. In Hawk, out of B processed items, K items are selected to be transmitted to the expert for labeling.

The scores given by the model define a soft partitioning of the batch into positives ($\mathcal{P}$), hard negatives ($\mathcal{HN}$) and easy negatives ($\mathcal{EN}$). "Easy" and "hard" are, of course, relative to the current model. An $\mathcal{EN}$ is far from the decision boundary of the current model, while an $\mathcal{HN}$ is much closer. Inferencing scores close to 1.0 suggest $\mathcal{P}$, while scores close to 0.0 suggest $\mathcal{EN}$. Everything deemed $\mathcal{P}$, but nothing deemed $\mathcal{EN}$, is transmitted for labeling. The "gray middle" is more difficult, because it may include some $\mathcal{P}$ with lower scores than they deserve.

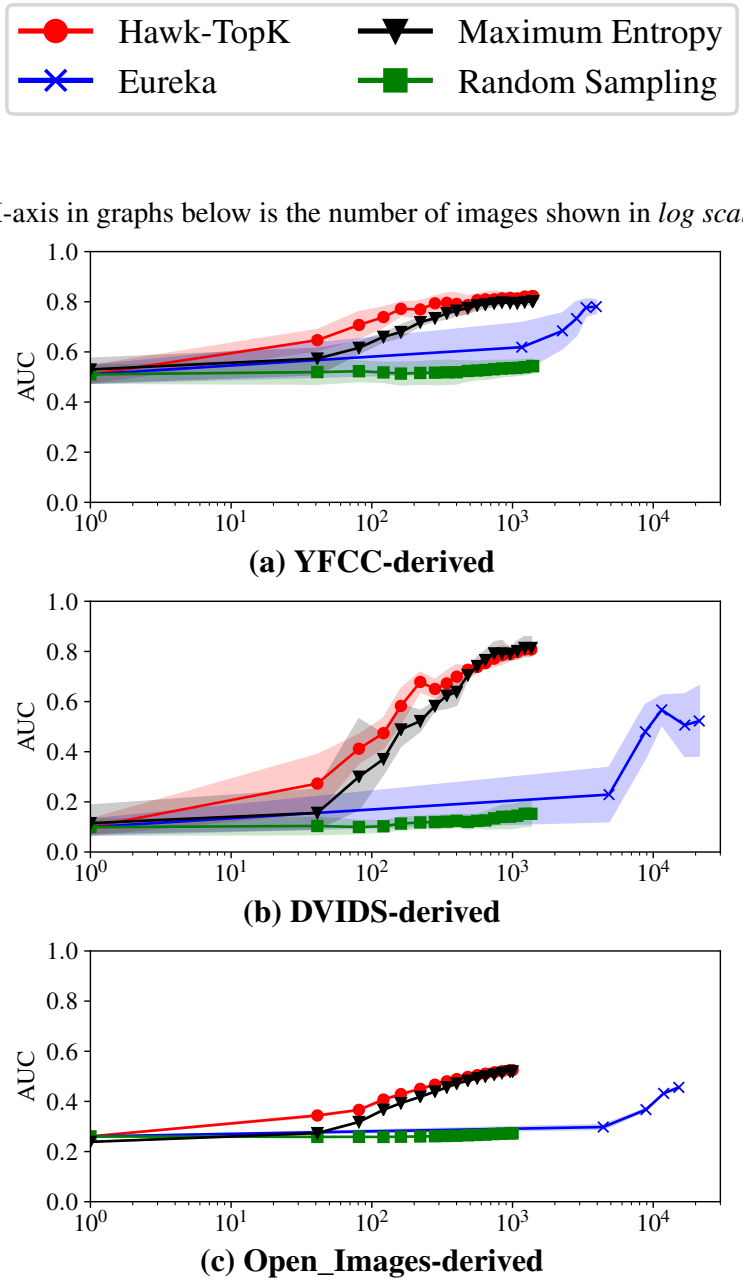
Previous work, Eureka, has a threshold-based selection scheme, it presents all data items that score above a threshold, as they are encountered in the stream. Having a fixed threshold results in more FPs being transmitted to the client, especially in the early stages of learning when the model performance is poor. Another challenge in this selection policy is that the process of finding a "good" threshold cutoff is highly manual and requires many iterations of trial and error.

Table 4.4 gives the results for multiple data selection schemes from active learning that we explored in Hawk. In particular we examined three data selection policies, namely "random," "threshold," "maximum-entropy," and "TopK". A "random" strategy transmits K random items, while a "TopK" presents the top K scoring data items encountered in a batch of size B. "Maximum-entropy" strategy transmits those K items about which the classifier is most uncertain. For binary probabilistic classifiers, this is equivalent to selecting K items whose probabilistic confidence is closest to 0.5. The "threshold" strategy is the same as the strategy used in Eureka, where items above a certain confidence threshold is transmitted to the annotator. In our experiments we use a confidence threshold of 0.6. In all three cases we transmit 20 items for labeling after processing a batch of 10,000 images.

As can be seen from the summary in Table 4.4 and Figure 4.7, maximum-entropy and TopK improved much faster than random sampling or thresholding policy in Eureka classifiers. Note, the X-axis of the graphs in Figure 4.7 are in logarithmic scale. As seen in Figure 4.7, though "TopK" and "Maximum-entropy" asymptotically converge to similar AUC, TopK improves much faster in the early stages of labeling as compared to maximum-entropy. Another interesting observation is the disparity in the number of positives discovered when using the two strategies. Using TopK, we transmit about 1.3X – 1.5X more positives as compared to maximum-entropy. Because maximum-entropy transmits examples it is most uncertain about, there may be scenarios where the number of positives transmitted is even less that no-learning strategy, as can be observed in Table 4.4(a). Thus, TopK may be the better choice if one plans to use the current classifier for ingest or search while labeling is still in progress.

In the rest of the chapter, we refer to the Hawk variant that uses TopK policy as Hawk-TopK. With an extremely low base rate, the conventional ML approach of i.i.d. random sampling will uncover very few positives, and is not very effective. Though Eureka and Hawk, converge to roughly the same AUC for the YFCC and Open_Images tasks, Hawk achieves this with 3X – 15X less human labeling effort than with Eureka.

These results suggest that Hawk's approach of only requesting labels for a small subset of the data items pays off handsomely. The fact that Eureka converges to a significantly lower AUC



Data points on curves show where an old classifier was replaced by a new one. Halo around each curve is the standard deviation across three runs of each experiment. The halos for Open_Images-derived are present but slight due to the averaging across 100 classes.

Figure 4.7: AUC Improvement with Labeling Effort

Method	Images Shown	Positives Found	Positives Found / Images Shown	AUC
No-Learning	1400	496 (11)	0.35 (0.01)	0.51 (0.03)
Random	1400	1 (1)	0.00 (0.00)	0.54 (0.03)
Maximum-Entropy	1400	436 (12)	0.31 (0.01)	0.80 (0.01)
TopK	1400	616 (11)	**0.44 (0.01)**	0.82 (0.02)
Eureka (Threshold)	3936	628 (10)	0.23 (0.15)	0.78 (0.03)

(a) YFCC-derived

Method	Images Shown	Positives Found	Positives Found / Images Shown	AUC
No-Learning	1360	128 (12)	0.09 (0.01)	0.10 (0.03)
Random	1360	2 (0)	0.00 (0.00)	0.15 (0.05)
Maximum-Entropy	1360	324 (5)	0.24 (0.00)	0.81 (0.03)
TopK	1360	499 (3)	**0.37 (0.00)**	0.81 (0.02)
Eureka (Threshold)	21228	557 (12)	0.03 (0.01)	0.52 (0.13)

(b) DVIDS-derived

Method	Images Shown	Positives Found	Positives Found / Images Shown	AUC
No-Learning	1000	204 (1)	0.20 (0.00)	0.26 (0.00)
Random	1000	2 (1)	0.00 (0.00)	0.27 (0.00)
Maximum-Entropy	1000	221 (0)	0.22 (0.00)	0.52 (0.00)
TopK	1000	295 (1)	**0.30 (0.00)**	0.53 (0.01)
Eureka (Threshold)	15209	410 (0)	0.03 (0.00)	0.46 (0.01)

(c) Open_Images-derived

Figures above are means of three experimental runs and the figures in parentheses are standard deviations.

Table 4.4: Data Selection Results

than the Hawk variants on the DVIDS task (0.52 versus 0.81) in spite of collecting more positives (557 versus 324/499), suggests that the FPs surfaced by Hawk also serve as valuable examples in the training set.

In our work, we assume a uniform base rate of 0.1% or lower, we acknowledge that this is not true of a real-world deployment where the data distribution is often non-uniform and there may be a possibility of burstiness in the occurrence of TPs. The burstiness could be temporal or spatial in nature. In such a scenario, where the TPs are more than the value of K, Hawk might miss or delay the transmission of some of the TPs. We do not address this problem in our work but leave it for future work (Section 8.2.7).

4.6 Rapid Early Improvement of Model Precision

Early in the mission, when the model controlling selective transmission is weak, rapid improvement of precision is crucial to reducing wasted network bandwidth. Any positives that are missed because of poor recall can be discovered by later visits to the discard pile (Section 4.7). To improve model accuracy for new targets from different viewpoints, we use the technique of transfer learning [8]. For rapid training, Hawk can be configured to use a cascade filter based on a support vector machine (SVM) [68]. Training an SVM can be done in a fraction of a second even on scout hardware. The SVM uses the weights of the penultimate layer of the DNN as a feature vector. We call this *shallow transfer learning*, as the model weights trained are small and shallow and the training requires fewer number of labeled examples to converge as compared to DNN training. For all design analysis experiments specified in this chapter, we use an SVM based linear classifier.

The use of an SVM strategy only works when there is a significant structural similarity between the phenomena being explored and the training set on which the DNN was trained. In scenarios where this is not true, we need to apply *deep transfer learning*, where weights of some DNN model parameters are trained on the new data. We explore this further in Section 6.3.

To evaluate whether Hawk can efficiently use human labeling effort, we explore how a model's AUC (the area under its precision-recall curve) improves as more data is labeled. Attaining higher AUC values with fewer labels indicates higher efficiency. Results in Table 4.4 shows how AUC improves in Hawk with labeling as compared to "No-Learning". The row corresponding to TopK is what we refer as Hawk-TopK."No-Learning" is the same as Hawk-TopK except no learning is done to improve Gen-0. On all three tasks, Hawk-TopK performs better than No-Learning. For YFCC-derived task, Hawk achieves an AUC of 0.82 vs. 0.51 in the case of No-Learning. Similarly for DVIDS (0.81 vs. 0.10) and Open_Images(0.53 vs. 0.26), Hawk outperforms No-Learning by a huge margin.

One reason for the low AUC performance in the Open_Images-derived task is the size of the target class in the images of the dataset. Open_Images dataset is an object detection dataset and the images in this task are highly cluttered with the target object at times occupying only a small fraction of the image. The model trained is unable to capture the features of this small target and are distracted by features of larger, dominant objects in the scene rather than the desired target.

4.7 Re-visiting Discards

A scout wastes little bandwidth on $\mathcal{EN}$s. Rigidly adhering to this principle would mean that a $\mathcal{P}$ that is misclassified as an $\mathcal{EN}$may never be transmitted for labeling and will be lost forever. This would be especially unfortunate because of the low base rate — every positive is precious. The obvious solution of lowering the selectivity of transmission is unacceptable because the increased level of FPs would overwhelm the limited bandwidth.

We therefore keep selectivity high, but periodically re-visit items that were discarded earlier. Training of a new model is triggered in Hawk upon accumulation of positive labels. The better recall of an improved model may help to find a "lost" $\mathcal{P}$. To understand the merits of re-visiting discarded data, we conducted experiments using the calibration tasks. Table 4.5 shows the total

Re-visit Approach	Total Images Processed	Total Positives Found	Percentage of Ground-Truth Positives Found
YFCC-Derived Task (700 ground-truth positives)			
None	700,000	616 (11)	88%
Top 100	715,300	629 (5)	89%
All	5,460,460	630 (1)	90%
DVIDS-Derived Task (680 ground-truth positives)			
None	680,000	523 (13)	73%
Top 100	693,600	552 (3)	81%
All	4,751,840	575 (1)	85%
Open_Images-Derived Task (500 ground-truth positives)			
None	500,000	201 (1)	40%
Top 100	510,500	219 (3)	44%
All	3,424,140	222 (2)	44%

Figures in parentheses are standard deviations across three runs.

Table 4.5: Value of Re-visiting Discard Pile

	REPLICATE_FULL		REPLICATE_POSITIVES	
Dataset	Positives Found	Final AUC	Positives Found	Final AUC
YFCC-derived	616 (11)	0.82 (0.02)	617 (5)	0.79 (0.02)
DVIDS-derived	499 (9)	0.81 (0.02)	489 (8)	0.80 (0.02)
Open_Images-derived	295 (1)	0.53 (0.01)	289 (0)	0.51 (0.01)

Figures in parentheses are standard deviations across three experimental runs

Table 4.6: Results using different data sharing policies

number of positives discovered for each task using three approaches: (a) no re-visit; (b) re-visit only the 100 top-scoring results in the discard pile of each previous batch; and (c) re-visit all previous discards.

Three observations follow from these results. First, there is indeed value in re-visiting the discard pile. For example, the bottom row for the YFCC-derived task in Table 4.5 shows that 14 additional positives were found, an increase of about 2.3%. Second, the last column shows that nearly 10% of the positives were still missed, suggesting ample opportunity for further improvement. Third, just re-visiting the 100 top-scoring items in the discard piles of each batch is almost as good as re-visiting all discarded data. For a modest increase in the number of images processed (barely 2%), almost the full benefit is obtained (629 out of 630 positives). The results for the DVIDS and Open_Images tasks in Table 4.5 confirm these observations. Based on these results, Hawk uses the policy of just re-visiting the 100 top-scoring items in discard piles. In effect, scores from an old model act as a result cache to guide the selection of items to rescore with a new model.

4.8 Randomized Use of Unlabeled Data

The use of $\mathcal{EN}$s in training is important because it reduces model drift over the course of evolution and helps improve the quality of the model. As mentioned earlier, when training is done in the cloud there is a large supply of $\mathcal{EN}$s available from cloud archives. Although unlabeled by the current mission, these can be assumed to be negatives ($\mathcal{EN}$s or $\mathcal{HN}$s) because the phenomenon being explored is new. In contrast, when training is done on the scout, the large supply of unlabeled $\mathcal{EN}$s may include some classification errors.

To reduce the impact of these errors, we randomize the selection of $\mathcal{EN}$s that are chosen afresh for each training session. Randomizing the selection helps overcome a major risk in using unlabeled data for training on the scout, namely the possibility of a $\mathcal{P}$ that is misclassified as an $\mathcal{EN}$ wreaking havoc. There is no way to fully avoid this danger — it is inherent in using unlabeled data. However, by randomization we reduce the chances that a misclassified $\mathcal{P}$ will be selected again from the huge volume of $\mathcal{EN}$s is low. A specific model may be affected, but it is unlikely that this will harm the long-term convergence of the model. Without consuming precious network bandwidth, freshly-collected $\mathcal{EN}$s are still able to play a useful role in training.

4.9 Leveraging Multiple Scouts

Hawk enables scouts to work collaboratively as a team. Examples include drone swarms for military reconnaissance, and space missions in which a swarm of scouts is launched from an unmanned orbiter to explore a planetary body [24]. The result streams from these scouts are merged in temporal order and delivered to the labeler. Internal bookkeeping ensures that every label is conveyed to the scout that generated that result. In Hawk, we explored two data sharing policies, namely, REPLICATE_FULL and REPLICATE_POSITIVES. REPLICATE_FULL is a bandwidth-hungry policy in which all labeled training data is shared across all scouts. In contrast, REPLICATE_POSITIVES only shares the TPs across all scouts. The negatives are locally obtained, and therefore different at each scout. A potential concern with REPLICATE_POSITIVES is that the accuracy of the model may suffer with respect to REPLICATE_FULL because negative examples are not being shared across scouts. Table 4.6, compares the performance of the two policies across the three datasets. For each dataset, the AUC for REPLICATE_POSITIVES is nearly identical to that of REPLICATE_FULL.

Chapter 5

Hawk Implementation

In the previous chapter, we described and evaluated the design of Hawk. As discussed, Hawk uses an adaptive training strategy, it selects the site of model training based on the available backhaul bandwidth to the Internet. For prompt response during live learning, Hawk quickly trains and deploys new improved but suboptimal DNN models using transfer learning. In the default mode of operation, Hawk uses TopK selection strategy to transmit items to the expert for labeling. In our results, We also show that Hawk-TopK always performs better than the "No-Learning" search strategy, which uses the initial model with no further learning or improvement.

Hawk is implemented in Linux using C, Python, and PyTorch. The entire implementation is in user space, and Docker is used to encapsulate the components that pertain to learning. There are no kernel modules or kernel modifications. For network communication we use ZeroMQ[23], a transport-agnostic asynchronous network messaging library which is cross-platform and supports 30+ languages.

The Hawk server is started on scouts, which are configured with user-specified mission parameters before they are launched. The configuration includes the type of training, selective transmission, and retraining strategies to be used for the mission. Once the scout reaches the location of exploration where relevant data will be encountered, the mission is started. The high-resolution data that is collected at the scout is inferred using the DNN model that is currently in use. Hawk effectively uses the scarce bandwidth to transmit only a small fraction of data to the user for labeling. Any result that is explicitly labeled is added to the training set for future training. The labeled results are shared across the scouts. Training of a new model is triggered when the total number of new positives reaches a threshold. Once training is completed, the new model immediately replaces the existing model on that scout. The improved accuracy of a newly-trained model helps reduce the volume of false positives transmitted and improves the value of human labeling effort.

We use the term HOME to refer to the location of the domain-expert. At HOME, the expert monitors the progress of the mission via a labeling GUI. The GUI displays thumbnails of the result stream to be labeled. Section 5.3 provides details about the labeling GUI. Since Hawk is designed to be used by domain-experts who are not trained in machine learning, the default settings of these parameters are sufficient for satisfactory operation.

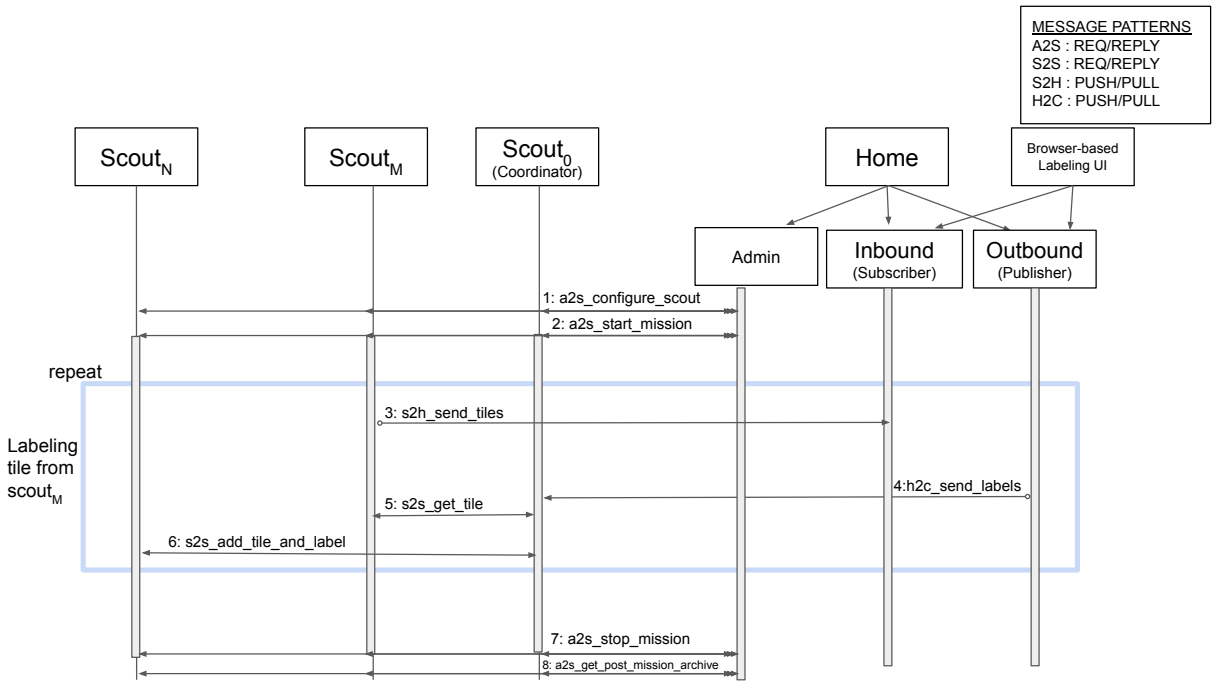


Figure 5.1: Sequence of API calls in Hawk.

5.1 Hawk API

The expert connects to the scouts using the GUI from the HOME. A Hawk mission involves multiple rounds of network communication between scouts and HOME. Over these rounds, Hawk continually trains and improves the DNN model to produce increasingly accurate results over time. Among the scouts, one of the scout is selected as the COORDINATOR which collects labels from user and distributes it to other scouts.

Hawk uses the ZeroMQ[23] messaging library for the communication between scouts and HOME. ZeroMQ is an asynchronous messaging library, which provides a message queue without a dedicated message broker. The ZeroMQ API provides sockets that carry atomic messages across various transports like TCP, UDP, in-process, inter-process, and multicast. In Hawk, we use two basic messaging patterns:

1. Request-reply (REQ/REPLY), a client connects to a server, requests info, then receives a reply. This is a remote procedure call and task distribution pattern. This pattern of messaging is used when there is good connectivity between the server and clients. It is a synchronous call, where the client wait for response from the server before processing continues on the client side.
2. Push-pull (PUSH/PULL) is a parallel task distribution and collection pattern. The messages are distributed in a round-robin fashion to the clients. The PUSH/PULL sockets are one-way only. This is equivalent to producer/consumer model. The results computed by clients or consumers are not sent upstream. PUSH/PULL sockets are asynchronous. In Hawk, we use PUSH/PULL messaging pattern between scouts and HOME.

The messages are serialized using Google's protocol buffers[2]. For experimental missions, we emulate constricted bandwidth-environment using the traffic shaping tool, FireQOS [73].

As shown in the Figure 5.1, the HOME process launches three worker processes: ADMIN, INBOUND, and OUTBOUND. The ADMIN process transmits control messages from HOME to participating scouts including how to configure the scouts for the mission. The INBOUND process collects results transmitted from the scouts. Finally, the OUTBOUND process is responsible for transmitting user labels back to the scouts. Depending on functionality Hawk API can be divided into four components:

- ADMIN to SCOUT API (A2S API)
- SCOUT to SCOUT API (S2S API)
- SCOUT to HOME API (S2H API)
- HOME to COORDINATOR API (H2C API)

Note, in this thesis we use the term "API" to refer to internal interfaces between various components in Hawk. The A2S API relays control messages from ADMIN to scouts. These control messages include mission configuration, start, stop, and so on. This API uses the REQ/REPLY messaging pattern. The configuration message sent via the A2S API specifies the type of training, selective transmission and retraining strategies to be used for the mission. It also transmits either the initial DNN model (M_0) or the initial bootstrap set which contains labeled images (L_0) used to train the initial model. During this configuration phase, we assume there exists good bandwidth between admin and scouts.

The S2S API is used to share knowledge of the learning between the scouts. This includes labeled results, or DNN model weights. We use REQ/REPLY messaging pattern for this API. The data selected for labeling are transmitted via S2H API. The transmitted results contain thumbnails if the training is done on scouts Otherwise, if training is performed on the cloud, full-resolution tiles are sent to HOME. These results also contain metadata needed to correctly convey the labels back to the relevant scout. The results labeled by the expert are sent to the COORDINATOR via the H2C API. Because of the low bandwidth between scout and home, we use a PUSH/PULL messaging pattern for S2H and H2C communication.

Figure 5.1 shows the sequence of API calls made during the labeling of results in Hawk. The configuration file from home is sent to the ADMIN, which constructs the configuration message. This message is transmitted to the scouts via `a2s_configure_scout` call. When a scout receives the configuration message, it initializes all modules needed for the mission which includes training, processing, selective transmission, and so on. The configuration message also includes the initial bootstrapping examples or the initial trained model. The ADMIN waits for the reply from all the participating scouts. On receiving responses from all the participating scouts, the ADMIN calls `a2s_start_mission` to begin the mission. In some cases, there may be a significant delay between mission configuration and mission start to allow for the probes to reach the correct location of exploration. The model processing the incoming tiles and the selective module selects a small subset for transmission. These tiles are published by the scouts using the `s2h_send_tiles` method. The INBOUND process receives these tiles along with their meta-data and saves them to the local file system. The OUTBOUND process transmits the labels provided by the user to the COORDINATOR. The COORDI-

NATOR calls `s2s_get_tile` to fetch the tile content from the parent scout if the label is a positive. It then invokes `s2s_add_tile_and_label` to distribute the labeled tile along with its content to other scouts. This continues till the end of the mission. The ADMIN calls `a2s_stop_mission` to explicitly stop the mission. Post mission, the ADMIN makes the `a2s_get_post_mission_archive` call to collect the model generation along with logs from the scouts. In cases where the mission terminated with probe loss, this call may never be called.

5.2 Hawk Workflow

Using Hawk, we aim to construct DNN models that can detect instances of a rare target with minimal human supervision. As mentioned in the previous section, the scouts are configured with the mission parameters which include the weight of the initial DNN model (M_0) or a small set of labeled images (L_0). The learning and data selection of the Hawk pipeline in scouts is formalized in Algorithm 2.

Algorithm 2 Hawk Learning Algorithm on Scout

$M_0, L_0, B, \cdots \leftarrow$ **configure_scout**()
if M_0 is present **then**
    $M_0 \leftarrow$ **load_model()**
else if L_0 is present **then**
    $M_0 \leftarrow$ **train_model()**
end if
start_mission()
$D \leftarrow$ incoming data
$T \leftarrow$ **split_tiles**(D)
$i = 0, L = L_0$
repeat
    $P_i = \{\}, S_i = \{\}$
    if $|P_i| < B$ **then**
        $P_i \leftarrow P_i \cup$ **infer_data**(M_i, T)
    end if
    $S_i \leftarrow S_i \cup$ **select_data**(P_i)
    $L \leftarrow L \cup$ **get_labels**(S_i)
    if $|L| > R_i$ **then**
        $M_{i+1} \leftarrow$ **train_model()**
        $R_{i+1} \leftarrow$ **retrain_condition()**
        $i = i + 1$
    end if
until stop_mission()

} in parallel in all scouts

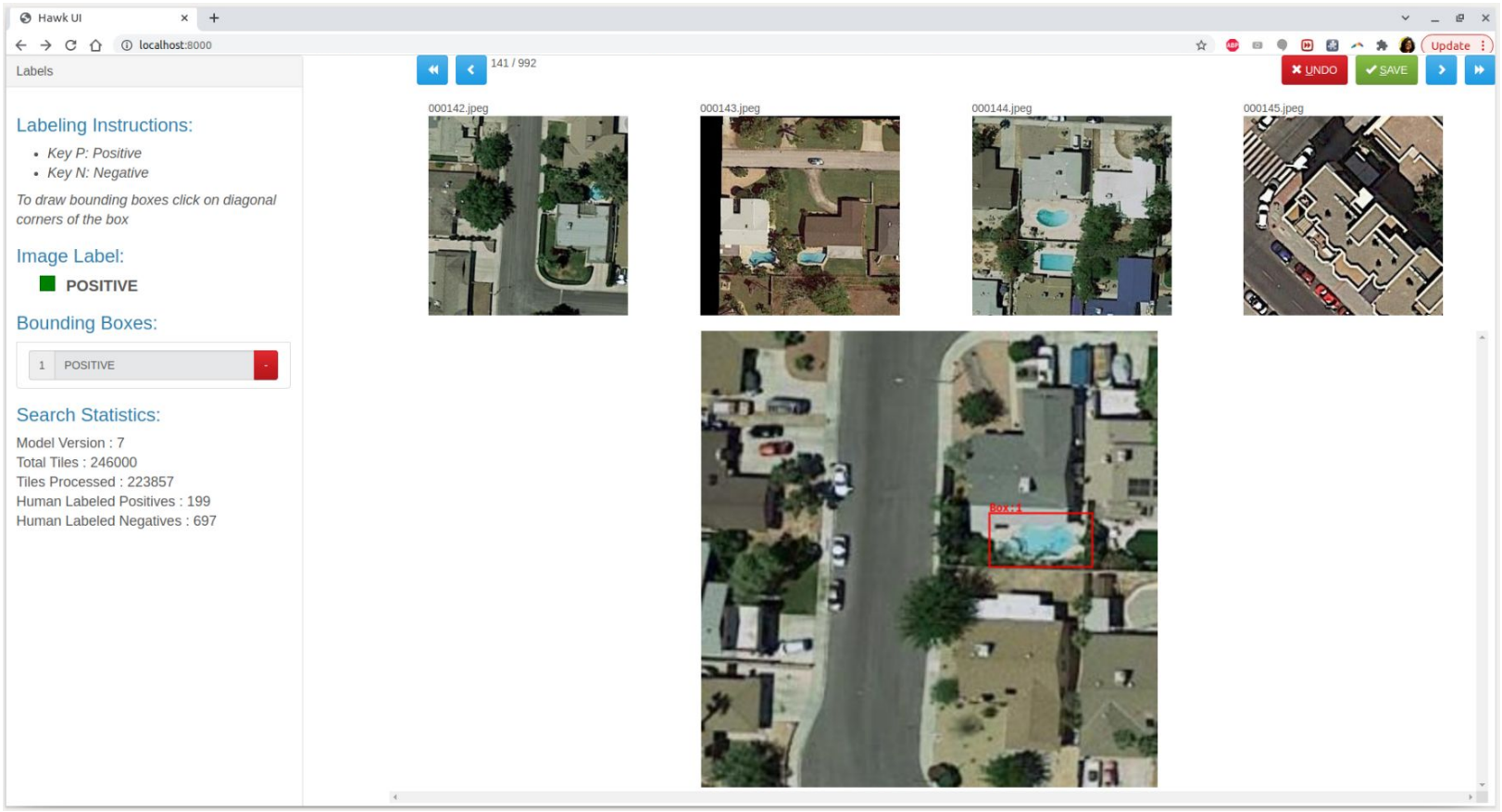


Figure 5.2: Screenshot of Hawk Labeling GUI

Once the scout is configured, it initializes the DNN model and other components needed for the mission. It then waits for the `start_mission` from home to start processing the incoming data, D. Each high-resolution incoming data frame is broken up into small tiles T. The tile size is a tunable parameter, it also depends on the learning algorithm used in the mission. For example, if image classification using ResNet50 is chosen, the tile size is 256x256 pixels. The tiles are then inferenced in small batches using the current model. After processing B tiles, Hawk's selection policy selects a small subset of tiles, S_i, are selected to be transmitted to the expert for labeling. The user sends back labels to the scouts, these labeled results are added to the training set, L. When the number of labeled tiles reaches a threshold, R_i, training of a new model is triggered. The current model is immediately replaced by the newly trained model. Inferencing, data selection, labeling, and training happens concurrently on Hawk. Hawk aims to improve the quality of the model over multiple rounds of labeling and training. Learning continues until it reaches a set `stop_criteria` or is explicitly stopped by the admin. The mission stop criterion is conditioned on collecting a certain number of positives or after a certain mission time has elapsed.

5.3 Hawk Labeling Interface

To the labeler, Hawk appears to just be a simple GUI. All of the complexity of training, inferencing, and bandwidth adaptation are hidden from the labeler. The labeling interface is a simple web application implemented using JavaScript. The interface can be viewed using any web browser; it has been tested on Chrome and FireFox. Using the UI the user can provide image tags or bounding box annotations to incoming result tiles.

Figure 5.2, provides a screenshot of the labeling interface. As mentioned earlier in Section 5.1, the INBOUND process collects image thumbnails and metadata of tiles transmitted from the scouts. These results are stored in the local file system. The expert may choose to label some of the results. As seen in the screenshot, we also provide a preview of some of the subsequent tiles. Using the UI, the expert can mark a result as a positive or a negative, he may also provide additional details such as bounding boxes. Results labeled by the expert are stored in the file system as well as transmitted by the OUTBOUND process to the scouts. These labels are added to the training set in the scouts for future learning. The GUI provides controls for tunable server parameters. Since Hawk is designed to be used by domain experts who are not trained in machine learning, the default settings of these parameters are sufficient for satisfactory operation. When multiple scouts are involved, their result streams are merged in temporal order as they are received. Internal bookkeeping ensures that every label is conveyed to the scout that generated that result.

5.4 Modes of Operation

Real use cases of Hawk typically start out with very few positive instances of the target class. Bootstrapping from these few examples to many more is, of course, the whole point of Hawk. An important implication is that there can be no stable test set against which to evaluate the

quality of models as they evolve in a Hawk mission. This makes it impossible to provide classic measures of quality of a classifier such as precision, recall, AUC (area under precision-recall curve), and ROC (receiver operating characteristics) curve. This is not a limitation of Hawk or of our evaluation methodology, but is an intrinsic constraint when no labeled test set is available.

In lieu of rigorous metrics of quality, weaker indicators must suffice. The number of true positives discovered, relative to the number of data items examined by the user is one such indicator. The rate of discovery of true positives per unit time, also known as *productivity,* is another. Later in a session, more time is spent confirming presumed-positive labels rather than correcting them. These indicators correspond to expert-perceived improvement in quality and correlate with precision, but give no indication of recall. The expert has no idea how many instances have been misclassified and not presented for labeling. Short of manual review of Hawk's discard pile, we cannot estimate this quantity. The missed instances are thus an *unknown unknown* rather than a *known unknown.*

We refer to the above mode of Hawk operation as *production mode.* In addition, Hawk also offers a *research mode* of operation for experimental investigations in machine learning and computer vision. This mode is enabled when a labeled test dataset is provided during the configuration phase. Calls to the method `get_test_result` during the experimental mission then return classic metrics relative to that test set. If no test data set is provided, Hawk operates in production mode, and only returns precision-related metrics.

Chapter 6

Evaluating Hawk

In the previous chapters, we described the working of Hawk and its various design choices. In this chapter, we evaluate how effective Hawk learning is in discovering instances of a novel target from incoming high-resolution image streams from a team of weakly connected scouts.
Specifically, we address the following questions:

- *In spite of extreme low bandwidth, can a scout discover many of the positives that it encounters during a mission?*
- *When additional bandwidth is available, is Hawk able to use it effectively to speed up the discovery of true positives?*
- *How close is Hawk to an ideal system?*
- *How does the site of model training affect the performance of Hawk?*
- *Is Hawk effective with bootstrap sets smaller than 20 positives, implying an even weaker Gen-0 model?*
- *Is Hawk model agnostic?*

We first describe the evaluation setup in Section 6.1. Then, we present the datasets we use to validate Hawk in Section 6.2. Finally, from Sections 6.4.1 to 6.4.6 to provide the results of our evaluations and provide answers to the above questions.

6.1 Evaluation Setup

Our evaluations use 7 physical machines as a scout swarm, each with a 6-core 3.6 GHz Intel Xeon processor, 32 GB memory, 4 TB disk storage for image data, and an NVIDIA GTX 1060 GPU. We use the FireQoS traffic shaping tool [73] to emulate the following bandwidth and RTT combinations on each scout's backhaul: 12 Kbps @4s, 30 Kbps @2s, and 100 Kbps @2s. Connectivity between scouts is 1 Mbps @200ms.

Before the start of the mission, we train the initial model using a labeled bootstrapping set. For all experiments unless specified, we use a bootstrapping set of 20 positives. This model is also referred as "Gen-0", and is uploaded to the scouts in the configuration phase. We configure the 7 scouts with mission parameters, such as revisit, training, and selection policies. As discussed in Section 4.7, we revisit the top 100 items from the discard pile when a new model

is deployed. For the retraining policy, we train a new model once 33% more positives than the previous training set have been labeled.

We use a TopK selective policy as discussed in Section 4.5. In this policy we only transmit the K top-scoring items after inferencing a batch of 1000 tiles. The goal is to fill, but not overfill, the end-to-end pipeline from incoming data to cloud. The bottleneck of this pipeline is the smallest of three values: (a) the ingest rate at a scout; (b) the inferencing rate on a scout; and, (c) the transmission rate at current bandwidth of an average-sized tile. To allow for some spare bandwidth, in our experiments we set K to be significantly lower than the bottleneck value: K=4 at 12 kbps; K=10 at 30 kbps; K=30 at 100 kbps.

During a mission, images from the unlabeled dataset are randomly delivered to the 7 scouts at an average rate of one every 20 seconds. This can be viewed as a low sampling rate of 0.05 frames per second. This is the maximum sustainable rate, given the modest hardware resources of a scout, the need to tile images, and the need to perform both inferencing and training. The dataset is striped across scouts at the input rate of 0.05 frames per second, and the experiment continues until the entire dataset has been delivered.

We emulate the human labeler by code that returns the ground-truth label for its input, after an optional delay for think time. We conduct three runs of each experiment, using different seeds for random number generation.

6.2 Validation Datasets

We validate Hawk on three publicly-available datasets from the domains of drone surveillance, planetary exploration, and underwater sensing. Each of these datasets was released within the past few years, and has been used in recent ML research publications in its domain.

As previously mentioned in Section 4.3, we split high-resolution images present in the dataset into smaller tiles. Most DNN models inference images at a lower-resolution, for example 256x256 pixels in the case of ResNet50. To remove ambiguity in the results, we discarded tiles where a labeled object straddles tiles. We also removed tiles in which the fraction of the area occupied by target instance is less than 1% of the tile.

6.2.1 Aerial Drone Surveillance (DOTA)

Dataset of Object deTection in Aerial images (DOTA) dataset [79] is a large-scale dataset for object detection in aerial images. Aerial object detection is useful for many applications such as remote object tracking and UAV navigation. Since its inception in 2018, the dataset has rapidly gained adoption in the computer-vision community as a challenging dataset for object detection and classification. The aerial images in the dataset are collected from multiple sensors and platforms such as Google Earth, GF-2 Satellite, and drones.

Popular image datasets such as ImageNet and MSCOCO are made of natural images taken from a human perspective where the target is often large and occupies a majority of the image. In contrast, the images in DOTA are captured from an aerial perception, so the targets in DOTA tend to be very small. There also exists a large variation in the scale, orientation, and shape of the targets in DOTA dataset.

The fifteen annotated classes in DOTA are: plane, ship, storage tank, baseball diamond, tennis court, swimming pool, ground track field, harbor, bridge, large vehicle, small vehicle, helicopter, roundabout, soccer ball field and basketball court. Figure 6.2 shows example images from DOTA classes.

DOTA dataset consists of 2806 labeled images, ranging in resolution from 800x800 pixels to 4000x4000 pixels, with a bias towards higher resolution. All images are broken into tiles of size 256x256 pixels, yielding a mission dataset of 252,231 usable tiles. During a mission, all the tiles from a single image in the mission dataset are randomly delivered to one of 7 scouts at an average rate of one every 20 seconds.

6.2.2 Planetary Exploration (HiRISE)

Images collected by the High Resolution Imaging Experiment onboard the Mars Reconnaissance Orbiter form the basis of the HiRISE dataset [19]. There are 73,031 labeled images, each cropped to 227x227 pixels. These images help in classifying Martian landmarks, such as craters and dunes.The eight target classes in the dataset are "Bright dune", "Crater", "Dark dune", "Slope streak", "Impact ejecta", "Spider", "Swiss cheese", and "Other".

We created a derived dataset by reducing the number of images of each class to achieve a 0.1% target base rate. We also reserve 10% of the dataset as a held-out test set. During a mission, 100 tiles in 20 seconds are delivered at random to each of the 7 scouts. Figure 6.3, shows example images of four target classes from this dataset.

6.2.3 Underwater Sensing (Brackish)

The Brackish dataset [51] is a publicly available dataset containing labeled images of marine animals in a brackish strait with varying visibility. The six target classes in the dataset are big fish, small fish, starfish, shrimp, jellyfish, and crab.

There are 14,518 images of 1080p resolution. We split each image into tiles of size 256x256 pixels, with an overlap of 50 pixels and remove tiles in which the size of the target instance is less than 20x20 pixels. The resulting derived dataset has 563,829 tiles. We created the dataset such that each target class has a base rate below 0.1%. During a mission, 100 tiles in 3 seconds are delivered at random to each of the 7 scouts. Figure 6.4, shows example frames from the dataset.

6.2.4 Experimental Notation

As previously discussed in Section 6.1, all our experiments begin with a model pre-trained on ImageNet and then trained via transfer learning on a certain number of bootstrapping TPs of the target class. This crude model is referred to as "Gen-0." The unlabeled mission dataset is striped across scouts at the specified input rate for the dataset, and the experiment continues until the entire mission data has been delivered. This takes roughly an hour to an hour and a half, depending on the dataset. We compare Hawk to three other alternatives, and the results use the following notation:

- *Hawk-xx-yy:* Hawk at a bandwidth of *xx* kbps, starting from a Gen-0 that was bootstrapped from *yy* TPs.

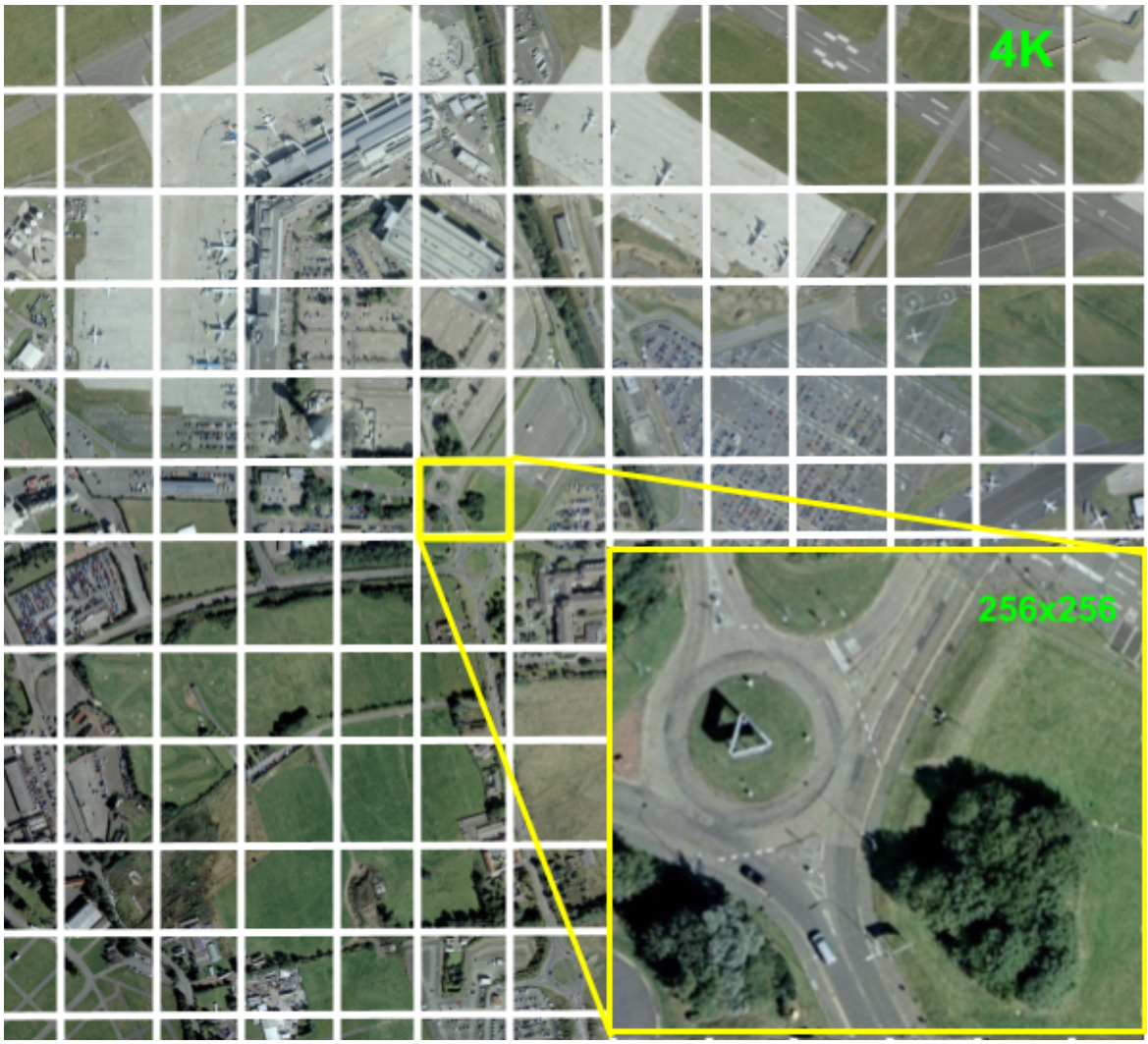
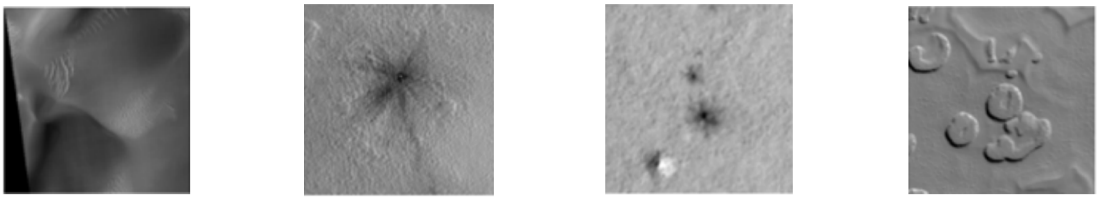


Figure 6.1: An example 4K frame and a tile containing target Roundabout (DOTA)

(a) Roundabout (TPs=336) (b) Swimming Pool (TPs=335) (c) Large Vehicle (TPs=357) (d) Airplane (TPs=350)

Each of the above examples is a small part of a 256x256 tile from a large high-res DOTA image. Ground truth bounding boxes are also shown. There are 252,231 unlabeled tiles in the mission dataset.

Figure 6.2: Examples of DOTA Target Classes



(a) Dark Dune (TPs=64) (b) Impact Ejecta (TPs=64) (c) Spider (TPs=64) (d) Swiss Cheese (TPs=64)

Each of the above examples is a 227x227 pixels in size. There are 73,031 unlabeled tiles in the mission dataset.

Figure 6.3: Examples of HiRISE Target Classes

(a) Starfish (TPs=370) (b) Shrimp (TPs=564) (c) Small Fish (TPs=564) (d) JellyFish (TPs=584)

Each of the above examples is 256x256 pixels in size. There are 563,829 unlabeled tiles in the mission dataset.

Figure 6.4: Examples of Brackish Target Classes

- *NoLearn-xx-yy:* same as *Hawk-xx-yy*, except that no learning is done to improve Gen-0.
- *BruteForce-xx:* at *xx* kbps, a DNN that uses the same architecture as Hawk but is trained offline in the cloud using unlimited resources and a fully-labeled version of the same dataset used in the mission. This model is an asymptotic limit towards which Hawk can aspire. Though trained on all TPs in the mission data, the BruteForce is not a perfect model. Thus, transmission of some FPs and omission of some TPs can occur during the mission.
- *Oracle-xx:* an oracle at *xx* kbps. The oracle is a perfect model that returns ground truth scores for all inputs, and a positive instance is transmitted for labeling as soon as it is encountered in the input stream.

6.3 Learning Strategy

The learning in Hawk allows a flexible declarative expression of how to train models. As discussed earlier in Section 4.6, Hawk rapidly learns models on newly labeled data to improve the quality of the model. There are two ways to perform transfer learning. Figure 6.5(a) shows the two approaches for the YFCC-derived task. pre-trained DNN as a fixed feature extractor and then train a linear classifier on top of it. Second, unfreezing some of the top layers of the pre-trained model and then re-training the model on new data with a low learning rate. The first approach is a shallow transfer learning strategy called `Strategy-SVM`, which does SVM training using feature vectors from a DNN pre-trained on ImageNet [59]. This allows us to learn models us-

ing newly labeled data in a fraction of a second even on scout hardware. A drawback of this approach is the diminished model accuracy because the weights of the DNN are not trained on the examples of the novel target. This can be solved by deep transfer learning, in which we unfreeze some of the top layers of the pre-trained model and then re-train the DNN model on the new data. This, however, is a slow process and may take hundreds of seconds. Thus, we use a hybrid approach called `StrategyDNN`, where we do less frequent releases of fine-tuned DNN models (slow training), interspersed with SVM releases (fast training). As seen in Figure 6.5(a), the rapid initial model improvement from SVM training helps both strategies. However, beyond a certain point, further improvement is only possible through DNN fine-tuning.

The use of a cascade SVM only works when there is a significant structural similarity between the phenomena being explored and the training set on which the DNN was trained. This is illustrated by Figure 6.5(b), which shows model evolution on DOTA data that has low structural similarity to the images in ImageNet dataset which were used to bootstrap Hawk. Because of low structural similarity between DOTA and ImageNet, there was no significant improvement using SVMs. Thus, for the experiments in this chapter, we use `StrategyDNN` without SVM training.

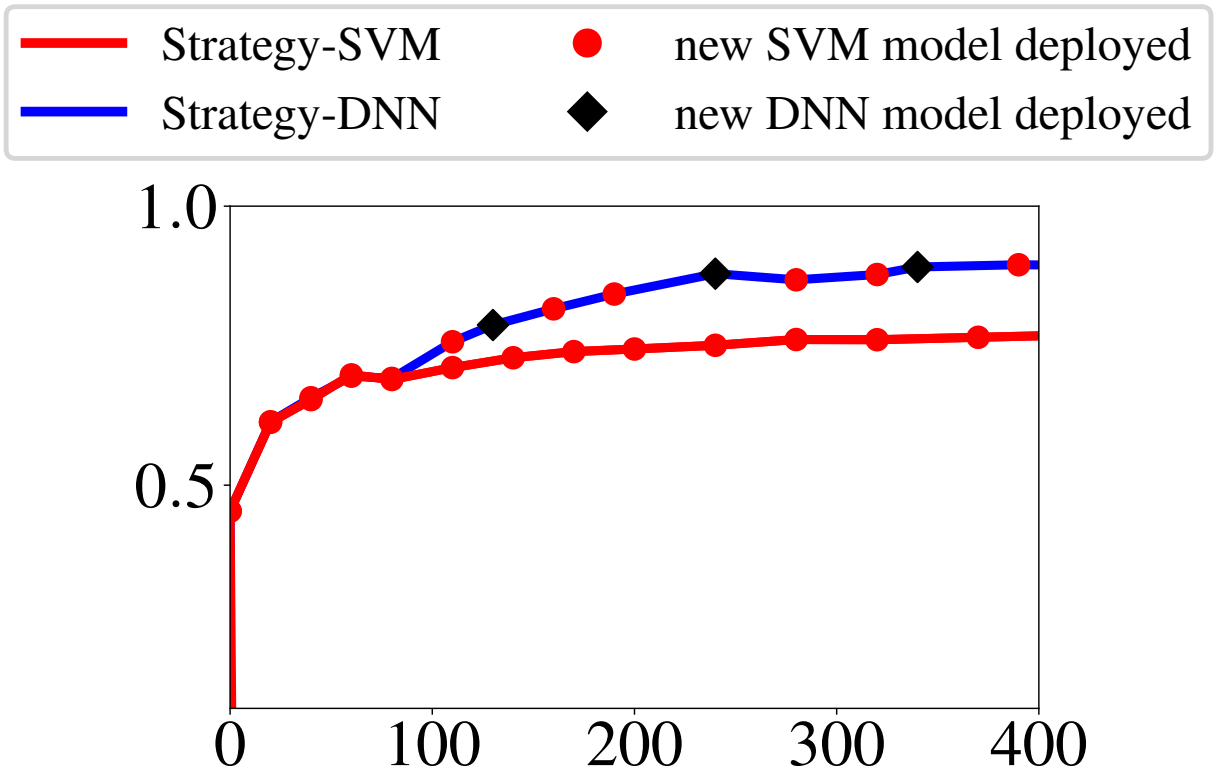
For all experiments, we use a ResNet-50 model that is pre-trained on ImageNet as "Gen-0" model. In Section 6.4.6, we evaluate the performance of Hawk using other DNN model architectures. Residual Networks (ResNet) introduced by He et al.[21], utilize "shortcut connections" in the neural network to reduce optimization difficulties. There are different versions of ResNets depending on the number of layers, ranging from 18-layers to 152-layers. In Hawk, we use the version of ResNets with 50-layers, as it provides good model accuracy with reasonable compute demand.

6.4 Evaluation

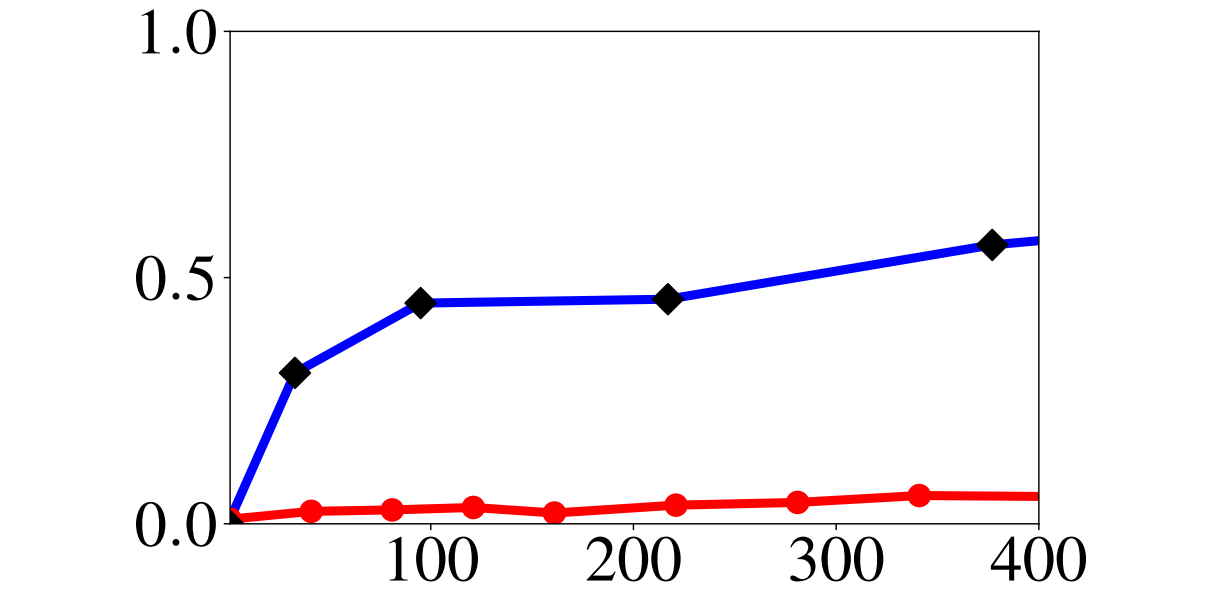
6.4.1 Bandwidth-Frugal Model Evolution

In this Section, we analyze the effectiveness of Hawk for an extremely low backhaul bandwidth of 12 kbps and training done on the scout. For the results presented in this section, we perform learning using a ResNet50 DNN model. We analyze the performance using other DNN models, namely YOLO-v5 and EfficientNet-B4 in Section 6.4.6. All results presented are the average of three runs using different seeds for a random number generator.

Table 6.1 shows the evolution of models for the class "Roundabout" in the DOTA dataset. This corresponds to the timeline shown earlier in Figure 4.6(b). The tables show the evolution of models ("Gen-0", "Gen-1", ...) during a mission. Each row of the table gives the state of the mission when a new model generation is installed on a scout. The first column of the table gives the generation number of the models on the scouts. The second column gives the approximate time at which the model is installed on a scout. The exact moment of replacement may vary slightly at different scouts. Column 3 indicates the number of true positive tiles sent to the cloud for labeling. The fourth column gives the total number of positive tiles encountered so far during a mission. Column 5 gives the total number of tiles transmitted to the cloud for labeling, which includes true positives and false positives. The last column gives the total number of tiles



(a) YFCC-derived dataset



(b) DOTA dataset

X-axis is number of seconds and Y-axis is AUC of model. Initial model was ImageNet-trained ResNet-50 DNN. The markers on the plot indicate when a new learned model is installed.

Figure 6.5: Hybrid Learning and Structural Similarity

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	277 (54)	8 (1)	19 (3)	47 (16)	17759 (713)
2	530 (59)	19 (1)	39 (4)	103 (16)	32797 (556)
3	768 (55)	28 (3)	59 (4)	152 (17)	42859 (624)
4	1168 (115)	50 (2)	95 (5)	241 (34)	65116 (703)
5	1579 (63)	72 (2)	126 (8)	335 (12)	94186 (433)
6	2201 (57)	97 (5)	165 (5)	455 (15)	124783 (542)
7	3010 (44)	137 (3)	218 (5)	604 (14)	166576 (366)
8	3529 (97)	187 (5)	283 (8)	772 (31)	207074 (676)
	4149 (32)	219 (6)	336 (0)	923 (3)	252231 (0)

DOTA Class: **Roundabout**
Column 1 corresponds to model number (Gen-0, Gen-1, Gen-2, . . .).
Column 2 corresponds to point in mission when new model was installed.
Columns 3–6 give totals across all 7 scouts.
Figures in parentheses are standard deviations from three runs.

Table 6.1: Model Evolution at 12 kbps (Training on scout)

processed so far during a mission.

At the start of the mission (time = 0 s), Gen-0 is initialized on the scouts. The table shows that at roughly 277 seconds into the mission, Gen-0 is replaced by model Gen-1 on all scouts. Since the model for each scout is trained independently on that scout, the exact moment of replacement may vary slightly across scouts. That variance, plus the use of a different random seed for each run, leads to the standard deviation shown (54 seconds). Gen-1 is replaced by Gen-2 at 530 seconds, and so on. The mission ends when mission data is exhausted.

By the time Gen-1 replaces Gen-0, Hawk scouts have discovered 8 TPs (Column 3) out of the 19 that they have encountered so far (Column 4). The 11 positives they have missed is indicative of the poor recall of Gen-0. The last two columns show how frugal Hawk is in bandwidth usage. Across all scouts, only 47 tiles (Column 5) have been transmitted for labeling, out of 17,759 encountered so far (Column 6). Gen-1 is replaced by Gen-2 at roughly 530 seconds when scouts have discovered about 19 positives, and so on. These general observations continue throughout the mission. By the end, 219 TPs out of 336 are discovered, with the scouts only transmitting 923 tiles out of 252,231 which is less than 0.4% of the number of tiles processed on the scouts. This shows how frugal Hawk is in its selection of tiles for transmission. In this case, Hawk was able to find 65% of the positives it encountered in the data stream.

We represent these results in a more compact format, the curve for Hawk-12-20 (red) in Figure 6.6 summarizes the data in Table 6.1. The circular markers correspond to models Gen-1, Gen-2, etc. in Table 6.1. The X-axis value of a symbol gives the time into the mission when that model was installed (Column 2 of Table 6.1). This Y-axis value corresponds to the total number of TPs discovered so far in the mission (Column 3 of Table 6.1). The curve for Oracle-12 (black) in Figure 6.6 gives the ground truth: i.e., the number of TPs encountered by the 7 scouts until that point in time (i.e., Column 4).

Figures 6.7 to 6.22 confirm similar model evolution for other DOTA classes. Table 6.4, summarizes the performance of Hawk across the other 14 classes in DOTA. For all classes, Hawk is effective is finding positives from the stream even for bandwidth as low as 12kbps. Figures 6.28 and 6.33 confirm that it also holds for the HiRISE and Brackish data sets. Across all datasets and classes, Hawk is typically able to find 50-60% of the TPs encountered. Its lowest success rate is 49% (class Starfish in dataset Brackish) and its highest is 73% (class Swiss Cheese in dataset HiRISE).

Since these results were all obtained at 12 kbps, it is clear that Hawk is very bandwidth-frugal. Our results give a strongly positive answer to the first question: *"In spite of extreme low bandwidth, can a scout discover many of the TPs that it encounters during a mission?"*

6.4.2 Ability to Use Additional Bandwidth

Holding all other variables constant, Table 6.2 shows the impact of increasing backhaul bandwidth to 30 kbps for the class "Roundabout" in the DOTA dataset. Because backhaul bandwidth is more plentiful, we transmit approximately 10 of the top tiles from each scout after processing 1000 tiles. Thus, a slightly higher pace of discovery of positives is maintained throughout the mission, leading to 16 additional TPs being discovered (235 rather than 219, out of 336). Because of the higher bandwidth, more tiles can be transmitted for labeling (1451 rather than 923 out of 252231, which is roughly 0.6%).

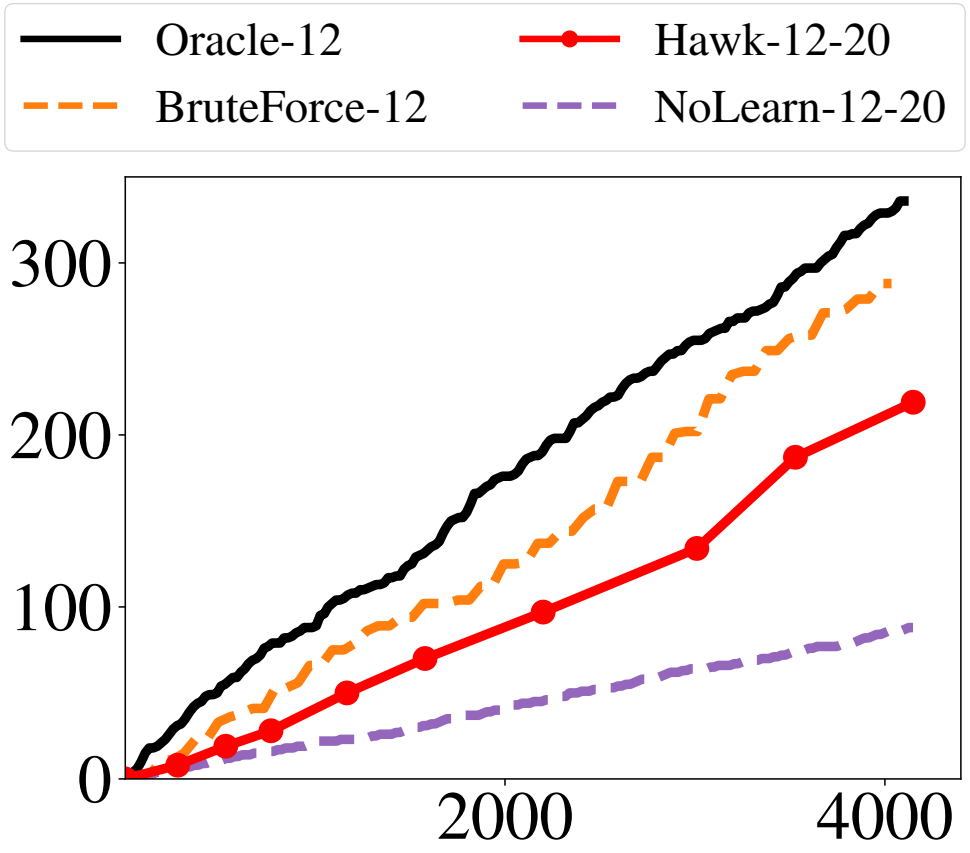


Figure 6.6: Model Evolution for DOTA Class: Roundabout (12 kbps, Scout Training)

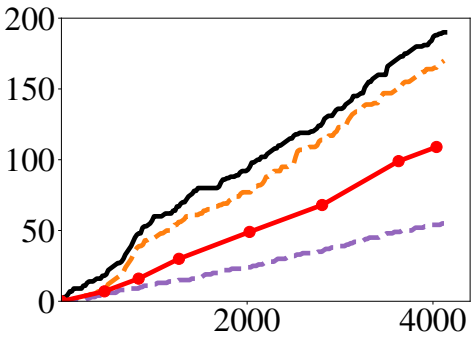


Figure 6.7: DOTA: Baseball Diamond

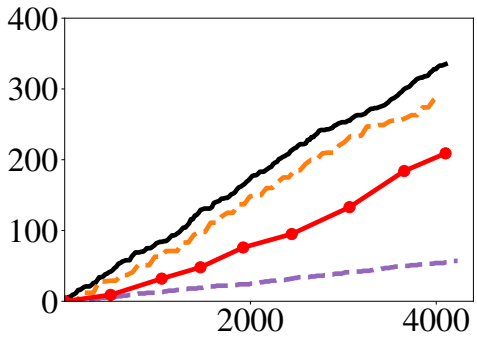


Figure 6.8: DOTA: Swimming Pool

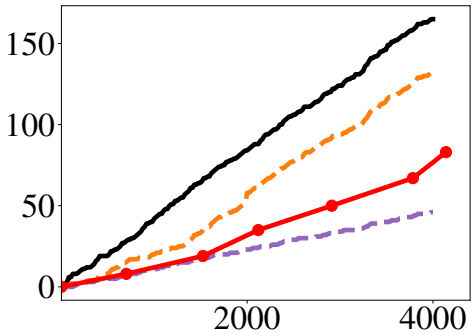


Figure 6.9: DOTA: Basketball Court

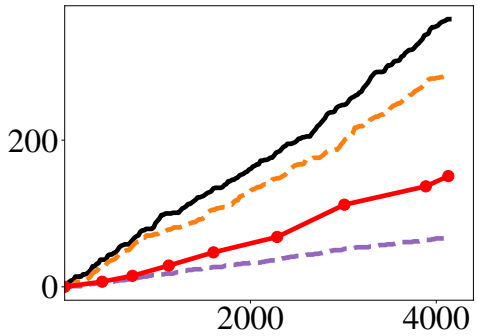


Figure 6.10: DOTA: Bridge

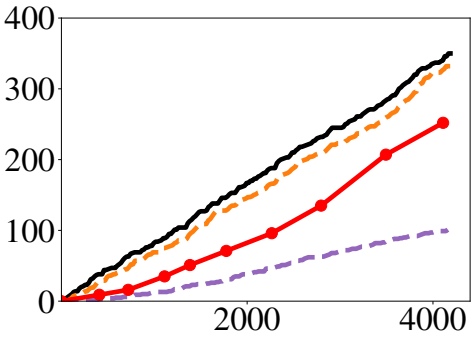


Figure 6.11: DOTA: Harbor

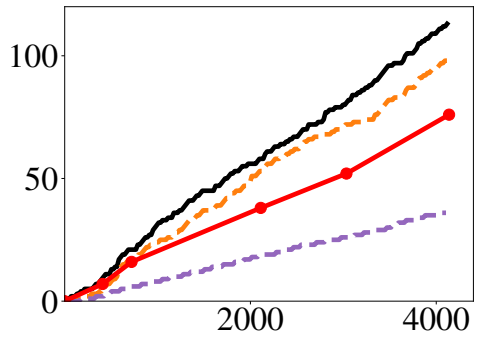
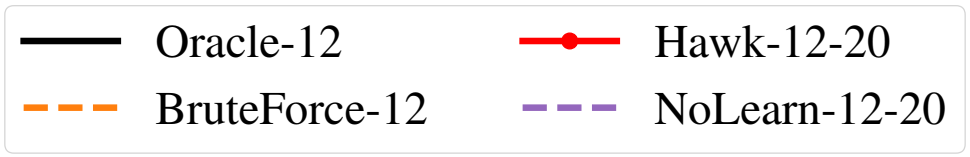


Figure 6.12: DOTA: Helicopter

Figure 6.13: Model Evolution (12 kbps, DOTA, Scout Training)



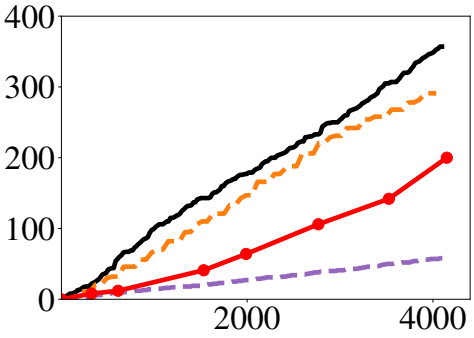


Figure 6.14: DOTA: Large vehicle

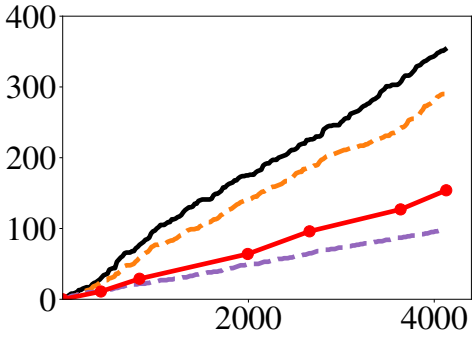


Figure 6.15: DOTA: Small Vehicle

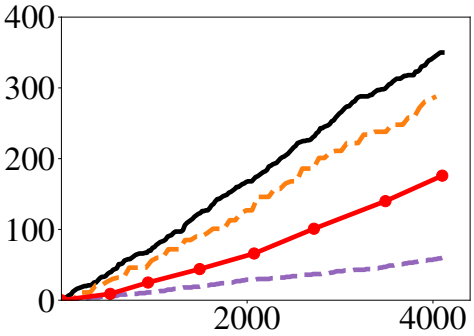


Figure 6.16: DOTA: Plane

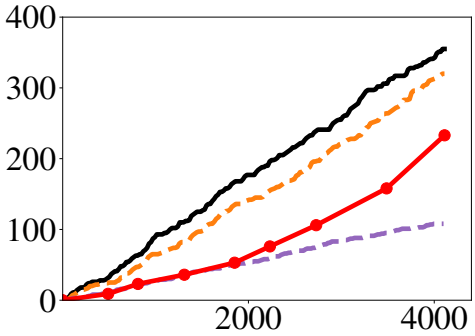


Figure 6.17: DOTA: Ship

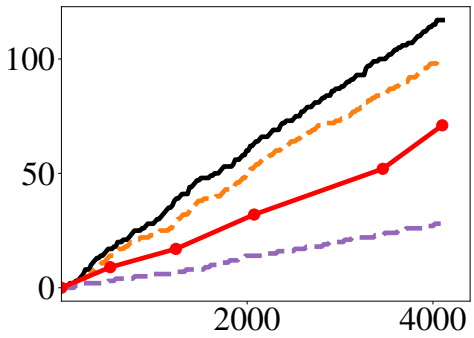


Figure 6.18: DOTA: Soccer Field

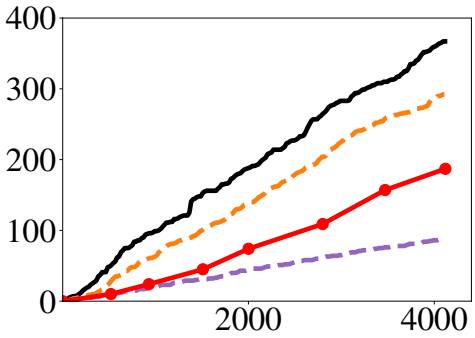


Figure 6.19: DOTA: Storage Tank

Figure 6.20: Model Evolution (12 kbps, DOTA, Scout Training)

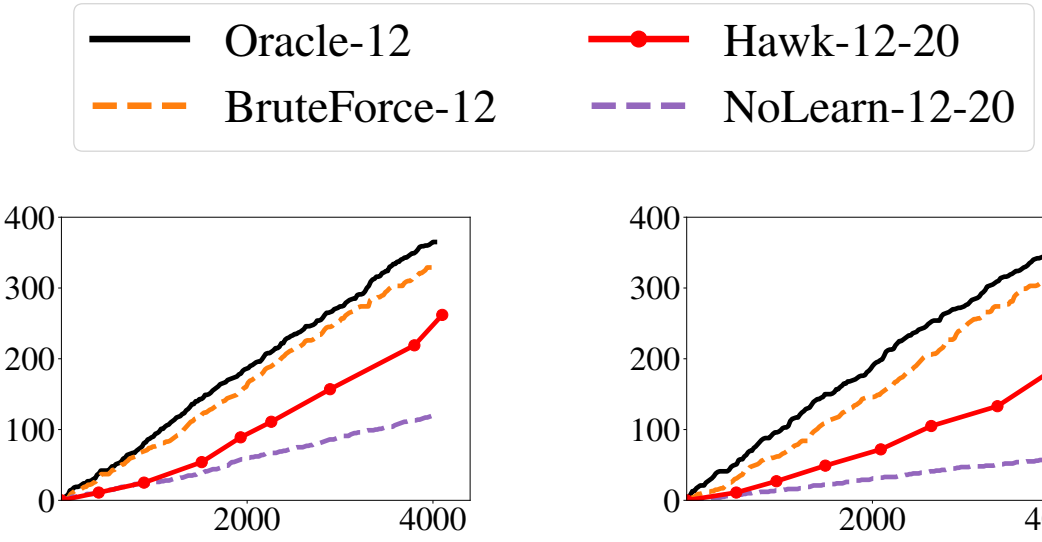


Figure 6.21: DOTA: Tennis Court

Figure 6.22: DOTA: Ground Track Field

Figure 6.23: Model Evolution (12 kbps, DOTA, Scout Training)

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	244 (49)	10 (2)	25 (2)	90 (35)	14502 (156)
2	413 (87)	20 (1)	43 (1)	160 (10)	21967 (240)
3	712 (105)	32 (2)	55 (5)	313 (65)	45424 (2161)
4	1063 (60)	48 (5)	90 (4)	540 (111)	68821 (1891)
5	1510 (104)	68 (2)	118 (7)	730 (159)	88620 (1086)
6	2264 (84)	99 (3)	156 (4)	1140 (294)	140508 (1378)
7	2884 (50)	135 (2)	207 (4)	1603 (185)	183283 (296)
8	3456 (125)	188 (3)	266 (2)	2063 (99)	221766 (1568)
	4143 (31)	235 (7)	336 (0)	1451 (44)	252231 (0)

DOTA Class: **Roundabout**
Format and notes identical to Table 6.1

Table 6.2: Model Evolution at 30 kbps (Training on scout)

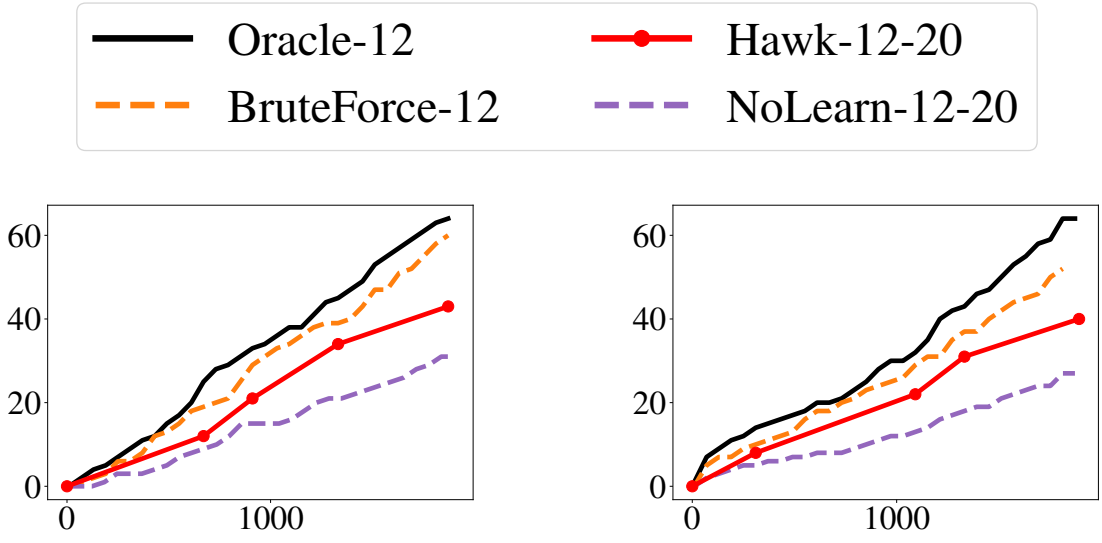


Figure 6.24: HiRISE: Dark Dune

Figure 6.25: HiRISE: Impact Ejecta

Figure 6.26: HiRISE: Spider

Figure 6.27: HiRISE: Swiss Cheese

Number of bootstrapping TPs = 20. X axis is time in seconds into the mission. Y axis is number of TPs discovered so far.

Figure 6.28: Model Evolution (12 kbps, HiRISE, Scout Training)

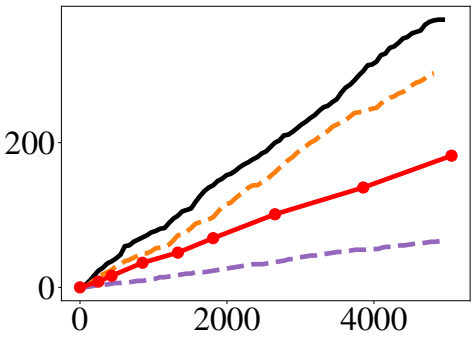


Figure 6.29: Brackish: Starfish

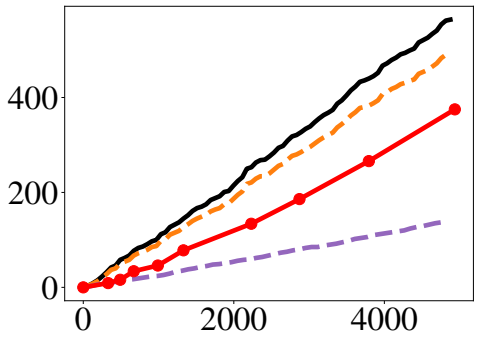


Figure 6.30: Brackish: Shrimp

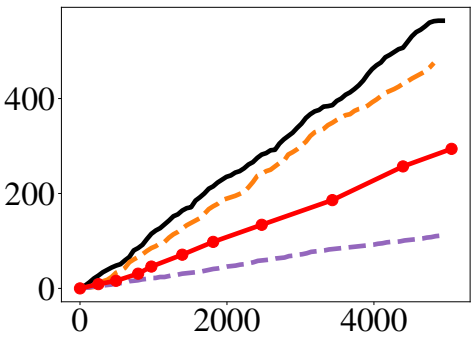


Figure 6.31: Brackish: Small Fish

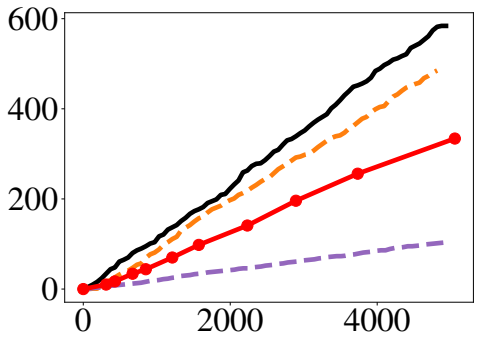


Figure 6.32: Brackish: Jellyfish

Number of bootstrapping TPs = 20. X axis is time in seconds into the mission. Y axis is number of TPs discovered so far.

Figure 6.33: Model Evolution (12 kbps, Brackish, Scout Training)

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	208 (11)	9 (0)	24 (2)	230 (173)	14824 (567)
2	374 (12)	24 (2)	40 (4)	520 (104)	21970 (875)
3	554 (14)	31 (4)	52 (4)	740 (121)	33633 (719)
4	1002 (61)	52 (3)	90 (7)	1570 (221)	64918 (1860)
5	1452 (68)	70 (8)	113 (5)	2453 (72)	89827 (2537)
6	2192 (42)	100 (3)	153 (2)	3674 (247)	136950 (3434)
7	2795 (57)	110 (4)	204 (2)	5207 (267)	179070 (1880)
8	3518 (60)	193 (7)	269 (6)	6235 (175)	219442 (2786)
	4147 (41)	246 (8)	336 (0)	7501 (128)	252231 (0)

DOTA Class: **Roundabout**
Format identical to Table 6.1

Table 6.3: Model Evolution at 100 kbps (Training on scout)

We observe similar results when the backhaul bandwidth is further increased to 100 kbps, as shown by Table 6.3. A further 11 TPs are discovered by the end of the mission (246 rather than 235, out of 336). The network bandwidth is more freely used, we transmit 7501 rather than 1451 tiles. Even at 100 kbps we transmit only a very small fraction of the total tiles processed, 7501 tiles out of a total of 252231 (a little over 3.0%).

The results for HiRISE and Brackish are consistent with that observed for DOTA dataset. Tables 6.4, 6.5, and 6.6, summarizes the performance of Hawk for the datasets DOTA, HiRISE, and Brackish respectively across varying backhaul bandwidth from scouts to the Internet. For all target classes, more positives are found with the increase in backhaul bandwidth. Thus, Hawk is able to effectively use the increase in network bandwidth to find more positives. In summary, these results strongly confirm a positive answer to the question: *"Can Hawk use additional bandwidth effectively?"*

6.4.3 Comparison to an Ideal System

In the results we compare with a "BruteForce" model. As explained earlier in Section6.2.4), a brute force model is a DNN model that was trained using all the mission data. Such a model cannot be implemented in practice because it assumes that (a) all mission data can be previewed in advance of mission start, and (b) there is sufficient bandwidth to transmit all mission data to the cloud for labeling and training. Yet, since it is a model that uses the same DNN architecture as Hawk, a comparison offers insights into how much improvement is possible in Hawk without improvement to the underlying DNN.

The gap between the curves labeled "Hawk-12-20" and "BruteForce-12-20" in Figures 6.6–

Target Class	Ground-Truth Positives	Positives found by				
		BruteForce-100	No-Learning-100	Hawk-12-20	Hawk-30-20	Hawk-100-20
Roundabout	336	296 (7)	198 (9)	219 (6)	235 (7)	246 (8)
Baseball Diamond	190	179 (5)	93 (3)	109 (7)	140 (12)	155 (13)
Swimming Pool	335	303 (6)	194 (4)	209 (8)	250 (5)	272 (9)
Basketball Court	165	149 (8)	79 (8)	78 (7)	104 (8)	121 (14)
Bridge	365	287 (10)	111 (5)	151 (6)	185 (9)	211 (12)
Harbor	350	332 (12)	218 (9)	252 (6)	288 (8)	313 (11)
Helicopter	113	99 (11)	36 (8)	51 (5)	76 (8)	88 (9)
Large Vehicle	357	327 (15)	161 (11)	200 (11)	217 (12)	264 (10)
Small Vehicle	353	290 (8)	125 (9)	154 (6)	242 (4)	275 (6)
Plane	350	328 (11)	126 (7)	176 (6)	213 (9)	253 (8)
Ship	355	324 (13)	249 (6)	263 (8)	295 (5)	329 (9)
Soccer Field	117	108 (10)	42 (8)	69 (4)	74 (7)	83 (9)
Storage Tank	367	320 (8)	121 (5)	167 (7)	220 (8)	251 (8)
Tennis Court	365	331 (12)	166 (4)	262 (7)	304 (6)	326 (9)
Ground Track Field	368	327 (14)	184 (7)	198 (6)	222 (9)	235 (11)

Table 6.4: Performance across DOTA Classes

Target Class	Ground-Truth Positives	Positives found by				
		BruteForce-100	No-Learning-100	Hawk-12-20	Hawk-30-20	Hawk-100-20
Bright Dune	64	63	42	48 (2)	51 (2)	56 (4)
Crater	64	60	45	42 (1)	49 (1)	55 (4)
Dark Dune	64	64	45	43 (1)	50 (2)	55 (3)
Impact Ejecta	64	60	41	40 (3)	45 (3)	51 (6)
Slope streak	64	62	38	45 (2)	49 (2)	54 (3)
Spider	64	58	44	45 (1)	50 (5)	53 (5)
Swiss Cheese	64	61	45	47 (2)	51 (2)	57 (3)

Table 6.5: Performance across HiRISE Classes

Target Class	Ground-Truth Positives	Positives found by				
		BruteForce-100	No-Learning-100	Hawk-12-20	Hawk-30-20	Hawk-100-20
Crab	600	558	231	396 (3)	462 (6)	511 (8)
Fish	600	549	224	327 (4)	434 (5)	488 (6)
Starfish	370	331	104	182 (2)	227 (3)	255 (4)
Shrimp	564	550	226	375 (8)	455 (5)	496 (7)
Small Fish	564	506	216	299 (4)	311 (9)	386 (6)
Jellyfish	584	527	215	334 (4)	387 (3)	440 (9)

Table 6.6: Performance Across Brackish Classes

6.33 shows the difference between the number of TPs found by the two approaches. By mission end, the gap is roughly 15-46% across all datasets and classes for a backhaul bandwidth of 12 kbps. The smallest gap is 13% for class Swiss Cheese of dataset HiRISE, and the largest gap is 46% for class "Bridge" of dataset DOTA. This gap can be attributed to the difference between fully supervised learning and the live learning strategy of Hawk.

The results in Figures 6.6–6.33 also illustrate the importance of learning during a mission. In each of these graphs, the curve labeled "NoLearn-12-20" shows the number of TPs that would have been discovered if Hawk did not implement continuous learning. The gap between the red and purple lines shows the importance of learning. In almost all cases, the number of TPs discovered in the NoLearn case is 50% or fewer than Hawk is actually able to discover. This justifies the need for Hawk. It also shows the importance of equipping scouts with hardware that is capable of doing DNN training. If inferencing using a static model was all that was needed, much weaker scout hardware would have sufficed. Training in the cloud allows use of weaker scout hardware, but Section 6.4.4 shows that it is much less bandwidth-frugal than training on the scout.

6.4.4 Cloud versus scout Training

As discussed in Section 4.4, Hawk is able to dynamically select between training on the scout and training in the cloud. The choice of site of training depends on various runtime factors such as end-to-end bandwidth, size of image tiles, and so on. In Hawk, the major disadvantage of cloud training is the need to transmit the resulting large model over low bandwidth. While model compression using a mechanism such as DeepIoT [82] has been very successful on some DNNs, we have shown earlier in Section 4.4 that it is much less effective on the kinds of DNNs typically used in Hawk. The achievable compression of 2–3 is helpful, but hardly adequate at 12 kbps for transmitting models that are tens of MB in size. At such low network bandwidth, far more aggressive compression is necessary to transmit these models to scouts. ResNet-50 models are 95 MB in size, with an aggressive compression of 10X, transmission size would be 9.5 MB and take roughly 100 minutes for a poor backhaul bandwidth of 12 kbps. This transmission time may be much longer than what is acceptable in a time-critical mission. Even if a compression technique to achieve 100X reduction in size were available, this would still require transmission of 0.95 MB. At 12 kbps, this would take nearly 10 minutes.

In this Section, we examine what the impact on Hawk would be if 100X and 10X model compression were possible. The results overstate the benefits of cloud-based training since they ignore the time required for model compression, and any associated loss of model accuracy. Yet, this deliberate handicapping of scout-based training is useful as an upper bound on the benefit of cloud-based training in Hawk.

Table 6.7 presents the results of cloud-based training followed by 100X model compression for a mission that is identical in all other respects to the mission presented earlier in Table 6.1. The bandwidth (12 kbps), dataset (DOTA) and class (Roundabout) are unchanged across the two experiments — only the training site is different. With cloud training, for a network bandwidth of 12 kbps even at 100X compression, it takes 10 minutes to transmit a ResNet-50 model of size 0.95 MB.

As a consequence, fewer models are installed during a mission — only 4 models in Table 6.7,

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	973 (94)	10 (3)	82 (6)	54 (30)	57148 (892)
2	1771 (108)	32 (7)	152 (11)	298 (84)	113045 (1764)
3	2531 (164)	66 (4)	223 (16)	450 (44)	160287 (3305)
4	3298 (156)	102 (13)	286 (21)	649 (47)	211825 (2500)
	4010 (15)	151 (14)	336 (0)	1009 (6)	252231 (0)

DOTA Class: **Roundabout**
Column 1 corresponds to model number (Gen-0, Gen-1, Gen-2, . . .).
Column 2 corresponds to point in mission when new model was installed.
Columns 3–6 give totals across all 7 scouts.
Figures in parentheses are standard deviations from three runs.

Table 6.7: Model Evolution at 12 kbps (Cloud-100x)

in contrast to 8 models in Table 6.1. On the plus side, inferencing is faster because scout hardware does not have to be shared between inferencing and training. Processing of mission data hence completes roughly 100 seconds sooner in Table 6.7 than in Table 6.1. Unfortunately, the net impact of these factors is not favorable for cloud training. Out of 336 TPs, Hawk is able to find only 151 in contrast to 219 (bottom rows of Tables 6.7 and 6.1).

At 30 kbps, cloud training with 100X compression fares better as shown in Table 6.8. It takes approximately 4.2 minutes to transmit the compressed model. At the end of the mission, we find a total of 269 positives as compared to 235 with scout training at the same bandwidth (Table 6.2). At higher bandwidth of 100 kbps, this improvement in recall persists. As seen in Table 6.9, we find 284 positives with cloud training versus 246 with scout training. Apart from increased compute power leading to longer training, the cloud also has access to all the human-labeled data leading to increased model accuracy. In scout training, we only share positives between the scouts.

We also tested cloud training using a reduced model compression of 10X. As expected, when 100X model compression is not feasible, cloud training fares much worse. At 12 kbps, model transmission with 10X compression takes so long that the results are meaningless. Matters improve as bandwidth rises. At 30 kbps, cloud-training finds 159 TPs (bottom row of Table 6.10), this is much lower as compared to the 235 TPS found during scout training (bottom row of Table 6.2). With increased bandwidth of 100 kbps, we find 263 positives (bottom row of Table 6.11) compared to 246 with scout training (bottom row of Table 6.3).

In Figure 6.34, we summarize the results for cloud vs. scout training for four classes in DOTA. As seen in the figure, when bandwidth rises, cloud training at 100X fares better than scout training. Similarly, we show the results for four classes of the HiRISE and Brackish datasets in Figures 6.35 and 6.36.

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	499 (30)	8 (5)	46 (4)	69 (8)	31379 (826)
2	859 (53)	22 (3)	74 (6)	299 (27)	52919 (1160)
3	1156 (80)	44 (5)	109 (9)	524 (31)	74392 (1898)
4	1935 (116)	62 (10)	165 (12)	802 (98)	121635 (2835)
5	2169 (149)	87 (8)	193 (13)	1067 (48)	138814 (2166)
6	2889 (222)	133 (5)	255 (16)	1396 (164)	181761 (3639)
7	3699 (357)	181 (10)	318 (18)	1786 (204)	233264 (3751)
	4019 (11)	269 (4)	336 (0)	2554 (9)	252231 (0)

DOTA Class: **Roundabout**

Format and notes identical to Table 6.7

Table 6.8: Model Evolution at 30 kbps (Cloud-100x)

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	262 (82)	7 (2)	25 (5)	271 (132)	18484 (288)
2	455 (149)	19 (4)	41 (4)	679 (219)	27150 (426)
3	636 (113)	37 (9)	56 (8)	1453 (282)	48416 (960)
4	936 (152)	51 (11)	90 (11)	2351 (197)	75859 (1764)
5	1297 (172)	84 (26)	130 (23)	3012 (174)	102039 (1295)
6	1746 (194)	115 (37)	174 (15)	3990 (157)	132468 (2367)
7	2390 (283)	160 (26)	233 (21)	5161 (217)	184297 (3305)
8	3330 (362)	229 (14)	286 (23)	6110 (869)	211825 (3915)
	4029 (29)	284 (11)	336 (0)	7593 (104)	252231 (0)

DOTA Class: **Roundabout**

Format and notes identical to Table 6.7

Table 6.9: Model Evolution at 100 kbps (Cloud-100x)

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	3080 (74)	7 (4)	273 (20)	70 (7)	198940 (305)
	4011 (7)	159 (13)	336 (0)	2446 (35)	252231 (0)

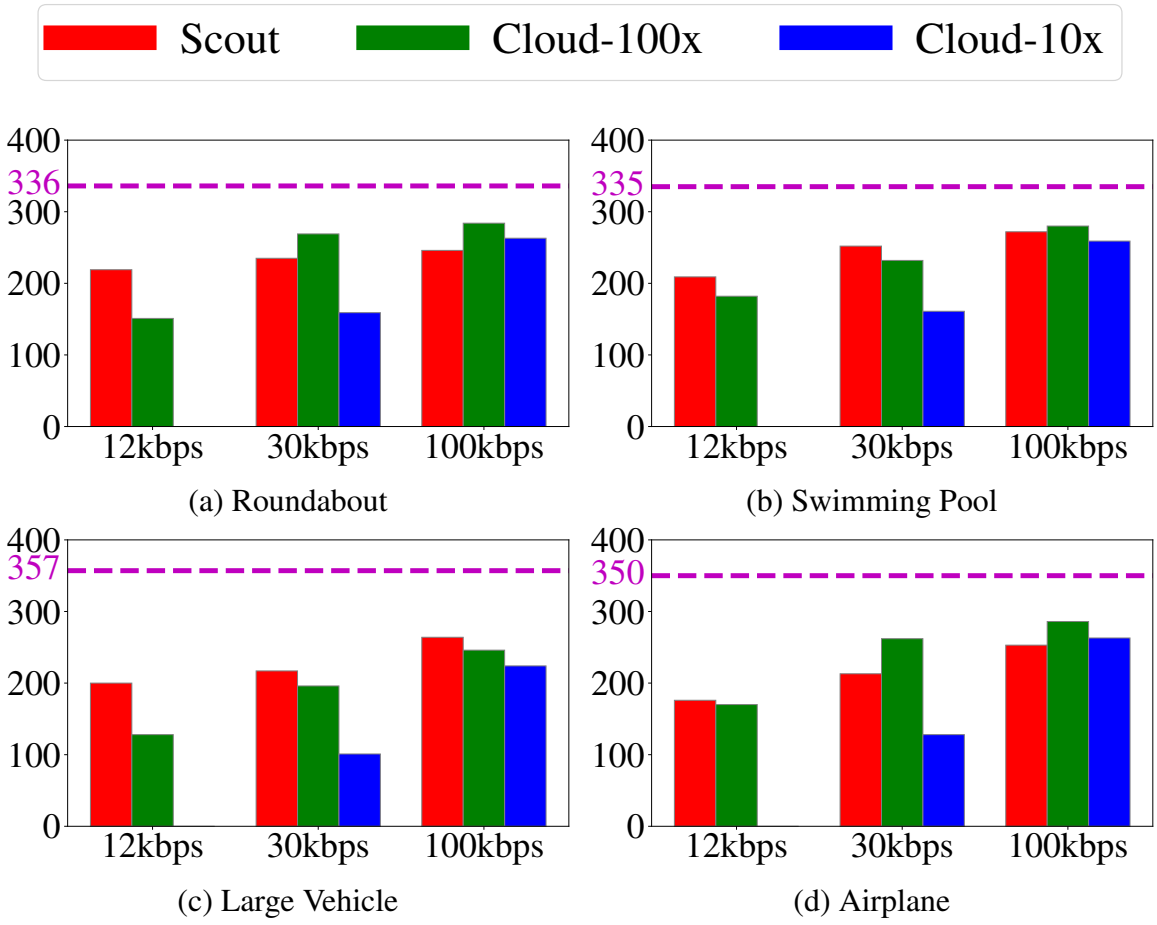
DOTA Class: **Roundabout**
Format and notes identical to Table 6.7

Table 6.10: Model Evolution at 30 kbps (Cloud-10x)

Gen	New model installed (seconds)	Positives discovered so far	Total Positives so far	Tiles transmitted so far	Tiles processed so far
0	0	0	0	0	0
1	998 (18)	7 (3)	94 (7)	211 (8)	65803 (1027)
2	1972 (22)	72 (4)	168 (12)	2031 (29)	125929 (1964)
3	2930 (57)	137 (13)	262 (13)	3751 (95)	186056 (3319)
4	3884 (32)	205 (15)	331 (18)	5502 (66)	246083 (3839)
	4019 (11)	263 (8)	336 (0)	7527 (41)	252231 (0)

DOTA Class: **Roundabout**
Format and notes identical to Table 6.7

Table 6.11: Model Evolution at 100 kbps (Cloud-10x)



(a) Roundabout

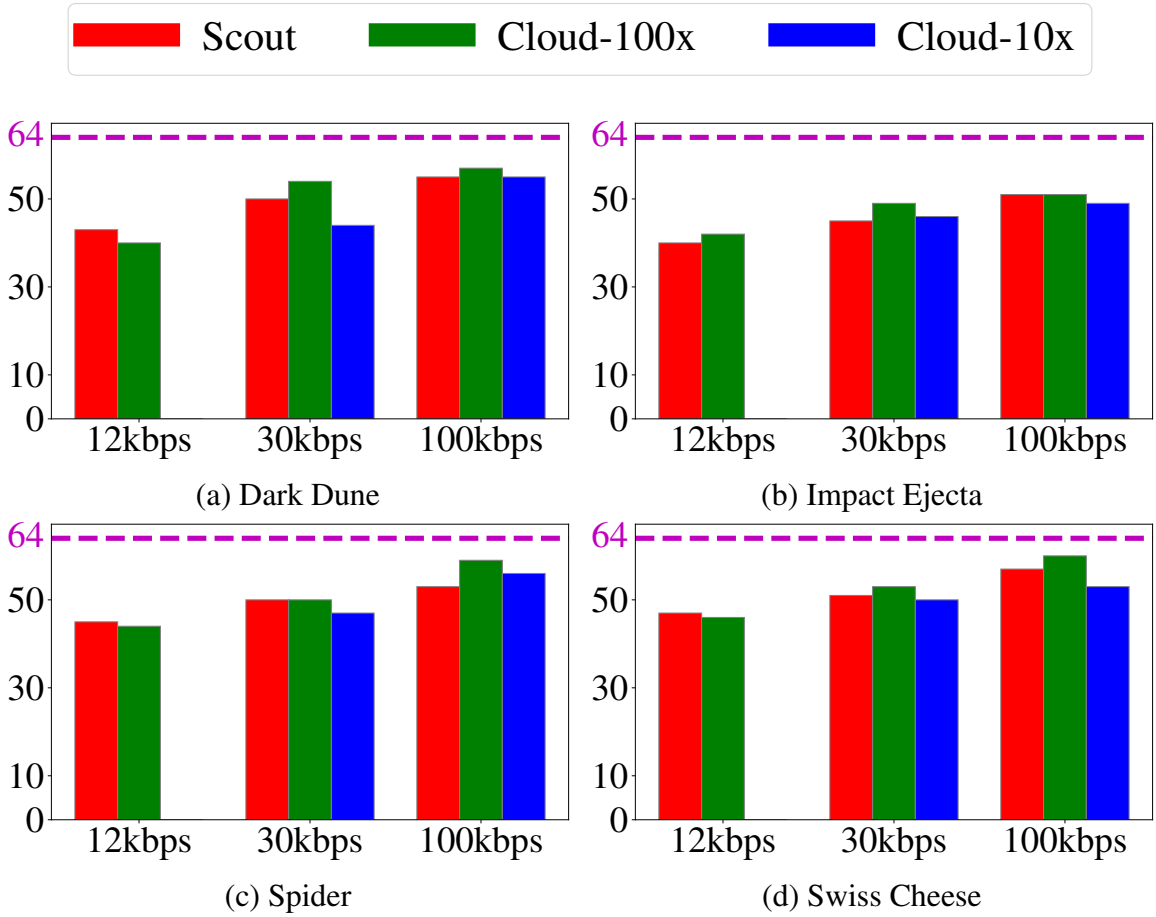
(b) Swimming Pool

(c) Large Vehicle

(d) Airplane

Y axis is number of TPs discovered.

Figure 6.34: Cloud vs. Scout Training (DOTA)



(a) Dark Dune

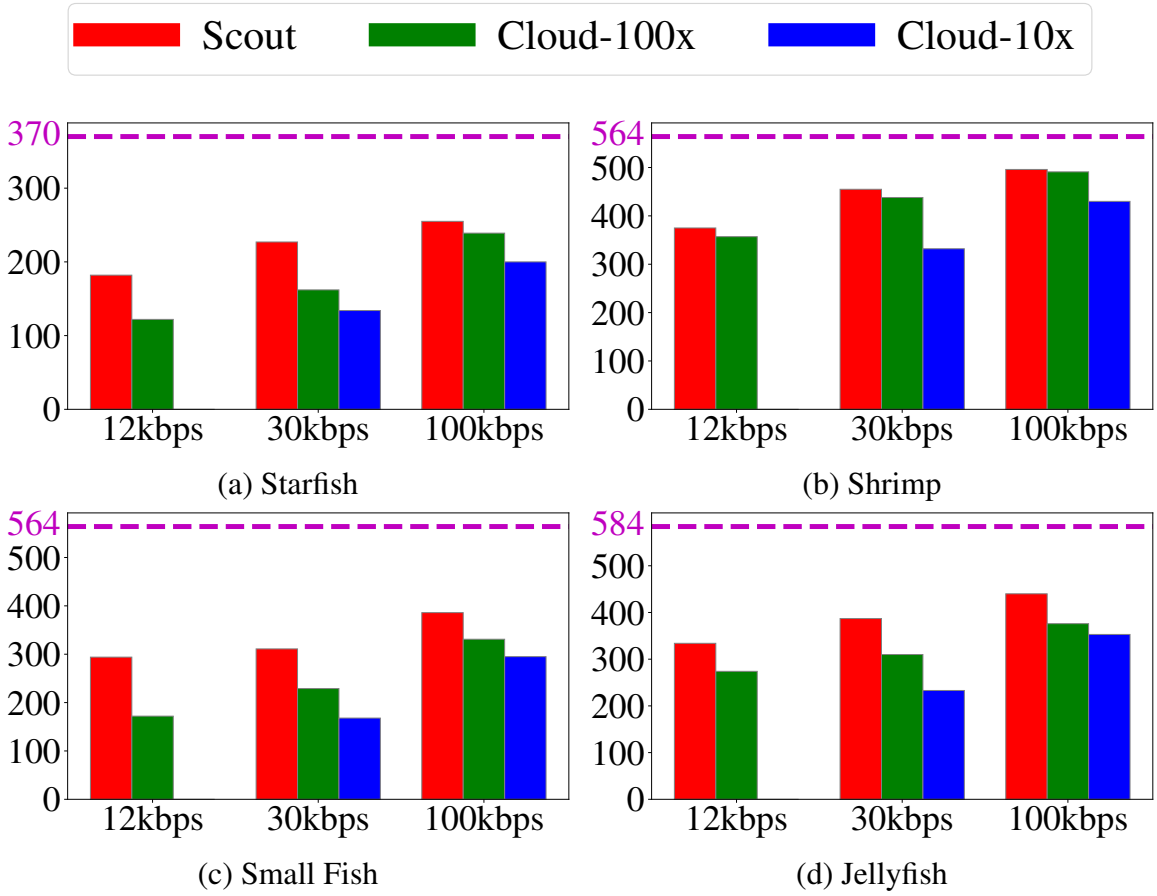
(b) Impact Ejecta

(c) Spider

(d) Swiss Cheese

Y axis is number of TPs discovered.

Figure 6.35: Cloud vs. Scout Training (HiRISE)



(a) Starfish

(b) Shrimp

(c) Small Fish

(d) Jellyfish

Y axis is number of TPs discovered.

Figure 6.36: Cloud vs. Scout Training (Brackish)

Parameter name	Description	Default Values
Initial Training Epochs	Number of training epochs for initial model	30
Rate of augementation	Rate of augmenting training set	4
Learning rate (LR)	Rate of model learning	0.01
LR warm-up epochs	Initial number of warm-up epochs	10
Learning rate schedule	Pre-defined LR schedule	cosine
Momentum	Percentage of gradient retained	0.9
Weight decay	L2 regularizer	1e-4
Optimizer	Name of optimization algorithm	sgd
Label Smoothing	Degree of smoothing	0.1
EMA momentum	Update momentum	0.6
Batch-size	Number of images in a batch	64
Image-size	Resolution of image feed	256
Layers to unfreeze	Number of model layers to train	3
Model architecture	Name of DNN model architecture	resnet50

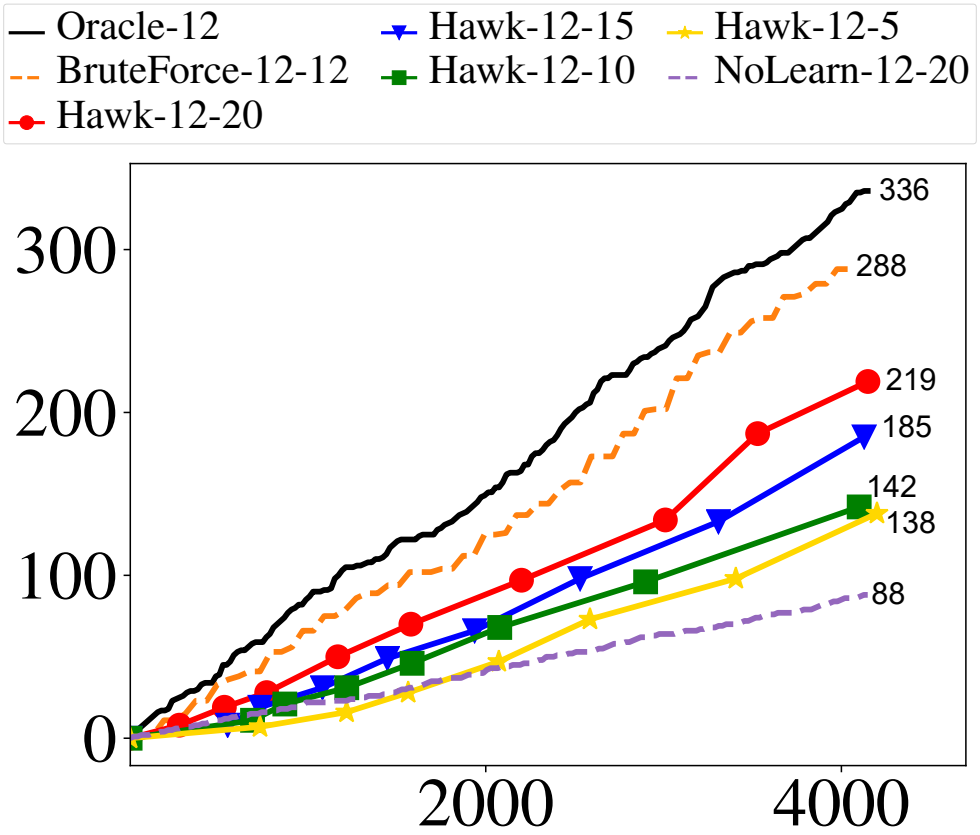
Table 6.12: Hyperparameter used for Section 6.4.5

6.4.5 Novelty of Phenomenon

From an ML viewpoint, the size of Hawk's bootstrap set implicitly defines the novelty of the phenomenon being explored. All results reported so far were obtained with a bootstrap set size of 20. In this section, we explore the performance of Hawk when fewer than 20 examples are used in bootstrapping Hawk models. We ask *"Is Hawk effective with even smaller bootstrap sets, implying an even weaker Gen-0 model?"*

Table 6.12, gives the hyperparameters for training Hawk models when fewer than 20 examples are used. Figure 6.37 shows how Hawk model evolution varies with bootstrap set size as it shrinks from 20 down to 15, 10 and 5. These results were obtained at 12 kbps for the Roundabout class of the DOTA dataset. The figure shows that bootstrap set size clearly impacts Hawk's effectiveness. As this size shrinks, Hawk's model evolution has a shallower slope and ends with discovery of fewer TPs. However, it is impressive that even with a bootstrap set size of 5, Hawk finds 138 out of 336 TPs by the end of the mission. This is in contrast to just 88 TPs found by Gen-0 without learning at a bootstrap set size of 20. As bootstrap set size is increased to 10, 15, and 20, Hawk finds 142, 185 and 219 TPs respectively.

As expected, the size of the bootstrap set affects the initial model accuracy and in effect the number of positives found at the end of the mission. For the class Roundabout, the AUC of the initial model at Hawk-12-20 is 0.18 which drops to 0.086 for Hawk-12-5. Lower model accuracy leads to misclassifying positives and longer time in retraining the next model generation. We see similar patterns for other classes of DOTA, and across the other datasets. In summary, Hawk is effective even with very novel phenomena.



DOTA Class: Roundabout X axis is time in seconds. Y axis is number of TPs discovered. The "NoLearn" curves for bootstrap set sizes of 15, 10 and 5 are below the curve for 20, and are omitted to reduce clutter.

Figure 6.37: Impact of Bootstrap Set Size (12 kbps, DOTA)

		12 kbps		30 kbps		100 kbps	
	Total TPs	TPs Discovered	Tiles Transmitted	TPs Discovered	Tiles Transmitted	TPs Discovered	Tiles Transmitted
Roundabout	214	141	570	165	1427	175	4280
Swimming Pool	214	149	571	168	1430	181	4285
Large Vehicle	225	123	546	158	1367	170	4100
Airplane	200	120	633	142	1433	168	4390

Table 6.13: YOLO as Bandwidth Varies (DOTA, Scout Training)

6.4.6 DNN-Agnostic Model Evolution

Hawk cleanly separates and encapsulates the model used for selective transmission from the rest of its machinery. Adding a new model to Hawk is relatively straightforward (Section 3.5). No model customization is needed. The methods in Hawk interface (Section 3.4.5) has to be implemented, and default hyperparameter settings have to be provided for the Wizard (Section 3.6). The tile size of input images (Section 4.3) may also need to be modified. Hawk's modularity thus positions it well to benefit from future DNN architectural improvements from the ML community. In this section we evaluate how sensitive Hawk learning is to the specific choice of model. We aim to answer the question, "Is Hawk model agnostic?" We repeated the experiments in Section 6.4.1 using YOLOv5-small [28] and EfficientNet-b4 [70] rather than ResNet50.

The You Only Look Once (YOLO) models were proposed by Redmon et al. for object detection [57]. YOLOv5 is a state-of-the-art single-stage detector that formulates the object detection problem as a single regression problem, where bounding box coordinates and class probabilities are computed at the same time. During learning YOLO uses multi-scale training, where the input image is transformed and analyzed at different resolution. This helps in detecting small objects in the image.

EfficientNet is a group of state-of-the-art image classification DNN models. This model emphasizes in keeping the number of parameters lower while increasing the accuracy. The appropriate scaling coefficient for different dimensions of the network are found using grid search and those coefficients are applied to scale up the baseline network to the desired target model size or computational budget. Compared to other DNN models, the EfficientNet models shows better efficiency by reducing parameter size and FLOPS by an order of magnitude.

In Hawk, the tile size is configured depending on the DNN architecture. The tile size is 256x256 for ResNet-50 and EfficientNet; it is 600x600 for YOLO. Table 6.13 and Table 6.14 show the results for YOLOv5 and EfficientNet respectively. As in the case of ResNet-50, many of the TPs that are encountered during a mission are discovered even at 12 kbps. The number of TPs found for each classes differs with the DNN model used. For example, Column 7 of Table 6.13 shows that YOLO finds more TPs of Roundabout (175) than Airplane (168) at 100 kbps. In contrast, Column 7 of Table 6.4 shows that Resnet-50 finds fewer TPs of Roundabout (246) than Airplane (253) at the same bandwidth. Many other subtle reorderings of this nature occur across Tables 6.13 and 6.14. The choice of the optimal DNN for a mission is thus difficult to automate. Therefore, Hawk defers this decision to mission personnel.

		12 kbps		30 kbps		100 kbps	
	Total TPs	TPs Discovered	Tiles Transmitted	TPs Discovered	Tiles Transmitted	TPs Discovered	Tiles Transmitted
Roundabout	336	185	948	216	2570	289	7860
Swimming Pool	335	203	940	275	2590	305	7650
Large Vehicle	357	174	916	206	2470	247	7620
Airplane	350	194	958	272	2430	297	7560

Table 6.14: EfficientNet as Bandwidth Varies (DOTA, Scout Training)

6.5 Chapter Summary and Discussion

In summary, our results show the effectiveness of Hawk in discovering positives for datasets from the domains of drone surveillance, planetary exploration, and underwater sensing. Even at bandwidths as low as 12 kbps and base rate of 0.1%, on average Hawk discovers close to 64.6% of the positives in the stream across the three datasets. Hawk also is bandwidth adaptive, it discovers more positives with increase in network bandwidth. In the Hawk setting, where data is unlabeled and bandwidth is very low, the optimal location for training may be in the cloud or at the scout. Results in Section 6.4.4, upend the conventional thinking that the cloud is always the right place for training if privacy is not an issue. The specific tradeoffs are highly dependent on factors such as model compression, bandwidth, and scout compute power. This showcases the need for a dynamic and bandwidth-adaptive approach in selecting the training location. Hawk is not dependent on any specific DNN model architecture and as new models are developed they may be added to the Hawk toolkit.

Chapter 7

Improving Recall using Diversity Sampling and Few-Shot Learning

Previous chapters describe the characteristics and effectiveness of Hawk in discovering instances of a rare target from unlabeled high-resolution image streams in a distributed edge setting. In Chapter 6, we validate Hawk on three challenging datasets at network bandwidths as low as 12 kbps. From the results, a team of 7 scouts is able to discover up to 86.3% of the target instances discovered using BruteForce model. A BruteForce model is created using prior knowledge of the mission. Though effective in discovering rare instances, there exists a persistent gap between Hawk and BruteForce as seen in Figures 6.7 to 6.33. The gap between the two search methods is as small as 13% in the best case, but higher in other cases. It can be viewed as the price of doing learning on-the-fly from sparsely labeled data. *Can we improve the performance of Hawk even further?*

There are three main reasons for the disparity between Hawk and BruteForce. First, the BruteForce model as stated earlier has knowledge of the mission, the model used for BruteForce model is trained using all mission data. On the other hand, the initial model in Hawk is trained using less than 20 positive instances of the target. Due to high variability in appearance even amongst instances of the same target object, there may be positive instances in the mission data that are very different from the instances available in the training set. Secondly, the positive instances presented for labeling may be too similar to the existing examples in the training set. This leads to little improvement in the model during training. Thus, some positive instances present in the mission data may never be transmitted for labeling. Finally, human-labeling guides the mission, any bias during labeling is reflected in the items transmitted by Hawk.

In this chapter, we look at techniques to improve recall and thus reduce the gap between positives discovered using BruteForce and Hawk. Particularly, we explore two techniques, Diversity Sampling and Few-Shot Learning, as ways of improving recall and hence, the performance of Hawk.

7.1 Diversity Sampling

Hawk aims to continuously improve its selective transmission capability during a mission. As mentioned in Section 4.5, Hawk uses TopK strategy to select results for transmission. The selection strategy serves dual purpose, at low network bandwidth it aims to transmit as many positive instances as possible and also select good items that improve the model quality. Thus, the top "K" most scoring items is a reasonable heuristic. It both helps in discovering positive instances, and the FPs transmitted are negatives that confuse Hawk model the most. However, using this selective strategy could lead to the model transmitting TPs that are too similar to the examples in the training set and lead to stagnation of model quality. Uncertainty sampling strategies in Active Learning (AL) such as TopK and Maximum Entropy, mainly focus on items close to the decision boundary and leads to confirmation bias [36].

This bias is inherent to Active Learning (AL) and Semi-Supervised Learning (SSL) and is a significant contributor to the gap between positives discovered in Hawk and BruteForce. The early TPs guide learning along a path that gives low scores to TPs that look very different. These are the proverbial "black swans," relative to what has been learned so far. Once SSL proceeds down this path, those low-scoring positives and others like them are excluded from ever being transmitted for labeling. The TPs transmitted for labeling will merely reinforce the existing bias, leading to a stagnation of model quality. This problem does not occur in fully supervised learning, because all data is labeled *a priori.* Thus, there is a need for transmitting diverse positives to the labeler. In this section, we show how *diversity sampling* strategies can be used to compensate for the lack of exploration of space.

As stated earlier, the TopK selection strategy does not remove redundant results during transmission. This can result in very similar results being labeled and can lead to little or no improvement in the quality of models trained. Diversity sampling helps select items of different kinds, thus, it is complementary to uncertainty sampling. We use clustering to partition the feature space and then select representative samples from each cluster. We build on recent work in *diversity sampling* [65, 83]. These work optimize for both model uncertainty and feature exploration. We choose an approach similar to Zhdanov [83], we use clustering to select diverse samples. In Zhdanov [83], the authors connect the problem of diverse selection to the Facility Location problem and uses K-means clustering as a solution. In Hawk, we cannot make assumptions on the number of clusters. We therefore use a density-based clustering scheme [14], rather than K-means clustering. We project the high-dimensional DNN-extracted features to a low-dimensional space using principal component analysis (PCA) before applying the clustering algorithm.

How much of the available bandwidth should we allocate to diversity-based samples, as opposed to TopK samples? This is a tricky question to answer, because it is highly dependent on the data, and the order in which it is encountered. Our experiments show that deviating from strict TopK at very low bandwidth (12 kbps) hurts rather than helps. Since K=4 at 12 kbps, one diversity sample and three TopK samples per batch is the smallest diversity allocation possible. The columns labeled "12 kbps" in Table 7.1 show that even using this smallest possible allocation for diversity hurts Hawk for all four classes of the DOTA dataset. Except for a few classes, we observe similar behavior for the classes in HiRISE and Brackish as given in Table 7.2 and Table 7.3 respectively.

When bandwidth rises, there is additional headroom for diversity. At 30 kbps, K=10; we

		12 kbps		30 kbps		100 kbps	
	Total TPs	TopK Only	+Diversity	TopK Only	+Diversity	TopK Only	+Diversity
Roundabout	336	219	203	235	237	246	269
Swim. Pool	335	209	194	250	262	272	313
Large Vehicle	357	200	182	217	217	264	271
Airplane	350	176	183	213	221	253	285

The mean of three runs of each experiments are shown. Standard deviations are not shown since they are less than 5.54%.

Table 7.1: Using Diversity Selection (DOTA, scout training)

		12 kbps		30 kbps		100 kbps	
	Total TPs	TopK Only	+Diversity	TopK Only	+Diversity	TopK Only	+Diversity
Bright Dune	64	48	44	51	49	56	57
Crater	64	42	37	49	47	55	55
Dark Dune	64	43	41	50	52	55	58
Imact Ejecta	64	40	43	45	49	51	57
Slope Streak	64	45	39	49	52	54	59
Spider	64	45	42	50	48	53	54
Swiss Cheese	64	47	48	51	51	57	61

Table 7.2: Using Diversity Selection (HiRISE, scout training)

		12 kbps		30 kbps		100 kbps	
	Total TPs	TopK Only	+Diversity	TopK Only	+Diversity	TopK Only	+Diversity
Crab	600	396	385	462	472	511	533
Fish	600	327	310	434	440	488	506
Starfish	370	182	177	227	235	255	267
Shrimp	564	375	366	455	462	496	517
SmallFish	564	299	281	311	320	386	404
Jellyfish	584	334	338	387	393	440	456

Table 7.3: Using Diversity Selection (Brackish, scout training)

transmit 7 TopK samples and 3 diversity samples. At 100 kbps, K=30; we transmit 20 TopK samples and 10 diversity samples. As seen from Tables 7.1 to 7.3, there is a significant improvement in the number of TPs found at these higher bandwidths. At 100 kbps, on average Hawk discovers 15 additional positives for the three datasets considered. We observe the most improvement for the target "Swimming-Pool", we discover 41 more positives using TopK+Diversity as compared to a pure TopK strategy. For the HiRISE dataset, the improvement in positives is smaller than other datasets, a reason for this could be that the samples are all clustered together leading to inefficient diversity sampling.

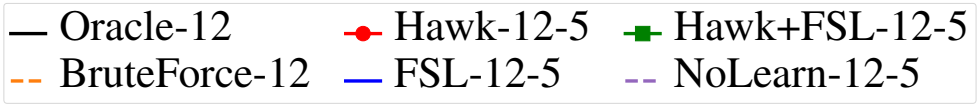
7.2 Few-Shot Learning

Few-shot Learning (FSL) is a subfield of machine learning that learns from a very small set of training samples. FSL methods aim to emulate a human's ability to identify new data from only a few examples. In essence, FSL methods "provide algorithms for transferring knowledge from a large set of source data, to a set of sparsely annotated target categories of interest" [25]. FSL is a complement to Hawk rather than a competitor. Its goal is to create the best model possible when only a few labeled TPs are available. This is indeed the situation at the start of a Hawk mission. However, Hawk's goal is to go well beyond the few-shot stage by discovering new TPs and adding them to the training set. This leads to two questions: (1) Can the learning in Hawk help it to catch up with FSL? (2) Can the strengths of FSL and Hawk be combined? We answer these questions using SnaTCHer [27], a state-of-the-art open-set recognition algorithm for FSL.

FSL models provide better feature representations as compared to a DNN model learned via transfer learning. In this section, we compare the performance of Hawk with FSL methods. Most FSL work only consider a closed-world problem setting where the test data contains classes that are known to the FSL model. However, this does not satisfy the setting we consider in Hawk where the majority of mission data are uninteresting and may not contain instances of the target class. This is the main reason for using *SnaTCHer* [27], which considers few-shot open-set recognition problem. In this work, the algorithm trains a feature transformer to identify unseen samples. SnaTCHer supports 1-shot and 5-shot FSL; we use the 5-shot version in our experiments.

At 12 kbps and a bootstrap set size of 5, Figures 7.1 to 7.14 compare Hawk-12-5 and FSL-12-5 for the classes in the three datasets. For reference, the Oracle-12, BruteForce-12, and NoLearn-12-5 lines are also shown. Two observations are salient. First, FSL-12-5 does much better than NoLearn-12-5. In other words, if no learning is done beyond the bootstrap stage, using FSL to create Gen-0 is definitely better than using transfer learning. Second, although Hawk-12-5 is noticeably worse than FSL-12-5 at the start of the mission, it is noticeably better by the end. This positively answers the first question: continuous learning during the mission can indeed overcome the advantage of a superior initial model that does not evolve.

To answer the second question, we compare Hawk-12-5 with a modified version of Hawk whose Gen-0 model is FSL-12-5. As mentioned earlier, FSL-12-5 discovers more positives as compared to the Gen-0 model used in Hawk-12-5, as can be observed from the NoLearn-12-5. Thus instead of the transfer-learned model, we use the FSL model as the initial model. FSL methods do not allow for continuous learning, that is the FSL model is never re-trained to include



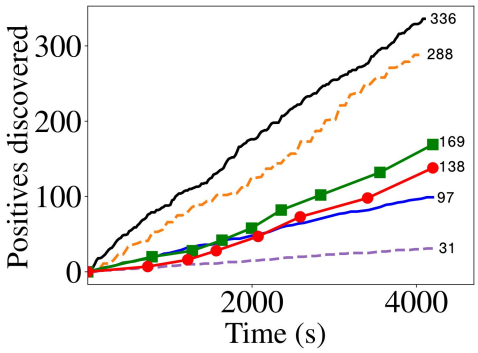


Figure 7.1: DOTA: Roundabout

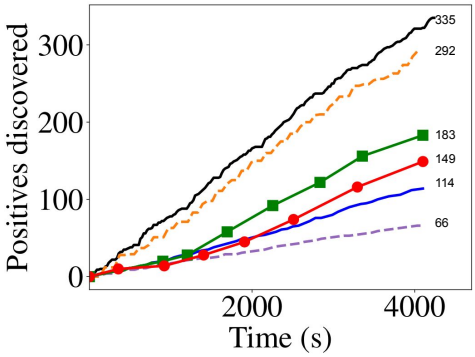


Figure 7.2: DOTA: Swimming Pool

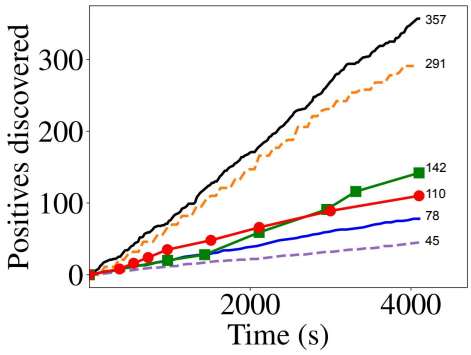


Figure 7.3: DOTA: Large Vehicle

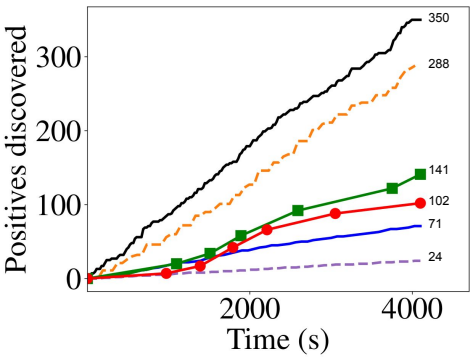
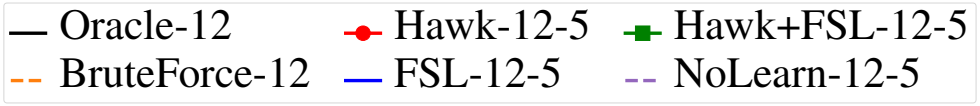


Figure 7.4: DOTA: Airplane

Number of bootstrapping TPs = 5.

Figure 7.5: Few-Shot Learning (12 kbps, DOTA, Scout Training)



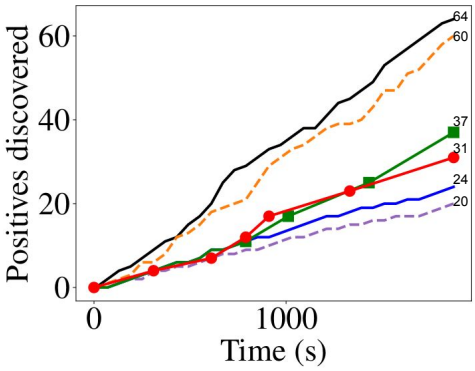


Figure 7.6: HiRISE: Dark Dune

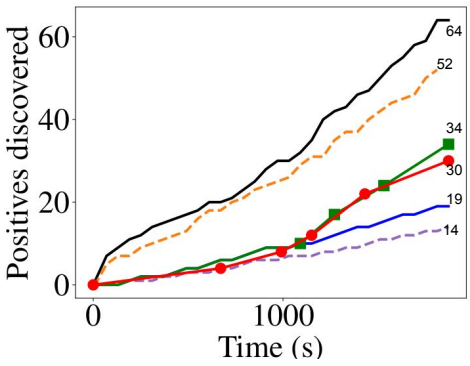


Figure 7.7: HiRISE: Impact Ejecta

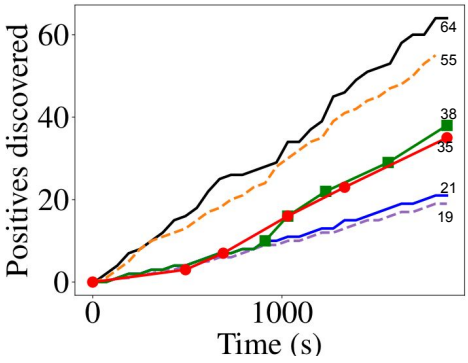


Figure 7.8: HiRISE: Spider

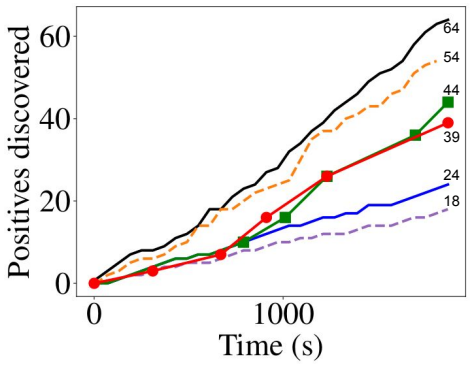
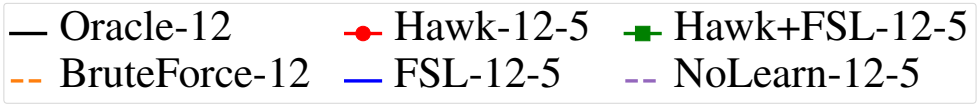


Figure 7.9: HiRISE: Swiss Cheese

Number of bootstrapping TPs = 5.

Figure 7.10: Few-Shot Learning (12 kbps, HiRISE, Scout Training)



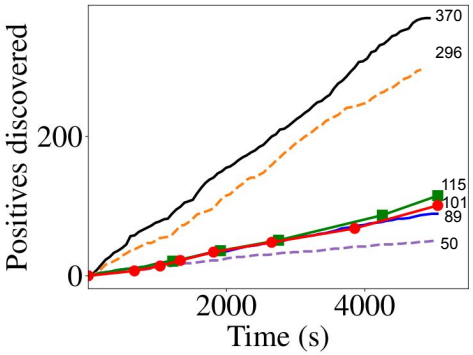


Figure 7.11: Brackish: Starfish

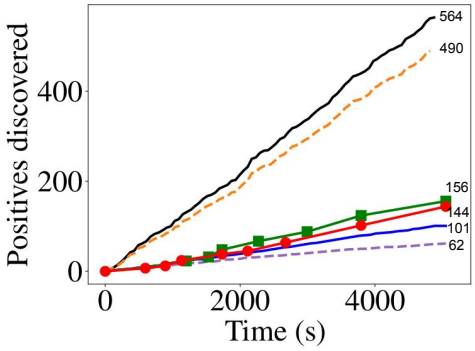


Figure 7.12: Brackish: Shrimp

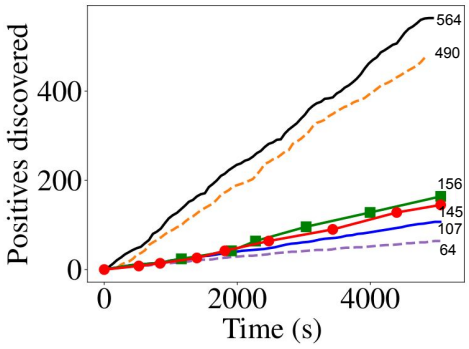


Figure 7.13: Brackish: Small Fish

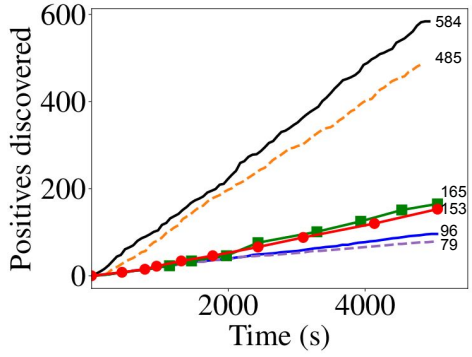


Figure 7.14: Brackish:Jellyfish

Number of bootstrapping TPs = 5.

Figure 7.15: Few-Shot Learning (12 kbps, Brackish, Scout Training)

the knowledge of newly discovered TPs. Therefore, to perform continuous learning we replace the FSL model with a transfer learnable model after sufficient TPs have been discovered by the Gen-0 model. In other words, once sufficient TPs have been collected to train Gen-1, Hawk switches to using transfer learning for Gen-1 and all further models. This hybrid approach is shown by the curve Hawk+FSL-12-5 in Figures **??** to **??**. For DOTA and Brackish datasets, we use Hawk after collecting 20 positives. This number is 10 positives in the case of HiRISE, because FSL finds fewer than 20 positives for some classes in HiRISE. From the mission start to model Gen-1, the superiority of FSL over simple transfer learning is operative. As mentioned earlier in Section 4.6, the influence of early training examples is especially significant. This leads to about 34 more positives being found for the four classes of DOTA, relative to Hawk-12-5. The improvement is smaller in the case of the other two datasets, about 5 in the case of HiRISE dataset and 14 for Brackish dataset. In effect, FSL provides Hawk with a higher starting point from which to benefit from improvements via continuous learning.

A possible future extension to Hawk would be to have a continual FSL approach, where we use a model trained using FSL technique for models beyond Gen-0 model, possibly to Gen-1, Gen-2, and so on. Efforts are being made to enhance few-shot learning by incorporating the capability for continuous lifelong learning [7, 58]. Such a learning technique would greatly benefit from the generalization capability provided by the FSL model and the ability of continual learning to incorporate the knowledge of newly available data without forgetting previous data.

7.3 Chapter Summary and Discussion

In this Chapter, we introduce diversity sampling and few-shot learning as techniques to improve the performance of hawk in discovering more positives relative to BruteForce. Hawk's default configuration optimizes for extreme low bandwidths by using a pure TopK selection strategy. Diversity sampling, helps in the exploration of the feature space and improvement in the model learned compared to TopK sampling. The fraction of K allocated to diversity can be adapted to match the current bandwidth. We also show the advantage of using FSL as the initial model for a bootstrap set having 5 positives. As stated earlier, the better feature representation of FSL provides Hawk with a higher starting point. Since FSL and diversity sampling (Section 7.1) apply to different phases of a Hawk mission, we can easily combine the two. This combination can further close the gap relative to BruteForce.

Chapter 8

Conclusion and Future Work

In this dissertation, we address the problem of efficient utilization of low network bandwidth for remote sensing of rare events. As mentioned in Section 1.1, the mismatch ratio between the incoming sensor data rate on probes, and much lower backhaul bandwidth from probes to the Internet, can easily be one to four orders of magnitude. We design and implement a system called Hawk that is extremely bandwidth-frugal in collecting data for training sets of rare phenomena. We also propose an efficient semi-supervised approach to continuously improve the quality of learning and in turn, improve the selective transmission capability in Hawk during a mission. This chapter concludes the dissertation with a summary of contributions, and discusses future research directions and challenges in this area.

8.1 Contributions

As stated in Chapter 1, the thesis validated by this dissertation claims:

> It is feasible and effective to create a distributed system that integrates selective transmission, human labeling, and model training to perform low base rate active learning in an edge computing setting. Such a system can be valuable in collecting training data of a novel phenomenon in extreme bandwidth-challenged environments. Such a system can transparently perform machine learning in the background and automatically select tactical parameters to match network bandwidths, data distribution, and computational loads on robotic probes.

To validate this thesis, we design and build a prototype implementation of the system Hawk. Hawk is an interactive model-agnostic live learning system that enables discovery of rare novel phenomena from a stream of extremely skewed unlabeled visual data capture on weakly-connected remote sensing probes. Live learning is crucial for remote sensing missions where discovery of targets are difficult due to factors such as low bandwidth, novelty of target, and extreme class imbalance.

Previous work in distributed and federated learning assume the incoming data is pre-labeled, which is not true in the case of real-world deployments. These work also do not consider the rarity of targets, which is inherent to video analytics or surveillance tasks. We do not know any

prior system that performs live learning.

In this work, we recognize the network bandwidth as the first-class design consideration in live learning. There exists a severe mismatch of 10^3 – 10^4 or more between the input and output data rates on remote sensing platforms. This severely limit the amount of data that can be transmitted back to the mission personnel. Thus, even a weak model trained on-the-fly using knowledge acquired from the data encountered during a mission is helpful to enrich the result stream to the mission personnel for inspection. This also helps prune away irrelevant data and thus, ensure that precious bandwidth is not wasted.

We also propose an efficient active learning and semi-supervised learning approach to improve the quality of the newly trained models and discover positives instances of a target from very few labeled data. Widely used techniques in active learning and semi-supervised learning, do not consider extreme class imbalance of 0.1% or less. Naively using one of the classical semi-supervised algorithms such as pseudo-labeling will deteriorate the quality of the model trained because of the rarity of the target class. In Hawk, we ensure that any positive data added to the training data is verified by the mission personnel as a TP. We develop an improved pseudo labeling (Section 3.2), to machine label ENs and thus prevent transmission of frivolous FPs. To ensure training is not affected by a stray mislabeled example, we use a fresh random subset of available pseudo-labeled negatives for each new generation of model training.

In Chapters 4 and 6, we extensively validate the design and effectiveness of the system under varying conditions across three challenging datasets. These are datasets similar to data collected during a remote sensing mission and are collected from aerial drones (UAVs), satellites, and underwater vehicles. In our results, we show the effectiveness of Hawk in collecting positive instances encountered from live data stream during a mission. By presenting results for three different DNN model architecture, we demonstrate the model-agnostic nature of Hawk. We show that even at bandwidths as low as 12 kbps, Hawk comes close to what is achievable through brute force.

Finally, we present results to showcase probable techniques to improve the productivity of Hawk. In the best case, we were able to find about 40 additional positive examples by extending Hawk to include diversity sampling capability.

8.2 Future Work

8.2.1 Optimizing Human Attention Bandwidth

There are two critical bottlenecks in the interactive labeling of rare targets from unlabeled visual streams: (a) the network bandwidth from the remote source to the human expert, and (b) the expert's labeling bandwidth. In this thesis, we focus on the efficient utilization of the backhaul bandwidth by transmitting better results to the expert. The selective transmission strategy in Hawk is frugal in its bandwidth usage, it fills but not overfill the end-to-end pipeline, and thus optimize the use of scarce network bandwidth.

Another area worth exploring is the efficient usage of human expert's time and attention during the labeling task. Most work in active learning use a labeling script in lieu of a human annotator and the key performance indicator is the performance of the trained model against

benchmark datasets. It may be valuable to study how factors such as human fatigue, and the difference in labeling quality can affect the training set collection. The system must also be designed to match the rate of result delivery to the labeling rate of the expert. When the delivery is too fast, it overloads the expert and results in wastage of system resources. On the other hand, too slow a delivery will result in wasting the expert's time because of the data stall. This work can benefit greatly from algorithms that help identify coresets that generalize and speed up training such that the trained machine learning model has approximately the same accuracy as when trained on the entire data [44, 50].

8.2.2 Improving Model Compression

In Section 6.4.4, we discuss the need for aggressive model techniques to be able to transmit DNN models that are tens of MBs in size over low kbps networks available during remote sensing. Model compression aims to remove the redundancies in large models and reduce the model size and inference time of a model with minimal impact on its performance. Some widely used techniques to achieve such compression are: pruning, knowledge distillation, and quantization [15]. However, the effect of these methods alone is limited.

With the increase in training data, the newly developed DNN models are getting bigger and more complex. For example, large language models are growing at the rate of 10X every 12 months, and the compute requirements are doubling every 3.4 months [60]. Even with prominent compression methods such as DeepIoT [82], which reports size reductions of 90–99% on sequential model architectures such as LeNet and VGG, we only observe 2–3X compression on recent DNNs such as YOLOv5 and EfficientNet. As shown in Table 4.3, we can attain a better compression ratio, around 10X, by training only the last layers of the model. However, from our results, it is clear that only with compression close to 100X we see an advantage of training in the cloud as compared to training on-site on the scouts. Therefore, it is worth exploring techniques to attain 10X or 100X compression in model sizes.

8.2.3 Addressing Labeling Bias

Machine learning models trained on a human-labeled dataset can learn the inherent biases of the labeler[49]. The labeler's biases such as language, expertise, and experiences can influence the quality of data labeling. Learning in Hawk is guided by the labels provided the expert, any bias during labeling is reflected in the results transmitted. In this thesis, we do not address the impact of imperfect human labelers or noisy labels. Even the most widely used ImageNet dataset has an error rate of 5.8% [48]. The presence of these noisy labels in training data can greatly reduce the accuracy of the learned models. Early identification of such erroneous labels is valuable to guarantee the accuracy of learning is not affected. Solutions that address this problem [12, 71], can help ensure the system is robust to such labeling bias or inaccuracies.

8.2.4 Adapting to new targets

In many exploration tasks, unforeseen phenomena or a possible threat, may be encountered during a mission. It is impossible to anticipate all types of queries before the start of the mission.

Therefore, it is beneficial to have a learning algorithm that can learn and detect new phenomena and act accordingly. Recent work in *incremental machine learning* may be useful in addressing this problem. One such work iCARL [54] extends the existing model's knowledge to adapt to new targets of interest while preserving previously acquired knowledge. An interesting extension of Hawk would be the addition of such an incremental learning algorithm.

8.2.5 Continual Few-Shot Learning

Another valuable future extension to Hawk would be the integration of a continual few-shot learning. Few-Shot Learning (FSL), as the name suggests are algorithms that predict new classes that have very few label data. These FSL models have better generalization capability compared to models trained using transfer learning with the same labeled examples. However, conventional FSL methods do not consider continuous learning to improve the model quality when new TPs are available. Such an extension to FSL methods will further improve the number of TPS that can be discovered by Hawk.

8.2.6 Extending Hawk Labeling Interface

In specialized fields such as pathology, there may be ambiguous data items that need a second opinion. In Hawk, currently the labeling interface only supports a single expert per mission. One way to support labeling by multiple experts would be to shard the results across multiple home (or clients). A better extension would be the capability to incorporate judgments of multiple experts for an ambiguous data items. When an expert displays uncertainty in labeling, the item is presented to multiple experts to collect their decision. The system may also choose to occasionally presents the same items to multiple experts to ensure consensus in labeling. Such an interface may have screen sharing and voice communication features to help experts reconcile differences, if any, during labeling. For cases where the disagreement needs to be preserved, multi-label classification [75] techniques may be useful.

8.2.7 Burstiness of TPs

As discussed in Section 4.5, in Hawk we assume the low base rate TPs have uniform data distribution. However, there could be burstiness in the distribution of TPs. This burstiness could be temporal in nature (e.g., in a search-and-rescue drone mission, all the survivors are clustered at a particular location) or spatial in nature (e.g., one of N scouts sees a target-rich environment, but the others see a target-poor environment). In such cases, the default TopK selection strategy in Hawk may miss transmitting some of the TPs or may transmit some of the TPs in the next iteration of selective transmission, provided the TPs have high scores. A naive approach, would be to have live actuation, upon discovery of a target the ground operator could instruct the flight program in the drone to fly closer for better inspection. Another approach, would be to have an on-demand pull request from the operator based on the hit-rate from a particular scout. Such that, instead of having the same K value across scouts, the value of K can be based on the number of TPs discovered at each scout.

Bibliography

[1] Video Surveillance Storage: How Much Is Enough? https://www.seagate.com/solutions/surveillance/how-much-video-surveillance-storage-is-enough/. 4

[2] Protocol Buffers - Google's data interchange format. https://github.com/protocolbuffers/protobuf, 2008. 5.1

[3] Defense Media Activity. Defense Visual Information Service. https://www.dvidshub.net/. Last accessed: June 12, 2020. 4.2.2

[4] David Adamy. *EW 103: Tactical Battlefield Communications Electronic Warfare*. Artech House, 2008. 1.1

[5] Airandspace. Bird's eye viewfinder: 160 years of aerial photography. 2017. 1

[6] Amjad Ali and Khalid Saifullah Syed. An outlook of high performance computing infrastructures for scientific computing. In *Advances in Computers*, volume 91, pages 87–118. Elsevier, 2013. 4.1

[7] Antreas Antoniou, Massimiliano Patacchiola, Mateusz Ochal, and Amos Storkey. Defining benchmarks for continual few-shot learning. *arXiv preprint arXiv:2004.11967*, 2020. 7.2

[8] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1790–1802, 2015. 4.6

[9] Mohammadamin Barekatain, Miquel Martí, Hsueh-Fu Shih, Samuel Murray, Kotaro Nakayama, Yutaka Matsuo, and Helmut Prendinger. Okutama-Action: An Aerial View Video Dataset for Concurrent Human Action Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017. 4.3

[10] Mesay Belete Bejiga, Abdallah Zeggada, Abdelhamid Nouffidj, and Farid Melgani. A convolutional neural network approach for assisting avalanche search and rescue operations with uav imagery. *Remote Sensing*, 9(2):100, 2017. 1

[11] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 2.2

[12] Mohamed-Rafik Bouguelia, Yolande Belaid, and Abdel Belaid. Identifying and Mitigating Labelling Errors in Active Learning. In *Pattern Recognition: Applications and Methods*, 2015. 3.0, 8.2.3

[13] Kevin Canini, Tushar Chandra, Eugene Ie, Jim McFadden, Ken Goldman, Mike Gunter, Jeremiah Harmsen, Kristen LeFevre, Dmitry Lepikhin, Tomas Lloret Llinares, et al. Sibyl: A system for large scale supervised machine learning. *Technical Talk*, 1:113, 2012. 2.3

[14] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 328–339. SIAM, 2006. 7.1

[15] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017. 8.2.2

[16] Cisco. Cisco annual internet report (2018–2023) white paper. 2022. 2.1

[17] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9268–9277, 2019. 3.2

[18] Department of Health and Human Services, US Government. Health Information Privacy, 1996. `https://www.hhs.gov/hipaa/index.html`. 2.4

[19] Gary Doran, Steven Lu, Lukas Mandrake, and Kiri Wagstaff. Mars orbital image (hirise) labeled data set version 3. *NASA: Washington, DC, USA*, 2019. 6.2.2

[20] Ziqiang Feng, Shilpa George, Jan Harkes, Padmanabhan Pillai, Roberta Klatzky, and Mahadev Satyanarayanan. Edge-based discovery of training data for machine learning. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 145–158, 2018. 1.5.2, 2.5

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 770–778, 2016. 6.3

[22] John Heidemann, Milica Stojanovic, and Michele Zorzi. Underwater sensor networks: applications, advances and challenges. *Philosophical Transactions of the Royal Society A*, 370:150–175, 2012. 1.1

[23] Pieter Hintjens. *ZeroMQ: messaging for many applications*. "OReilly Media, Inc.", 2013. 5, 5.1

[24] Troy Howe. SPEAR Probe - An Ultra Lightweight Nuclear Electric Propulsion Probe for Deep Space Exploration. `https://www.nasa.gov/directorates/spacetech/niac/2020_Phase_I_Phase_II/SPEAR_Probe/`, 2020. 4.9

[25] Shell Xu Hu, Da Li, Jan Stühmer, Minyoung Kim, and Timothy M Hospedales. Pushing the limits of simple pipelines for few-shot learning: External data and fine-tuning make a difference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9068–9077, 2022. 7.2

[26] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *Proceedings of IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017. 4.3

[27] Minki Jeong, Seokeon Choi, and Changick Kim. Few-shot open-set recognition by transformation consistency. In *Proceedings of the IEEE/CVF Conference on Computer Vision*

and Pattern Recognition, pages 12566–12575, 2021. 7.2

[28] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, AlexWang1900, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, February 2022. URL `https://doi.org/10.5281/zenodo.6222936`. 6.4.6

[29] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54, 2019. 1.3

[30] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960. 1.2

[31] Jaehyung Kim, Youngbum Hur, Sejun Park, Eunho Yang1, Sung Ju Hwang1, and Jinwoo Shin. Distribution Aligning Refinery of Pseudo-label for Imbalanced Semi-supervised Learning. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 3.2

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3.3.1

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1.4

[34] Matthias Langer. *Distributed Deep Learning in Bandwidth-Constrained Environments*. PhD thesis, La Trobe University, 2019. 2.3

[35] Dong-Hyun Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896, 2013. 3.1, 3.2

[36] Germain Lefebvre, Christopher Summerfield, and Rafal Bogacz. A normative account of confirmation bias during reinforcement learning. *Neural computation*, 34(2):307–337, 2022. 7.1

[37] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14, 2010. 2.1

[38] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3), May 2020. 2.4

[39] Yu Li, Tao Wang, Bingyi Kang, Sheng Tang, Chunfeng Wang, Jintao Li, and Jiashi Feng. Overcoming classifier imbalance for long-tail object detection with balanced group softmax. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020. 3.2

[40] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for

dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 3.2

[41] Spencer K. Lynn and Lisa Feldman Barrett. 'Utilizing' signal detection theory. *Psychological science*, 2014. 1.3

[42] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision (ECCV)*, 2018. 3.2

[43] Andrew McCallum, Kamal Nigam, et al. Employing em and pool-based active learning for text classification. In *ICML*, volume 98, pages 350–358. Madison, 1998. 2.2

[44] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020. 8.2.1

[45] Alexander Narr, Rudolph Triebel, and Daniel Cremers. Stream-based active learning for efficient and adaptive classification of 3d objects. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 227–233. IEEE, 2016. 2.2

[46] NASA. Communications with Earth. `https://mars.nasa.gov/msl/mission/communications/`. 1.1

[47] Brian D Noble, Mahadev Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R Walker. Agile application-aware adaptation for mobility. *ACM SIGOPS Operating Systems Review*, 31(5):276–287, 1997. 2.1

[48] Curtis G Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. *arXiv preprint arXiv:2103.14749*, 2021. 8.2.3

[49] O'Reilly. Arguments against human labeling. 2022. 8.2.3

[50] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34:20596–20607, 2021. 8.2.1

[51] Malte Pedersen, Bruslund, Joakim Haurum, Rikke Gade, and Thomas Moeslund. Detection of marine animals in a new underwater dataset with varying visibility. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019. 6.2.3

[52] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11, 2013. 2.3

[53] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proceedings of the VLDB Endowment*, 11(3), November 2017. 1.5.2, 2.5

[54] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 8.2.4

[55] William Reckling, Helena Mitasova, Karl Wegmann, Gary Kauffman, and Rebekah Reid. Efficient drone-based rare plant monitoring using a species distribution model and ai-based object detection. *Drones*, 5(4):110, 2021. 1

[56] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL `http://arxiv.org/abs/1804.02767`. 4.2.1

[57] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 6.4.6

[58] Mengye Ren, Michael L Iuzzolino, Michael C Mozer, and Richard S Zemel. Wandering within a world: Online contextualized few-shot learning. *arXiv preprint arXiv:2007.04546*, 2020. 7.2

[59] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015. 1.2, 6.3

[60] Sambanova. The impact of large language models and deep learning. 2022. 8.2.2

[61] Mahadev Satyanarayanan. Accessing information on demand at any location. mobile information access. *IEEE personal Communications*, 3(1):26–33, 1996. 2.1

[62] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001. 2.1

[63] Mahadev Satyanarayanan, Wei Gao, and Brandon. The Computing Landscape of the 21st Century. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications (HotMobile'19)*, Santa Cruz, CA, February 2019. 2.1

[64] Burr Settles. *Active Learning*. Morgan & Claypool Synthesis Series on Machine Learning, 2012. 2.2

[65] Jingyu Shao, Qing Wang, and Fangbing Liu. Learning to Sample: an Active Learning Framework. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2019. 7.1

[66] Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. Variational adversarial active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5972–5981, 2019. 3.0

[67] Jasmina Smailović, Miha Grčar, Nada Lavrač, and Martin Žnidaršič. Stream-based active learning for sentiment analysis in the financial domain. *Information sciences*, 285:181–203, 2014. 2.2

[68] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Publishing, 2008. 4.6

[69] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 3.3.2

[70] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 6.4.6

[71] Hidetaka Taniguchi, Hiroshi Sato, and Tomohiro Shirakawa. A machine learning model with human cognitive biases capable of learning from small and biased datasets. *Scientific reports*, 8(1):1–13, 2018. 8.2.3

[72] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. YFCC100M: the new data in multimedia research. *Communications of the ACM*, 2016. 4.2.1

[73] Costa Tsaousis and Costa Tsaousis. FireQOS Reference. `https://firehol.org/fireqos-manual.html`. 5.1, 6.1

[74] European Union. General data protection regulation, 2016. `https://gdpr-info.eu/`. 2.4

[75] Hamed Valizadegan, Quang Nguyen, and Milos Hauskrecht. Learning classification models from multiple experts. *Journal of Biomedical Informatics*, 46(6):1125 – 1135, 2013. 8.2.6

[76] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018. 4.2.1

[77] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. A Survey on Distributed Machine Learning. *ACM Computing Surveys*, 53(2), March 2020. `https://doi.org/10.1145/3377454`. 2.3

[78] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient Live Video Analytics for Drones via Edge Computing. In *Proceedings of the Third IEEE/ACM Symposium on Edge Computing (SEC 2018)*, Bellevue, WA, October 2018. 4.3

[79] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. DOTA: A Large-Scale Dataset for Object Detection in Aerial Images. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 6.2.1

[80] Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. A survey on deep semi-supervised learning. *arXiv preprint arXiv:2103.00550*, 2021. 3.1

[81] Yao-Yuan Yang, Shao-Chuan Lee, Yu-An Chung, Tung-En Wu, Si-An Chen, and Hsuan-Tien Lin. libact: Pool-based active learning in python. *arXiv preprint arXiv:1710.00379*, 2017. 2.2

[82] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework. In *Proceedings of SenSys '17*, 2017. 4.4, 6.4.4, 8.2.2

[83] Fedor Zhdanov. Diverse mini-batch active learning. *arXiv preprint arXiv:1901.05954*, 2019. 7.1