U-Net Ship Detection in Satellite Optical Imagery

by

Benjamin Smith
B.Sc., Vancouver Island University, 2018

A Thesis Submitted in Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

U-Net Ship Detection in Satellite Optical Imagery

by

Benjamin Smith
B.Sc., Vancouver Island University, 2018

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Sean Chester, Academic Unit Member
(Department of Computer Science)

ABSTRACT

Deep learning ship detection in satellite optical imagery suffers from false positive occurrences with clouds, landmasses, and man-made objects that interfere with correctly classifying ships. A custom U-Net is implemented to challenge this issue and aims to capture more features in order to provide a more accurate class accuracy. This model is trained with two different systematic architectures: single node architecture and a parameter server variant whose workers act as a boosting mechanism. To extend this effort, a refining method of offline hard example mining aims to improve the accuracy of the trained models in both the validation and target datasets however it results in over correction and a decrease in accuracy. The single node architecture results in 92% class accuracy over the validation dataset and 68% over the target dataset. This exceeds class accuracy scores in related works which reached up to 88%. A parameter server variant results in class accuracy of 86% over the validation set and 73% over the target dataset. The custom U-Net is able to achieve acceptable and high class accuracy on a subset of training data keeping training time and cost low in cloud based solutions.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Supervisor Dr. Yvonne Coady, for mentoring, support, endless excited encouragement, and patience. Pushing me to achieve my goals.

UrtheCast, for funding me with a grant and giving me access to amazing satellite data.

But what does Socrates say? 'Just as one person delights in improving his farm, and another his horse, so I delight in attending to my own improvement day by day.'
Epictetus, Discourses, 3.5.14

Chapter 1

Introduction

The ability to detect ships in optical satellite imagery provides issues surrounding clouds, landmasses, man-made objects, and highly reflective objects that introduce false positives. This has been noted as a significant area of interest within the remote sensing community. In this work, the goal is to exceed related works accuracy in the domain of ship detection using optical satellite imagery. Using a training/validation dataset distribution matched to that of a target distribution, a custom U-Net model will be trained in order to predict segmentation masks of ships of varying sizes. The target dataset is provided and requested to be used from industry partner UrtheCast, Vancouver, B.C. via their Theia MRC satellite. Since no ship detection dataset exists with the target dataset, a dataset taken by the SPOT-5 satellite will be used and the resolution re-sampled to that of the target dataset.

This custom U-Net model implements deeper layers and bottlenecks that aim to provide a better feature representation during the encoding phase and result in a improved class accurate segmentation mask than other works. This claim will be supported both empirically through class accuracy scores and visually by interpretive analysis of segmentation masks on validation and target data. The custom U-Net will be trained with two different systematic approaches: a single node architecture that closely relates to standard training and a parameter server variant. This variant implements the workers as weak learner boosters via gradient accumulation in order to train over the dataset. The parameter server variant aims to explore multiple gradient paths via multiple workers and result in a more generalizable model. The outcome of increased class accuracy detecting ships in satellite optical imagery can provide beneficial use-case applications for industry and governmental bodies in areas of economic, ecological, and security domains.

1.1 Agenda

Chapter 1 contains a statement of the claims which will be proved by this dissertation followed by an overview of the structure of the document itself.

Chapter 2 describes in details the background and related works which this dissertation will use to provide motivation and baseline goals.

Chapter 3 gives the new research, its methodology, the algorithms involved, the new solution, the new work done.

Chapter 4 is where the experiments and the methodology for them is fully described.

Chapter 5 includes the evaluation of the data presented above and the comparisons with the work of others.

Chapter 6 contains a restatement of the claims and results of the dissertation. It also enumerates avenues of future work for further development of the concept and its applications.

Chapter 2

Background

A prominent area of interest within the remote sensing domain is ship detection in optical satellite imagery. This application comes with many implications involving ecological, environmental, and financial. However, ship detection comes with its set of challenges. Of these, false positive occurrences have recently been approached with the use of deep learning. At a higher level, the other challenges include satellite data being systematically expensive to implement into learning applications. Due to its size, cloud based approaches are often used as a solution. Cloud-based parameter servers make use of task scheduling to reduce the occurrence of straggling workers and increase throughput through different algorithmic approaches.

2.1 Ship Detection Using Satellite Imagery Data and Deep Learning

Research surrounding the detection of ships using satellite optical imagery is extensive regarding the many satellites, algorithmic approaches, and deep learning approaches used [3]. This survey disseminates through the different types of satellites and the multitude of approaches used to detect ships as well as the atmospheric difficulties that surround such satellite data. In regard to maritime object detection the more generic term of "vessel" is unused as it refers to other objects that are not distinctly ships such as floating docks and canoes. Locating ships in aerial imagery is typically performed using synthetic aperture radio (SAR) data [4, 5, 3]. SAR receives high degrees of accuracy since the data provided are not affected by sunlight or clouds. This is a particularly useful attribute as optical imagery from satellites often strug-

gle with clouds when processing the data and being able to distinguish them from highly reflective man-made surfaces [6, 7]. However, utilizing satellite optical imagery provides more identifiable information about ships [3]. Ship detection in coastal and open bodies of water has a wide range of applications [3]. Economically it aids in tourism and fishing. Security applications involve maritime safety with regard to traffic, piracy, irregular migration, border control, and illegal fishing. Environmentally, ship detection gives insight to maritime pollution and side effects on ecological habitats through traffic.

Using satellite optical imagery for ship detection with deep learning brings the accompanying issues around highly reflective objects being falsely identified as ships. This is prominent in cloudy scenes as well as scenes that contain ships very close to shore. Since optical satellite imagery can cover a massive amount of ground, the size of actual objects within these scenes are fairly small [8] which increases the difficulty of detection and also the possibility of false positives. This is amplified by complex backgrounds that include heavy cloud, choppy waters, and/or landmasses. Ship detection in the remote sensing domain is a very prominent problem [9, 8]. This problem has been approached by more programmatic methods and only recently has been tackled by use of deep learning.

Traditional, non-deep learning, approaches for ship detection have been more analytical using a combination of algorithms. Ground based Automatic Identification Systems (AIS) are typically used to give accurate ground truth (or ground control points(GCPs)) information about the location of ships [10]. AIS paired with satellite optical imagery can give a promising estimate of ships as used with Chinese GF-4 data from their geostationary satellite's near infrared band [10]. Typically rational polynomial coefficient (RPC) algorithmic approaches are used to isolate ships with AIS and imagery [10, 11]. These approaches manage to achieve 68.85% [11] and 74.5% [10] precision in detecting ships.

However, AIS relies on the ship to broadcast its location. If the ship were to exist maliciously, then it is most likely to hide itself from AIS to avoid policing. Therefore using AIS as GCP ground truth limits the capabilities of ship detection to a subset of total possible ships within satellite optical imagery

Previous work implementing object segmentation models (focus on U-Net) have seen accuracies reach 57% [12] and for the same dataset used in this work, 85% [9]. The work in [9] implemented F2 score for accuracy which is similar to IoU with more emphasis on the true positive examples, so it will tend to score higher. Work

implementing a VGG-19 plus custom filter selections manages to achieve precision of 93.53% in high resolution validation of ariel images [13]. However, this approach uses a threshold of 0.5 - 50% confidence - as well as inner-atmosphere imagery that does not endure the issues surrounding atmospheric corrections that satellite imagery has. A R-CNN model was implemented over sub-meter high resolution GF-2 satellite optical imagery and achieves 95.79% precision and an IoU of 0.3 [14]. Using such a low confidence level, or threshold, allows models to accept more false positives and reduces the distinction between ship and no-ship instances. Although, using sub-meter satellite optical imagery like GF-2 and GF-4 satellites allows for a much higher pool of features for models to learn from. High resolution satellite imagery tends to be more expensive as it requires much more processing to produce a usable image. Another approach implements other model types, as well as their own, that are typically used for object detection (YOLO, Faster R-CNN, SSD, etc) ranged from 62.4% to 88.3% accuracies [8]. This accuracy score was focused on the number of correctly classified ships in orientated bounding boxes (implementing an IoU threshold of 0.7 for the center point of a ship) and based on higher resolution images.

2.1.1 Ships and Algae

Environmental concerns surrounding the traffic and usage of ships directly affect algae blooms in case 2 coastal waters. This is seen in the transportation of harmful algae via ship ballasts [15, 16]. These algae being transported from one ecosystem to another and contribute to increased concentrations at the destination location. Algae defines many living organisms of which exists in both fresh, brackish, and salt water. The focus is on the salt water algae, specifically along the coastal region of Western British Columbia (B.C.). Over the last decade, the occurrences of harmful algae blooms (HAB's) have become significant to coastal ecological and socioeconomic members [17, 18, 19, 20]. Though thousands of microaglal organisms exists, about 100 [20] of these emit natural toxins under certain seasonal (notably weather events of spring and summer) [20] circumstances that can be harmful to humans and animals [17, 18, 19, 20], and of specific interest is the Amnesic Shellfish Poisoning (ASM) from the traces of Domoic Acid (DA) [19] or Paralytic Shellfish Poisoning (PSP) [20]. Coastal B.C. first found traces of DA in 1992 in razor clams, native littleneck clams, manila clams, horse clams, Pacific oysters, California mussels, blue mussels, geoducks, Dungeness crabs, and red rock crabs [19]. DA has become even more

widely distributed since [19]. PSP occurrences were first documented in 1793 at Poison Cove from contaminated mussels - resulting in five sick and one death (an onset of neurological symptoms resulting in paralysis or death through respiratory arrest [18]) [20]. In B.C. the most prevalent paralytic shellfish toxins (PST) are that of *Alexandrium* blooms [18], though no illnesses have been reported since 2005 due to biotoxin monitoring program effective closures of oceanic agricultural and fishery areas, and public education about the hazards of shellfish poisoning [18]. Closing down oceanic agriculture and fisheries causes the potential of bankruptcy, layoffs, and lost harvest [18]. Toxins can remain in seafood, even after toxin levels have decreased in the surrounding water [17]. This allows for toxic accumulation in local sea-life (fish, seal varieties, whales, etc) and specifically those that are of most concern exist in the human food chain. This is important since the biotoxins are not destroyed through cooking or processing of seafood [17].

2.2 Geo-spatial Data

Satellite data is collected in a swath process which manages to cover large amounts of ground in a single image as the satellite passes over [3]. There exist many types satellites: multi-spectral, hyper-spectral, and panchromatic [3]. These senors can be as basic as registering the red, green, and blue (RGB) channels, as typical optical imagery techniques use. At the other end of the complexity spectrum, multi-spectral sensors range over many bandwidths of the light spectrum. The more band widths the satellite has to offer, the more complex and bigger the processed data is. For example, Theia MRC data ranges from 1.2 GB to 1.7 GB of imagery data per scene once processed from raw unusable data; where a scene is a single instance of a swath. The two multi-spectral satellite data sources used in this work can be found in Table 2.2.

Satellite	Owner	Bands	Swath	Resolution
Theia	UtheCast	4	50 km	5 m
SPOT-5	CNES (France)	5	60 km	3-5 m

Table 2.1: Satellite and Band Count [1, 2]

In references to the size of satellite imagery: the amount of data to be processed analytically to make any inferences, classification, or object detection involves specialized techniques/approaches or a large amount of compute power. This is especially

the case for when scenes intersect, or a larger area (for example, the entire west coast of Canada) is analyzed. Again, this processing complexity is further increased by temporal spatial data where scenes from different capture times overlap.

2.3 Distributed System Scheduling

Distributed systems can be utilized in order to improve the computational power of a system [21]. By horizontally scaling the number of compute nodes, the amount of computational power increases. This allows systems to process a larger amount of data/tasks as compared to a single machine. Vertically scaling is typically more expensive, thus the focus on horizontal scaling. With the increase of nodes and compute power, there is also an increase in the issues surrounding distributed systems.

The general scheduling problem can be written as $\alpha|\beta|\gamma|\mu$ [22]. The first attribute, α, is a single entry and description of the execution environment. The attribute β can have from zero to multiple entries, and describes what is the processing to be done. The γ attribute details the description of the objective to be optimized. Finally, μ is the charging criteria of models adopted in the system: hourly based charge, an auction system, per-transaction charge, data-access charge, and so on.

By distributing computation, there is a need for oversight and regularization of tasks for the system to process, this is where scheduling comes in. Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over a given time period and its goal is to optimize one or more objectives [23]. Implementing a distributed system requires cost for hardware support and agreements on service expectations. Optimizing tasks through proper scheduling helps reduce the overall cost of computation while increasing the value customers receive.

2.4 Architecture

A system is centralized when it contains a central node or process that performs the necessary delegations to other nodes — this node is called the primary node [22]. The other delegated nodes are considered workers. Decentralized distributed systems offer a more obscure abstraction of primary/worker relations. A node may be a primary of a centralized subset of the network and simultaneously exist as a worker of the larger network, or it may be a worker in its entirety while existing at the same level as a

primary with its own worker pool. The primary nodes communicate between each other in order to achieve the system goal.

Another type of distributed system architecture falls into a hierarchical structure. This hierarchical distributed system offers a tiered layer where a node may communicate with only its predecessor or successor (either one level above or below).

The organization of the architecture may play a role in the scope of the scheduler. Not only should we consider the architecture connectivity of primary, workers, and their resources but also policies and access controls when implementing a scheduler. In a cloud based environment, these access controls limit the network and physical resources a worker can communicate with.

2.4.1 Clusters

A cluster, in this realm, is a collection of dedicated computational hardware that can communicate; usually over a dedicated private network (but distributed systems also exist over public networks with heightened security protocols) [22]. Careful consideration towards task allocation can have immediate effects on the efficiency of a system. The type of system is what dictates the best scheduling approach. In a synchronous scheduling of tasks, the network is only as fast as its slowest worker. The same applies to asynchronous architectures, but typically with a faster rate of overall system completion. The rate of completion also depends on the nature of the task itself. If tasks are not dependent on each other, then the system may progress concurrently. However, if tasks do depend on each other, then the system may experience a varied progress rate where at times it may be zero.

2.5 Tasks

A task is a logical unit that makes progress toward a unified goal of a system [24]. One such set of tasks could be images that a distributed model is learning or predicting on. Another such set of tasks could be a set of computations offloaded to multiple nodes in small chunks and sent off to workers to be processed. A task may end at the worker node — perhaps a state change — or a response may be communicated back to the primary. There are two types of tasks: dependent and independent.

Dependent tasks rely on the response from a task ahead of itself before they can be executed. These tasks are better suited for a synchronous system that waterfalls

from one task to the next depending on dependency. The system makes progress at a more determinable rate due to the predictive analysis of task succession.

Independent tasks can be run in parallel and/or concurrently. These tasks run best in an asynchronous system. The downfall to consider is resource contention if a particular set of tasks require the same network resource (say, accessing a file on a shared storage) — this situation has workarounds: eventual consistent databases/servers. The system makes unified progress at an undetermined rate as tasks may run simultaneously toward the system goal.

2.6 Task Allocation Through The Scheduler

The process of managing task allocation, to where and to whom, is the responsibility of a scheduler. The scheduler of a distributed system performs akin to the process scheduler on any operating system.

Task-graph scheduling is considered an NP-hard problem [25]. Scheduling tasks from the primary to workers can occur at multiple stages. There can be global scheduler that directs all tasks to all connected workers. There can also be local schedulers that handle incoming and outgoing tasks/replies on both the primary and worker nodes. Organizing this dynamic scale of events is a major component of the system efficiency. In a centralized distributed system, it is more common to see a global scheduler managing the allocation of resources and tasks between primary and workers. However, in a decentralized distributed system, there may exist multiple schedulers that are responsible for part of the overall system. Management of this decentralized state may be distributed over resources. Each scheduler in a decentralized system can be considered the same as that of the scheduler in a centralized system. Schedulers are considered to follow one of two main categories: List or Cluster scheduling [24]. List scheduling is composed of two parts: a priority/ordering of tasks and a mapping to a processing node. Cluster scheduling extends List scheduling by mapping collections or sets to processing nodes.

2.7 Cloud-Based Scheduling

Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS) are common models for user services and each have their own scheduling requirements [22].

IaaS — the most popular in cloud environments — offers physical and virtual environments to deploy computations and/or applications at scale on a public or private based setting. Some of these features are restricted through access controls that are subject to cost-per-use or subscription-based. For IaaS, the scheduler is responsible for "allocating virtual machine requests on the physical [architecture]" [22]. For PaaS and SaaS, the scheduler is responsible for allocating the aforementioned applications and/or computational chunks to workers in order to be executed/served.

Part of optimizing the objectives of cloud-based scheduling includes the Service Level Agreement (SLA) of users with their end clients (workers). The throughput and latency of a system directly affects users first and third party SLA agreements — these SLA agreements can be synonymous to Quality of Service (QoS) expectations.

To achieve high SLA/QoS, a scheduler needs to be robust in order to reduce execution time and cost. A robust scheduler must be able to handle a change in application topology (new instances dynamically being added and removed from the pool), resource/software configurations, input data, and data sources. This is not a comprehensive list of possible dynamics a scheduler should be robust to. One of the biggest nuances of distributed systems is the ability of a schedulers' robustness against failure occurrences.

Handling failure occurrences is a major proponent of distributed systems. The scheduler must be able to salvage, redistribute, and/or re-implement these occurrences in order to maintain system QoS. Granted, if a failure occurrence is large enough, there will be interruptions in service. However, if a failure occurrence is small scale and handled properly, an end user may never even notice.

2.8 Distributed Deep Learning Scheduling

There are many kinds of scheduling algorithms that each satisfy different constraints of a system. Choosing the most effective algorithm ensures the optimization of the system goals are achieved.

During prediction, production models will most likely be served in an app on a scalable cluster. This cluster can be either on a heterogeneous or homogeneous system and tasks will be allocated with the above type algorithms.

However, Machine/Deep Learning task allocation during training are more specifically scheduled depending on the architecture of the framework. Framework architectures may be decentralized, such as AllReduce, or centralized, such as the Parameter

Server. The following two algorithms are focused on the Parameter Server architecture. They aim to reduce the limitations imposed by straggling workers (workers that have fallen behind the others progress rate).

2.8.1 Cyclic Scheduling

Each worker has the same task ordering as other workers, however the start task is shifted by a cyclic shift operator [26]. This repeats for consecutive workers. This allows workers to minimize the contention on resources so they minimize deadlocks. Naturally a learning schema would likely have more than four tasks to allocate, so having more than four workers is possible. The number of workers is limited to the number of tasks. However, this can be exploited by limiting workers to receive a subset of tasks, then the upper bound on number of workers becomes the total number of cyclic orderings of tasks. An example of Cyclic Scheduling can be seen in Table 2.2.

Worker	Task Order
0	1,2,3
1	2,3,4
2	3,4,1
3	4,1,2

Table 2.2: Cyclic Ordering Example

The task ordering matrix is given by the following form in Equation 2.1.

$$C_{CS}(i,j) = \text{g(i + j - 1) for i } \varepsilon \text{ [n] and j } \varepsilon \text{ [r]} \tag{2.1}$$

Where n is the number of workers and r is the number of distinct entries in each row. The function g is a mapping of integers to integers defined as follows in Equation 2.2.

$$g(m) = \begin{cases} (m, if 1 \leq m \leq n \\ m - n, if m \geq n + 1 \\ m + n, if m \leq 0 \end{cases} \tag{2.2}$$

2.8.2 Staircase Scheduling

The Staircase scheduling algorithm introduces an inverse computation on the task orders at the workers [26]. This task ordering matrix is given by the following form in Equation 2.3.

$$C_{SS}(i,j) = g(i + (-1)^{i-1}(j-1)) \tag{2.3}$$

Where i is iterated through the number of workers and j is iterated through the number of distinct entries for each row. An example can be seen in Table 2.3

Worker	Task Order
0	1,2,3
1	2,1,4
2	3,4,1
3	4,3,2

Table 2.3: Staircase Ordering Example

2.8.3 Overcoming Stragglers

When provided with tasks, workers sometimes either become corrupt (maybe by a broken process) or another process has started (out of the control of the user and worker node) that demands more computational power that causes the computation of the task to take longer; these workers are referred to as *stragglers* [27, 26]. Perhaps it is the task itself that is computationally demanding that causes a worker to take longer. In other (probably most) cases, it is the network that bottlenecks task completion communication between the worker and primary [21].

When one (or more) workers begin to experience these disruptions, they fall behind the other workers in terms of progress over assigned tasks — these are considered straggling workers. The product these straggling workers output may become stale if the application requires operations on current data and therefore useless. If stale data is used there is the possibility that it will hinder the progress made by other workers.

Persistent Stragglers

It's fairly intuitive to reason about workers that are persistently straggling — they fell behind and are in a constant state of catch up and are persistently falling behind due

to lack of computational power/network bottlenecking compared to other workers. For example, a worker is consistently slower at processing an image than the other workers. Due to their persistent behaviour, solutions to overcoming this deficit can be more easily analyzed.

Non-Persistent Stragglers

Non-persistent types of straggling workers are harder to predict and analyze solutions for [26, 27]. Non-persistent straggling workers are intermittently straggling but may not always be in as much of a deficit as compared to other workers as they may complete a significant portion of the assigned tasks by the time other workers have completed theirs. For example, an intermittent process keeps locking up resources that the worker is trying to use for the assigned task.

2.9 Parameter Server

A parameter server is a cluster involving a primary and worker processes. The primary process can be a single instance or belong to a group of gradient servers. The workers provide gradient updates that publish to the gradient, or primary, server to progress the global model. They are also subscribing the global model weight updates from the primary server to make forward passes on.

The architecture of a parameter-server system, typically centralized, can involve multiple layers of server groups, schedulers, and workers [28]. This architecture is used to leverage more compute power against massive datasets, such as datasets found in industry that can go from TBs to PTs [28]. The worker processes will read the current state of the global model (weights), the position in the dataset, and the end point for the requesting worker. The scheduler typically gives each worker process an up front chunk of the training dataset to iterate over. Once the worker process has this information, the workers are used calculate gradients of the global model. Once calculated, the worker will push the gradients to the primary process, wait for the update to occur, and then pull down the most current version of the global model. This process will continue until the dataset has been iterated over a set number of times. The push-pull process has been improved over the years to permit asynchronous requests [29, 28]. Attempts to provide better solutions for convergence with parameter servers involve a process that makes use of accumulated

gradients [30]. Due to the nature of the push-pull communication between server groups and workers and the delays for computation of the workers and updating of the global model at the primary server, the workers may pull stale weights. Parameter servers do not typically converge at the same rate as synchronous SGD approaches and requires further research into more efficient asynchronous approaches and focus on computational loads [31, 32].

2.10 Chapter Summary

Tackling the issues surrounding ship detection in satellite optical imagery with deep learning implements different approaches that are challenged by false positive occurrences. The ecological, environmental, financial, and safety concerns give motivation to find solutions that focus on the application of ship detection. Deep learning approaches have focused on non-rotated and rotated bounding boxes as well as U-Net mask segmentation within satellite optical imagery with greatly varying accuracies. Making use of expensive satellite data requires considerations toward cloud-based solutions such as distributed approaches for task scheduling and parameter servers. This brings their own set of issues to the bigger problem, but recent works aid in minimizing the carryover. The following chapter provides the methodology for the experiments that leverage this background information.

Chapter 3

Methodology

Two architectures were explored as part of this process: a single node and a variant on the parameter server architecture that leverages gradients from workers at the primary node. Both architectures trained the same custom U-Net model with the goal to achieve ship segmentation masking on satellite imagery. Additionally, a classifier was refined on the training dataset and implemented to build future commercially usable datasets made from the target dataset distribution.

3.1 Custom U-Net Model Architecture

The custom U-Net model in Figure 3.1 was implemented with the PyTorch deep learning framework and can be thought about in two sections: encoder and decoder. It goes an extra layer deeper than the traditional U-Net model [33] in aim to extract more features from the images. This increases the complexity of the model and aims to capture more distinct features in examples (see Table 3.1). Also implemented are bottlenecks in the encoding blocks. These bottlenecks are used to gather the best features of the distribution and therefore minimize loss. Regarding the implementations of the bottleneck in the encoder block, the input channels are subtracted from the output channels as a means of dimensionality compression to gain the most relevant features through a 1x1 kernel (or filter), then processed via a 3x3 kernel to the same number of reduced channels before being processed to the target number out channels via 1x1 kernel. The reduction in dimensionality before the expensive 3x3 kernel sized convolution helps reduce the computational resources (number of parameters to process) from increasing too much. The numbers above each box in

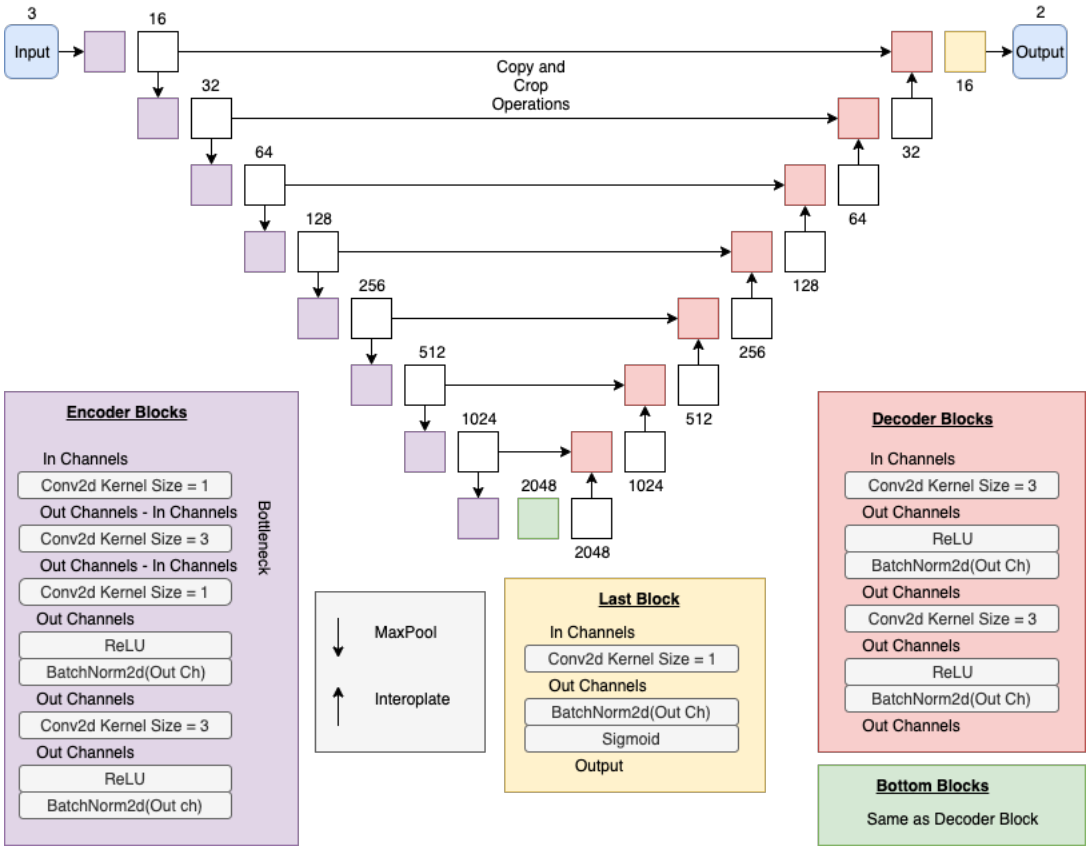


Figure 3.1: Custom U-Net Architecture

Figure 3.1 indicate the number of channels/layers the tensor has at that layer of the U-Net model. As such, the input images are RGB and therefore 3 channels : (3,X,Y). This is run through an encoder block and transformed into a deeper feature tensor (3 channels to 16 from input to the first layer of the U-Net). This transformed layer is then put through a 'MaxPool' convolution to the next U-Net layer and the process repeats until the bottom of the U-Net where the features of the images are encoded into the 2048 channel tensor. The bottom layer at the end of the encoder holds the reduced dimensionality feature space of the input batch. It runs the tensor through a 3x3 kernel 2D convolutional layer twice to extract as many features as possible before being interpolated up a U-Net layer to the start of the decoding block. At each interpolation, the equally deep encoded layer is cropped and copied over (to enhance the segmentation to the objects that are to be detected) and then put through the decoder block convolutions. This process is repeated up until the top layer of the U-Net model where the tensor is refined through a 1x1 kernel 2D convolutional

layer to the number of classes requested and passed through the sigmoid function to normalize the output values between 0 and 1. This gives a probability for each pixel of what the U-Net thinks is the desired object class. The architectures used in this research utilized stochastic gradient descent as the optimization method for model convergence.

Model	Trainable Parameters
Original U-Net	31,042,434
Custom U-Net	306,270,726

Table 3.1: Trainable Parameters of U-Net Versions

3.2 Cloud Environment

The following work was performed on Microsoft Azure cloud computing resources. This cloud environment imposes cost-related considerations that are determined by pay-as-you-go pricing. To eliminate inter-regional data transfer costs, all resources used were created and deployed in the US East region to satisfy available virtual machines that had GPU's for deep learning training.

3.3 Azure Fileshare

An Azure Fileshare is a structured file system that can be mounted on to virtual machines [34]. Its purpose is to host data that can be replicated across multiple virtual machines or server as a single point of data storage in a network.

3.4 RedisAI as a Data Server

Delivering data to the training cycle was handled by a Redis connection client. A module of Redis, RedisAI [35], that within it's datatype structure can manage tensors via the Redis standard key-value approach is used as the server structure. This is useful to communicate large amounts of tensors back and forth since Redis can handle massive amounts of requests - it can easily handle 10,000 clients as of Redis version 2.6 [36]. Leveraging RedisAIs in-memory caching system allows for training data to be delivered rapidly. One downfall to this approach is that task delivery is limited

by the network bandwidth as with all network bound communications. However, this delivery service allowed for the training data to be available and in a ready format in memory without the use of file I/O operations. This was particularly useful as the single node architecture was trained concurrently with the following parameter server variant with both using the same data server. Also, this method of data delivery allowed for concurrent reads of key-values. This means that, in a parameter server architecture, any worker may read the global model weights concurrently as another worker. This eliminates downtime due to resource locks. In an Azure virtual private network, specific inbound port rules have to be assured to prevent malicious interaction/use from outside users.

3.5 Single Node (SN) Architecture

The single node architecture is a typical training scheme deployed over a single Azure NC24 virtual machine. This virtual machine is host to four Tesla K80 GPU's with 12GB memory for each GPU (48 GB total) and 220 GB of RAM [37]. To take advantage of four GPU's, the PyTorch data parallel functionality was implemented. This distributes the model across all four GPU's and delivers equally divided batches to each GPU. With this process, there is the upfront overhead of loading model replicas to each GPU, but once loaded they stay in GPU memory for the training session. Retaining GPU memory allows for quick updates to GPU-local models. The GPU-local worker processes the batch and returns its gradients for which the main loop will average across all GPU's for the backward propagation of the model. This is a fairly standard approach to training deep learning models.

Utilizing the Azure cloud environment, a Fileshare is mounted to the virtual machine to allow for model and metric save points that can be accessed if the VM were to die. An overview of the SN architecture can be seen in Figure 3.2.

3.6 Parameter Server Variant (PSv) Architecture

The single node deployed model is extended by being distributed over a network of nodes. This architecture compromised of five Microsoft Azure NC series virtual machines connected over a private virtual network. Each one mounting the same Fileshare for model and metric save points. It was implemented as a centralized distributed system, where one primary node hosted the global trainable model and

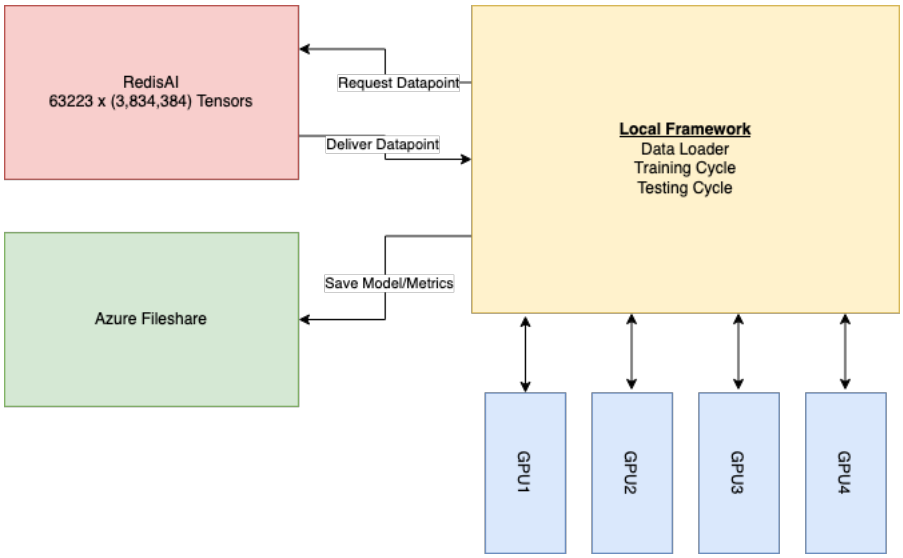


Figure 3.2: Single Node Architecture

the registered workers would implement gradient training to return to the primary node and global model.

With PyTorch offering built in functionality to easily return a models current state in an ordered dictionary, uploading this information to a data server was simple by using the name of the parameter as the key and the data as the value. The data server was used to host the training and test datasets in memory for fast access. It also hosts the global model values for all updated layers as well as hosting each workers gradients for the parameter server to utilize. An overview of the PSv architecture can be seen in Figure 3.3.

The primary node was implemented on a NC24 virtual machine which hosted four Tesla K80 GPU's, just like the single node architecture. This VM also hosted the primary process as well as the scheduler and the data server (due to the large amount of RAM available it was cheaper than deploying on a memory optimized virtual machine or separate VMs for each process). Each worker was implemented on a NC6 virtual machine which hosted a single Tesla K80 GPU with 12 GB of GPU memory. The connections are identical for each worker despite only Worker_001 is shown to be connected in Figure 3.3.

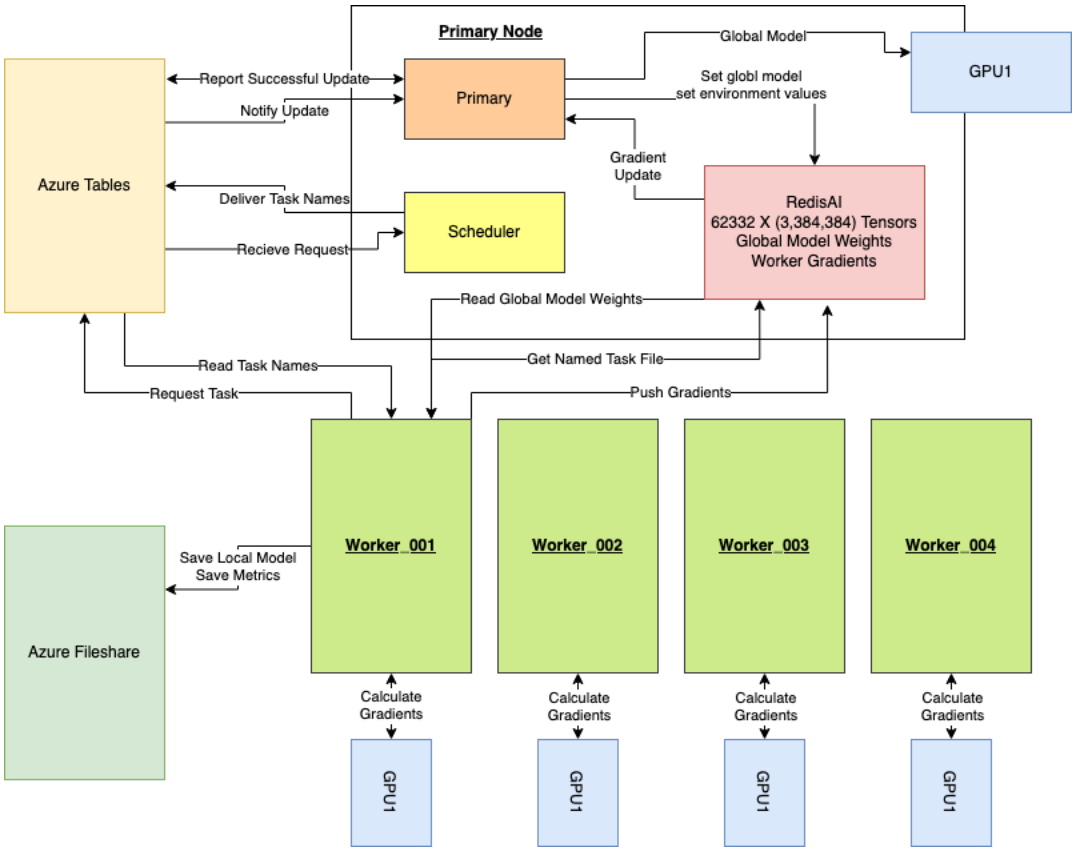


Figure 3.3: Parameter Server Variant

Primary

The parameter server is a variant of the standard approach by leveraging the gradients the workers pushed. These gradients are leveraged by processing a single batch forward pass which maps the workers accumulated gradients into a back propagation function before an optimizer step that updates the global models weights. Since parameter servers are known to take longer to converge, this approach aims to provide a more directional nudge towards convergence and minimize time for results. This process allows gradients of other possible solutions to steer the global solution and act as a boosting mechanism.

The initialization process involves the primary uploading the initialized global model to the data server for all workers to set their local models. After initializing the environment (total epochs, learning rate, batch size, momentum, and weight decay that the workers read as well as the global model), the primary waits for worker updates to come in by querying the update table in Azure Table Service. Once worker

updates are received, the key passed in the table is used to pull all the gradient values from the data server. These values are set into the global model and then deleted from the data server for memory management as to ensure stale gradients are not accidentally read at a subsequent interval if a key corruption occurred. The primary node then processes a single batch which accumulates and leverages the workers gradients and links to a back propagation function for the model before the global model weights are updated. This step is what varies from standard parameter server architecture. Once updated, the global model new weights are then uploaded to the data server and the recent worker is notified that their update has been processed via table communication. The worker will then read the global model, while the primary continues waiting for new updates from other workers. Unlike the single node architecture, the primary did not utilize a learning rate scheduler as the gradient accumulations from workers were the effective steps toward solutions and the primary would not have sufficient steps to imitate the behaviour of single node architecture. Instead, the U-Net had to modify the momentum of its 'BatchNorm2d' layers. The momentum had to be increased since 'BatchNorm2d' layers do not properly accumulate gradients with its default value. This increase of momentum aims to make the updates of the running statistics of the batch smoother.

Workers

Each worker is used to accumulate gradients over a set of batches from the training dataset. Initially each worker will register on a table that is meant to act as a heartbeat. Before worker startup, the primary node will have pushed the starting point for the global model - the weights that it was initialized with. On startup, the workers will read these weights from the data server and load them into their local model. This will also work at subsequent registering if a worker were to join later in the training process pulling the most recent global model weights pushed rather than the initial weights. This process is accomplished by a request propositioned in the request task table, which the scheduler will read and reply with a batch of keys (the names of the files) for the data server. Once the task has been set, the worker will read over the list and build an input batch. Next the workers will iterate over a set of batches (mini-batch) that they have requested and had delivered from the scheduler. This process allows for the workers to explore different solutions with their currently loaded weights. The worker processes this mini-batch with no optimization

step (updating model weights) to allow for gradient accumulation - this approach simulates larger batch sizes when there are limitations on memory. Once the mini-batch is processed, the worker then uploads the local models gradients and signals through the Azure Table service that there is an update ready. The worker then waits until its update has been processed before reading global model weights. Since any non-waiting worker may be reading weights while the primary is updating these same weights, there is no guarantee that the weights the worker is reading are the most up to date. So, each worker will read the weights within different stages of the global models life-cycle and may be up to the number of workers updates behind in the worst case scenario. Therefore, the workers perform lazy-learning of global model weights. Once the local model is updated with the recently read global model weights, the training cycle continues another iteration. Over each iteration of the training dataset, each worker will have observed a quarter of the dataset. When this occurs (as there are four workers and this signals the end of an epoch), the worker will save the current metrics and local model for redundancy.

Scheduler

This scheduler operates under the 'Cluster' category by assigning a collection of keys and mapping it to a worker. The scheduler also keeps track of the current positioning of the dataset. It receives requests from workers for the next batch that are read from the request task table, to which it will deliver the keys of the datapoints in that batch. Once delivered, the positioning in the dataset is updated, as well as the progress of the number of epochs iterated through. At the turn of each epoch, the scheduler shuffles the dataset so workers will not receive identical batches.

If a worker were to disconnect and reconnect, the data it contained would be lost and forgotten since it would at most be a single batch. When it reconnects, it will receive the next batch in the workflow that the other workers had reached whilst the disconnected worker restarted. This eliminates large progress losses from lost workers since the architecture progresses concurrently.

3.7 Offline Hard Example Mining (OffHEM)

There are a few different approaches to bootstrapping [38]. As a post-training refinement for the model, this process focuses on hard examples. During this pass, the

model is in evaluation mode and does not learn, it simply assesses the loss for each datapoint individually through the training dataset. Using a threshold of 20% all keys with losses that are in the top 20% range of the previously calculated losses are then joined to the original dataset to create a new training dataset that repeats the top 20% hardest examples. This is decided that the higher the loss value, the harder the example is for the U-Net model to correctly create a segmented mask for. The model is then refined in a single iteration of this process. The entire OffHEM process is repeated ten times.

3.8 Training Dataset

The term layers and channels are synonymous in regard to the following description about the datasets. The training dataset is composed of 63,223 datapoints and built from part of the Kaggle Airbus Ship Detection competition. Each datapoint is a (3,384,384) image (RGB) that contains a ship that was taken by the SPOT-5 satellite [2] and organized by AirBus [39]. This dataset was of the resolution five to three meters, meaning each pixel represented anywhere from five to three meters on the ground dependent on the quality of the post-processing. This had to be re-sampled to five meter resolution in order to match the distribution of the target dataset distribution: UrtheCast Theia MRC satellite optical images.

Originally the dataset contained 48 thousand (3,768,768) ship chips and 128 thousand non-ship chips. However, regarding ship and no-ship as the classes, the original dataset was severely imbalanced. As of this, the 128 thousand non-ship chips were discarded and the remaining 48 thousand were segmented into quarters - resulting in the 63 thousand ship chips of size (3,384,384). The segmented chips that did not contain a ship were also discarded like the other non-ship chips. The chips were quartered and not resized as to retain the correct resolution and not to compress already small ships into meaningless blurs. Even in the newly created ship chips, the class imbalance of ship to no-ship is drastic. Only 1.035e-6% of the overall dataset contains ships. The remaining pixels are no-ship. Examples of the ship chips used for training can be seen in Figure 3.4. Each chip contains the original RGB channels as well as a segmentation mask. This segmentation mask is a [0,1] classification of no-ship and ship classes respectively. In order to combat this, a Focal Dice Loss was implemented for the model which focuses more on the smaller classes within data for object segmentation [40]. This loss is a combination of Focal Loss and Dice Loss

that down weights larger classes. Where Focal Loss is a log space version of Binary Cross Entropy and Dice Loss gauges the similarity of two sets (predictions and labels) [41, 42].

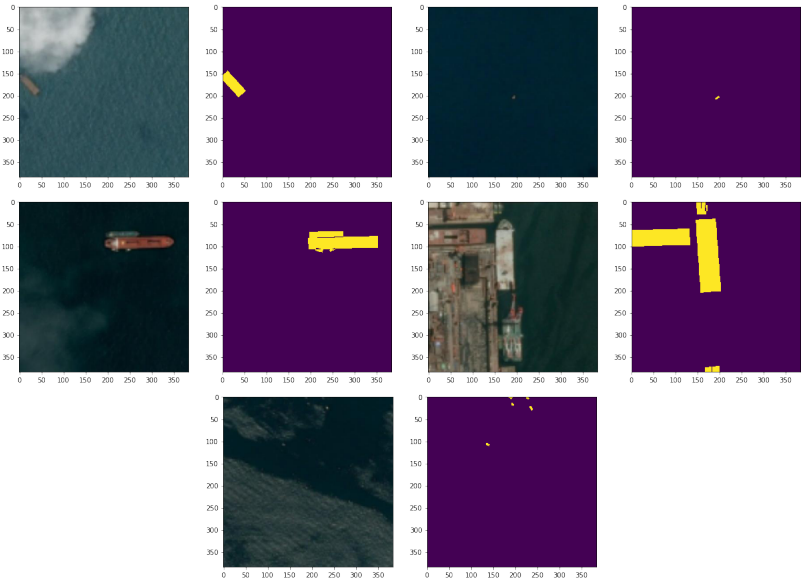


Figure 3.4: SPOT-5 Ship Chips and Label

Additionally, the dataset underwent augmentation: vertical and horizontal flipping, and no augmentation. At each batch process, a random augmentation was performed on the chips. This effectively triples the size of the trainable dataset the model sees.

3.8.1 Intersection over Union

The ground truth labels (see Figure 3.4) associated with the optical images are box segmentation masks that are slightly larger than the actual ship. This provides a general area for the U-Net model to learn during the training process. To measure the accuracy of the training/test predictions to the ground truth label an IoU (Intersect over Union) score is calculated.

$$IoU(A, B) = \frac{A \cup B}{A \cap B} \tag{3.1}$$

Predictions assessed by this method of accuracy are performed with a threshold of 0.9 meaning that all prediction values over 0.9 are pushed to 1.0 and are otherwise set to 0.0. IoU tends to lean more toward the worst-case scoring of a comparison,

meaning it penalizes single instances more than a more average approach over predictions. This is in relation to an F2 score which puts more weight on the true positives. However, during accuracy assessments, the U-Net theoretically could fit a segmentation mask tighter to the actual ship and therefore result in a lower accuracy despite a having higher precision in reality. Also, IoU will focus only on true positives and will not account for other class classifications including false positives and true and false negatives. This must be taken into consideration when evaluating the accuracy of the U-Net. To record a more accurate precision on test predictions between classes, a confusion matrix is calculated to properly inform about true negatives, false negatives, true positives, and false positives across both classes (ship and no-ship). Calibration of this confusion matrix to assess class accuracies uses a threshold of 0.9 of the predictions provided by the model against the ground truth. This sets a confidence of 90% in predictions. This threshold is higher than that of the related works and aims to provide a more strict classification between classes.

3.9 Target Dataset

A target dataset is built using the optical imagery generated by Theia, the MRC satellite optical camera [1]. It is a five-meter resolution camera, meaning that each pixel represented five meters on the ground. This dataset consisted of 79 chips that were built from an MRC scene over Burrard Inlet, Vancouver, British Columbia seen in Figure 3.5. The reason that this dataset is not used for training is that it is extremely small. Theia MRC data has never been used for this use-case (ship detection) and is desired for commercial usage. In order to use this data as a commercial product, two obstacles must be overcome: grow a trainable dataset and show that this data can be used for the use-case of ship detection.

This small target dataset was built with both a (3,384,384) and (4,384,384) shape - RGB and RGB with near-infrared layers respectively. For this work, only the 3-channel optical images will be considered. Creating a dataset from a large scene into (3,384,384) chips was accomplished by windowing the scene in Figure 3.5 and saving the chips. After this step, the ships needed to be labeled. Each ship in each chip was manually labeled using GIMP [43]. If a chip did not contain a ship then it was discarded. This was a lengthy and manual process. This dataset was used for testing against an unseen satellite dataset in order to determine the generalizable capabilities of the trained model. Examples of the four layered chips can be seen in Figure 3.6.

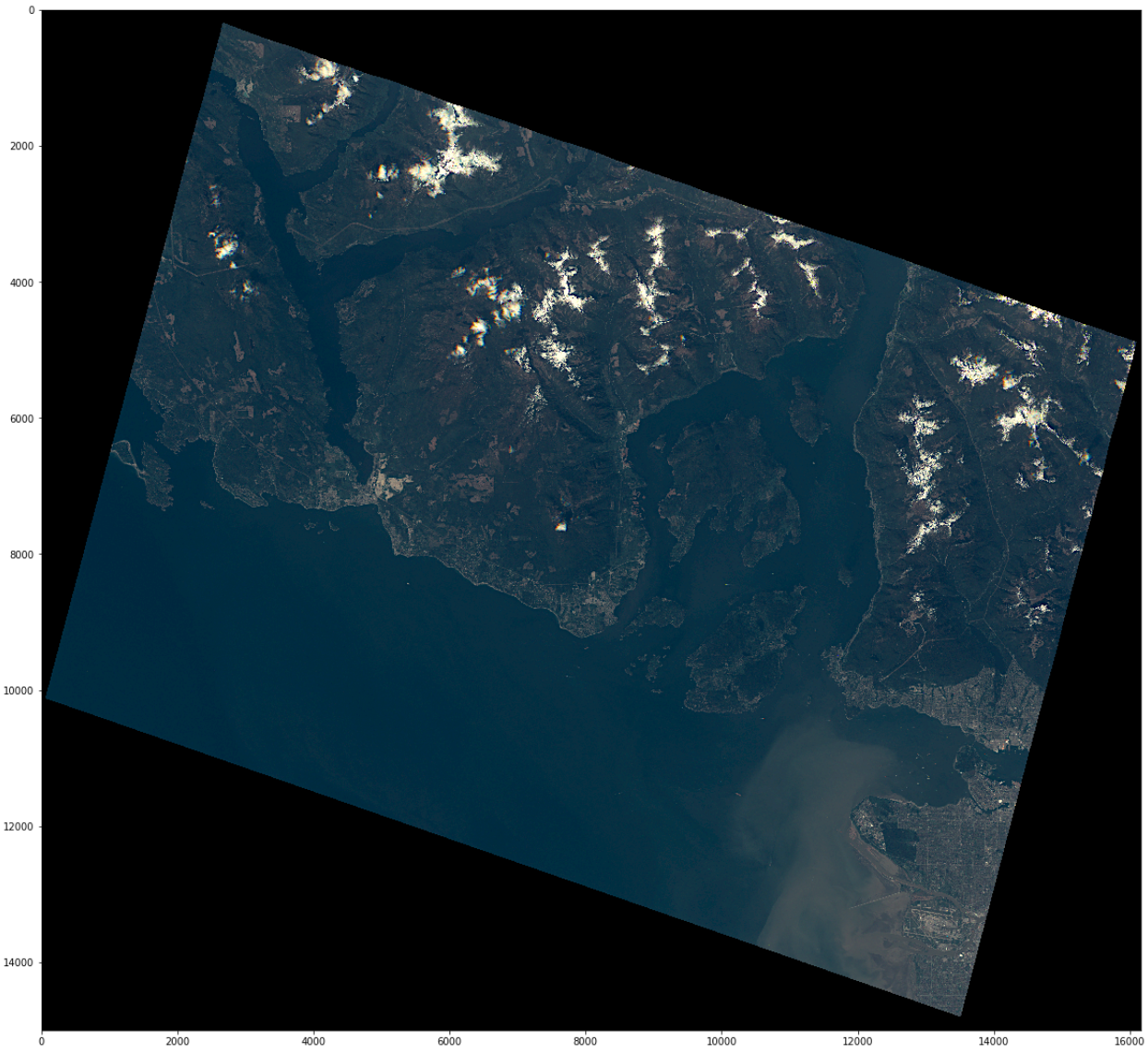


Figure 3.5: Burrard Inlet, Vancouver, British Columbia, MRC

Like the training/validation dataset, the ship class is severely imbalanced and only accounts for 0.2508% of the dataset.

3.10 Ship Classifier

To further explore the capabilities of processing large GeoTiff satellite scenes, and to eliminate wasteful ship mask prediction processing, a refined pre-trained ResNet101 classifier was refined and employed. This refined classifier was passed (3,384,384)

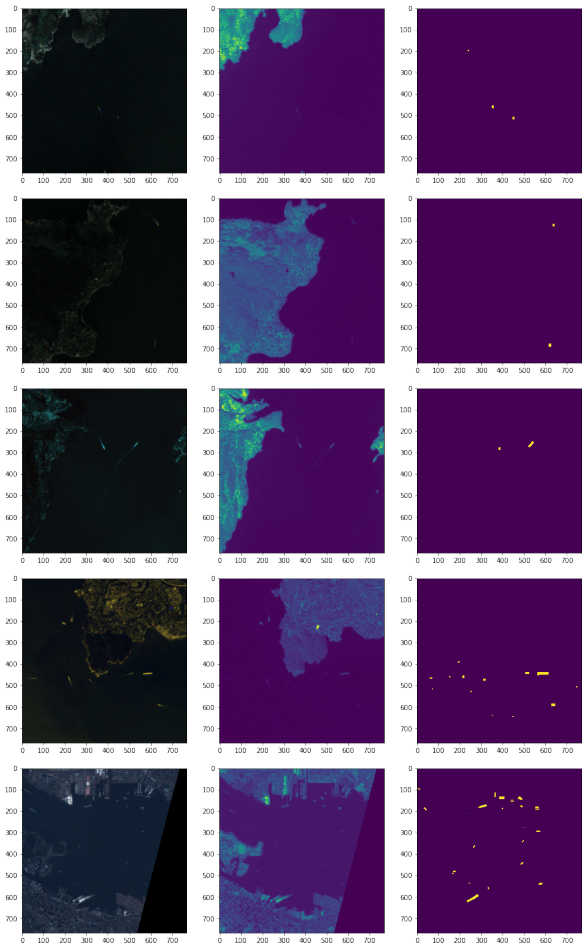


Figure 3.6: Theia MRC Ship Chips

windows from an entire satellite imagery scene to determine if a ship was within this window. To eliminate false positive from similar objects close to or on land, a water mask was implemented that isolated the water and eliminated non-water bodies from the classification process. A single Theia MRC scene is of the shape (4,14999,16156) (RGB and Nir layers) and the scene used for windowing can be seen in Figure 3.5.

A pre-trained (on ImageNet) ResNet101 classifier from the PyTorch library was loaded and refined on the same training dataset as the U-Net model with the difference that the accuracy was calculated on the label and output sum for each pixel across all layers instead of IoU. The pre-trained classifier was refined on a sample size of 2000 SPOT-5 chips (65% contained ships and 35% contained no ships). Once refined on the distribution that the ship chips belong to, it was then used to predict on the target Theia MRC chips.

Predictions made on windowed and water masked Theia MRC chips are given in Figure 3.7.

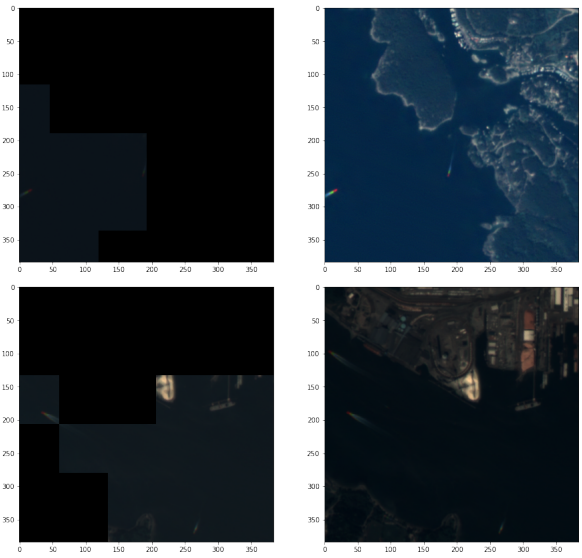


Figure 3.7: Water Masked MRC (3,384,384) Window Chips

The purpose of this classifier is to determine window chips containing ships that can be used to both enlarge the current Theia MRC ship dataset size for training and also to allow for ship masking by the U-Net for chips that only contain ships.

3.11 Chapter Summary

Implementing a custom U-Net model that is deeper and implements feature enriching approaches aims to increase the quality of class predictions for ship detection. This model is trained over two systematically different approaches which leverages a Redis based data server. These two approaches involve the standard training mechanisms over a single node and a PSv over five distinct virtual machines. The PSv employs workers as a boosting mechanism for a global model at the primary process/node. The following chapter documents the results of experiments from these approaches.

Chapter 4

Results

The two architectures and custom U-Net model underwent experiments to determine the best approach for ship detection. The data server was also tested against traditional file I/O operations in order to identify its validity for use in the architecture experiments. The SN architecture went through three experiments training a subset of the training dataset (10k and 20k) over 30 and 100 epochs to comply with time and cost restricted cloud-based training. The PSv architecture performed a single experiment to be compared against the SN training performance over a subset of the training data (10k) and 30 epochs. To possibly refine and increase the accuracy of the models, the OffHEM approach is applied to the best performing SN model and the PSv model.

4.1 Data Serving

Unlike traditional imagery learning, both frameworks utilized a RediaAI server to load chips. One of the most effective ways of serving traditional files for deep learning is using a HDF5 format. Each chip was saved as their own individual HDF5 file originally before being uploaded to the RediaAI server. Since a PyTorch data loader will typically open individual files and concatenate them into a batch that is served to the model, a test against RedisAI tensor serving was performed.

For this test, each method of delivery iterated through individual files (both identical Numpy array format) and converted into float tensors that a model could receive. However, the model step was skipped as to just compare file serving methods.

It is intuitive that utilizing the data server is the more effective approach since

Method	Time
RediaAI	0:02:32.101122
HDF5 I/O	3:37:02.746008

Table 4.1: File Serving Time Comparison

it processed all 63 thousand chips within a few minutes, whereas HDF5 I/O took over three hours. The speed at which data is read to the model pairs effectively with PyTorch DataLoaders which, when more than one is specified, implement multi-process batch gathering. The use of a data server increases the usage of the GPU to 95-99% consistently as per the nvidia-smi command, whereas file I/O would see dips to 0% (although microseconds) when workers are not able to process an item or are limited by the number of cores versus the batch size in relation to the overhead of file I/O.

4.2 Single Node Architecture

During training, hyperparameters for Focal Dice Loss and a learning rate scheduler were adjusted through micro-testing. The Focal Dice Loss hyper parameters alpha and beta were set to 10.0 and 2.0 respectively. When the alpha parameter was lower than 10.0, the loss function would not pick up on the smaller class size. The learning rate was set to 1e-2, momentum to 0.9, and weight decay to 1e-5 as the hyperparameters for training. A PyTorch cyclic learning rate scheduler was also implemented with the minimum as 1e-2 and the maximum as 1e-1 in order to prevent the model from getting stuck in local minimums. At the end of each epoch of training, the model was saved to a mounted Azure Fileshare. The SN architecture was trained over dataset sizes of 10 and 20 thousand for 30 epochs, and there was also a run with dataset size of 10 thousand over 100 epochs. The focus on using a subset of the data is that of time and cost constraints using virtual machines in the Azure cloud space as well as the aim to use OffHEM to further refine the model which offsets the early stopping of training. The focus of training 10k datapoints over 100 epochs instead of 20k over 60 (to reach the same time frame) was to achieve a more refined model over more epochs and then implement OffHEM to refine nuances further.

The training and validation metrics, seen in Figure 4.2, show the training loss and IoU accuracy for each approach. Implementing 10k datapoints over 30 epochs achieves 5.83 loss and 47.8% IoU accuracy. With the 20k datapoints over 30 epochs,

the model achieves 5.83 loss and 52.8% IoU accuracy. Finally, the 10k datapoints over 100 epochs model achieves 5.80 loss and 56.2% IoU accuracy.

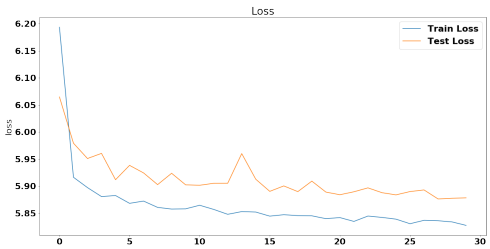
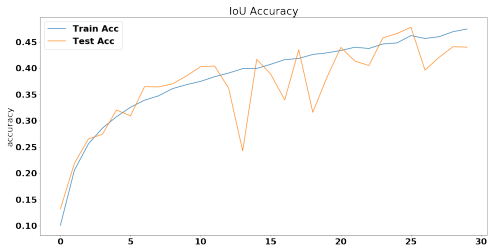
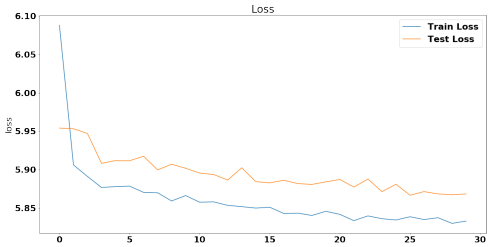
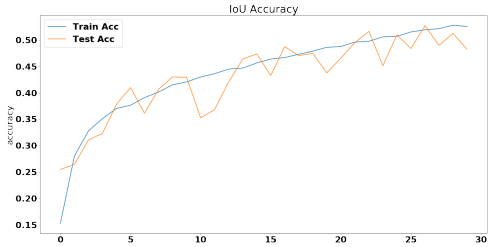
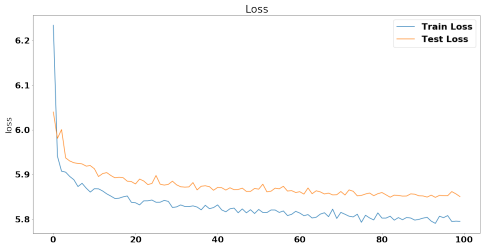
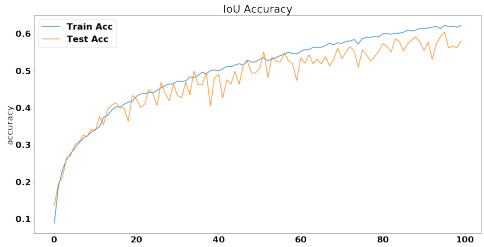
10k Dataset and 30 Epoch		
20k Dataset and 30 Epoch		
10k Dataset and 100 Epoch		

Table 4.2: SN Architecture Loss and IoU Accuracy

Over the training cycle, various metrics were recorded to give insight to the performance of the architecture. One metric involved the time in seconds it took to complete each batch, seen in Table 4.3. This process involves retrieving the batch items, forward passing through the U-Net model, calculating loss and accuracy, performing a backward pass through the U-Net model to calculate its gradients, and finally performing an optimization step to update the U-Net weights ready for the next batch.

Approach	Min	Mean	Max	Variance
10k dataset and 30 epoch	2.06	2.16	83.19	0.82
20k dataset and 30 epoch	2.07	2.16	84.83	0.45
10k dataset and 100 epoch	2.08	2.18	84.93	0.35

Table 4.3: Time Taken to Prepare and Train a Single Batch (seconds)

Table 4.4 shows the metrics across approaches for how long in seconds it takes to

iterate through the entire dataset once. At each epoch end, the dataset is shuffled so the model avoids overfitting.

Approach	Min	Mean	Max	Variance
10k dataset and 30 epoch	784.84	676.23	670.71	189.01
20k dataset and 30 epoch	1342.67	1351.55	1426.31	203.61
10k dataset and 100 epoch	675.92	682.85	756.77	61.51

Table 4.4: Time Taken to Train a Single Epoch (seconds)

At the end of each epoch, the model is saved as a measure of redundancy if anything were to go wrong. Table 4.5 shows how long it takes in seconds to save the model to the mounted Azure Fileshare via 'torch.save'.

Approach	Min	Mean	Max	Variance
10k dataset and 30 epoch	38.31	56.46	90.13	188.04
20k dataset and 30 epoch	31.43	59.93	111.16	405.93
10k dataset and 100 epoch	31.90	52.73	104.51	174.51

Table 4.5: Time Taken to Save Model (seconds)

To accompany IoU accuracies, Table 4.6 shows the confusion matrices and accuracies for each approach across class predictions.

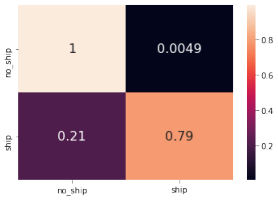
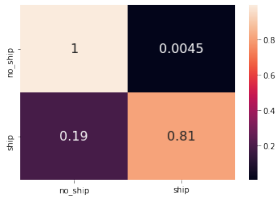
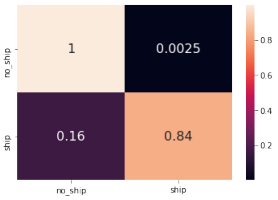
10k Dataset and 30 Epoch	20k Dataset and 30 Epoch	10k Dataset and 100 Epoch
		
89.27%	90.41%	91.87

Table 4.6: SN Confusion Matrix Accuracies

Finally, Table 4.7 shows a side-by-side comparison of predictions from each of the approaches. This type of visual analysis allows for interpretation for model direction.

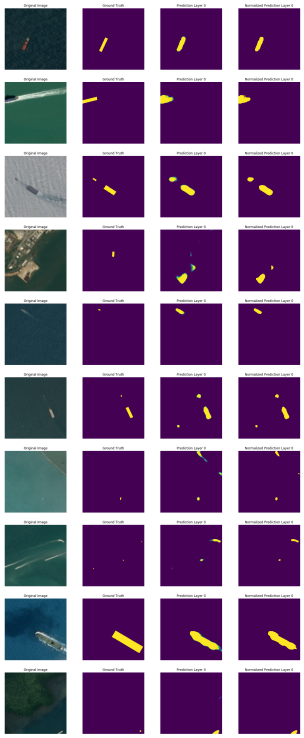
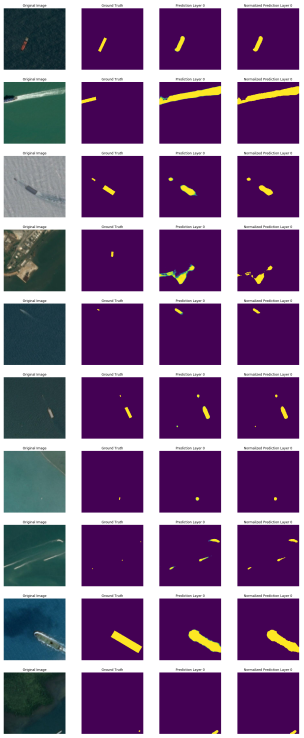
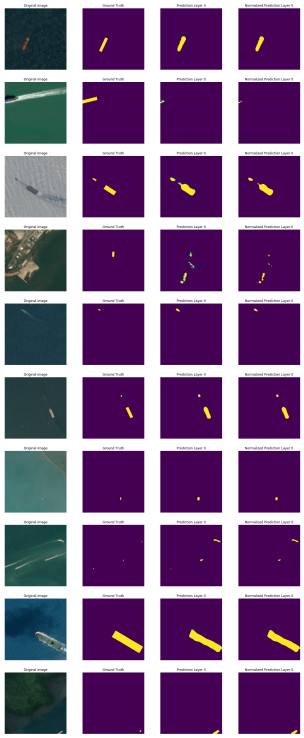
10k Dataset and 30 Epochs	20k Dataset and 30 Epochs	10k Dataset and 100 Epochs
		

Table 4.7: SN Architecture Predictions

4.3 Parameter Server Variant Architecture

The PSv was run over 30 epochs using the dataset size of 10 thousand points and implemented gradient accumulation four times at each of the four workers before they pushed gradients. The reason only four batch gradient accumulations were chosen was because 8 and 16, although allowed the model to iterate through batches much faster, resulted in poorer convergence. Only 10 thousand datapoints were used (identical to that used in the SN architecture) as to reduce time and cost of the training run. Due to the boosting nature of this architecture, a learning rate of 1e-3 was implemented, as well as momentum of 0.9 and weight decay of 1e-5 like the SN architecture. The same hyperparameters were also used for the Focal Dice Loss as the SN architecture.

Figure 4.1 shows the loss and IoU accuracy for the training run. The IoU accuracy,

gathered over model evaluation, achieved 14%, however still showed a slow decrease in loss value and thus still converging. The workers reach a training loss of around 1.28 and a validation loss around 6.54.

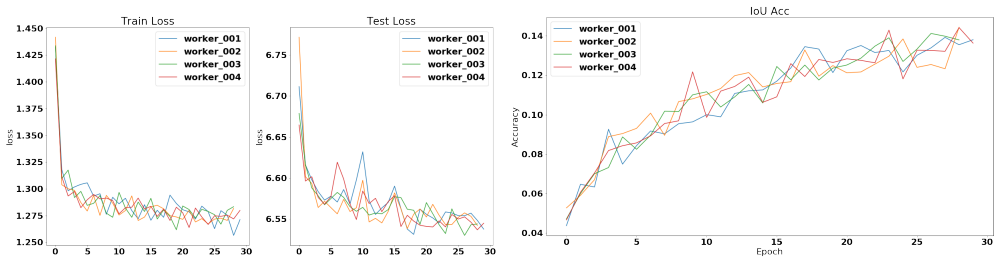


Figure 4.1: PSv Loss and IoU Accuracy

The confusion matrix seen in Figure 4.2 demonstrates an accuracy of 85.96% across classes.

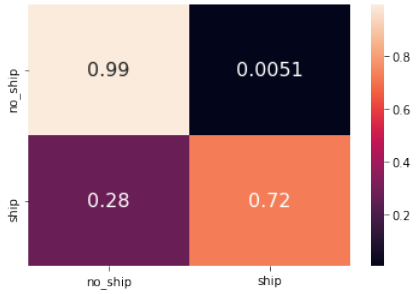


Figure 4.2: PS 10k Dataset Over 30 Epochs and 4 Accumulations: 85.96% Accuracy

At the beginning and after each consecutive gradient push, the worker will wait and then read the global models updated weights in order for the next set of batches to be processed. Table 4.8 shows the time taken in seconds for that initial and consecutive reads.

Entity	Min	Mean	Max	Variance
Worker_001	18.58	22.36	30.65	3.25
Worker_002	18.35	22.49	31.30	2.83
Worker_003	19.45	22.70	31.46	4.41
Worker_004	18.48	21.97	31.24	2.89

Table 4.8: Time Taken to read Global Model (seconds)

Before a worker is able to process a batch, it must first request the keys from the scheduler from the current point of the dataset. Table 4.9 shows the time it takes

in seconds for the worker to put in a request for a new batch to the Azure table the scheduler reads.

Entity	Min	Mean	Max	Variance
Worker_001	0.035	0.101	1.35	0.008
Worker_002	0.035	0.099	1.23	0.008
Worker_003	0.034	0.098	3.80	0.009
Worker_004	0.036	0.094	1.99	0.007

Table 4.9: Time Taken to Request Task (seconds)

The scheduler may be busy processing another workers batch, so the requesting worker will have to wait until theirs in processed. Table 4.10 shows the time in seconds each worker is waiting for their batch request to be processed by the scheduler.

Entity	Min	Mean	Max	Variance
Worker_001	0.012	0.080	1.24	0.006
Worker_002	0.014	0.078	0.90	0.006
Worker_003	0.012	0.079	3.78	0.008
Worker_004	0.014	0.073	0.74	0.005

Table 4.10: Time Taken to Wait for Task (seconds)

While the workers wait for their requests to be processed, the scheduler is busy performing other requests. Table 4.11 shows the time in seconds taken by the scheduler to receive, process, and deliver the task back to a requesting worker.

Entity	Min	Mean	Max	Variance
Scheduler	0.00025	0.0135	3.74	0.0024

Table 4.11: Time Taken for Scheduler to Process Request (seconds)

Once the worker has received the task, it must prepare it for model input. Table 4.12 shows the time in seconds it takes for that preparation and model processing of that batch.

Once processed and accumulated over the four batches, the worker will push the local models gradients to the data server for the primary process to read. Table 4.13 shows the time in seconds it takes to push those gradients to the data server.

Once pushed, the worker will wait until the primary process has read, updated the global model, and pushed the updated weights to the data server ready for the

Entity	Min	Mean	Max	Variance
Worker_001	0.498	1.16	2.53	0.068
Worker_002	0.116	1.11	3.05	0.064
Worker_003	0.600	1.21	2.76	0.087
Worker_004	0.333	1.12	2.19	0.065

Table 4.12: Time Taken to Build and Process Input (seconds)

Entity	Min	Mean	Max	Variance
Worker_001	5.37	9.27	35.40	26.23
Worker_002	5.35	9.00	34.41	21.38
Worker_003	5.36	9.68	35.72	32.40
Worker_004	5.21	8.99	30.81	23.93

Table 4.13: Time Taken to Push Gradients to Data Server From Worker (seconds)

Entity	Min	Mean	Max	Variance
Worker_001	12.64	19.34	54.37	30.62
Worker_002	12.41	19.89	52.98	27.71
Worker_003	12.29	18.35	56.24	34.82
Worker_004	12.23	19.13	66.68	41.70

Table 4.14: Time Taken to Wait for Global Weight Updates (seconds)

worker to read and use in the next cycle. Table 4.14 shows the time in seconds the worker waits for the primary to execute that process.

While the worker is waiting, the primary process reads and updates the global model. Table 4.15 shows the time in seconds it takes for the primary process to perform that action.

Entity	Min	Mean	Max	Variance
Primary	5.41	8.82	14.75	0.367

Table 4.15: Time Taken to Read Gradients and Update Global Model (seconds)

Once updated, the global model updated weights are then pushed to the data server and the worker is then signalled to read those weights. Table 4.16 shows the time in seconds it takes to push those updated weights and signal the worker.

Entity	Min	Mean	Max	Variance
Primary	2.98	4.76	7.55	0.97

Table 4.16: Time Taken to Push Updated Global Weights to Data Server (seconds)

This process encapsulates the sub-processes of an epoch. Table 4.17 shows the time records in seconds for the time taken to complete epochs over the entire 30 epoch training run.

Entity	Min	Mean	Max	Variance
Worker_001	2432.45	2660.84	3064.24	39696.18
Worker_002	2425.73	2672.78	3040.29	42061.82
Worker_003	2441.96	2677.38	3213.13	50324.72
Worker_004	2413.94	2628.33	3016.29	27690.05

Table 4.17: Time Taken Per Epoch Completion (seconds)

At the end of each epoch the scheduler will initialize a save for each worker to allow for redundancy if the primary were to fail. Table 4.18 shows the time in seconds it takes to save the local model with the workers most recent weight reads to disk.

Entity	Min	Mean	Max	Variance
Worker_001	15.30	25.25	42.52	56.81
Worker_002	13.35	30.49	56.78	142.78
Worker_003	12.34	25.88	46.31	59.27
Worker_004	16.48	32.48	54.51	82.29

Table 4.18: Time Taken to Save Local Model via torch.save (seconds)

Finally, Table 4.3 shows predictions made by the trained global model. This type of visual analysis allows for interpretation for model direction.

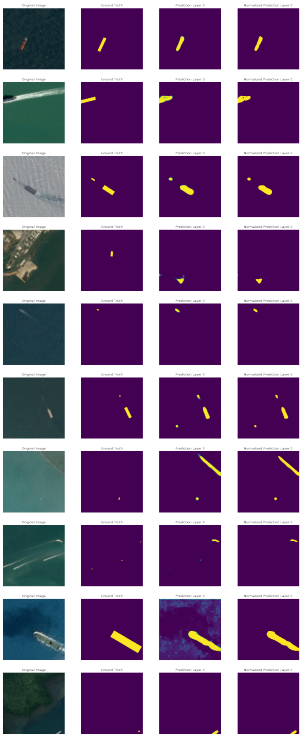


Figure 4.3: PSv Predictions

4.4 System Completion Time

Table 4.19 states the completion time in seconds and cost for all run experiments. The SN architecture made use of a single NC24 virtual machine for training. The PSv made use of a NC24 and four NC6 virtual machines. However, the data server was hosted on the primary node of the PSv architecture - this was a cheaper solution than having it run on a individual virtual machine. Since the SN architecture used the data server, it can also be suggested that the virtual machine that it trained on could also host the data server in an isolated training cycle and would absorb the cost. The cost is calculated based on hourly usage of each virtual machine [44]. In these experiments, the promotional version of the virtual machines were used for a reduced cost while they were available, but the cost will also be calculated for full-price usage. The NC6 Promo cost \$0.396/hr and NC24 Promo \$1.584/hr. The non-Promo NC6 cost \$0.90/hr and non-Promo NC24 cost \$3.60/hr.

Approach	System Time	Promo Cost	Full Cost
SN (10k/30ep)	06:24:47.519815	$10.16	$23.09
SN (20k/30ep)	12:04:27.297251	$19.13	$43.47
SN (10k/100ep)	21:26:59.642056	$33.98	$77.22
PSv (10k/30ep/4accum)	22:10:41.253272	$70.26	$156.68

Table 4.19: Training System Completion Time and Cost

4.5 OffHEM

The process of OffHEM was conducted with the best performing SN model (10k datapoints over 100 epochs) and the PSv model. The confusion matrix accuracy was calculated as the precision between classes was the main focus of this process - aiming to refine over the hardest examples and therefore should see that reflected in class accuracies. Table 4.20 shows the confusion matrices and accuracy post OffHEM training of the top 20% hardest examples reintegrated as duplicates into the training dataset. The SN model decreased from 91.87% to 88.46% accuracy and the PSv model decreased from 85.96% to 83.28% accuracy.

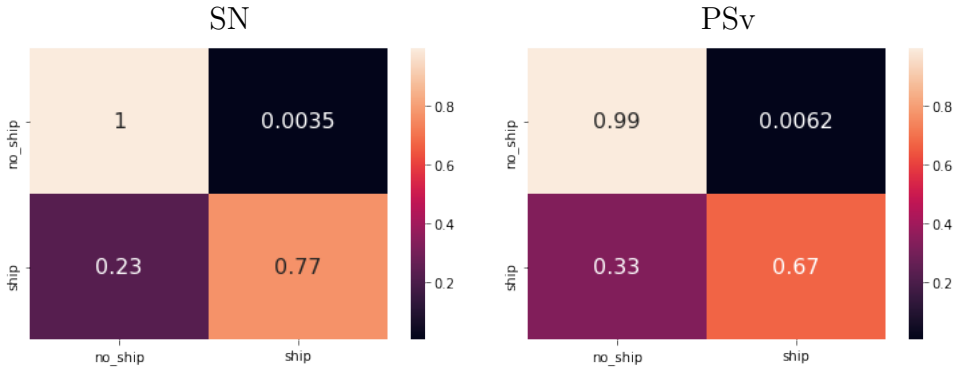


Table 4.20: Post OffHEM Confusion Matrices

Figure 4.4 shows some examples of what both models considered as 'hard'. They include cloudy examples as well as near to shore and very small ship examples.

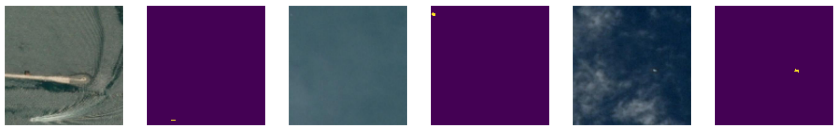


Figure 4.4: OffHEM Examples Across Both Architectures

4.6 Theia MRC Predictions

The 79 chip (3,384,384) was tested with each of the four architectures: SN, PSv, and their accompanying OffHEM versions. Table 4.21 shows the percentages of the true positives, true negatives, false positives, and false negative of the prediction versus the ground truth. In the final column is the overall accuracy. These columns are calculated using a confusion matrix.

Architecture	TP	TN	FP	FN	Accuracy	IoU mAP
SN	36%	100%	64%	0.26%	68.01%	15.03%
SN OffHEM	31%	99%	69%	0.92%	65.65%	10.36%
PSv	47%	100%	53%	0.38%	73.37%	19.15%
PSv OffHEM	17%	100%	83%	0.26%	58.26%	9.19%

Table 4.21: Theia MRC Test Results

Table 4.22 shows a subset of 6 examples that are representative of the target dataset and the accompanying predictions from the two best models: PSv and SN architecture. The associated IoU accuracy scores can be found in Table 4.23.

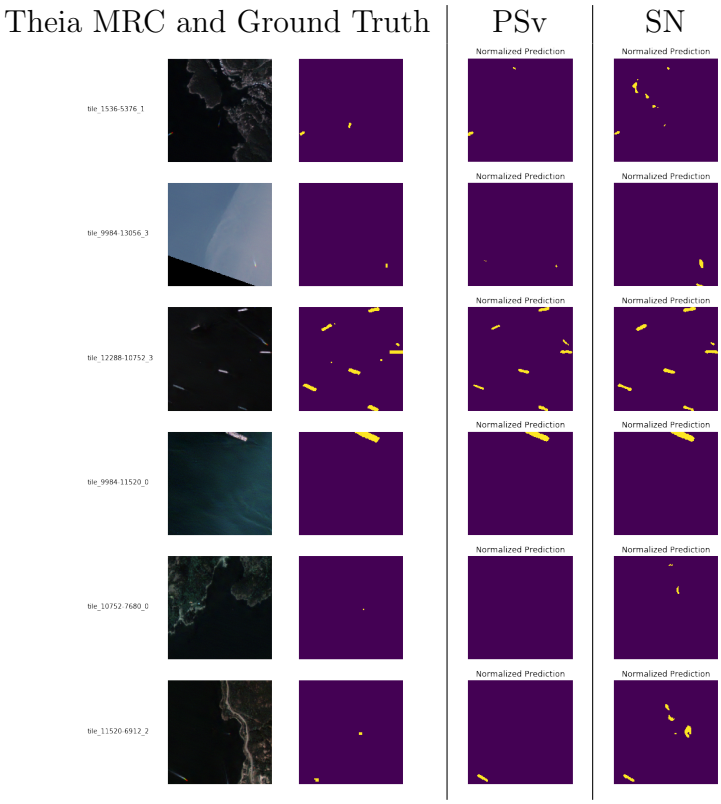


Table 4.22: Theia MRC Predictions

Theia MRC Name	PSv Accuracy	SN Accuracy
tile_1536-5376_1	20.81%	9.85%
tile_9984-13056_3	8.79%	13.38%
tile_12288-10752_3	63.98%	38.45%
tile_9984-11520_0	78.47%	44.54%
tile_10752-7680_0	0.023%	0.0026%
tile_11520-6912_2	15.02%	6.78%

Table 4.23: Theia MRC IoU Prediction Accuracy

4.7 Ship Classifier

Figure 4.5 shows the refining of a pre-trained ResNet101 for ship classification. The validation accuracy reaches 90% accurate.

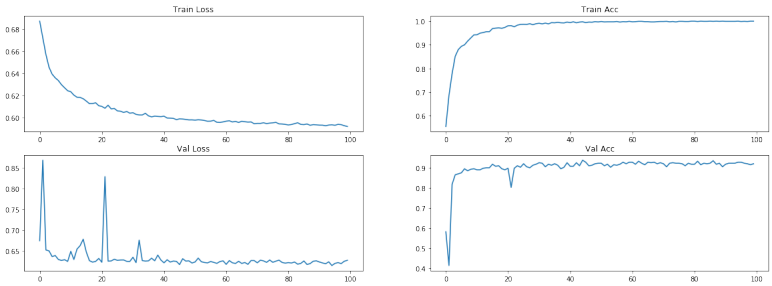


Figure 4.5: Refined Classifier Loss and Accuracy

Assessing between ship and no-ship classes, the confusion matrix in Figure 4.6 shows an accuracy of 90.56%. Pixels containing ships were correctly classified 91% of the time and no-ship pixels 90% of the time.

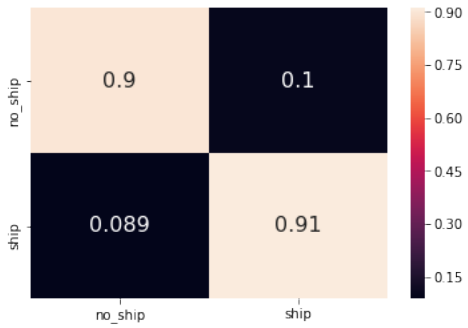


Figure 4.6: Refined Classifier With Accuracy 90.56%

4.8 Chapter Summary

Performing these experiments gives insight to the systematic performance and use-case accuracy for ship detection in satellite optical imagery. The SN best accuracy over the validation dataset resulted in 91.87% across classes, whereas the PSv resulted in 85.96%. The accuracies performed inversely on the target dataset where the SN model achieved 68.01% accuracy across classes and the PSv achieved 73.37% accuracy. The process of OffHEM resulted in decreased accuracy scores in both the validation and target dataset as compared to the non-OffHEM models equivalents. The SN architecture performs as a faster system rate than the PSv as the PSv incurs communication bottlenecks dealing with the large custom U-Net model parameters. The experiments performed are discussed and compared in the following Results chapter.

Chapter 5

Evaluation, Analysis and Comparisons

PSv and SN architectures are systematically compared in both systematic performance and resulting accuracies over the validation and target datasets. Extending the empirical analysis, visual analysis on the PSv and SN model predictions over the validation and target dataset aim to provide insights into the behaviour of the training approaches. The performance of OffHEM experiments are also discussed and the probable causes for the resulting outcome.

5.1 Data Serving

Since I/O operations are vastly more expensive than in-memory retrieval of data, this is why a data server was chosen over standard image reads for serving up datapoints. Iterating and transforming over the entire 63223 image chips and converting them into learning-ready tensors only took 2 minutes and 32 seconds while the file I/O operations took over 3 hours. This comparison to read and prepare is the reason serving data to models from a data server was the chosen approach. Also, utilizing a data server allowed for uninterrupted GPU usage that allows for a more cost effective approach to training.

5.2 Single Node Architecture

All three approaches did not converge due to early stopping because of the time and resource limitations imposed. However, all approaches reached a training loss of about 5.8 and a validation loss of about 5.9. Since the resultant losses are around the same at the end of the training cycles, it is intuitive to suggest that the models all perform similarly. It is important to note that given an extra 60 epochs, the resultant loss only decreased by 0.03 even with a cyclic learning rate scheduler to help avoid local minima. This slow convergence rate could indicate the complexity and difficulty of learning such a class imbalanced dataset despite using a specialized loss function meant to focus on smaller classes.

This same behaviour can be seen in the IoU accuracy, especially in regard to the validation dataset. The validation accuracy fluctuates whilst gradually increasing. These fluctuations describe the existence of noise that the model is experiencing within the validation set. This noise could be interpreted as the model having difficulty distinguishing between false positive rich examples like clouds, man-made objects, and landmasses. The IoU accuracies are fairly low in terms of expectation. This could be accredited to a poorer performance from the use of a subset of the original dataset or an indicator of IoU downfall when U-Net fits tighter segmentation masks than the ground truth provides. It's fairly common to issue a ground truth larger than the object desired so that the model can pick up on surrounding data and allow for that cleaner segmentation mask.

It is expected that the SN approaches achieve similar times for their functionalities. Consideration toward the batch training time needs to be taken in regard to the maximum and the variance. The small variance indicates that the average batch training time is consistently around the mean time. However, the maximum time is excessively larger than the mean. This is due to the preparation of the dataset at the beginning of each epoch. The SN architecture initializes built in (PyTorch support) dataset and data loader objects, which results in a large time overhead. Intuitively, training on a 20k dataset takes around twice as long as when training with a 10k dataset. The timing records for the 10k dataset over 100 epochs are lower than the other two most likely because the total number of recordings are numerously greater than the 30 epoch training runs. Though, it is interesting that the variance of the time it took the 20k dataset over 30 epoch training run to save the model to the Azure Fileshare is quite large compared to the other two approaches. This high variance

could suggest that there was more fluctuation in network traffic between the compute node and the storage location at the time of this training run and at the end of each epoch when the model is saved since the standard deviation is about 11 seconds for Worker_002 and about 7-9 seconds for the other workers. Though, it is more likely there was internal process contention delaying the initialization of the saving process within the VM that may also be paired alongside network bandwidth that causes such a large discrepancy. That same hypothesis could be applied to the other two approaches also at a much lesser degree since the variance is not very small.

The confusion matrix accuracy indicates the percentage of class accuracy between ship and no-ship over the validation set. Each model is easily successful in correctly classifying the no-ship class (100% across all three approaches), which is no surprise as it is the dominating class. The number of false negatives is small enough to be considered negligible and number of true and false positives are the best indicator of the models performance. Table 4.6 shows relatively similar accuracies over the 30 epoch training runs with 79/81% true and 19/21% false positives to the 10k and 20k datasets respectively. The training run of 10k dataset over 100 epochs achieved 84% true and 16% false positives, better than either of the other two approaches. Lower false positives indicate lower chances of mislabelling clouds, man-made objects, and landmasses as ships. This is apparent in Table 4.7 as the 100 epoch model manages to mask less of the ships wake and landmass than the other two runs resulting in cleaner and more refined segmentation masks. It should be pointed out that the model using 20k dataset managed to isolate all three small ships in row 8 that neither 10k dataset model masked. All three models have no issue masking open water ships but struggle with harder examples that are closer to land or where the ship size is small. This is noticeable by having false positives or not even registering the small ships. The fact that the models pick up on the wakes of ships and beaches show that it is difficult to distinguish highly reflective surfaces. The false positive accuracies support the data complexity. Each model exceeds the range of accuracies achieved in previous works around ship detection found in the background section. To restate, the class accuracy scores were calculated using a threshold of 0.9, rather than 0.7 in previous works, and suggests that the class classification distinctions are more precise than that of the previous work.

The cost of running deep learning training for the SN architecture is relatively cheap. Especially for early stopped training runs or runs that do not require such complex data.

5.3 Parameter Server Variant

The loss of all the workers reached as low of about 1.28 training and about 6.54 validation loss. This discrepancy in loss is indicative of the local training environment and the global model. The validation loss is higher but similar to that of the SN model. The higher validation loss value suggests that it makes slightly more mistakes than the SN model. The PSv training loss is much lower and this is likely because of the method of how the model was trained. This is likely due to the running statistics from the 'BatchNorm2d' layers that required manual adjustment of its momentum (regularization) factor to properly account for gradient accumulation. The accumulation of gradients means that the model imitates larger batch sizes while using the same weights for each batch accumulated. This suggests that a larger batch size with this data would be beneficial in learning ship detection. Since no resize is performed on the input images (which would allow for larger bath sizes), accumulation is the only valid approach to achieve larger batch sizes since GPU memory is maximized at batch size 16. However, since the training loss is much lower than the validation loss, this method of gradient accumulation could easily lead to overfitting. Since the validation loss is similar to the SN model (standard approach to training), leveraging the workers as weak learners helps avoid the global model from overfitting to the training dataset. Though, since both losses decrease and are not diverging, this is not indicative of an overfitting case and that the workers are providing quality gradients that the global model leverages in its forward pass.

The small IoU accuracy value (about 14%) is representative of the larger validation loss and makes sense that the global model would perform more poorly than the SN model over the validation dataset. This is notable when discussing the confusion matrix accuracy between classes. The PSv model had 72% true positives and 28% false positives for the ship class. Again, it's no surprise that the no-ship class is classified so accurately (99.49%) when it is the dominating class. Although lower than the SN models, the achieved 85.96% class accuracy still reaches the upper range of ship detection accuracies from other object detection models of 62.4% to 88.3% [8]. The PSv model also beats the traditional U-Net performance (85%) [9] on the same dataset. To restate, the class accuracy is based on the threshold of 0.9, as compared to 0.7 and exists within the top end of the accuracy ranges (62.4% to 88.3%) provided [8]. This strict classification suggests that the 85.96% accuracy achieved by the PSv model is more precise than that of the related works accuracy scores.

There were an array of timing metrics gathered throughout the training process of the PSv model. Starting at the beginning of the pipeline is the reading of the global model to initialize the local models.The variance of records is small enough to indicate that there were no major networking or local issues at each read. Operating over a virtual private network within a region helps reduce network latency. Reading of the global model is one of the larger portions of overhead from the system. However, operating with a Redis based data server allows for concurrent reads and therefore reducing contention around resources. This also reduces the chance of stragglers occurring while reading large amounts of data for the global model. The time taken to read the global model is indicative to the models large size for this application.

The next step in the pipeline is for workers to request a task. The scheduler receives a request, delivers the task, and then the worker processes that task into a batch by performing a forward pass through the model. Requesting the task is a simple entry to a table that the scheduler reads. While the variance across all workers is really small indicating that the mean value is close to the true time, it is interesting that worker_003 maximum value was recorded as three times as long as the other workers. This is most likely a rare event (considering the variance is so small), and probably due to some network traffic congestion from the compute node to the storage account the tables are hosted on. The process of waiting for the task to be delivered and the time taken to process a task request are interoperable and dependent. The reason the worker wait times are bigger than the scheduler process times are due to the scheduler processing worker tasks synchronously as they are received. Once the worker has received the task, it is read from the data server, built into a batch, and a forward pass is processed. The times taken, all with relatively small variances and so we can trust the mean is close to the true time, are all faster than that of the SN architecture. Implementing a PyTorch data loader is convenient, however, it introduces an overhead. This step can be multiplied by the number of accumulations to represent a more accurate "batch" training time, but it is assessed individually with respect to the SN architecture performance comparison. Implementing a request-deliver type scheduling algorithm for batches (tasks) reduces contention over resources and the chances of stragglers occurring at the task scheduling stage. With measures to reduce the straggling of workers through scheduling and data server approaches, it is network overhead that contributes a large portion of the system overhead. Although it takes half as much time to prepare and process each batch, the accompanying overhead from requesting tasks and pushing/pulling global model updates causes the

PSv architecture to operate slower than the SN architecture.

After building an input and performing a forward pass over the accumulated gradients, they are delivered to the primary process. This upload for the large U-Net model varied over the training run as per the larger variance value in Table 4.13. This deviation from the mean could be attributed to the complexity or size of the update or it could be network bound from uploading a large amount of data. Most likely, it is a combination of both of these things that cause the overhead. The minimum time for each worker suggests an ideal upload traffic load (close to zero) between the worker node and the primary node. Whereas the maximum time is the opposite and worst case scenario.

Once pushed, the worker will begin waiting for the primary process to retrieve and update the global model weights before pushing them for the worker to read. The process is synchronous at the primary process as gradients are applied as they come in and recognized at each query. This means, that if a worker were to update just after the primary process queried the update table, then it would have to wait longer until the next query which would occur after the current query updates had been applied. This is reflective in the maximum wait times of the workers. The minimum wait time occurs when the primary process recognizes and implements that workers update first. Since this operation is synchronous and intermittent between worker update queries, this causes the variance to grow larger. The worst case scenario would be a worker having to wait for up to six updates before having theirs processed due to missing a query involving the other three workers and then having their update recognized at the bottom of the following query.

A portion of this waiting is actually the primary process retrieving gradients then updating and pushing the global model weights (which are relatively reliable as they are not network traffic dependent since the data server is hosted at the primary node). This is observed in the small variance between timing records for reading gradients and pushing weights from Tables 4.15 and 4.16. These two functionalities are synchronous and the combined timing effects the workers wait time heavily.

Each epoch takes around twice as long as the SN architecture, which is seen in Table 4.17, and varies significantly over the 30 epochs as per the extremely large variance value. All the timing effects discussed previously accumulate into this epoch completion time. There are multiple reads of the global model by workers, pushing gradients, and pushing weights that all accumulate into a larger overhead than the SN architecture. This architecture was performed using four workers with capabilities

to scale beyond that. However, there would be a limit as the primary process would become more of a bottleneck than it currently is since workers must wait for their global model updates to complete.

Again, like the SN architecture, at the end of each epoch each worker saves their recently pulled model to disk as a measure of redundancy in case there is a system failure. The reason that the PSv architecture takes half as long to save the respective model is the missing optimizing state dictionary that the SN architecture retains.

Despite poorer IoU accuracy, predictions over the validation set returned visually acceptable segmentation masks. Referencing Figure 4.3, the PSv model picked up similar false positives to the SN model, though to a bigger degree in regard to landmasses. For smaller ships, the PSv model picked up a potentially unlabelled ship in row 6. However, it is more likely that this is a false positive. The clear observation of false positives assure are assured by the confusion matrix accuracy. However, the ability of the PSv model to recognize ships of different sizes reinforces that the PSv architecture approach as a solution is valid.

This approach cost over twice as much as the SN architecture within the same amount of time of training. This is obvious since the PSv architecture used more resources. Even using 5 virtual machines to run a distributed parameter server (variant), it is an affordable approach for distributed deep learning training given that the application justifies monetary expense.

5.4 U-Net

The custom U-Net model achieved higher class accuracies than those of related works in ship detection. It can be said that the customization of the model is effective for this type of application and distribution. Having feature-extracting bottlenecks that refine the encoded layers of the input whilst also reaching a deeper reduction of dimensionality, the custom U-Net is able to distinguish between ship and no-ship classes more effectively and thus minimize false positive classifications of pixels. Satellite data is rich with features that encompass a wide variety of textures from open water, airborne water, landmasses, and man-made structures. Being able to accurately distinguish between these textures/features and our targeted ship class, which is dominated by the no-ship class occurrences, through the use of more complex feature extraction, is why the custom U-Net was successful over the standard architecture approach.

5.5 OffHEM

The aim of this process was to refine the chosen models to further capture difficult scenarios during predictions. Even with a small subset (one tenth of the training dataset) being classified as hard examples both models resulted in decreased accuracies across class predictions. This result suggests that the use of the most difficult examples pulled both models attention from the average scenario and more toward the more difficult scenarios since both models saw more of the more difficult cases. While this may be useful if the target dataset consisted heavily of smaller, cloudy, or close to shore ships, it is not generalizable to all cases.

5.6 Target Dataset Predictions

The Theia MRC dataset consisted of a lot of fast and small moving ships. These fast ships would show as individual RGB pixels. Both models managed to capture these fast ships with varying segmentation masks. Just like with the training and validation dataset, both models excel in correctly classifying no-ship class due to its extremely weighted portion of each training example. In the examples given in Figure 4.7 and 4.3, the PSv model segmented more false positives than the SN model (28% to 16% respectively as per the confusion matrix). Interestingly, the PSv model managed to exceed both IoU mAP and confusion matrix accuracies on the target dataset compared to the SN model, which scored higher on the validation set. Also, the PSv model, although still high, predicted less false positives than the SN model on the target dataset - opposite to the validation set. It is also possible that because the validation score of the PSv model was so low, the PSv model was ineffective on the training/validation dataset and is coincidentally more precise on the target dataset. However, the increase in accuracy over the target dataset suggests that the PSv model learned associations within the training dataset that are more applicable to the target dataset than the SN models did. It could be suggested that the boosting of weak learning by leveraging worker gradients in a forward pass on the global model resulted in a more generalizable model. This is supported by the loss values provided by the workers being lower and therefore offering higher quality predictions than other experiments. Despite the resolution re-sampling of the training/validation data to match the distribution more closely of the target dataset, there could exist nuances in the target dataset that do not exist in the training dataset.

Both models struggled with small ships close to shore/landmass which is what the OffHEM process indicated. They excelled at segmenting larger ships and those in open water, regardless of the count. One interesting case is that, although resulting in a small IoU score, both models were able to recognize a small ship hidden a soft cloud (see Table 4.22).

The OffHEM models under-performed the non-OffHEM models reinforcing that they refined toward a solution that is not generalizable.

5.7 Classification Prediction Usage

The classifier was used to build an effective dataset of about 8000 Theia MRC image chips of size (384,384) that contained ships. These images were delivered to industry member UrtheCast to be submitted to AWS SageMaker for automatic labelling of region of interest (RoI) bounding boxes. This is so they could have an internally commercially usable dataset to re-train the U-Net model on as a product in their pipeline.

5.8 Chapter Summary

The data server demonstrates a speedup worth its implementation into the experiments of the two architectural approaches. The SN architecture and model outperforms that of the PSv in the training and validation experiments. However, the PSv outperforms that of the SN on the target dataset. This discrepancy could be attributed to the variant approach of the PSv architecture leveraging workers gradients to improve the generalizable quality of the global model predictions since the PSv workers attributed much lower loss scores which does not diverge from the validation loss scores. In both cases, the PSv and SN models perform within the higher end of acceptable and even exceed related works accuracies in ship detection, respectively. The work performed is summarized in the following chapter.

Chapter 6

Conclusions

6.1 Conclusion

Ship detection using optical satellite imagery is a difficult process. The examples of clouds, landmasses, man-made objects, and highly reflective objects pose as issues and are a cause of false positives. The application in this work of using a custom U-Net model that is deeper and implements bottlenecks to extract more features demonstrates an improvement over related work. The custom U-Net exceeds class accuracy of related works by achieving 91.87% accuracy. Training this model with two different systematic approaches provided acceptable and exceeding class accuracies compared to related works. The SN architecture provided an excellent class accurate model exceeding that of related works. The PSv architectural approach, though slower than the SN architecture, provided a model that was acceptable within the training/validation dataset and exceeded the SN model approaches in the target dataset. The goal of the PSv architecture and model is reached by providing a generalizable model toward the target dataset within the related works acceptable class accuracy range.

6.2 Future work

Communication bottlenecks are a major concern for the PSv architecture. Overcoming the primary process bottleneck would require a tiered server architecture to overcome and therefore introducing complexity to the system. This involves having workers subscribe to a gradient accumulation server, which would then push to a

higher level and pull down the available global model weights. This architecture may introduce a higher degree of weight staleness but could be considered a systematically efficient improvement over the current PSv architecture. Investigations into the effect of stale weights and object detection could be considered for future work. Reducing systematic overhead within the PSv architecture would improve its efficiency in order to exceed standard training times. Within this same vein, implementing an asynchronous task loader communicating with the data server would be very beneficial in regard to architectural performance. However, the biggest bottlenecks lie at the model reads and uploads of both gradients and weights. Future work would be best served investigating efficient mechanisms to asynchronously have workers push gradients, primary update weights, primary push weights, and workers read weights on a layer-by-layer basis. This could drastically reduce the communication and waiting overhead currently implemented. Eliminating the bottleneck of a centralized primary node may force considerations towards ring-AllReduce approaches for gradient sharing.

Other approaches for efficiency increase could lie at the gradient accumulation stage. Determining the effects and overcoming these issues at larger accumulation rounds could increase the performance of epoch completion. However, it is unknown how these larger accumulation rounds would affect convergence given the current experiments.

Other approaches for refinement of the models via OffHEM should be explored by including softer examples with the hard examples as to keep the refinement process in line with the generalizable training goals. The approach used in this work aimed at extreme hard examples in regard to larger loss values. Other metrics for considering what a "hard" example is should be explored.

Naturally, implementing a training cycle using the entire training dataset would be case for future work with both the SN and PSv architectures to determine their performance over unrestricted time and cost. Since the SN model is able to achieve 92% class accuracy, it would be beneficial to determine the minimum sized dataset to achieve the highest class accuracy score. The minimum sized dataset would achieve time, and computation and financial expenses to train a successful model. This information could give insight into similar satellite optical imagery datasets focused on object detection.

Bibliography

[1] ESA. Iss utilization: Urthecast cameras and instruments on the iss. `https://directory.eoportal.org/web/eoportal/satellite-missions/i/iss-urthecast`, 2020. [Online; accessed 5-July-2020].

[2] ESA. Spot-5 - eoportal directory - satellite missions. `https://earth.esa.int/web/eoportal/satellite-missions/s/spot-5`, 2020. [Online; accessed 5-July-2020].

[3] Urška Kanjir, Harm Greidanus, and Krištof Oštir. Vessel detection and classification from spaceborne optical images: A literature survey. *Remote Sensing of Environment*, 207:1 – 26, 2018.

[4] O. Karakuş, I. Rizaev, and A. Achim. Ship wake detection in sar images via sparse regularization. *IEEE Transactions on Geoscience and Remote Sensing*, 58(3):1665–1677, 2020.

[5] A. Grover, S. Kumar, and A. Kumar. Ship detection using sentinel-1 sar data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-5:317–324, 2018.

[6] Jacob Høxbroe Jeppesen, Rune Hylsberg Jacobsen, Fadil Inceoglu, and Thomas Skjødeberg Toftegaard. A cloud detection algorithm for satellite imagery based on deep learning. *Remote Sensing of Environment*, 229:247 – 259, 2019.

[7] Eija Parmes, Yrjö Rauste, Matthieu Molinier, Kaj Andersson, and Lauri Seitsonen. Automatic cloud and shadow detection in optical satellite imagery without using thermal bands—application to suomi npp viirs images over fennoscandia. *Remote Sensing*, 9(8):806, Aug 2017.

[8] Jinlei Ma, Zhiqiang Zhou, Bo Wang, Hua Zong, and Fei Wu. Ship detection in optical satellite images via directional bounding boxes based on ship center and orientation prediction. *Remote Sensing*, 11(18):2173, Sep 2019.

[9] D. Hordiiuk, I. Oliinyk, V. Hnatushenko, and K. Maksymov. Semantic segmentation for ships detection from satellite imagery. In *2019 IEEE 39th International Conference on Electronics and Nanotechnology (ELNANO)*, pages 454–457, 2019.

[10] Libo Yao, Yong Liu, and You He. A novel ship-tracking method for gf-4 satellite sequential images. *Sensors (Basel, Switzerland)*, 18(7):2007, Jun 2018. 29932145[pmid].

[11] Yong Liu, Libo Yao, Wei Xiong, Tian Jing, and Zhimin Zhou. Ship target tracking based on a low-resolution optical satellite in geostationary orbit. *International Journal of Remote Sensing*, 39(9):2991–3009, 2018.

[12] Mennatullah Siam, Mostafa Gamal, Moemen Abdel-Razek, Senthil Yogamani, and Martin Jagersand. Rtseg: Real-time semantic segmentation comparative study. *2018 25th IEEE International Conference on Image Processing (ICIP)*, Oct 2018.

[13] S. Alashhab, A. Gallego, A. Pertusa, and P. Gil. Precise ship location with cnn filter selection from optical aerial images. *IEEE Access*, 7:96567–96582, 2019.

[14] Shaoming Zhang, Ruize Wu, Kunyuan Xu, Jianmei Wang, and Weiwei Sun. R-cnn-based ship detection from high resolution remote sensing imagery. *Remote Sensing*, 11:631, 03 2019.

[15] Vlado Valković and Jasmina Obhoaš. Sediments in the ship's ballast water tank: a problem to be solved. *Journal of Soils and Sediments*, 20(6):2717–2723, Jun 2020.

[16] Linan Jia, Dawen Jiao, Yin Yue, Xiaotao Shi, and Lin Sun. Mechanism and influence of removing algae with chlorine dioxide treatment for ship ballast water. *IOP Conference Series: Materials Science and Engineering*, 631:022002, nov 2019.

[17] Elisa Berdalet, Lora E. Fleming, Richard Gowen, Keith Davidson, Philipp Hess, Lorraine C. Backer, Stephanie K. Moore, Porter Hoagland, and Henrik Enevoldsen. Marine harmful algal blooms, human health and wellbeing: challenges and

opportunities in the 21st century. *Journal of the Marine Biological Association of the United Kingdom*, 96(1):61–91, 2016.

[18] Alan J. Lewitus, Rita A. Horner, David A. Caron, Ernesto Garcia-Mendoza, Barbara M. Hickey, Matthew Hunter, Daniel D. Huppert, Raphael M. Kudela, Gregg W. Langlois, John L. Largier, Evelyn J. Lessard, Raymond RaLonde, J.E. Jack Rensel, Peter G. Strutton, Vera L. Trainer, and Jacqueline F. Tweddle. Harmful algal blooms along the north american west coast region: History, trends, causes, and impacts. *Harmful Algae*, 19:133 – 159, 2012.

[19] Vera L. Trainer, Stephen S. Bates, Nina Lundholm, Anne E. Thessen, William P. Cochlan, Nicolaus G. Adams, and Charles G. Trick. Pseudo-nitzschia physiological ecology, phylogeny, toxicity, monitoring and impacts on ecosystem health. *Harmful Algae*, 14:271 – 300, 2012. Harmful Algae–The requirement for species-specific information.

[20] Rita A. Horner, David L. Garrison, and F. Gerald Plumley. Harmful algal blooms and red tide problems on the u.s. west coast. *Limnology and Oceanography*, 42(5part2):1076–1088, 1997.

[21] H. Yang and J. Lee. Secure distributed computing with straggling servers using polynomial codes. *IEEE Transactions on Information Forensics and Security*, 14(1):141–150, 2019.

[22] Luiz F. Bittencourt, Alfredo Goldman, Edmundo R.M. Madeira, Nelson L.S. da Fonseca, and Rizos Sakellariou. Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, 30:31–54, Nov 2018.

[23] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization, 2018.

[24] Muhammad Khurram Bhatti, Isil Oz, Sarah Amin, Maria Mushtaq, Umer Farooq, Konstantin Popov, and Mats Brorsson. Locality-aware task scheduling for homogeneous parallel computing systems. *Computing*, 100(6):557–595, 2018.

[25] Zhiqiang Xie, Xia Shao, and Yu Xin. A scheduling algorithm for cloud computing system based on the driver of dynamic essential path. *Plos One*, 11(8), Apr 2016.

[26] Mohammad Mohammadi Amiri and Deniz Gunduz. Computation scheduling for distributed machine learning with straggling workers. *IEEE Transactions on Signal Processing*, 67(24):6270–6284, Dec 2019.

[27] Emre Ozfatura, Deniz Gunduz, and Sennur Ulukus. Speeding up distributed gradient descent by utilizing non-persistent stragglers. *2019 IEEE International Symposium on Information Theory (ISIT)*, Jul 2019.

[28] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 19–27. Curran Associates, Inc., 2014.

[29] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011.

[30] Joeri Hermans, Gerasimos Spanakis, and Rico Möckel. Accumulated gradient normalization, 2017.

[31] Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. Integrated model, batch, and domain parallelism in training neural networks. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, page 77–86, New York, NY, USA, 2018. Association for Computing Machinery.

[32] J. Zhang and O. Simeone. Lagc: Lazily aggregated gradient coding for straggler-tolerant and communication-efficient distributed learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2020.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, page 234–241, 2015.

[34] Microsoft. What is azure fileshare. https://docs.microsoft.com/en-us/azure/storage/files/storage-files-introduction, 2020. [Online; accessed 5-July-2020].

[35] RedisAI. Redisai. https://oss.redislabs.com/redisai/, 2020. [Online; accessed 5-July-2020].

[36] Redis. Redis client handling. https://redis.io/topics/clients, 2020. [Online; accessed 5-July-2020].

[37] Microsoft. Nc-series. https://docs.microsoft.com/en-us/azure/virtual-machines/nc-series, 2020. [Online; accessed 5-July-2020].

[38] J. O. D. Terrail and F. Jurie. On the use of deep neural networks for the detection of small vehicles in ortho-images. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 4212–4216, 2017.

[39] AirBus. Optical and radar data. https://www.intelligence-airbusds.com/optical-and-radar-data/, 2020. [Online; accessed 5-July-2020].

[40] Chung A.C.S. Wang P. Focal dice loss and image dilation for brain tumor segmentation. *In: Stoyanov D. et al. (eds) Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support. DLMIA 2018, ML-CDS 2018. Lecture Notes in Computer Science, vol 11045. Springer, Cham*, 2018.

[41] Xiaoya Li, Xiaofei Sun, Yuxian Meng, Junjun Liang, Fei Wu, and Jiwei Li. Dice loss for data-imbalanced nlp tasks, 2019.

[42] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[43] GIMP. Free and open source photo editor. https://www.gimp.org/, 2010. [Online; accessed 5-July-2020].

[44] Microsoft. Linux virtual machines pricing. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/, 2020. [Online; accessed 20-July-2020].