AFRL-RY-WP-TR-2020-0109

TRAINING CONVOLUTIONAL NEURAL NETWORK CLASSIFIERS USING SIMULTANEOUS SCALED SUPERCOMPUTING

Joshua Kaster
University of Dayton

MAY 2020
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
May 2020	Thesis	28 February 2020 –28 February 2020

4. TITLE AND SUBTITLE
TRAINING CONVOLUTIONAL NEURAL NETWORK CLASSIFIERS USING SIMULTANEOUS SCALED SUPERCOMPUTING

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
N/A

6. AUTHOR(S)
Joshua Kaster

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Dayton
300 College Park
Dayton, OH 45469

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYAT

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2020-0109

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
PAO case number 88ABW-2020-0783, Clearance Date 28 February 2020. © 2020 University of Dayton. Submitted to The School of Engineering of the University of Dayton in partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering. The U.S. Government is joint author of this work and has the right to use, modify, reproduce, release, perform, display, or disclose the work. Report contains color.

14. ABSTRACT
Convolutional neural networks (CNN) are revolutionizing and improving today's technological landscape at a remarkable rate. Yet even in their success, creating optimal trained networks depends on expensive empirical processing to generate the best results. They require powerful processors, expansive datasets, days of training time, and hundreds of training instances across a range of hyperparameters to identify optimal results. These requirements can be dicult to access for the typical CNN technologist and ultimately wasteful of resources, since only the most optimal model will be utilized. To overcome these challenges and create a foundation for the next generation of CNN technologist, a three-stage solution is proposed: (1) To cultivate a new dataset containing millions of domain-specific (aerial) annotated images; (2) to design a flexible experiment generator framework which is easy to use, can operate on the fastest supercomputers in the world, and can simultaneously train hundreds of unique CNN networks; and (3) to establish benchmarks of accuracies and optimal training hyperparameters. An aerial imagery database is presented which contains 260 new cultivated datasets, features tens of millions of annotated image chips, and provides several distinct vehicular classes. Accompanying the database, a CNN-training framework is presented which can generate hundreds of CNN experiments with extensively customizable input parameters.

15. SUBJECT TERMS
big data, deep learning, hyperparameter tuning, image classification, image processing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Hamilton Clouse
Unclassified	Unclassified	Unclassified	SAR	64	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

TRAINING CONVOLUTIONAL NEURAL NETWORK CLASSIFIERS USING SIMULTANEOUS SCALED SUPERCOMPUTING

Thesis

Submitted to

The School of Engineering of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Master of Science in Electrical Engineering

By

Joshua Montana Kaster

Dayton, Ohio

Dec, 2019

TRAINING CONVOLUTIONAL NEURAL NETWORK CLASSIFIERS USING

SIMULTANEOUS SCALED SUPERCOMPUTING

Name: Kaster, Joshua Montana

APPROVED BY:

Eric J. Balster, Ph.D. Advisory Committee Chairman Associate Professor and Chair, Electrical and Computer Engineering	Patrick Hytla, Ph.D. Committee Member Senior Image Processing Engineer, Group Lead UDRI

Vijayan Asari, Ph.D.
Committee Member
Professor, Electrical and Computer
Engineering

Robert J. Wilkens, Ph.D., P.E. Associate Dean for Research and Innovation Professor School of Engineering	Eddy M. Rojas, Ph.D., M.A., P.E. Dean, School of Engineering

ABSTRACT

TRAINING CONVOLUTIONAL NEURAL NETWORK CLASSIFIERS USING SIMULTANEOUS SCALED SUPERCOMPUTING

Name: Kaster, Joshua Montana
University of Dayton

Advisor: Dr. Eric J. Balster

Convolutional neural networks (CNN) are revolutionizing and improving today's technological landscape at a remarkable rate. Yet even in their success, creating optimal trained networks depends on expensive empirical processing to generate the best results. They require powerful processors, expansive datasets, days of training time, and hundreds of training instances across a range of hyperparameters to identify optimal results. These requirements can be difficult to access for the typical CNN technologist and ultimately wasteful of resources, since only the most optimal model will be utilized. To overcome these challenges and create a foundation for the next generation of CNN technologist, a three-stage solution is proposed: (1) To cultivate a new dataset containing millions of domain-specific (aerial) annotated images; (2) to design a flexible experiment generator framework which is easy to use, can operate on the fastest supercomputers in the world, and can simultaneously train hundreds of unique CNN networks; and (3) to establish benchmarks of accuracies and optimal training hyperparameters.

An aerial imagery database is presented which contains 260 new cultivated datasets, features tens of millions of annotated image chips, and provides several distinct vehicular classes. Accompanying the database, a CNN-training framework is presented which can generate hundreds of CNN experiments with extensively customizable input parameters.

It operates across 11 cutting-edge CNN architectures, any Keras-formatted database, and is supported on 3 unique Linux operating systems - including two supercomputers ranked in the top 70 worldwide. Training can be easily performed by simply inputting desirable parameter ranges in a pre-formatted spreadsheet. The framework creates unique training experiments for every combination of dataset, hyperparameter, data augmentation, and super computer requested. The resulting hundreds of trained networks provides the performance to perform intensive qualitative analysis to highlight key input requirements for optimal results. Finally, as proof of this performance, select sample benchmarking results of both the accuracies on aerial image datasets and their training hyperparamters are presented to illustrate the framework's utility.

For my wife Emily; my motivation and my love - without her support this work would have never been complete. And for my children Zoe, Chloe, Theo, and Leo, my greatest pride and joy – without whom this thesis would have been complete 2 years earlier.

ACKNOWLEDGMENTS

I would like to thank the Air Force Research Labs (AFRL) and the University of Dayton Research Institute (UDRI) for supporting my research and maturing my career. I would like to thank Dr. Eric Balster and Dr. Vijayan Asari for teaching and guiding me through my coursework, equipping me for my research, and for their exemplary professionalism in the research community. I would like to thank Dr. Patrick Hytla for his exceptional leadership of our UDRI engineering team - he guides with wisdom, respect, and success at every step. I would like to thank Dr. Hamilton Scott Clouse, James Patrick, and Robert Mash for inspiring and funding my research with AFRL. They showed me how to confidently tackle impossible tasks, apply technology to difficult problems, and to find fun and passion at work. These outstanding professionals contributed to any success I can claim – I am grateful simply to know them and to have the good fortune of working with them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Convolutional neural network (CNN) techniques have exploded across the technological community in the last decade. With the exponential increase in processing power and availability of massive datasets, researchers across a diverse number of research fields have the key tools to approach long established problems and make significant improvements. Where once human developers worked to recognize notable features in data and design algorithms to achieve a desired result, CNNs flip that paradigm by allowing algorithms to develop and recognize features in data and humans dictate the desired results.

1.1 History

The use of CNN architectures with multiple layers has blossomed at the head of CNN techniques. In the computer vision (CV) community, CNN architectures have achieved state-of-the-art (SotA) performance in object classification applications on challenging benchmark image datasets such as ImageNet [3] and PASCAL VOC [4], as well as speech recognition applications. While CNNs originally appeared in the late 1990's and were successful in digit recognition applications [5], their current accelerated progress can be attributed to three unique changes: the 2006 insight of Hinton and Salakhutdinov [6] of treating each layer in a deep feedforward neural network as an unsupervised restricted Boltzmann machine, the proliferation of vast amounts of data via the internet, and the exponential progress in general purpose graphics processing units (GPUs) for massively parallel computations.

This revolution off convolutional neural networks (CNN) first met widespread public success with the CNN AlexNet [7] - which won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) of 2012. After its landside improvement of 10% over the competition,

each following ILSVRC winner is also a CNN design. Each shows novel techniques and approaches to CNNs which all feature trade-offs in decrease in error for increase in processing power, training time, and architecture complexity. Each proves better at completing the ILSVRC, but which is best for other datasets with limited resources? The resulting accuracy of a trained model is highly dependent on its input datasets and the hyperparameters selected to train it. How does one pick these reliably? These are the questions that inspire this work.

1.2 Direction

Along with these questions, the goal of this work is to develop performance in training deep CNNs on aggregate sensor data acquired from both passive and active sensors. This work allows researchers to fully understand the computational challenges associated with successfully training deep CNNs on relevant aerial datasets. The potential payoff is a next generation of algorithms that can be transitioned for object detection, classification, and tracking applications. Or put more succinctly: to establish baseline datasets, experimentation frameworks, and CNN algorithms for Electro-Optical (EO) Wide Area Motion Indicator (WAMI) imagery with special interest in detection, classification, and tracking of vehicular targets in aerial datasets.

A three-stage solution is proposed: (1) To cultivate a new dataset containing millions of domain-specific (aerial) annotated EO images; (2) to design a flexible experiment generator framework which is easy to use, can operate on the fastest supercomputers in the world, and can simultaneously train hundreds of unique CNN networks; and (3) to establish benchmarks of accuracies and optimal training hyperparameters. Results will show an established CNN-Training Framework (CTF) with the performance to operate across 11 cutting-edge

CNN network architectures, 260 cultivated datasets featuring tens of millions of aerial image chips, and on 3 unique computer operating systems. A select sample benchmark of this performance will perform 100s of CNN trainings and show accuracies across a variety of testing paradigms – including test on withheld training data subset, transfer learning to similar datasets, and fine-tuning on new datasets.

The following chapters further clarify the CTF development and results. Chapter II explores the foundational background of convolutional neural network and related CNN technology. Chapter III explains the 1st stage solution methodology of curating new datasets. Chapter IV discusses the 2nd stage solution methodology to simultaneously scale CNN training across super computers and create exhaustive hyperparameter testing. Chapter V identifies the final software tools that enable the CTF and describes the results of implementing the 1st and 2nd stage solutions. Chapter VI shows results from the 3rd stage solution of a select sample benchmark of aerial imagery and performance of initial experiments. Chapter VII summarizes with concluding remarks and thoughts on worthwhile future research efforts.

CHAPTER II

BACKGROUND

2.1 Origin and Application of Neural Networks

The human mind has natural abilities that excel beyond all computers. Classification of imagery is a great example of one of these abilities. Verifying the presence of a coffee mug in any given image would be tediously simple for a human to accomplish. But to recreate this performance in a computer is a surprisingly complex endeavor. And if more than one class are required to be identified, the complexity grows quickly beyond a single algorithm's performance. With this observation of human and computer differences in mind, scientists sought to recreate this human performance of complex classification in a computer by modeling their approach on the latest understanding of brain structure and processing. The resulting system is referred to as a neural network to honor its original inspiration.

In the last decade, major improvements in neural network technology have allowed for more and more complicated endeavors to be undertaken. One great example is Open AI's Dota 2 challenge. In 2017, Open AI trained a bot with neural networks to play the competitive video game [8]. Dota 2 is multi-player online battle game similar to capture the flag – but virtual and much more complicated. It requires teams of players, and battles of powerful 'heroes' to remove a construction called an 'Ancient' from enemy bases. Being a competitive eSport, training a bot to play is vastly more complicated than one-on-one board games like Chess and Go. The bot is able to defeat a world class professional in two consecutive rounds and marked significant increase in neural net performances. Similarly, neural networks have been revolutionary in the LSVRC ImageNet Competitions. Every year since 2012, a neural network has won the 1000-category image classification challenge. The

competitive nature of this highly visible and popular technological event has lead developers to constantly design new and better architectures for neural networks.

2.2 Definition of Neuron

The neural network concept is built around the concept of a neuron inside a physical human brain. Each neuron is expected to accept multiple inputs which can activate a unique output into other neurons. An visual representation of this can be seen in Figure 2.1. M number of inputs, x_i, are available to the neuron.

Before each input can reach the neuron, it must pass through a synaptic junction, which either amplifies or attenuates the input intensity. This effectively acts as a signal multiplier and can be represented mathematically as w_i for each input. Once the attenuated inputs reach the neuron they are combined with the neuron's existing bias, which can be defined as b_0. All received inputs are summed to a final value within the neuron, v, as shown in Eq. 2.1. With all these signals incoming, the neuron 'fires' and sends off an output under specific condition called the activation function – defined as $f(v)$ in Eq. 2.2. Activation functions vary across neural network models, but mathematical models typically use one of the following functions: linear, rectified linear unite, hyperbolic tangent, sigmoidal, or step. The final neuron output is defined as y.

$$v = \sum_{i=1}^{M} x_i w_i + b_0 \tag{2.1}$$

$$y = f(v) \tag{2.2}$$

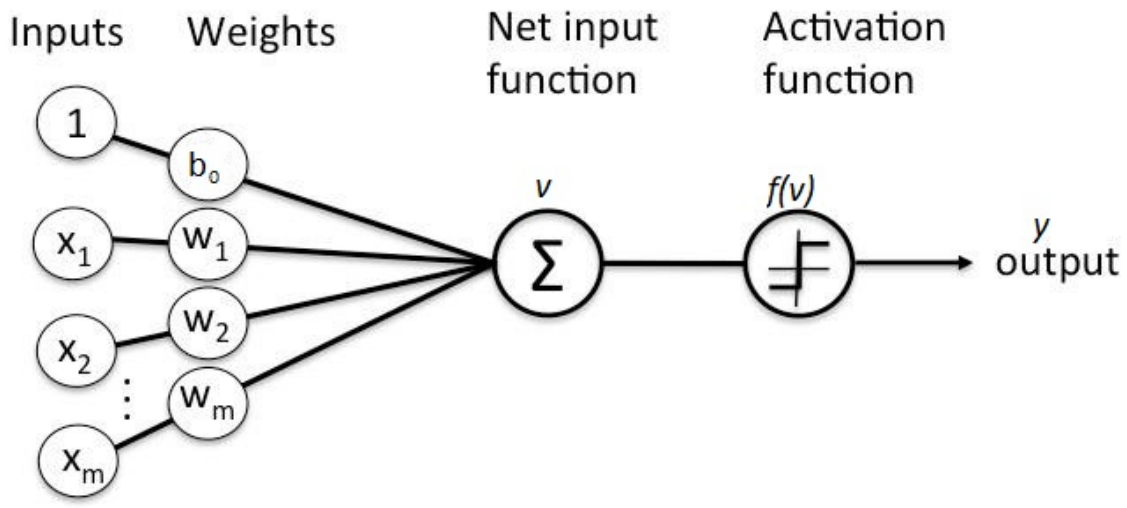

Figure 2.1: Diagram of Neuron

Whenever an activation function is attached to a linear layer, it needs to be non-linear in nature. This ensures that complicated functions are approximated instead of simple linear relationships. When multiple neurons are connected, the result is called a neural network. Typically, these connections are made in distinctive layers, and any parallel-connected neurons are called fully-connected layers.

2.3 Back-Propagation

Training neural networks requires working backwards from its performance and adjusting the weight values to better match desire results. This is called back-propagation and works by taking a derivative of a cost functions - written in terms of weights. In an ideal cost function, the solution space is parabolic and thus only has a single minimum solution. With this simplistic case, there are no other local minimums to compete with the final solution. However, there are more complex cases where more robust cost functions should

be used. A general cost function formula is given in Eq. 2.3, where the error is ϵ, the expected value is β, and the calculated result is ρ. A co-efficient is present simply to lower computation requirements, which are reduced when the cost function derivative is taken.

$$\epsilon = \frac{1}{2}(\beta - \rho)^2 \tag{2.3}$$

Speaking broadly, machine learning techniques can be categorized as either supervised or unsupervised learning. With supervised, both the input and expected output are known pieces of data. With unsupervised, only the input is known. The majority of successful neural network trainings focus exclusively on supervised methods. They use the difference between the known output and predicted outputs to estimate error and influence a change in neural network weights. Each new data input and resulting predicted output lead to slight adjustments in the weight's values. How quickly these adjustments occur can be controlled by a hyperparameter known as the learning rate. This process repeats until every sample input is presented to the network; a full iteration through the training dataset is called an epoch.

The primary goal of this process is to severely reduce the error by adjusting the weights of the neural network. Finding a solution's global minimum is optimal, and can be better found by taking the error's derivative with respect to the weights. This will point to the closest minimum by calculating the slope. Weight changing equations can be found below in Eq. 2.4 and 2.5. The weight value for each neural network connection is w_i and Δw_i is the change in weights value. η is the learning rate parameter and multiplies the derivative-determined change in order to influence how drastically the weight values are changed.

$$w_i = w_i + \Delta w_i \tag{2.4}$$

$$\Delta w_i = -\eta \frac{\delta \epsilon}{\delta w_i} \tag{2.5}$$

The process to train neural network and update weights can be repeated infinitely as the solution approaches a minimum, so it's important to identify stopping criteria. Too early an interruption and neural network accuracies will still be quiet poor due to under-learning. To late a stop may see neural networks so closely resembling the dataset that the phenomenon is called over-fit; these are rarely accurate on similar datasets. Perhaps the most effective stopping conditions is to calculate total error per epoch and complete training the change in error is less than a pre-set threshold. An additional method is to test every epoch with a validation dataset and use its accuracy results to determine stopping conditions.

2.4 Convolutional Neural Networks

The original model for a neural network is novel in its implementation, but not well suited to processing imagery due to their much larger resolutions. Images must be broken down into more simplistic representation and processed through fully connected layers. The most successful method of breaking down images is with convolutional kernels and can be described by the equation Eq. 2.6 below. This concept shows that convolutional layers are effective at extracting features before the fully connected layers. The maximum number of layer filtered by convolutions can be represented by M and a convolutional kernel by k. Large variability of input sizes and stride lengths can affect the kernel. Stride length directly controls the distance a convolution covers and ultimately adjust the final output dimensions. In the equation, $*$ represents the 2D convolutional operator and u represents the activation function. Finally, the output is defined as y_j.

$$y_j = u(b_j + \sum_{i=1}^{M} k_{ij} * x_i) \tag{2.6}$$

This combination of convolution and filtering proves to be very powerful in application. Data-adapting feature extractors can be combined within several layers and map the original input to a simpler representations- all while avoiding over-fitting. This reduction also lowers the data diversity requirements and create convolutional neural networks (CNN) that can handle more dense data inputs.

2.5 Image Feature Filtering

One of the greatest advantages of CNNs are their ability to detect and extract features within datasets. This is a computationally expensive process with a large output when it works, and unhelpfully empty output when it fails. Filtering and passing useful features to the next layer while ignoring broken returns is a process called pooling. The majority of current application use maximum pooling (as opposed to average or minimum pooling). This compares multiple neuron results at a given layer and passes along the largest. This removes superfluous values from computation stream and removes any sparse or small data. With this process in place between layers, a full end-to-end process can be run and the network results are analyzed to see if a solution emerges.

CHAPTER III

DATA CURATION

To achieve the 1st stage solution of cultivating a new database containing millions of aerial annotated EO images, a significant effort must be undertaken to gather, annotate, chip, and augment datasets. Focus should first be on gathering several existing annotated datasets for training and convert both imagery and truth into usable formats. Whenever several historical EO annotated datasets are identified and collected it is likely each will have a unique truth format, image size, and data qualities. Software scripting must be developed to read in the imagery and truth, transform truth to fit EO, and display bounding boxes. This can be very difficult when truth data is not easily understood and may require multiple conversations with teams who generated the data to achieve a working result.

If annotations are only available for a few frames within a full motion video (FMV), the incomplete truth data can be estimated by interpolating sparse truth points and then manually corrected for accuracy. All image frames originating from sequential FMV will benefit from image registration and required homography transformations to align truth across multiple frames. Additional datasets should be collected whenever discovered, with a focus on cultivating from successful sources such as publicly released datasets, successful data collections, and readily adaptable previous work. Successfully completing this process effectively is critical to future trainings. Without large and diverse data for CNNs, any trained results will be unreliable and can easily become overfit - only operating on the training data but inapplicable to other datasets.

3.1 Data Gathering

The first step in an effective data curation is the process of gathering all potential datasets. The wide variety of available imagery will require extensive image processing such as calculating homographies for datasets, converting raw files to imagery standards, and aggregating truth files into centralize file with a project-standard format. All data should be formatted to make data accessible in an standard annotation structure. Ideally the annotation software will allow quick and effective annotation of datasets with universally accessible imagery and truth files. To benefit from available built-in tracking and interpolation algorithms, all image sequences must be registered and homography-transformed to match the first key image. Annotation software requires all images to be PNG format, which is utilized for its efficient file size with lossless compression, high bit depths, and open standard. Annotation software truth format are often held in a SQL database and scripts to import and export are crucial to preserve and access annotations.

Ideal imagery should feature a diverse collection of vehicles in a wide variety of settings and backgrounds – all with an effective ground sampling distances (GSD) of 0.5 - 1.5 ft. per pixel. This is the desired resolution for vehicle detections and best matches existing datasets. More image diversity in represented vehicles and backgrounds will lead to more robust and accurate trained CNNs. Look for both high quantity datasets and extremes in features such as bright colors, partial obstructions, and low contrast imagery. There will be likely be hundreds on grey sedans in a parking lot, but if that's the only class a CNN is trained on, it will be unable to identify yellow cars in grass or boats on trailers. Most data collections do not focus on characterizing the data (and unique features), so it is helfpul to develop both the ability to read truth from project-specific comma-separated value (CSV) files and structured query language (SQL) formats and to quickly visualize. These tools will

also prove useful in matching all available imagery with its truth and visa-versa. Scripting should be developed to complied a list of all image and truth locations, then verify each has a match. Whenever annotations are missing, the unmatched image should be manually truthed or dropped from the dataset.

It's important to mature and optimize the scripting and tools on smaller datasets - once a few large datasets are gathered, lengthy processing times and storage limitations will be encountered. It's recommended to access a shared storage system accessible across an entire network of machines. Image and annotations tend to be on the order of gigabytes and multiple datasets can add up quickly. It's also crucial to check the number of available inodes in a shared storage system. Inodes are the number of file names storable by a system - if there's a large number of files of small size, like annotation in text form, it can quickly fill up all available inodes without using up all available space. System admins can typically adjust inodes and available storage space when requested. Care must be taken to not fill up storage files or data can be lost. Also be careful to permission data files properly. It's recommended to create a unique group ID that is only available to users connected to the project.

3.2 Annotate and Chip

Once new annotated datasets are reliably accessible and scripting is in place to manipulate imagery, focus should be directed to creating an image chipper that can scale across all image datasets. Effectively scaling across millions of images will require adding error checks and optimizing for increases in the processing speed and to avoid potential RAM crashes. As more data is converted to the standard truthing format, focus will shift to processing the remaining edge and corner cases. Depending on the reliability of dataset source, the first

error check may need to be an image quality checker to identify any significant minority of the imagery that is oversaturated, under-saturated, or missing from listed location. This identification can often be most easily done with histograms, with any atypical distributions removed from the final datasets. Most chippers will work well on a small scale but quickly demonstrate memory errors when ingesting large datasets. This is correctable by organizing the imagery to be chipped into digestible batches, clearing and closing all opened figures, and periodically clearing all working memory when opening new batches.

As the majority of largest datasets are processed, focus can turn to supplement existing data by manually truthing entirely new datasets. High-resolution imagery from publicly available YouTube quad-copter videos are an excellent unannotated source. Building from now-established tooling and optimized code developed for previous datasets, the conversion and truthing process should be significantly faster. A single individual can find a video, split to frames, convert to truthing format, add annotations - generating around 70 GB or more vehicle datasets in 2 weeks.

3.3 Ontology and Negatives

With a massive increase in annotated datasets, more diverse data characteristics and objects will be observed. Simple ontologies like those used in the low size, weight and power (SWaP) proof-of-concept implementation of in Part A of this Thesis [9] are intentionally simplified to just car, truck, van/SUV, and background. Now that millions of vehicle examples exist, annotations can be expanded to new characteristics such as color, occlusion, non-standard vehicles, and pedestrians. Greater granularity in ontology is highly desirable because it will result in trained models with more descriptive classification on test data. However, it requires a significant effort to update all annotations of existing datasets to

more detailed ontologies. To update efficiently, existing annotation can first be updated with default values of all new parameters - for example a car can now be a sedan with black color and no occlusion. Truthers can then quickly move through existing labels and correct new parameters in approximately 25% of the time originally required to find, draw a bounding box, and properly annotate. However, any new detailed ontology that adds new classes will require entirely new entries to the annotation dataset and will be time consuming. Ultimately, available datasets and diversity of objects will drive chosen ontologies and it's useful to finalize ontology when near to finalizing datasets.

Once a final database and ontology is established, it can be supplemented with false positives from a CNN object detector. The idea is to train an object detector with existing imagery and annotations, then test on the same datasets. Any new detections will find objects missed by annotators or find features that closely resemble established classes. These can be aggregated into a confuser or negative class and added the training datasets to improve classifier object discrimination. Both FasterRCNN [10] and SqueezeDet [11] are great choices for object detectors to build a negative class. FasterRCNN as seen in [9] performs on simpler ontology and remains highly capable of identifying confusers. SqueezeDet is based on a unique dataset called KITTI [12] which features an unusual aspect ratio due to its original design as an object detector from a moving car on wide angle views of streets. A new script will need to be developed to effectively convert all annotation and imagery from AVA format to KITTI format for these trainings.

With final annotations and conversions, data curation can be concluded and focus moved toward CNN training with the new database. Each individual annotated image should be chipped into three pad sizes of 0.0m, 2.5m, and 5.0m to provided different amounts of

background and neighboring objects associated with the object to be classified. This is a pre-processing data augmentation step that should be performed outside of training.

CHAPTER IV

SIMULTANEOUSLY SCALED SUPERCOMPUTING

The 2nd stage solution requires designing a flexible experiment generator framework which is easy to use, can operate on the fastest supercomputers in the world, and can simultaneously train hundreds of unique CNN networks. Each of these is a complex task worth detailed thought and development before final implementation. Methodology and suggestions on how to achieve this are presented below.

4.1 Ease of Use: Interface

In order for a framework to be easy to use, it should have a intuitive interface for operation and robust software behind the interface. Intuitive interfaces will be accessible and readily used by the general computer user and not only CNN experts. Robust software should operate reliably in all scenarios it's expected to be used and provide useful error messages when failure occurs.

For the general computer user, one of the most familiar and accessible interfaces is a spreadsheet. With a pre-formatted set of column headings and example row data, a user can change quickly individual cell values to set their desired inputs. Any minimal framework for CNN training should allow a user to input an experiment name, training dataset, validation dataset, and test dataset. Each dataset must be indexed and readily available to be called by the user. A more flexible framework would allow the user to change the network architecture used for training and finetune method. Any callable network should also be indexed and available. Customizables hyperparameters for the psuedo-random seed number, number of epochs, batch size, number of GPUs, and initial learning rate would also be desirable. Each of these will allow a user to substantially control the training and repeat the most

successful results. Finally, a variety of data augmentations such as rescale, channel shift range, zoom range, rotation range, fill mode, horizontal flip, vertical flip, width shift range, height shift range, constant fill value, and comments would allow for significant expansion of the annotated dataset. Each of the parameters should allow for setting both single or multiple values; likewise, the individual datasets should be able to be added together to create new datasets unique to each experiment.

4.2 Ease of Use: Robust

With this highly customizable interface design, robustly setting up the software tools required for training CNNs is critical to ease of use. The first step is to insure matching software dependencies and packages across various computing machines. This is a very complex problem that has plagued software developers for years and can be made even more challenging if there is slow, restrictive, or unavailable internet access. A solution is required that can be built offline and reliably transferred to new networks, all while supporting multiple operating systems. One such solution is environment modules [13], which are utilized for their ability to turn on and off software installations on command, while also creating stable instances that can be separated by version. A large emphasis should be placed on learning to build these modules across multiple operating systems (OS) and software packages – each OS with unique configurations and requirements to become operable. If built correctly in the shared storage of a network, they will be accessible for all compatibly OS's that can connect. The ideal test of proper environmental module operation is the successful ability to train a new CNN. This test should be performed with small annotated datasets on every system before attempting to scale.

To improve code robustness, local training should be performed on a small scale to help develop code for massive distribution. All starting scripts and input parameters should be saved for comparison across multiple machines. With each attempted training, all accuracies and under-performances should prompt analysis and improvements to be made. When possible, try repeating previous successful trainings to compare dataset in and classifications accuracies out, then make minor adjustments to code. Specific adjustments to check include: verify validation and training dataset classes match, adjust epoch to match available GPU memory, confirm trainings are using GPU memory and not CPUs, and watch for training to update weights. It is also important to enable offline training to operate without internet access. When distributing trainings across multiple machines and nodes, each training does not always have individual access to online resources - even when interactive instances to those machines might have internet. To compensate for this limitation, it's best to download any required model architectures and pre-trained weights for offline access. In the framework, prioritize training from this offline source location before requesting online download which can subsequently fail.

4.3 Supercomputers: Identify and Access

In order for a framework to operate on the fastest supercomputers in the world, several challenges will need to be overcome. First, access to the desired computers must be obtained. Second, all data must be transferred to the new machine. Third, all software dependencies will need to be built on the new machines. Fourth, software must be modified to better integrate with the unique OS and utilize any jobs queue system available.

For access, it's recommended to start with a locally networked Linux GPU machine where the user has relatively high control over resources, data, and software. This will act

as a test bed to work out all development issues without the restrictive offline and non-super-user permissions of a supercomputer. This network should have a terabyte or more of shared storage across machines, internet access for downloading software, and at least one other accessible GPU machine to test the first machine's software builds.

While setting up these local GPU machines, identify and apply for access to desired supercomputers. Each system will have a unique application requirements which can take weeks to process. One such option is to access High Performance Computing (HPC) at DoD Supercomputing Resource Centers (DSRCs) [2]. According to their publically available website, this requires a sponsorship from a qualified employee of the US government who has an active project with allocated hours on one of the HPCs. Once an account is created, a user will be able to reach 10 unclassified supercomputers - 5 of which rank in the top 100 supercomputers world wide: Mustang, Excalibur, Thunder, ONYX, and Topaz [14]. Each of these debuted with a high rank in the top 50, but with the incredible rate of change in technologies, faster supercomputers have taken the top slots. Each supercomputer has distinct systems, CPUs, OSs, and performance performances (publically available online and a subset shown in Table 4.1). Of these options, ONYX is the fastest available supercomputer, and Thunder has more established software builds and environment modules for CNN training.

4.4 Supercomputers: Data Transfer

Once an appropriate supercomputer is selected for development, the next major obstacle to scale HPC use is that all code and data needed to be built on the selected system. Especially on ONYX, it is a completely fresh Cray Linux environment without software, data, or existing environment modules; all have to be built from scratch. To further complicate

Table 4.1: HPC Top Supercomputers [2]

2019 Rank	Site	Name	System	CPU	Operating System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Debut Rank	Debut Year
58	ERDC DSRC	ONYX	Cray XC40	Xeon Phi 7230 64C 1.3GHz	Cray Linux Environment	160304	3409.6	5865.5	29	2017
59	ERDC DSRC	Topaz	SGI ICE X	Xeon E5-2699v3 18C 2.3GHz	SUSE Linux Enterprise Server 11	124200	3318.9	4570.6	15	2015
68	AFRL DSRC	Thunder	SGI ICE X	Xeon E5-2699v3 18C 2.3GHz	SUSE Linux Enterprise Server 11	152692	3126.2	5610.5	18	2015
64	AFRL DSRC	Mustang	HPE SGI 8600	Xeon Platinum 8168 24C 2.7GHz	Red Hat Enterprise Linux	55296	3221.4	4777.6	48	2018
91	ARL DSRC	Excalibur	Cray XC40	Xeon E5-2698v3 16C 2.3GHz	Cray Linux Environment	100064	2485.0	3682.4	19	2014

progress, standard data transfer methods such as secure copy protocol and online connections to supercomputers can be inconsistent and unreliable. Even with correct syntax, it is common to be unable to start a transfer or for file transfers to be terminated midway. Due to the intensive processing demands and high performance requirements, all supercomputers are often down for maintenance – both emergency and routine. Given the need to transfer 100's of GBs of imagery for training, and the multiple days of training required for optimal results, these are difficult challenges to face. One method to reliably overcome is to download all possible needed resources locally (architectures, models, imagery, annotations, environment modules, dependencies, software, and code), and utilize the 'rsync' command [15] to send it directly from to supercomputers and keep a change log. This works better than compression methods, such as tar balls, because even if there are still several drops in transfer and data due to maximum sync time limits and undesirable system restarts, these can all be remedied by multiple syncs, checking file times and sizes, and reviewing sync logs. New data can be added relatively easily without sending an entire dataset again.

With a reliable method of syncing data established, the primary resulting problem with transferring is the sheer size of the dataset. Beyond the expected problem of limited hard drive space, the sheer quantity of files can far exceeded normal allowances. To overcome this, work with network system administrators on both sides of the transfer to confirm there is enough hard drive space and inodes allowance. Each time an inode cap is hit, it's possible

to lose all data transfer progress and have to restart each times to pick up from previous cap and add to dataset. This can occur even with deceptively small datasets of less than 5.0 GB when the number of files is in the million for each chipped pad size. Even simple requests to list all files can take over 30 minutes to display. It's critical to understand the size and quantity of database before attempting to transfer. It's recommended to discuss with system administrators and dedicate user groups with shared directory that can accommodate all files. With appropriate sizes allocated, a full data transfer and check can be complete within a week.

4.5 Supercomputers: Software Environments

When data is successfully transferred, all software dependencies and environment modules will need to be built fresh on the new machines. As expected when building a cutting-edge CTF, there are a few unique obstacles to overcome. One major obstacle is that many desired software packages may be complex and difficult to build, especially on new OS's without internet or elevated privileges. Build scripts must be tested for robustness on more accessible local machines before attempting to port to a supercomputer. Another obstacle is that software versions and dependencies need to be locked and saved offline whenever successful builds are made. Any offline saves should include a copy of all shared libraries and verify all symlinks point to files within the save. With newer cutting-edge software, every week can see new changes to online code bases that can break environment module builds that rely on older dependencies and build tools. Especially on supercomputers, version numbers are critical because available build tools are controlled by system administrators and are not guaranteed to match the latest software. Work with the administrators

to request and update tools where possible; build desired modules with older tools when available and stable on supercomputers.

4.6 Simultaneous Scaling

In order to scale software to simultaneously train hundreds of CNNs, a framework must be modified to better integrate with each unique supercomputer and utilize any jobs queue system available. The framework should provide experiment generation scripts and tooling which can read in multiple parameters from a single experiment list, create all desired experiments with repeatable results, and ultimately operate on HPCs using their system-specific parameters. Each experiment should define hundreds of potential trainings, each individual training known as a job, and each training should have a unique parameter combination. One method to achieve this is by iterating through the experiment list and generating two files for each potential job: an executable bash file to start the training and a YAML Ain't Markup Language (YAML) configuration file which defines all training parameters. These two files allow a user to manually start a specified job manually or for an autonomously start by a job queue. This iterative generation will allow for hundreds of trainings to be defined rapidly.

To simultaneously start all generated jobs on a HPC, an additional file should be generated by the framework. For all HPCs at DSRC, a computer software known as portable batch system (PBS) is used to perform job scheduling and allocate tasks across resources. The framework should create one PBS script per experiment which can define multiple jobs by calling the experiment generator scripts. PBS scripts have many options, are open source [16], and are well documented (and unclassified) for each HPC [2]. These should

all be set by the framework when it generates the PBS, and some options are critical for proper operation on HPCs.

Each HPC has a variety of queues available to submit jobs [2]. Each queue has a job priority, unique wall time (maximum time a job can be run), maximum number of cores, available number of CPUs, and available number of GPUs. Each job can then request an interactive GPU node from the HPC and run its respective custom training. If operating on a Cray Linux Environment instead of SUSE Enterprise Server 11, add 'ccmrun' before all GPU command when in a batch node. All spawned process that are submitted through 'subprocess' should be set in blocking mode. Otherwise a job node may close due to inactivity before a process-intensive training has a chance to start up. All trainings should store progress messages and output files in unique folder names to avoid name conflict.

4.7 Finalize Framework

With the recommended steps, the general flow of the framework should have multiple layers of software, bash scripts, and YAML scripts to support operation. A final flow starts with a single experiment generator bash script which sets global variables and launches an experiment generator. This then can creates a single PBS script per line as defined in a spreadsheet. Each of these PBS can hold an array of jobs and can request a single GPU node per job. Inside each GPU node, a unique YAML configuration file for training and an associated unique bash script can be created. The bash can set the YAML path and start the CNN training. With all software and data in place, several experiments can be submitted for simultaneous training. Several weeks of training can be simultaneously submitted and foundational baseline experiment are ready to be performed.

CHAPTER V

RESULTS: DATA AND CTF

This chapter identifies the final software tools that enable the CTF and describes the results of implementing the 1st and 2nd stage solutions. The final cultivated database and the final design of the CTF are explored. For the 1st stage solution, quantifiable metrics such as the total dataset size and number of images are discussed. For the 2nd state solution, a full description of the final CTF design and all of its custom parameters is included. Each achievement adds cutting-edge technology and performance.

Building on the success found in 'Convolutional Neural Networks on Small Unmanned Aerial Vehicles' [9], high performance had been built with the open source framework called CAFFE (Convolutional Architecture for Fast Feature Embedding) [17], developed by the Berkeley Vision and Learning Center (BVLC) with community contributors. Soon after, the neural network package TensorFlow [18] (TF) was released by Google and promised to be the new SotA framework for training CNNs. In order to further increase performance in classifiers, TF was selected for deeper research. More specifically, interest focuses on training models developed using the TF-Slim interface to support the development of the CTF. TF-Slim allows the use of abstracted code to both create new networks and use established networks developed by others. It promised to simplify the technical knowledge required to train up networks, but proved unreliable in high-level access to training and multiple-GPU implementation. However, after discovering these deficiencies with the TF-Slim tool and functionality, the framework is now refactored utilizing Keras (to TF) to speed development. Keras provides the critical software hooks to read in YML configuration settings and initialize a TF training.

During the one year course of this effort, there were extensive changes in all software; especially with TF, which progressed weekly from v0.9 to v1.4.1. There was a coincidental effort by Anaconda experts and HPC admins to build an anaconda-supported version of TF for HPCs, which started midway though the CTF effort. Despite their single task and extensive expert talent, it was released a few months after the CTF was complete and over 45 days later than the proposed complete date due to the complexity of a reliable TF build. This is but one example challenge faced and overcome in creating an integrated CTF. The full resulting database and CTF are described below.

5.1 Data Cultivation

The 1st stage solution is to cultivate a new database containing millions of domain-specific (aerial) annotated EO images. In total, the final database consists of 260 unique annotated EO datasets with over 80 million unique image chips. As comparison, ImageNet contains 14 million images. This annotated EO aerial imagery with vehicle-level GSD is backed up and hosted on multiple shared networks. A partial sample of the final list can be found in Table 5.1.

The final cultivated database is sourced from VEDAI [19], DLR Munich [1], several small unmanned aervial vehicle (SUAV) collections, several aerial YouTube videos, and a variety of additional EO sensor collections. The specified sensor imagery is contained in 42 datasets and is approximately 10 GB when tar compressed. One dataset is segregated for Transfer-Learning (TL). All specified sensor datasets also have a negative class featuring confusers detected by a trained SqueezeDet model and are approximately 700 MB when tar compressed. The largest majority of annotations come from the remaining sensor datasets. For purpose of training classifiers with varying backgrounds, all 80+ million unique chips

src_	set_	pad_	source_name	set_name	path	classes	set_check
2	1	1	DLR_Pos	dlr_munich_pad_0.0	$SDATA/d	aircraft,background,bike-r	dlr_munich_p
2	1	2	DLR_Pos	dlr_munich_pad_2.5	$SDATA/d	aircraft,background,bike-r	dlr_munich_p
2	1	3	DLR_Pos	dlr_munich_pad_5.0	$SDATA/d	aircraft,background,bike-r	dlr_munich_p
2	2	1	DLR_Neg	dlr_munich_pad_0.0	$SDATA/d	background	dlr_munich_p
2	2	2	DLR_Neg	dlr_munich_pad_2.5	$SDATA/d	background	dlr_munich_p
2	2	3	DLR_Neg	dlr_munich_pad_5.0	$SDATA/d	background	dlr_munich_p
5	1	1	VEDAI_Pos	vedai1024_pad_0.0	$SDATA/d	aircraft,background,bike-r	vedai1024_pa
5	1	2	VEDAI_Pos	vedai1024_pad_2.5	$SDATA/d	aircraft,background,bike-r	vedai1024_pa
5	1	3	VEDAI_Pos	vedai1024_pad_5.0	$SDATA/d	aircraft,background,bike-r	vedai1024_pa
5	2	1	VEDAI_Neg	vedai1024_pad_0.0	$SDATA/d	background	vedai1024_pa
5	2	2	VEDAI_Neg	vedai1024_pad_2.5	$SDATA/d	background	vedai1024_pa
5	2	3	VEDAI_Neg	vedai1024_pad_5.0	$SDATA/d	background	vedai1024_pa
6	1	1	Youtube_Pos	a_drones_perspective_of_traffic_jams_(	$SDATA/d	background,car,suv,truck,	a_drones_per
6	1	2	Youtube_Pos	a_drones_perspective_of_traffic_jams_(	$SDATA/d	background,car,suv,truck,	a_drones_per
6	1	3	Youtube_Pos	a_drones_perspective_of_traffic_jams_(	$SDATA/d	background,car,suv,truck,	a_drones_per
6	2	1	Youtube_Pos	aerial_drone_view_on_highway_in_calif	$SDATA/d	car,cargo,suv,truck,van	aerial_drone_
6	2	2	Youtube_Pos	aerial_drone_view_on_highway_in_calif	$SDATA/d	car,cargo,suv,truck,van	aerial_drone_
6	2	3	Youtube_Pos	aerial_drone_view_on_highway_in_calif	$SDATA/d	car,cargo,suv,truck,van	aerial_drone_

Table 5.1: Subset of Available Datasets for CTF

are padded at 0.0m, 2.5m, and 5.0m of neighboring space. Each is then copied into 3 new datasets with unique padding sizes.

A sample of a full-size image from the publicly available DLR Munich dataset can be seen in Figure 5.1 and image chips from its annotations can be seen in Figure 5.2. Currently, 8,614 annotations are available from this dataset, with nearly 99.26% being sedans. Similarly, in the publicly available VEDAI dataset, 1,952 annotations have been created. 36.65% are sedans and 25.42% are pickup trucks. Of the sedans, only 8.00% are occludes, and of the pickup trucks, only 11.31% are occluded. Due their small size, these two datasets are used consistently as sample training and validation sets to test out CTF development and new OS configurations.

While each dataset has distinctive collection properties and characteristics which boosts database diversity, all publicly available dataset are required to share specific commonalities to be included in the final distribution. First, all imagery is taken from an overhead or aerial perspective with a depression angle ranging from 35 degrees to nadir. The majority is nadir,

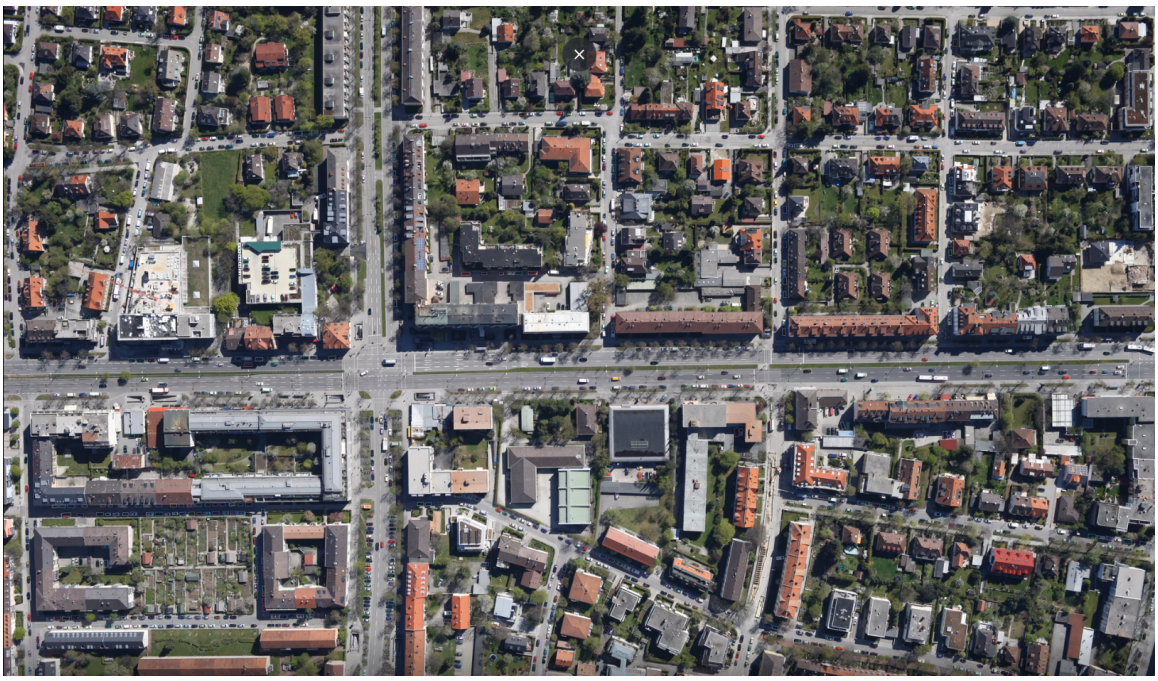

Figure 5.1: Sample Image Scene from DLR Munich [1]

while a large minority is 45 degrees. Second, all imagery features a vehicle-level GSD of 0.5 - 1.5 ft. per pixel. Satellite imagery is excluded by this requirement. Third, all vehicles are annotated using a common ontology. This ontology includes vehicle-level classes of sedan, SUV, van, pickup truck, semi truck, bus, RV, boat, and airplane. It also includes sub-class descriptions such as relative size, color, and occlusion. Non-publicly available datasets differ in several key ways which are intentionally not described in this paper.

5.2 CTF Overview

The 2nd stage solution is to design a flexible experiment generator framework which is easy to use, can operate on the fastest supercomputers in the world, and can simultaneously train hundreds of unique CNN networks. The final version of the CTF successfully uses TF and Keras to compile hyper-parameter grid-search experiments into PBS scripted

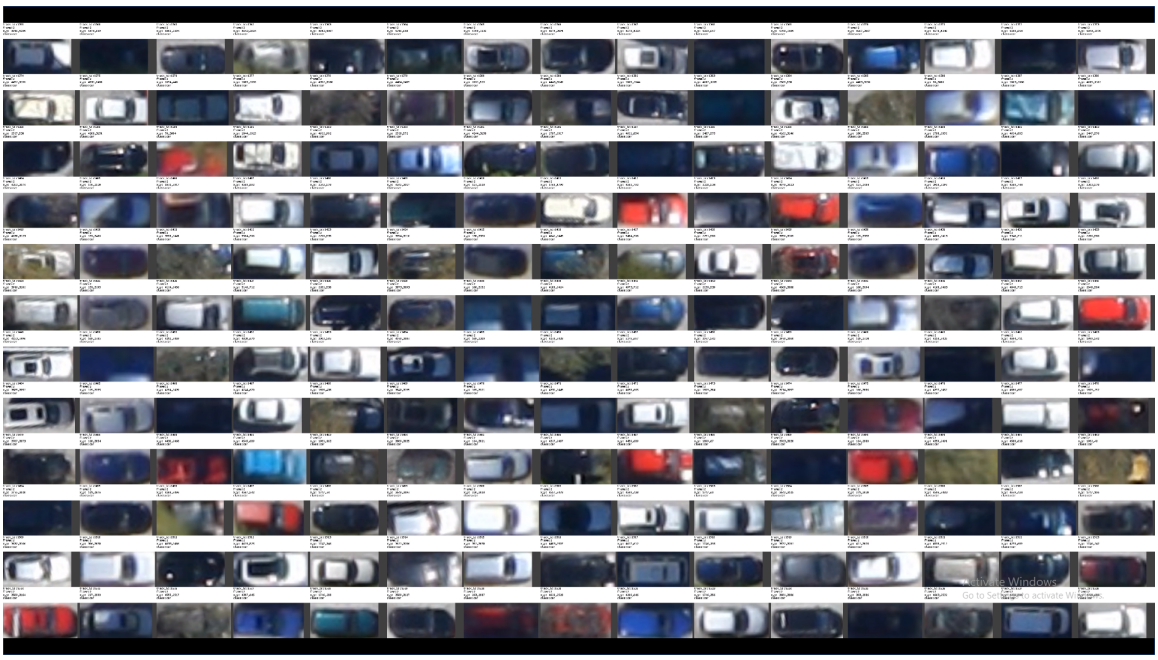

Figure 5.2: Sample Image Chips from DLR Munich [1]

arrays. After generating these scripts, the framework then uses the PBS scheduling tool to simultaneously execute hundreds of unique CNN-trainings on the DSRC's HPCs. The CTF features an easy to use spreadsheet interface, operates across 11 cutting-edge CNN architectures, offers extensive data augmentation, has proven performance on 3 operating systems and 30 machines, and can scaled across several HPCs GPU nodes.

5.3 Interface

To make simultaneous scaled CNN-training on HPCs more accessible to the typical user, an excel spreadsheet titled 'experiment_generator.xlsx' is utilized as the CTF interface. Each row of the 'experiments' sheet defines an entire experiment which can submit simultaneous jobs on the HPCs based on that row's input parameters. This is shown in Table 5.2, which is the left half of the 'experiments' sheet. Each column of the 'experiments' sheet accepts

customizable inputs that determine the datasets, networks, and hyperparameters to use in training. In all columns, multiple values can be listed within the cell by using comma separation. This is seen in 'batch_size' column and '02_weight_init_scratch' where two batch sizes are requested. For dataset-related columns ('Training', 'Validation', 'Test'), individual datasets (represented by their number found on 'datasets' sheet) can combine into aggregate datasets by using the '+' symbol or listed sequentially with a comma as separator. This allows for extensively customization in which dataset is used and opens up an exponential number of potential training combinations.

Experiment Name	seed	Training	Validation	Test	Networks	Finetune	epochs	batch_	gpus	learning_rate
01_Accuracy	100	5.1.1+4.1.	5.1.1	5.1.1	1.1, 2.1, 3.1, 4.1	all	50	128	1	0.01
02_weight_init_scratch	1	2.1.1, 3.8.	2.1.1+6.3.1	3.1.1+4.	1.1, 2.1, 3.1, 4.1	FALSE	1	32, 64	1	0.01, 0.001, 0.0
03_weight_init_final	1	2.1.2, 3.8.	2.1.1+6.3.1	3.1.1+4.	1.1, 2.1, 3.1, 4.1	final_layer	1	64	1	0.01, 0.001, 0.0
04_weight_init_all	1	2.1.3, 3.8.	2.1.1+6.2.1	3.1.1+4.	1.1, 2.1, 3.1, 4.1	all	1	64	1	0.01, 0.001, 0.0
05_Coarse_Grid_01	1	5.1.1+4.1.	5.1.1	5.1.1	1.1, 2.1, 3.1, 4.1	FALSE	100	64	1	0.01, 0.001, 0.0
06_FT_Offline	1	5.1.1	5.1.1	5.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	all, FALSE, fi	100	32	1	0.001
07_Pos_Scratch	1	2.1.1 + 3.1	5.1.1	4.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	FALSE	100	64	1	0.001
08_Pos_Final	1	2.1.1 + 3.1	5.1.1	4.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	final_layer	100	64	1	0.0001
09_Pos_All	1	2.1.1 + 3.1	5.1.1	4.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	all	100	64	1	0.0001
10_PosNeg_Scratch	1	2.1.1 + 3.1	5.1.1 + 5.2.1	4.1.1 + 4	1.1, 2.1, 3.1, 4.1, 5.1,	FALSE	100	64	1	0.001
11_PosNeg_Final	1	2.1.1 + 3.1	5.1.1 + 5.2.1	4.1.1 + 4	1.1, 2.1, 3.1, 4.1, 5.1,	final_layer	100	64	1	0.0001
12_PosNeg_All	1	2.1.1 + 3.1	5.1.1 + 5.2.1	4.1.1 + 4	1.1, 2.1, 3.1, 4.1, 5.1,	all	100	64	1	0.0001
13_Transfer_Learning	1	4.1.3	4.1.3	4.1.3	1.1, 2.1, 3.1, 4.1, 8.1,	all	100	32	1	0.001

Table 5.2: CTF Experiments List A

Detailed requirements about input formats and the full configurations options are listed in the 'other_config_options' sheet which is shown in Table 5.3. Each input parameter provided key detail toward the final training experiments. The 'Experiment Name' parameter allows for a unique string to be assigned to all generated experiments, files, and results. The 'seed' parameter allows a user-selected integer to be sent to the random number generators utilized in the frame – which allows for precise repeat-ability of any experiment results. The 'Training' parameter is perhaps the most crucial input as it allows a formatted string to describe all datasets to be used in training CNNs. Similarly, the 'Validation' parameter

Option	Type	Range or Allowable Set	Comment
Experiment Name	str		Should be unique
seed	int		
Training	str		
Validation	str		
Test	str		
Networks	str		
finetune	str	(False, final_layer, all)	
epochs	int	[1, 512]	
batch_size	int	[16, 512]	
num_gpus	int	[0, 32]	
learning_rate	float	[1e-10 1.0]	
rescale	float	[1.5259e-5, 10.0]	
channel_shift_range	float	[0, 50]	15 seems reasonable
zoom_range	float	[0, 1]	0.1 is reasonable
rotation_range	int	[0, 359]	all rotations for nadir r
fill_mode	str	(constant, nearest, reflect, wrap)	
horizontal_flip	bool	(True, False)	
vertical_flip	bool	(True, False)	
width_shift_range	float	[0, 0.5]	< 0.2 is reasonable
height_shift_range	float	[0, 0.5]	< 0.2 is reasonable
cval	int	[0, 255]	Default 0

Table 5.3: CTF Experiments List Configurations

allows for a specific dataset to be used to check the trained CNN model's accuracy after each epoch and print out progress messages for user assessment throughout the training process. The 'Test' dataset defines the final check of trained CNN model's accuracy after all epochs completed. This is the only parameter not intrinsically linked to the training process. Other datasets can be used to test outside of the CTF after a CNN is fully trained and available. The 'Networks' parameter allows a string to describe all networks (CNN architectures) to be used in this training.

The 'finetune' parameter allows for multiple methods to update trained models weights. Option 'FALSE' will start with randomly initialized weights and update all layers. This

is also known as training from scratch and is always the slowest approach but often the best method for training a new model. Option 'final_layer' will preserve all weight layers except the last. This the fastest method and works by taking a pre-trained model and only updating the classification layer to match desired classes while preserving model weights related to feature extraction. Option 'all' will start with a pre-trained model's weights and update all weight layers. This is ideal method for jump starting the training process and is often the best method of training.

The 'epochs' parameter allows the user to set a total number of epoch to complete before finishing the training. The 'batch_size' parameter allows for setting the number of images processed at once. A larger batch leads to faster epochs, but requires more memory; this can crash GPUs and ruin trainings if not set properly. The 'learning_rate' parameter allows for adjusting the strengths of weight updates by back-propagation calculations from loss and linearly decays to smaller values as epoch grows larger. Starting with a larger learning rate will dramatically change layer weights and avoid getting stuck in local minimums, while a smaller learning rate will reliably change weights to more perfectly match training data. The remaining parameters allow for adjustments to data augmentation and are described in a following section.

5.4 Network Architectures

The CTF supports training the following networks from either random initialization or via fine-tuning on pre-trained models: VGG16, VGG19 [20], InceptionV3 [21], Inception-ResNetV2 [22], Densenet121, DenseNet169, DenseNet201 [23], MobileNet [24], ResNet50 [25], Xception [26], and NASNet [27]. Each architecture is included with the final CTF due to three reasons: they contain SotA designs that excel in varying aspects of CNN

performance, they readily integrate with CTF's architecture calls, and they're available offline for training on HPCs. Their results are analyzed on aerial imagery in VI and more detailed descriptions can be found in their publications and cited works. These architectures are readily supported by TF and Keras and specified by listing the desired name in experiment parameters. Available networks can be found in the 'networks' sheet of 'experiment_generator.xlsx' which is shown in Table 5.4. For offline use, each has a path defined to the architecture location and the ImageNet-trained weights. The emphasis of this list is less on the architectures themselves, but more on the ability to call whichever is desired. When identical hyperparameters are utilized in training and only the architecture alters, a more direct comparison is available for analysis.

net_id	wgt_id	name	arch_path	wgt_path	comment
1	1	vgg16	$SDATA/dl_i	$SDATA/dl_	ImageNet, top:
2	1	vgg19	$SDATA/dl_i	$SDATA/dl_	ImageNet, top:
3	1	inception_v3	$SDATA/dl_i	$SDATA/dl_	ImageNet, top
4	1	inception_resnet_v2	$SDATA/dl_i	$SDATA/dl_	ImageNet, top
5	1	densenet121	$SDATA/dl_i	$SDATA/dl_	ImageNet, top:
6	1	densenet169	$SDATA/dl_i	$SDATA/dl_	ImageNet, top:
7	1	densenet201	$SDATA/dl_i	$SDATA/dl_	ImageNet, top:
8	1	mobilenet	$SDATA/dl_i	$SDATA/dl_	ImageNet, top:
9	1	resnet50	$SDATA/dl_i	$SDATA/dl_	ImageNet, top
10	1	xception	$SDATA/dl_i	$SDATA/dl_	ImageNet, top
11	1	NASNet	$SDATA/dl_i	$SDATA/dl_	ImageNet, top

Table 5.4: CTF Experiment CNN Architectures

5.5 Data Augmentation

Standard data augmentation methods defined by Keras are available and customizable in the CTF. Example hyperparameter inputs to adjust the data augmentation can

be found in the second half of the 'experiments' sheet as shown in Table 5.5. These include rescale, channel_shift_range, zoom_range, rotation_range, fill_mode, horizontal_flip, vertical_flip, width_shift_range, height_shift_range, and cval. Data augmentation adds feature diversity, overcomes look angle deficiencies, and grows the size of training datasets. The 'comments' column provides suggestions for the user when selecting a hyperparameter value.To establish the initial benchmarks created in chapter VI, data augmentation is used extensively for all experiments and is held to standard values. Evaluating effects of data augmentation manipulation is left to future work. For more details on these hyperparameters, please see the Keras documentation [28].

Experiment Name	rescale	channel_	zoom	rotation	fill_mode	horizontal	vertical	width_	height_	cval
01_Accuracy	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
02_weight_init_scratch	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
03_weight_init_final	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
04_weight_init_all	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
05_Coarse_Grid_01	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
06_FT_Offline	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
07_Pos_Scratch	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
08_Pos_Final	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
09_Pos_All	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
10_PosNeg_Scratch	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
11_PosNeg_Final	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
12_PosNeg_All	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0
13_Transfer_Learning	0.003922	15	0.3	20	mirror	TRUE	TRUE	0.2	0.2	0

Table 5.5: CTF Experiment List B - Data Augmentation

5.6 Hardware and OS

Identifying, accessing, and utilizing hardware capable of CNN training at the desired SotA performance is a large portion of this effort. Each of the machines listed in Table 5.6 is used in part to test and develop the CTF. Computational processing unit (CPU) count and memory are listed for single compute node while graphical processing unit (GPU) and

memory are reported per GPU compute note. CNN training primarily utilizes GPUs and their metrics are most critical for performance. The average GPU node consisted of 2 GPUs and has 12 GB of memory. In total, HPC Thunder has 178 GPU compute nodes (356 GPUs), HPC Lightning has 32 GPU compute nodes (32 GPUs), and HPC Topaz has 32 GPU compute nodes (32 GPUs).

Machine	CPUs, Memory	GPUs, Memory
HPC Thunder	28, 128GB	2 x K40, 12GB
HPC Lightning	10, 32GB	K40, 12GB
HPC Topaz	28, 128GB	K40, 12GB
HP z840	12, 64GB	2 x Maxwell Titan X, 12GB
HP z840	24, 64GB	2 x Maxwell Titan X, 12GB
HP z840	12, 64GB	2 X 1080 TI, 11GB
HP z840	24, 96GB	2 x Quadro K6000, 12GB
HP z840	10, 64GB	Maxwell Titan X, 12GB and Pascal Titan X, 12GB
HP z840	10, 64GB	Quadro P6000, 24GB and Pascal Titan X, 12GB
HP z840	12, 64GB	2 x Quadro K6000, 12GB
HP z840	12, 64GB	Maxwell Titan X, 12GB
HP z840	12, 64GB	Quadro M6000, 12GB
HP z820	12, 64GB	2 X Maxwell Titan X, 12GB
HP z820	12, 64GB	2 X Maxwell Titan X, 12GB
HP z840	12, 64GB	2 x Pascal Titan X, 12GB
HP z840	12, 64GB	2 x Pascal Titan X, 12GB
HP z840	10, 64GB	2 x Quadro P6000, 24GB
HP z840	10, 64GB	2 x Quadro P6000, 24GB
HP z840	12, 64GB	1080 TI, 11GB, Quadro M4000 8GB

Table 5.6: CTF Available Hardware

The priority of supporting updating software versions and implementing on increasingly powerful systems lead to building multiple modules for the multiple software packages (and hundreds of dependencies). All environment modules are tested on each desired OS, for all desired CNN architectures, and for all required software. Final CTF modules used in experiments are version locked to Keras/2.1.1, Keras/2.1.2, TF/1.4.0, and TF/1.4.1 for

reliability. Each is built on the following operating systems and networks: Ubuntu 16.04, Ubuntu 14.04, Linux SUSE on DSRC's HPC Thunder, and Linux Cray on ERDC's HPC ONYX. Non-HPC hardware is critical for software development and stabilization before porting to HPCs. Reliable build scripts and tooling is available to support on any network with these OS's on any network, including other top 100 HPCS such as HPC Topaz, HPC Mustang, and HPC Excalibur.

5.7 Simultaneous Scaling

The fully functioning CTF can scale the number of jobs to simultaneously train CNNs. The final version of the CTF successfully compiles hyper-parameter grid-search experiments (as defined in an experiment definition spreadsheet) into PBS scripted arrays of individual CNN-training instances. After generating PBS scripts, the framework then uses the PBS scheduling tool to execute 100s of unique CNN-trainings on the HPCs specific parameters to submit jobs to the HPC. These parameters are defined on the 'hpc_args' sheet in 'experiment_list.xlsx' and are shown in Table 5.7. Each queue provides variation in priority and GPU availability, and it is necessary to adjust the HPC configurations to match. As allocated HPC hours are used up, a new (often slower) queue is selected to allow training to continue until new hours are provided. Currently, the CTF is deployed on ONYX and Thunder - ONYX is chosen as the fastest available supercomputer, and Thunder is chosen for its stable builds of TF and large GPU hardware.

5.8 Summary

The effort to create a CTF is ambitious, complex, and on the bleeding edge of technology. Building on the preceding CAFFE framework success [9], all matching algorithms and

hpc	queue	CPUs	GPUs	walltime	comments
thunder	GPU	28	2	72	178 Nodes w/2 x K40s
onyx	standard_lw	22	1	72	32 Nodes w/1 P100
topaz	standard_lw	22	1	72	32 Nodes w/1 P100

Table 5.7: CTF HPC Configurations

data have been ported to TF to replicate results. By achieving technological parity in a demonstratively more powerful and flexible framework of TF, technical competency has allowed for the incorporation of drastically more datasets, several more CNN architectures, and significantly advanced performance. The bleeding edge nature of these software tools requires consistent troubleshooting and debugging to overcome rapidly outdated code and tutorials. These challenges are overcome by incrementally increasing technical capacity, developing reliable build scripts with robust error checks, and weekly reviews of latest code and techniques. This performance is increased by an order of magnitude by remotely submitting multiple jobs to HPCs with multi-GPU threading with TF. This all culminates in a CTF which allows a non-technical user to reliably access, define, and distribute SotA machine learning on supercomputers

CHAPTER VI

RESULTS: EXPERIMENTS

This chapter shows results from the 3rd stage solution of a select sample benchmark of aerial imagery and performance of select experiments. In particular, detailed descriptions of CTF's initial experiments, full experiments, time-restrained transfer learning, and future work are given. Each experiment illustrates a potential usage of the CTF and establishes a sample benchmark for future technologists. This massive adaptation of parameters allows the user to run hundreds of individual CNN trainings from a single row of inputs. Because CNN methods require empirical verification to identify the optimized hyperparameters per dataset, the CTF is an effective approach to show every combination and to determine conclusive results. These exhaustive results coupled with interchangeable input datasets provides the valuable tools required for intensive benchmarking analysis.

6.1 Calibration Experiments

The first phase of experiments are defined in #01-06 of 'experiments' sheet shown in Table 6.1. They are designed to be a lightweight trainings that can quickly and thoroughly test CTF functionality of the simultaneous scaling and iterations across multiple hyperparameters, such as dataset, epoch, and learning rate. They also establish initial benchmarks and hyperparameters to use in full experiments. Each calibrates a particular key functionality and are critical for validating the CTF on a new OS where all ported software, data, and scaling require testing.

6.1.1 Initial Calibration

The first experiment, '01_Accuracy', is an accuracy demo which trains only four architectures on previously established hyperparameters to confirm CTF accuracy matches expectation. This was run on each new OS where the CTF is ported as a calibration step. It's single training dataset is an aggregate that includes DLR Munich, VEDAI, and TL; validation dataset is VEDAI, and test dataset is VEDAI. It trains 4 separate architectures VGG16, VGG19, Inception v3, and Inception ResNet V2 by finetuning all layers (from ImageNet) for 50 epochs at a single learning rates of 0.01. It also uses a higher than typical batch size to evaluate memory availability of the training GPU, which. In practice, this experiment ended after a few epochs due to most GPUs only had enough memory for a lower batch size and this initial test revealed the upper limits quickly.

Experiment Name	seed	Training	Validation	Test	Networks	Finetune	epochs	batch_	gpus	learning_rate
01_Accuracy	100	5.1.1+4.1.	5.1.1	5.1.1	1.1, 2.1, 3.1, 4.1	all	50	128	1	0.01
02_weight_init_scratch	1	2.1.1, 3.8.	2.1.1+6.3.1	3.1.1+4.	1.1, 2.1, 3.1, 4.1	FALSE	1	32, 64	1	0.01, 0.001, 0.0
03_weight_init_final	1	2.1.2, 3.8.	2.1.1+6.3.1	3.1.1+4.	1.1, 2.1, 3.1, 4.1	final_layer	1	64	1	0.01, 0.001, 0.0
04_weight_init_all	1	2.1.3, 3.8.	2.1.1+6.2.1	3.1.1+4.	1.1, 2.1, 3.1, 4.1	all	1	64	1	0.01, 0.001, 0.0
05_Coarse_Grid_01	1	5.1.1+4.1.	5.1.1	5.1.1	1.1, 2.1, 3.1, 4.1	FALSE	100	64	1	0.01, 0.001, 0.0
06_FT_Offline	1	5.1.1	5.1.1	5.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	all, FALSE, fi	100	32	1	0.001
07_Pos_Scratch	1	2.1.1 + 3.1	5.1.1	4.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	FALSE	100	64	1	0.001
08_Pos_Final	1	2.1.1 + 3.1	5.1.1	4.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	final_layer	100	64	1	0.0001
09_Pos_All	1	2.1.1 + 3.1	5.1.1	4.1.1	1.1, 2.1, 3.1, 4.1, 5.1,	all	100	64	1	0.0001
10_PosNeg_Scratch	1	2.1.1 + 3.1	5.1.1 + 5.2.1	4.1.1 + 4	1.1, 2.1, 3.1, 4.1, 5.1,	FALSE	100	64	1	0.001
11_PosNeg_Final	1	2.1.1 + 3.1	5.1.1 + 5.2.1	4.1.1 + 4	1.1, 2.1, 3.1, 4.1, 5.1,	final_layer	100	64	1	0.0001
12_PosNeg_All	1	2.1.1 + 3.1	5.1.1 + 5.2.1	4.1.1 + 4	1.1, 2.1, 3.1, 4.1, 5.1,	all	100	64	1	0.0001
13_Transfer_Learning	1	4.1.3	4.1.3	4.1.3	1.1, 2.1, 3.1, 4.1, 8.1,	all	100	32	1	0.001

Table 6.1: Experiments List A

Experiments #02-04 are similar calibration test focusing on checking ability to iterate through multiple datasets and learning rates. The individual experiments differ primarily in their finetune methodology - training from scratch (random weights), just the final classification layer, and from ImageNet respectively. The motivation to separate these into distinct

rows instead of a single, iterated experiment is to allow for quicker analysis of results due to each finetune results saved to an experiment-specific folder and easily separable. All are trained separately on DLR Munich(differing chip sizes of 0.0m, 2.5m, and 5.0m), and on a single SUAV dataset, '191440z_02_pad_0.0'. They validate on the aggregate of DLR Munich with a single Youtube dataset, 'aerial_drone_view_on_highway_in_california_1080p_pad_0.0'. It is then tested on an aggregate of DLR Munich, the first SUAV dataset ('141128z_02_pad_0.0'), and TL. All train for only a single epoch to minimize HPC hours utilized while deploying 96 unique trainings with both validation and test data results. The number of training instances can be calculated by multiplying the number of unique (comma separated) hyperparameters across each experiment. In the example case of #04, there are 24 trainings: (1 seed) x (2 train datasets) x (1 validation) x (1 test) x (4 networks) x (1 finetune method) x (1 epoch) x (1 batch) x (1 GPU) x (3 learning rates). Multiply this by 3 experiments and the second batch size of 32 in #02 accumulates in 96 unique trainings. Analysis of these trainings reveal were CTF may fail and where developers can focus first efforts to ensure new OS compatibility.

6.1.2 Extended Calibration

Experiment '05_Coarse_Grid_01' is the first long-term training. By holding other hyperparameters constant, it provides feedback about which learning rate is ideal for which network and which finetune method. It is very similar to #01 in hyperparameters, differing in a corrected value of 64 for the batch size, allows for a full epoch size of 100 and 3 learning rates. Once run to completion, results confirmed theoretical knowledge that a starting learning rate of 0.001 is ideal when training from scratch (finetune=FALSE), while a learning rate of 0.0001 is better when finetuning final layer or all layers. A larger learning rate is

desirable when bigger changes in weight values are desired - such as training from random. A smaller learning rate is desirable when incremental changes in weight are described - such as finetuning a final layer or from ImageNet. As expected, accuracy values are inflated due to testing on training data and not reported.

The final calibration experiment, '06_FT_Offline', is a quick verification test of the offline performances of all 11 available networks architectures.Offline instances of the network architectures are stable archived alternatives to latest online architectures which is required to utilize a dedicated HPC GPU node. It minimizes difference in network accuracies by using a single set of hyperparamenters and training, validated, and tested only on VEDAI (a small dataset), which provides key insights on where some networks failed. In addition to checking all architectures, it also checks all finetune methods. After 10-20 epochs, most trainings can be manually canceled as networks establish a reliable pattern of loss convergences or fail to improve. Networks that error out, fail to update training weights, or fail to improve accuracy are all discoverable. This experiment led to several minor fixes of faulty offline network layers, refined Keras configurations, and informed design of full experiments. Again accuracy is inflated due to testing on training data. These six calibrations can be submitted sequentially or simultaneously to reliably verify CTF operation. Once all resulting concerns are discovered, addressed, and corrected, full experiments can be usefully undertaken.

6.2 Full Experiments

The second phase of experiments are defined in #07-12. Their design is to be exhaustive trainings over several days on the entire cultivated database, across each architecture, and with each finetune method. All other hyperparameters are held to previously established

best defaults. This is a standard approach to fully training CNNs and final results are available for analysis after trainings complete.

Every positive non-sensor dataset, every network, and every finetune method is in #07-09. For #10-12 this is repeated with the addition of the negative, confuser class datasets. Validation is performed on VEDAI and testing on TL - which are both withheld from training dataset to assess the effectiveness of transfer learning on similarly aerial EO data collected on different sensors and scenes. All trainings are left to run for 168 hours, which is the maximum walltime allowable by HPC. It should be noted that the when the allowance of computing hours on HPCs expire, it becomes necessary to submit all experiments to a slower, more crowded background queue. After being submitted, a typical 168-hour CNN-training may wait a week in the queue before the training started.

From the six experiments, 60 individual trainings are created. With periodic checking on validation accuracies throughout the 168 hour windows, poor performing or successfully converged results can be manually exited as needed to save HPC hours. If all are allowed to complete, a total of 10,080 training hours will be used. Select results primarily from experiment '09_Pos_All' are shown in Table 6.2 below and compared in the last two columns with results from '08_Pos_Final' and '07_Pos_Scratch'. Results from #10-12 are not shown here as they do not significantly differ in accuracies, but are available for further data analysis.

6.2.1 Training Analysis

Experiment '09_Pos_All' emphasized finetuning all network layers from ImageNet (the default for all accuracies shown below). The network architectures used for training are listed in the first column. Four distinct networks are shown to illustrate the diversity of

	Train Learning			Transfer Learning		Finetune Method	
Networks	DLR Munich	SUSEX	YouTube	VEDAI	TL	Final Layer	Scratch
vgg19	61.31%	40.46%	56.09%	24.74%	50.43%	1.90%	1.90%
inception_v3	86.71%	99.38%	99.82%	52.67%	61.04%	77.59%	26.87%
mobilenet	80.78%	99.39%	99.61%	59.36%	60.60%	59.50%	10.03%
resnet50	83.92%	99.46%	99.87%	50.04%	58.11%	16.87%	16.87%

Table 6.2: CNN-Training on Postive Annotations

dataset and relative strengths of each. The next five columns display the accuracies of the trained networks when tested on unique datasets listed in the column header. The group header 'Train Learning' indicates that DLR Munich, SUAV, and YouTube test datasets are similar to the trained dataset – either sharing a video source or is a subset. These are the strongest results and highlight the value of stratifying datasets for validation and testing. In particular, SUAV and YouTube test datasets had the best results of 99% accuracy because there are more SUAV and YouTube training datasets present - combined, they are approximately 80% of the training dataset. DLR Munich tested well in the 80% accuracy range, which is commendable considering it consists of only 7 unique image frames and less than 10,000 image chips of the total dataset. Results were consistent from 3 of the 4 networks, but VGG19 accuracies were significantly lower. This is likely caused by the slower weight-training speed of VGG19 which was still increasing accuracies and had not converged to a final training weight in 100 epochs or 168 training hours.

In the next group header, accuracies from 'Transfer Learning' are displayed. These dataset feature completely different sensors, GSDs, lighting, and viewing angles. These result are noticeably weaker at 50% - 60% overall, but still show potential as a starting point for further finetuning and this is explored in the last experiment. There is also some performance lost due to class mismatch. Since VEDAI and TL are isolated datasets, some

new classes are present in testing that did not occur often or at all in training and visa-versa. This subtracted from the final accuracy score and is a ready area for CTF improvement. VGG19 again performs at lower accuracy for similar reasons.

6.2.2 Finetuning Analysis

The last two columns in group header 'FineTune Method' displays the percent increase in accuracy when finetuning just the final layer from ImageNet weights, as defined in '08_Pos_Final', and finetuning from scratch (random initial weights) as defined in '07_Pos_Scratch'. These accuracies are determined by testing on DLR Munich, chosen for its generally high accuracies while not being potentially overfit. Each architecture responded differently to these finetuning methods, but overall fairly poorly. The best accuracy for final layer finetuning comes from Inception v3 at 77.59% then MobileNet at 59.50%. The most likely reason for this lower accuracy is the fine tune method itself. By only changing the weights of the final layer in deep CNN with several distinctive layers, only ImageNet-familiar features will be extracted and mapped to the new classes defined in the final layer. With training image chips that different distinctly from ImageNet (overhead vehicles instead of front facing dogs), the existing feature extraction cannot reliably map to new classes.

Even more pronounced, the best accuracy for scratch finetuning comes from Inception v3 at 26.87% and then ResNet50 at 16.87%. In theory, training from scratch could provide the most accurate results since weight changes are purely influenced by the training dataset. This may still hold true, but with 100 epochs and 168 training hours all converged to non-optimal solutions for each of the provided architectures. Two potential solutions to overcome this is to experiment with a higher learning weights and with a wide variety of randomly initialized weights. The small learning rate selected in lightweight calibration training didn't

allow for significant weight changes to avoid local minimums in the problem space. It's also probable that a different random initialization of weights would lead to better converged solution - especially if the weights look closer to ImageNet. These two columns confirm that finetuning all layers from ImageNet is consistently the best and fastest method.

6.3 Time-Restrained Transfer Learning

In addition to the calibration and full experiment results, another short experiment '13_Transfer_Learning' is designed to explore rapid transfer learning on the TL dataset. Building from a pre-trained weights trained in '09_Pos_All', multiple networks are finetuned on the TL dataset for a time-restrained 6 hours. Once the number of epoch achieved, the latest accuracy on validation dataset (TL) and the current loss in training are displayed in Table 6.3. The faster networks are able to converge with high accuracy within the tight time limit, and a general convergence is occurring for all featured networks.

Network	Epoch	Val_Acc	Loss
vgg19	8	51.58%	7.9872
inception_v3	7	69.85%	0.5454
inception_resnet_v2	3	86.03%	0.4843
mobilenet	26	98.23%	0.0710
resnet50	10	70.03%	0.4731

Table 6.3: Time-Restricted Validation on TL Dataset

All networks are successfully at rapidly minimizing loss and improving accuracy scores on the TL dataset, with the exception of VGG19 due to it's previously stated configurations. When comparing to previous results, Inception v3 has improved to 69.85% (increase of

8.81%), MobileNet to 98.23% (increase of 37.63%), and ResNet50 to 70.03% (increase of 11.92%). Each of these is an impressive improvement in such a short time frame. The loss values provide a snapshot view which indicate there's even more potential for improving. Inception v3, Inception ResNet v2, and ResNet50 have loss between 0.47 - 0.54 which can ideally be decreased similar to the 0.07 found in MobileNet with just a few more hours of training. MobileNet is the best performer in time-restricted scenarios primarily because it's the fastest network to train. It completes 26 epochs in the same time as the next fastest architecture, ResNet50, completes only 10 - approximately 2.6 times faster. This validates that the TL dataset can be used for training, and previous experiment test accuracies are due to distinct differences in dataset characteristics and transfer learning.

6.4 Future Work: Planned Experiments

With a flexible CTF and established sample benchmarks, future work leads to more diverse use cases to provide insight on the optimal CNN training. In an effort to remove reliance on exhaustive empirically training across 1,000s of GPU hours and 1,000s of hyper-parameter combinations, several potential experiments have been brainstormed and planned for future work. The first two are designed to explore the stability of current datasets. First, experimentation can be performed with the multiple classification layers available in the ontology - size of vehicles, colors of vehicles, and type of vehicles. By training on one type and testing on another, transferability between size, color, and type can be determined. If critical training classes properties can be separated from non-critical, valuable time in training, annotating, and data cultivation can be save. The second experiment plans to explore the effects of variation in hyperparameters across a range of suitable values. For results presented here, initial default values were used in all trainings which occasionally

provided poor accuracies. A massive simultaneous training across all reasonable hyperparameters could generated 1,000s of unique results which could be analyzed for optimal inputs. This would allow for key insights toward ideal hyperparameters for each unique dataset and architectures.

Additional experiments are intended to vary the training datasets. One aims is to train a complete network from all existing datasets, starting from weight initialization from ImageNet and with no data withheld for validation. Next, an experiment will explore unsupervised or weakly-supervised DL techniques to remove the massive annotation requirement for most CNN-training. Another experiment will fully assess the potential of transfer learning – the practice of training on one dataset and testing on another. A further experiment will build on the yet-unused orientation information in the annotations. It's planned to analyze object orientation by training on select views angles of an object (such as 0 - 270 degrees), and then testing on the unseen views of the object (such as the remaining 90 degrees). An experiment is designed to analyze dataset compression by training on large image resolutions and testing on small resolutions - and vise-versa. An experiment to dynamically adjust the size of training sets to see how performance varies and find minimum quantities to achieve effective results. An experiment could better quantify the benefits of data augmentation and diagnose the most influential changes made on training data. Each of these planned experiments can easily spin-off into a new project and establish several useful benchmarks for all future CNN progress.

Finally, to increase hardware performance and total number of training, the CTF can readily be added to new linux OS and HPCs. Reliable build scripts and tooling is available to support on any network with tested OSs on any computer network, including other top 100 HPCS such as HPC Topaz, HPC Mustang, and HPC Excallibur. The performances

provided by the three stage solution of a massive cultivated dataset, customizable CTF on HPCs, and sample benchmarks are foundational in exploring the future of CNNs.

CHAPTER VII

CONCLUSION

To conclude, a three-stage solution has been presented: (1) To cultivate a new dataset containing millions of domain-specific (aerial) annotated EO images; (2) to design a flexible experiment generator framework which is easy to use, can operate on the fastest supercomputers in the world, and can simultaneously train hundreds of unique CNN networks; and (3) to establish benchmarks of accuracies and optimal training hyperparameters. Results show an established CTF with the performance to operates across 11 cutting-edge CNN network architectures, 260 cultivated datasets featuring tens of millions of aerial image chips, and across 3 unique computer operating systems. A select sample benchmark of this performance was performed for 100s of CNN trainings and accuracies displayed for a variety of testing paradigms – including test on withheld training data subset, transfer learning to similar datasets, and fine-tuning on new datasets. The tools, performances, and knowledge developed will benefit future projects for year to come.

BIBLIOGRAPHY

[1] T. Koch, P. d'Angelo, F. Kurz, F. Fraundorfer, P. Reinartz, and M. Körner, "The tum-dlr multimodal earth observation evaluation benchmark," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016.

[2] HPC, "Hpc unclassified systems," https://centers.hpc.mil/systems/unclassified.html, 2018.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[4] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[5] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," *Circuits and Systems (ISCAS) 2010 IEEE International Symposium*, p. 253–256, 2010.

[6] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[8] OpenAI, "Openai five," https://blog.openai.com/openaifive, 2018.

[9] J. Kaster, J. Patrick, and H. Clouse, "Convolutional neural networks on small unmanned vehicles," *NAECON*, 2017.

[10] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[11] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," *CoRR*, vol. abs/1612.01051, 2016. [Online]. Available: http://arxiv.org/abs/1612.01051

[12] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[13] E. Whitney and M. Sprague, "Drag your design environment kicking and screaming into the 90's with modules!" 2001. [Online]. Available: http://modules.sourceforge.net/docs/MC2_whitney_paper.pdf

[14] Top500, "Top500 list - june 2019," https://www.top500.org/list/2019/06, 2019.

[15] J. Langford, "Multiround rsync," 2001.

[16] P. Pro, "Pbs professional open source project," https://www.pbspro.org/, 2019.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[19] S. Razakarivony and F. Jurie, "Vehicle Detection in Aerial Imagery : A small target detection benchmark," *Journal of Visual Communication and Image Representation*, Mar. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01122605

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for largescale image recognition," 2014.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[22] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[23] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[26] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016. [Online]. Available: http://arxiv.org/abs/1610.02357

[27] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017. [Online]. Available: http://arxiv.org/abs/1707.07012

[28] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.