



NATIONAL TECHNICAL UNIVERSITY OF ATHENS
MASTER IN GEOINFORMATIC

**Ship Detection in Satellite Images With Deep Learning
and
a Pythonic Interface on a Hadoop HDFS Platform**

MASTER THESIS

Nikolaos G. Peppes

Supervisor : Mitrou Nikolaos
Professor N.T.U.A

Athens, October, 2018



NATIONAL TECHNICAL UNIVERSITY OF ATHENS
MASTER IN GEOINFORMATIC

Ship Detection in Satellite Images With Deep Learning and a Pythonic Interface on a Hadoop HDFS Platform

MASTER THESIS

Nikolaos G. Peppes

Supervisor : Mitrou Nikolaos
Professor N.T.U.A

Approved by the three-member committee of inquiry at 26th October 2018.

.....
Nikolaos Mitrou
Professor N.T.U.A

.....
Dimitris Argialas
Professor N.T.U.A

.....
Marinos Kavouras
Professor N.T.U.A

Athens, October ,2018

.....

Nikolaos G. Peppes

Master In Geoinformatics National Technical University of Athens

Copyright © Nikolaos G. Peppes, 2018.

All rights reserved.

The author is the first owner of the copyright in the original material which they create. Any subsequent reproduction of a copyright owner's property can only take place with permissions. Taking even a small except of someone else's intellectual property and including it in your thesis (print or digital) must be permitted in law if your work is not to infringe on anyone else's rights.

*«If you want to choose the pleasure of growth, prepare yourself for some pain»
Irvin D. Yalom*

ABSTRACT

Big Data terminology indicates the massive or complex sets of data, in comparison with conventional “small” datasets, in a way that traditional processing operations are inefficient to manage them. Big Data ecosystems help making more accurate analysis, higher quality decision-making, higher operational effectiveness, cost minimizations and decreased risks for the infrastructures. Big Data combines extremely large volume, great velocity of changes and diversity of data and forms, offering greater capability of extensions.

Hadoop is an open-source framework that permits processing over big data as well as archiving them in a distributed environment across computer complex using programming models in an understandable way. It is constructed to be escalated from a single server to thousands of them or computer machines, where regional computation and storage processes comes forward. Hadoop is a Java Software that operates with thousands of nodes with gigabytes or even petabytes of data maintaining data-intensive distributed applications.

Hadoop is composed of two crucial components: HDFS (Hadoop Distributed File System) which is located on top of the filesystems of the fundamental operating systems, and MapReduce, a framework that offers the capability of distributed processes, by splitting the application into smaller operational chunks, in such a way that any node in the cluster can execute (or re-execute each of) them.

Convolutional Neural Networks (CNN) are the leading architecture in Deep Learning that are used for Image Processing Techniques. Convolutional Neural Networks are a category of Neural Networks seen to be more promising when working on image data. They work on images in a manner similar to the human brain: by finding smaller details and then working their way up to more abstract features.

The objective of the present master thesis is the deployment of a Hadoop Distributed File System Interface using the Python Programming Language to highlight the advantages of Big Data Technology. Following this scope, we develop an Artificial Intelligence Application using Deep Learning Technology, which detects ships in satellite images, using all the fundamental tools of Hadoop Ecosystem, in a user-friendly manner.

Chapter 1 provides an introduction to Big Data and the Hadoop Ecosystem. **Chapter 2** presents a revision of the theoretical framework that is used for the processing of Hadoop, Hbase and Deep Learning Applications. In **Chapter 3 and Chapter 4** described is the development procedure of a pythonic interface for interaction with the Hadoop Distributed File System (HDFS) and the Hbase environment, respectively. **Chapter 5** presents a deep learning framework, capable of detecting ships in satellite images. The thesis is concluded in **Chapter 6**, where some proposals for future research, based on the state of the art at this research field, are suggested.

Key Words

Hadoop, Hbase, HDFS, MapReduce, Artificial Intelligence, Deep Learning, Big Data, Neural Networks, Image Recognition, Satellite Images, Convolutional Neural Networks, Python, hdfs3, Happybase.

Acknowledgements

This project would not have been possible without the support of many people. I would like to thank my adviser, Nikolaos Mitrou, who helped me make some sense of the confusion by offering his valuable advice and guidance. Finally, thanks to my parents, my sister Christina and my girlfriend Evgenia who stand by me during this long process, always offering support, courage and love.

Nikolaos G. Peppes
October 2018

Table of Contents

1. Introduction.....	12
1.1 Motivation.....	12
1.2 Goals.....	13
1.3 Outcome.....	13
2. Theoretical Framework.....	15
2.1 The Big Data Phenomenon.....	15
2.2 Types of Big Data.....	16
2.3 Differences between Big Data and Traditional Data Sources.....	16
2.4 Apache Hadoop Ecosystem: An Open Source Big Data Project.....	17
2.5 Data Storage and Analysis Problems.....	17
2.6 RDBMS and Hadoop Databases.....	18
2.7 The Apache Hadoop Project.....	19
2.8 Deep Learning: A Sub-field of a Broader Family of Machine Learning....	20
2.9 Machine Learning and Intelligence.....	21
2.10 The ‘deep’ term in Deep Learning.....	21
3. Hadoop With Python.....	25
3.1 Hadoop Distributed File System (HDFS).....	25
3.2 Overview of HDFS Architecture.....	25
3.3 Hadoop Distributed File System Ecosystem Configuration.....	26
3.4 Working with HDFS using Python Programming Language.....	29
4. HBase Infrastructure Development.....	37
4.1 Limitations of the Traditional Databases.....	37
4.2 Architecture of HBase.....	37
4.3 Comparison Between HBase and RDBMS.....	38
4.4 Interaction with HBase by Developing a Pythonic Interface.....	39
5. Ship Detection In Planet Satellite Imagery Using Deep Learning.....	47
5.1 Application Description.....	47
5.2 Ship Detection Dataset Parameters.....	47
5.3 Convolutional Neural Network Architecture.....	49
5.5 Prediction Results.....	52
5.6 Visualization of Prediction Results.....	53
6. Conclusions.....	59
APPENDIX A. ΠΕΡΙΛΗΨΗ.....	60
APPENDIX B. Interaction Between HDFS and DL Model.....	64
REFERENCES.....	68

Table of Figures

Figure 1. The Three V Description of Big Data.....	15
Figure 2. Hadoop Ecosystem sub-projects.....	20
Figure 3. Artificial Intelligence, Machine Learning and Deep Learning relation.....	20
Figure 4. Machine Learning Programming Paradigm.....	21
Figure 5. Deep representations learned by a digit-classification model.....	22
Figure 6. Parametrization of neural network using its weights.....	22
Figure 7. Measurement of network's output quality using loss function.....	23
Figure 8. Loss score is used as a feedback signal to adjust weights.....	23
Figure 9. HDFS Cluster Architecture.....	26
Figure 10. HDFS Initialization.....	27
Figure 11. YARN Initialization.....	28
Figure 12. Java Virtual Machine Process Status Tool (JPS).....	28
Figure 13. NameNode Overview Using Web User Interface Communication.....	29
Figure 14. Hadoop UI Defaults TCP Ports.....	29
Figure 15: Connection to HDFS Cluster using IP 192.168.0.10.....	31
Figure 16. List of the Contents Files and Destination Folders in Root of HDFS.....	31
Figure 17: Create New Direction in HDFS.....	31
Figure 18: Getting Information for Path called '/shipsnet'.....	32
Figure 19. Getting Information for Path called '/scenes'.....	33
Figure 20. Import Unstructured Data from Local to HDFS.....	33
Figure 21. Retrieve Unstructured Data from HDFS to Local.....	34
Figure 22. Batch Data Import from Local to HDFS.....	34
Figure 23. Change of Owner and Group on a Path or a File.....	35
Figure 24. Change of Permissions on a Path or a File.....	35
Figure 25. HBase Architecture.....	38
Figure 26. HBase Table Schema.....	40
Figure 27. HBase Master and Slave Initialization.....	40
Figure 28. HBase Master Web User-Interface.....	41
Figure 29. HBase Slave Web User-Interface.....	41
Figure 30. Remote Connection to HBase Host.....	42
Figure 31. List of All Available Tables in HBase.....	42
Figure 32. Regions and Column Families Infos for a Specific Table.....	42
Figure 33. Existence, Creation and Delete of Tables into HBase.....	43
Figure 34. Storing and Deleting Data in HBase Table.....	44
Figure 35. Basic Functions for Retrieving Specific Values from HBase Table.....	44
Figure 36. Full Table Scan Iteration.....	45
Figure 37. Batch Storage in HBase Table.....	45
Figure 38. Storage Direction of Dataset on Hadoop Distributed File System.....	48
Figure 39. 'Ship' Class Sample Images.....	48
Figure 40. 'Non-Ship' Class Sample Images.....	49
Figure 41. Appropriate Libraries for Ship Detection Network Configuration.....	49
Figure 42. Typical Structure of Convolutional Neural Network.....	50
Figure 43. Model's Hidden Layer Configuration Output.....	51
Figure 44. Ship Detection Learning Process Configuration.....	52
Figure 45. Model Compilation & Prediction Results of Ships Detection Application.....	52
Figure 46. Accuracy and Loss Plots Between Training and Validation Data.....	53
Figure 47. Save Keras Models into HDFS for Future Load.....	53
Figure 48. Visualizations of 'Ships' Labels Predictions.....	54

Figure 49. Visualizations of ‘Non-Ships’ Labels Predictions.....	54
Figure 50. Searching on Satellite Images of Bays for Ships Existence.....	55
Figure 51. Creation of Output Storage Direction into HDFS.....	55
Figure 52. Inporting of Output Data into HDFS.....	56
Figure 53. Searching for Directory Contents in HDFS.....	56
Figure 54. Ship Detection Model using Hadoop Ecosystem Interaction Schema.....	57
Figure 55. Interaction Between HDFS and Compilation Process.....	64
Figure 56. Model Weights Re-Load from HDFS.....	65
Figure 57. Retrieve Satellite Image and Make Prediction Process.....	66
Figure 58. Output Results Stored into HDFS Directory.....	67

Chapter 1

Introduction

1. Introduction

The world has turned into information society that highly relies on data. Great amount of data production in an increasing rate was caused by the rise of diversity among network platforms, digital conversions by a massive number of procedures, evolution on wearable, portable devices and miscellaneous categories of sensors. Since information systems generate enormous amounts of records every minute, every second, it seems the world is reaching the level of data overload. It is obvious now, that for processing such volumes of data, an enormous capacity of storage and computing resources is required. Whereas the growth of capacity is limited by evolution of hardware and technologies, the growth of the data volume is in fact unlimited.

Governments, businesses and education operations, even lifestyle, has been affected from the rapidly usage and growth of the Internet. Nowadays, the high rate and the variety of the types of data that is daily produced exceed the traditional data storage methods. Over the past two decades, every scientific (and not only) field generated an amazing production of data establishing this way new pathways to social learning. Analysing and manipulating these datasets, can help infrastructures, organizations and businesses on taking decisions in a right as well as competitive way.

In the beginning, infrastructures adopted transaction processes that naturally were composed and working with Relational Data Base Management Systems (RDBMS) and simple data analysis methods via Structured Query Language (SQL) queries for their daily procedures for planning and decision making purposes. The expansion of the data, and more importantly the unstructured form of it rendered traditional types of storage inadequate to handle and manipulate these types, as a result making more important the requirement for new storage techniques and analytic methods on the field of big data.

1.1 Motivation

Discussions on a set of related topics mentions that the growth of data is unlimited. What the world is going to do about the data overload, which seems unstoppable? How to handle and process all data? How can it be possible to retrieve the relevant information, within a specified time? What is the balance between cost of retrieval and value of information?

Additionally to the above challenges is the requirement to manipulate and visualize the information in such a way that the result is comprehensive and understandable.

Adopting new technologies requires to process, discover and analyse these massive datasets that cannot be dealt while working with traditional types of databases architectures due to the lack of capacity resources in terms of computation and storage [1].

An important challenge for the present and the future is related to storage and fetch processes among a heavy quantity of structured and unstructured data within a desirable time interval. Traditional storage methods reveal many limitations on manipulation and processing at the huge amount of data, which forced to the appearance of the Big Data terminology[2]. Internet technology evolution caused a great boost on big data field, as the collection and sharing of the data in a raw format became easier. Storage, process and manipulation of the above type of huge volume of data consists the main scope of Big Data field, in accordance with minimum time delay and high precision.

1.2 Goals

The Big Data Phenomenon, which is characterised by rapid growth of volume, variety and velocity on data information assets, thrives the paradigm shift in analytical data processing.

The aim of the current master thesis is the deployment of a user-friendly Hadoop Ecosystem interface using the Python Programming Language in order to highlight the advantages of Big Data Technology. Following this scope, an Artificial Intelligence Application was developed using Deep Learning Technology, a sub-field of machine learning, which detects ships in satellite images, using all the fundamental tools of Hadoop ecosystem in a user-friendly way.

1.3 Outcome

The scope of the thesis is dedicated to experimentation and deployment of a Hadoop Interface and a Deep Learning application, which detects ships on images received from satellites, displaying port highlights including ships and docks. This way, we use a lot of utilities of Hadoop in order to manipulate the appropriate datasets for the deep-learning model.

The first (theoretical) part of the thesis summarises the state-of-art of this problem, defines the drivers and consequences of Big Data Phenomenon, and introduces paths for manipulation of Big Data, Hadoop Ecosystem and Deep Learning techniques.

The practical part of the thesis summarises the development in Hadoop environment, usage of common properties with RDBMS, manipulation of Hbase as an alternative approach to column-oriented database, and finally fully exploitation of their capabilities, along with the prediction results of a Deep Learning application. The experiment demonstrates the application of selected methods that are discussed in the theoretical part. The first part of the experiment includes generic operations for data storage and retrieval in a Hadoop platform, while the second part defines the model and applies its results in appropriate testing datasets.

Chapter 2

THEORETICAL FRAMEWORK

2. Theoretical Framework

Over the last years, around 90% of the total data was produced extensively and rapidly, making the 'Big Data' terminology the most widespread one in businesses across many industry sectors. Chapter 2 focused on the advantages of big data in addition to Hadoop Ecosystem and Deep Learning Technologies.

2.1 The Big Data Phenomenon

Big Data, Business Analytics and Data Mining terminologies are frequently used as advanced analysing tools among infrastructures that come along with huge data sources. The Big Data term is differentiated from the other two when more complex interchanges and bigger data quantities demand distinguished technologies and approaches.

According to Doug Laney (2003), the Big Data description includes the 'three-V' definition (Volume, Velocity and Variety), a term which is illustrated in Figure 1.

- Volume: Giants bulks of information data sized up to terabytes even zettabytes.
- Velocity: Large amounts of data with high speed and refresh proportion.
- Variety: Data can show up in diverse types of structure: structure data, as database tables for example, semi-structured data as JSON or XML forms and unstructured data, as images, videos and audio formats (etc).

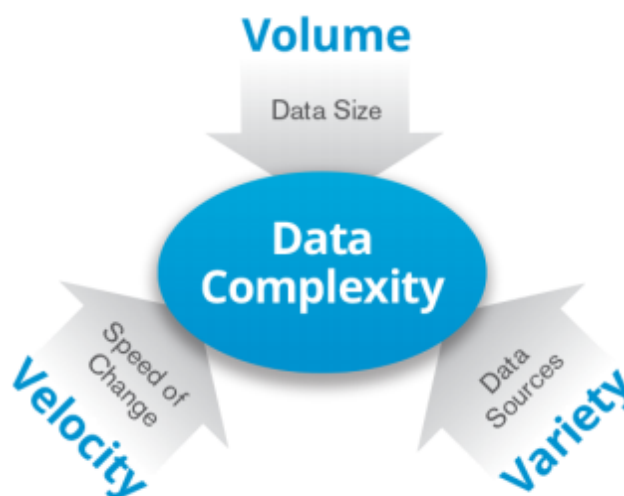


Figure 1. The Three V Description of Big Data
(Copyright 1995-2015 GRT Corporation)

According to Gartner (2012) : «Big Data is high-volume, high-velocity and/or high variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making and process automation. While big data certainly involves having a lot of data, big data does not refer to data volume alone. What it means is that you are not only getting a lot of data. It is also coming at you fast, in complex format and by a variety of sources» [1].

2.2 Types of Big Data

There is a big amount of traditional data that has been used for a long time among businesses, companies and organizations. At this moment, new types of data are captured alongside with the highly increased development of new database environment fields, such as:

- Text data: One of the most commonly used types of data, where specific patterns are extracted from texts for further analytical procedures.
- Web data: Diverse types of data are collected in a high rate from web sources, as searching results, reading reviews and advertisement results. Consequently, customer segmentation and focused advertisement can be improved.
- Social network data: Nowadays, social network applications, as Facebook, Twitter, Instagram and WhatsUp implement network analysis, which can provide important information, such as customers interests, targeted advertising and/or real-time information sharing process.
- Time and location data: Wi-Fi compatible devices, GPS, Mobile Phones as well as other related types of smart devices, set time and location related data as an increasing 'Big Data origin' during recent years. The manipulation of the time and location data from the companies can provide time and location services, as weather applications, but the information sensitivity and the user privacy must be protected from unauthorized accesses.
- Smart grid and sensor data: At the time being, Internet of Thing (IoT) technology provides a huge amount of informations from sensors used in cars, industries, meteorological stations even on forests (as an example are used for fire detection purposes), supplying by this way more efficient methods for analysis and problem diagnosis procedures.

Holding and manipulating a big amount of data series provides the capability of combine the output results from variant sources.

2.3 Differences between Big Data and Traditional Data Sources

Following Bill Frank's book 'Taming the big data tidal wave', there are some crucial differentiations among big data and conventional (traditional) data sources [2].

Big data consists of new types of data format, compared to traditional data sources commonly used so far among businesses and organizations. Higher data speed and frequency requirements highlight the necessity of the existence of a more analytic and efficient level of data manipulation. The capability of analysis procedures on big data sources consists a great tool for any user, where variant actions can be accomplished on it, reaching in an extensive data mining process.

Furthermore, the massive generation of semi-structured and unstructured data over the recent years, defined an additional difference between big and traditional data, based on the type of data that is being processed by each type of source.

Structured data is a type of a standardized information data format with specified data fields and labels. Some structured data examples are files that hold relational databases or spreadsheets informations.

On the other hand, unstructured data formats does not contain a specific or pre-defined structure, resulting in difficulties during analysis or processing. Some basic examples of unstructured data are audio, video and photos formats, presentations or e-mail messages.

Semi-structured data defined a different data type between structured and unstructured data that it does contains some structure, making by this analyse and manipulation processes more feasible. XML files and Not Only SQL (NoSQL) records are some representative examples of semi-structured data.

2.4 Apache Hadoop Ecosystem: An Open Source Big Data Project

According to Gartner: «Big Data necessitates a new type of data management, which bears the trademark of highly scalable, massively parallel and cost-effective» [1],[3].

In 2014, one of the largest installed Hadoop cluster was operating at 455 petabytes. Till then, neither any data warehouse nor a parallel relational database architecture had even approach this numbers. Hadoop's performance increases when unstructured data format, such as video, audio or photo, is processed [4].

It is important to be clarified that new technology approaches, such as Hadoop, are not intended to completely replace but work alongside with relational databases systems, meeting the desirable spot for a hybrid-parallel platform, which will come through structured data handling in combination with large datasets of unspecified structure, acting as unstructured data.

The Hadoop ecosystem is regularly appointed to compromise with big data processes containing a lot of elements from the stack below:

- Amazon Web Service, known as AWS, for infrastructure (in the cloud)
- Apach Hadoop Distributed File System, known as HDFS, for distributed file system
- MapReduce or Spark for distributed programming model
- HBase or Cassandra for non-relational distributed database management system
- Hive for execute SQL on top of Hadoop
- Mahout for Machine Learning and math library, on top of MapReduce
- R for data analytincs and visualization

Most of the widely used analytical techniques fall into one of the following categories:

- Statistical methods, forecasting, regression analysis
- Database querying
- Database warehouse
- Machine learning and data mining

2.5 Data Storage and Analysis Problems

As Tom White mentioned in his book '*Hadoop:The Definitive Guide* (2009)', a main problem occurs when: «the storage capacities of hard drive disks have expanded massively over the years, access speeds—the data rate which can be read from drive shave not kept up.

One typical drive from 1990 could store 1370 MB of data and keep a transfer speed of 4.4 MB/s, thus someone could read all the data from a full drive in around five minutes. Almost twenty (20) years later one terabyte drives consists the norm, while the transfer speed is almost at 100 MB/s, spending more than two and a half hours to completely read the data off the disk» [5], [6].

By combining a lot of pieces of hardware, the probability of failure in any of them becomes much higher. A replication process, where replicas of the used data are generated is frequently used, nowadays.: duplicated parts of the data are backed-up by the system in case of failure, resulting in copies ready for use at any time or circumstance.

Another main problem consists of the required combination of one disk's data read alongside with existing data from any other hard disk source. The combination of data received from miscellaneous distributed systems or disk origins in a correct way, consists of one of the leading challenges for data storage and analysis problems. MapReduce provides a programming model that summarises a complete read and write disk output, converting it into a computation process over sets of keys and values. MapReduce has also reliability built-in processes, as Hadoop Distributed File System (HDFS).

This constitutes Hadoop's main advantage: a reliable shared storage and analysis system, where Hadoop Distributed File System (HDFS) implements storage processes and MapReduce provides analysis process capabilities. These two process capabilities are Hadoop's kernel, even if a decent number of components also exists [5],[6].

2.6 RDBMS and Hadoop Databases

Seek time is improving slower than transfer rate. Seeking process describes the movement of the disk's head to a distinct place to read or write data, demonstrates the latency, whereas associates the transfer rate to a specific disk bandwidth.

The domination, by seeks, of the data access patterns results in longer periods of time demand during read and/or write processes over larger datasets or parts of them, whilst increase the disk transfer rate. Contrarily, the usage of a traditional B-Tree for the update procedure on a small proportion of records in a conventional database, provides better results. MapReduce is more efficient than B-Tree, for the updating process on the majority of a database, by implementing sort/merge functions to rebuild the database [8].

Hadoop's MapReduce is appropriate for problems that require the manipulation of the entire dataset, in a batch fashion, especially for ad hoc analysis. An RDBMS is pretty good for implementing queries, deletes or updates, where the dataset has been pointed to allocate low-latency recovery and update times on a small quantity of data. Applications where the data is written once, and read many times are coordinated better with MapReduce, whereas a relational database is good for datasets that are frequently updated [9].

Another contrast between MapReduce and an RDBMS is related to the structure of the datasets that they operate on. MapReduce performs much better on unstructured or semi-structured data, since it is designed to interpret the data during processing time, while relational data is designed to keep its purity, and get rid of redundancy.

Table 1 figures the comparison between relational databases and MapReduce :

	Traditional RDBMS	Hadoop (MapReduce)
Data Size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	NonLinear	Linear

Table 1. Comparison between RDBMS and Hadoop

As relational databases begin integrating and mixing some of the ideas from MapReduce and, in the other direction, as higher-level query languages embed in MapReduce make MapReduce systems become more convenient to traditional database programmers.

2.7 The Apache Hadoop Project

Hadoop consists of a combination of sub-projects, that are hosted by the Apache Software Foundation and are responsible for distributed computing. Even if, Hadoop is acknowledged for MapReduce and its distributed file system, called HDFS, the other components provide equivalent services, as they are structured on the core to add higher-level abstractions.

Figure 2 describes some major components of the Apache Hadoop Ecosystem [7]:

- **Core:** A set of components and interfaces for distributed file systems and general I/O procedures.
- **Avro:** A data serialization system appropriate for efficient and continuous data storage.
- **MapReduce:** A distributed data processing model and execution environment that runs on large clusters of commodity machines.
- **HDFS:** A distributed file system that runs on large clusters of commodity machines responsible for storage processes.
- **Pig:** A data flow language that runs on Hadoop Distributed File System clusters responsible for executions and analysis procedures on very large datasets.
- **Hbase:** A distributed, column-oriented database responsible for underlying storage, as well as queries and batch implementation processes.
- **ZooKeeper:** An embedded service implementation that provides fundamentals which can be used for building distributed applications.
- **Hive:** A distributed data warehouse that provides the capability of querying process on the dataset by the usage of a query language based on SQL.
- **Chukwa:** A distributed data collection and analysis system that uses HDFS for storage procedures data in HDFS and MapReduce for analysis processes.

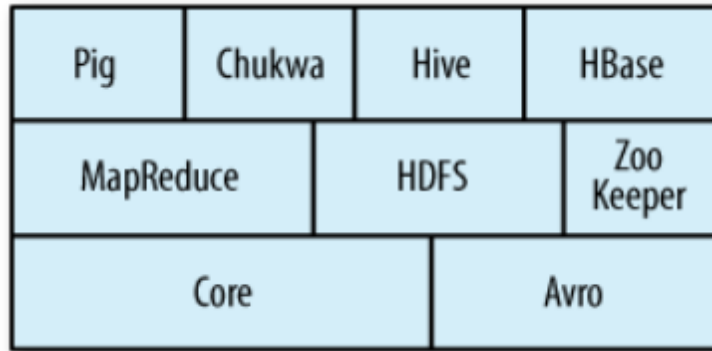


Figure 2. Hadoop Ecosystem sub-projects

(Copyright 2015, Tom White, Hadoop: The Definitive Guide, 4 Edition)

2.8 Deep Learning: A Sub-field of a Broader Family of Machine Learning

Artificial Intelligence (AI), machine learning and deep learning technologies arise in an increasingly way in many articles, publications as well as practical applications. We frequently hear about a future that will contain intelligent chat-bots, self-driving cars, and virtual assistants, where human jobs will be inadequate and the most parts of economic activities will be handled by robots or AI agents. The recognition of the noise signal, is critical, for a machine-learning engineer or data scientist, in order to highlight any alteration among the existing AI application releases [10].

Figure 3 defines the relationship between Artificial Intelligence, Machine Learning and Deep Learning environments.

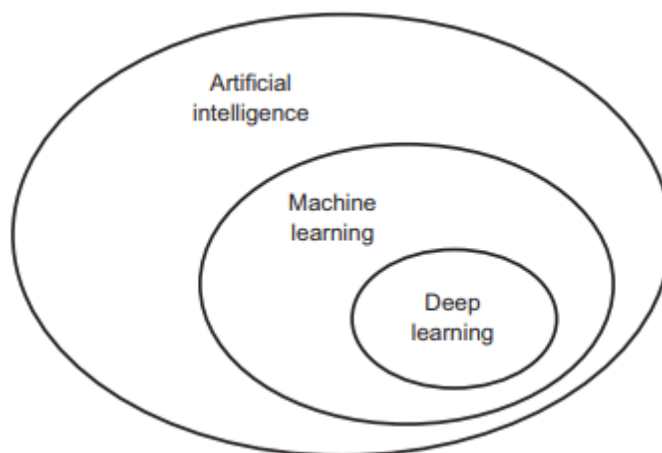


Figure 3. Artificial Intelligence, Machine Learning and Deep Learning relation

(Copyright 2015, Francois Chollet, Deep Learning with Python)

2.9 Machine Learning and Intelligence

In conventional programming models, data and rules are imported and combined into them, while answers are extracted as output results. On the other hand developing and implementing machine learning models, require data as well as answers as input, while processing them, results in rule extraction. Later, these rules can be re-applied in order to re-produce original answers on new datasets. Figure 4 displays input and output flows processes during a Machine Learning Model execution.

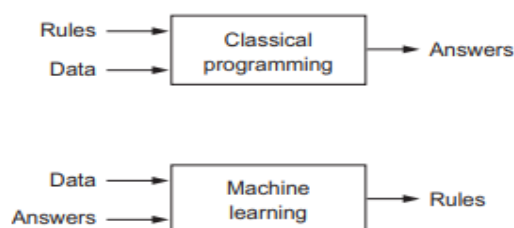


Figure 4. Machine Learning Programming Paradigm

(Copyright 2015, Francois Chollet, Deep Learning with Python)

A machine-learning training process takes place by the execution of a specific programming implementation. It is developed with a massive amount of cases suitable for a specific assignment. As an illustration, someone may desire to automate the process for tagging pictures. He, could generate a machine-learning algorithm with many cases of pictures that are previously tagged by humans, where the system would learn statistical rules for combining specific tags to distinct pictures.

Machine learning technology inclines to deal with large as well complex datasets for which classical statistical analysis such as Bayesian would be infeasible. As a result, machine learning, and especially deep learning theory is based on discipline where regularly, rules and ideas are demonstrated experimentally than theoretically [21].

2.10 The ‘deep’ term in Deep Learning

Deep learning consists of a new path on learning conditions from data which focus on learning consecutive layers of additional delegations. The ‘Deep’ term in ‘Deep Learning’ terminology represents the defined number of layers that are partly responsible for the construction of the model which will result in further manipulation process of the data. In other words, the ‘Deep’ term represents the exact depth of the model. Modern deep learning models frequently contain tens or even hundreds of consecutive layers, meanwhile other machine learning model approaches tend to focus on learning processes that use one or two representation layers of the datasets.

The above mentioned representation layers usually implement learning as part of models called neural networks. The ‘Neural’ term in ‘Neural Network’ terminology is related to neurobiology. Some of the basic approaches in deep learning were grown partially by drawing ideas during analysis over specific sections of the brain. Nevertheless deep learning are not models of the brain but a mathematical framework that provides learning representations from the data [21].

The aim of the existence of layer representations is to discover a suitable method of data representation in order to accomplish a machine learning task. The situation of what a layer performs to its input data is stored in the layer's weights, which practically are a batch of numbers.

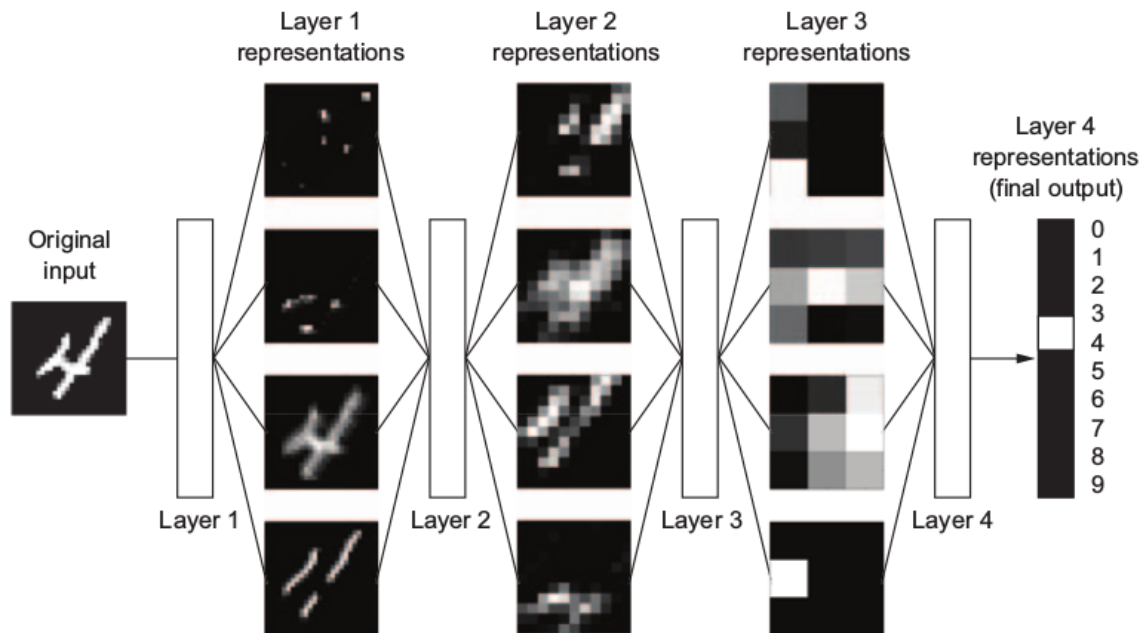


Figure 5. Deep representations learned by a digit-classification model
(Copyright 2015, Francois Chollet, Deep Learning with Python)

Approximately, the implementation of conversion process by a layer is configured by its stored weights, well-known as the parameters of a layer. The 'learning' term denotes the decision making process of a set of values for the weights among all layers in a deep learning network, in a way that it will correctly determine inputs to their associated outputs. Many times, a deep neural network model may contain a massive amount of parameters consisting the pre-mentioned proceedings as a time-consuming work.

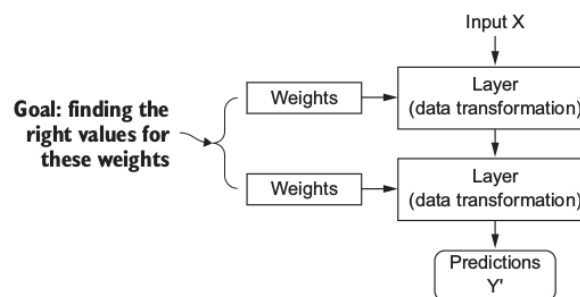


Figure 6. Parametrization of neural network using its weights

(Copyright 2015, Francois Chollet, Deep Learning with Python)

The loss function of the network, still known as the objective function is responsible for receiving the prediction results of the model network and calculating a distance score, describing this way the deviation of the output/predicted results that the model generated from the expected results or, in other words specifying the error that the model performed

during the estimation process. The purpose of the specific function is the regulation of the output of a deep learning model (or generally the output of a neural network).

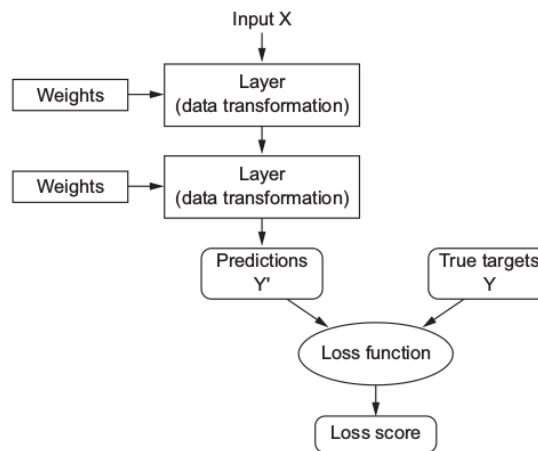


Figure 7. Measurement of network's output quality using loss function

(Copyright 2015, Francois Chollet, Deep Learning with Python)

The responsibility of the optimizer function is the maximization or the minimization of the loss function, by using its gradient. The weights of the network delegate random values, so that the network executes a sequence of arbitrary conversions. Practically, the network's output is far from what it should ideally be, while the loss score is also high. After every training loop, the weights are approaching the correct values, while the loss score is decreasing. As far as the loss function is minimized the output results comes closer to the expected results.

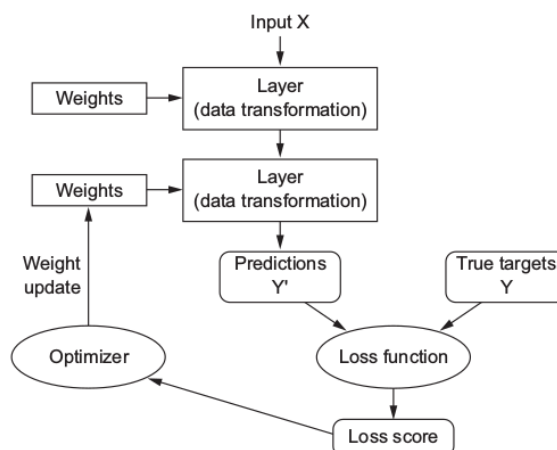


Figure 8. Loss score is used as a feedback signal to adjust weights

(Copyright 2015, Francois Chollet, Deep Learning with Python)

Chapter 3

HADOOP WITH PYTHON

3. Hadoop With Python

Chapter 3 introduces and describes the core concepts of Hadoop Distributed File System (HDFS) using Python Library, `hdfs3`, which is a slight python envelopment situated on the C/C++ programming language and `libhdfs3` library, supplying direct access on both from Python.

`Hdfs3` is a Python package, that provides a Python client library, allowing Hadoop's Filesystem (HDFS) to be accessed in programmatic way, from Pythonic Interfaces or applications, using all the built-in commands.

3.1 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System, known as HDFS, is a Java-base distributed, portable and expandable filesystem constructed for spanning large clusters of servers. HDFS designation is based on GFS, Google's File System [7]. HDFS can occupy a large amount of data, so as to grant access to many clients across network, while its storage capability on very large files, in a reliable and scalable way, makes it excel to other similar distributed filesystems.

Usage of a block-structured filesystem accomplished the ability on HDFS to save a massive amount of data, regularly up to petabytes. Files are separated into blocks with fixed size that can be stored among installed-machine-clusters. Generally, files gathered from a various number of blocks, that are not included totally, on a one and only host.

HDFS provides trustworthiness by replicating imported data on block-files and distributing duplicated files over the cluster. During Hadoop installation and configuration every fixed-size created block exists three times on the cluster, as replication factor during configuration process equalled to three. Block-level replication enables data restoration and availability even if the any of the machine may fail.

3.2 Overview of HDFS Architecture

The architecture of HDFS is expressed by two main processes: a process established as the Master Node (or NameNode), which keeps the important metadata information for the operation of filesystem, and one or more Slave Node (DataNode) processe(s), that used for the storage of the block files. The Master and SlaveNode (NameNode and DataNode) processes can implement on a single host, but for HDFS clusters generally preferred, a dedicated server executing the MasterNode's processes whilst one up to thousands of hosts executing the SlaveNode process [11],[12],[13].

The MasterNode (or NameNode) is the most major and necessary host in HDFS. For any given file in `hdfs` masternode knows not only the list of blocks, but also the location of blocks. The metadata informations for the whole filesystem, filenames that includes the location of every block, file as well as file grants and permissions, is collected from it. The MasterNode stores the entire metadata information structure in memory, in order to succeed fast access to it, so needs larger memory in comparison with SlaveNodes machines. The replication factor of blocks, is tracked from MasterNode protecting from data loss during machine failures. Because the MasterNode constitutes a single point of failure, a common used recovery strategy is the usage of a secondary MasterNode to regenerate snapshots of the primary NameNode's memory structures, limiting by this way the risk of data loss in a possible MasterNode failure[5].

SlaveNodes (or DataNodes) defined as the machines that physically store the blocks within HDFS, which they named so, because each node holds the actual data for the cluster. SlaveNodes are typically commodity machines with large storage capacities. Each SlaveNode knows the list of blocks it is responsible for, whilst does not care about the other blocks or SlaveNodes. HDFS operation continues normally, even if a SlaveNode fails, the lost blocks are replicated from the MasterNode, ensuring that the minimum replication factor demand comes along with every distributed block.

Figure 9 illustrates the mapping of files to blocks in the NameNode, and the storage of blocks and their replicas within the DataNodes.

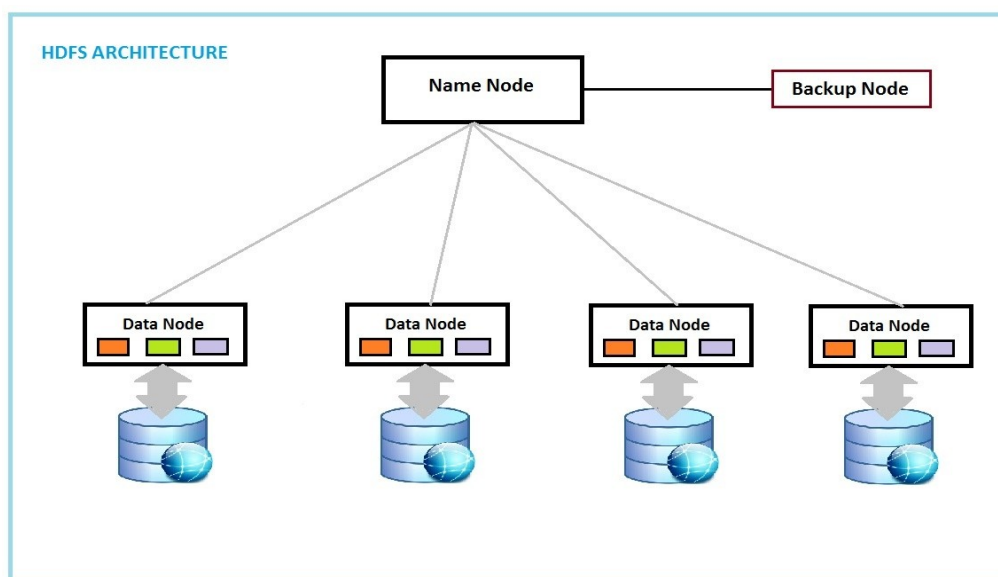


Figure 9. HDFS Cluster Architecture

(Copyright, Lavish Jain, Hadoop 2.x (HDFS and YARN features))

3.3 Hadoop Distributed File System Ecosystem Configuration

HDFS provides redundant storage for big data by storing that data across a cluster of cheap, unreliable computers, thus extending the amount of available storage capacity that a single machine alone might have. However, due to the networked nature of a distributed file system, HDFS configuration is more complex than traditional file systems.

Hadoop cluster architecture is responsible used for high computing operations and estimations, whilst is more effective when it separated into a decent total of files with big size. During the development of the Deep Learning Application, which detects ships across satellite images (subject deployment in Chapter 5), we stored a modest number of images to train our AI model, into HDFS folder destinations, using a Pythonic programming interface that includes all the built-in functions of HDFS ecosystem.

As mentioned in, above, 3.2 paragraph, Hadoop clusters are comprised of three different node types: master nodes, worker nodes, and client nodes.

Master nodes oversee the following key operations that comprise Hadoop: storing data in the Hadoop Distributed File System (HDFS) and running parallel computations on that data using MapReduce. The NameNode relates HDFS with the data storage operations, while the JobTracker supervises and relates data's parallel processing using MapReduce capabilities.

Worker nodes arise as the majority of virtual machines and accomplish the job of data storage and computation executions. Each worker node runs both a DataNode and TaskTracker service that communicates with, and receives instructions from their MasterNodes(s). The TaskTracker service is supplementary to the JobTracker and the DataNode to the NameNode.

Client nodes contain Hadoop framework installations with the appropriate cluster configurations, although they are neither master nor worker nodes. Client nodes' obligation is to surrender MapReduce jobs expressing the way that data should be processed, and after that displays the outputs of the job when processing just after completion.

In order to store and process data, firstly we need to complete the deployment of the Hadoop distribution cluster. All implementations and experiments are based on Hadoop cluster. As the scope of the current Master Thesis is not the installation of Hadoop Environment, there was not extensive reporting for the installation process, but only for the most important points to understand the (virtual) Hadoop infrastructure.

For our experimental research, was chosen a client-server architecture, using Oracle's virtual machine (server side) to create a Hadoop cluster in a pseudo-distributed operation, which runs all modes in one system, but on separate Java Virtual Machines (JVM) environments, in order to simulate, the preferred from enterprises way, Fully-Distributed Mode where each master and slave service running in separate system and different JVM. During research process, Hadoop 2.7.3 and Java 8 releases were selected, installed and configured in the way of the pre-mentioned Pseudo-Distributed mode.

With reference to the above virtual cluster installation included a memory of 3000 MB and storage capacity of 100 GB. Shell scripts are used for starting related Hadoop Daemons, while SSH is also needed to be installed on (each) host.

Replication factor set to three (3), as well as a data block kept the default size of 64 MB.

Both NameNode and DataNodes started at the IP:192.168.0.10 and logging to the defined path, while a secondary NameNode took place, to secure redundancy methods in case of failure, on the primary NameNode.

Figure 10 illustrates the initialization of HDFS into the virtual cluster.

```
hduser@test-VirtualBox:~$ start-dfs.sh
Starting namenodes on [192.168.0.10]
192.168.0.10: starting namenode, logging to /usr/local/hadoop-2.7.3/logs/hadoop-
hduser-namenode-test-VirtualBox.out
localhost: starting datanode, logging to /usr/local/hadoop-2.7.3/logs/hadoop-hdu
ser-datanode-test-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop-2.7.3/logs/had
oop-hduser-secondarynamenode-test-VirtualBox.out
```

Figure 10. HDFS Initialization

Subsequently, the initialization of Yet Another Resource Negotiator (or YARN), one of Apache Hadoop's core components, which is responsible for distributing system resources to the miscellaneous applications running in a Hadoop Cluster and scheduling tasks to be performed among distinct cluster nodes, executed as shown in figure 11.

```
hduser@test-VirtualBox:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource
manager-test-VirtualBox.out
localhost: starting nodemanager, logging to /usr/local/hadoop-2.7.3/logs/yarn-hd
user-nodemanager-test-VirtualBox.out
```

Figure 11. YARN Initialization

Using Java Virtual Machine Process Status Tool (JPS), a command to examine the operation process of the entire Hadoop daemons as NameNode, DataNode, ResourceManager, NodeManager. As Pseudo-Distributed operation was selected, all the daemons seems to run on the same cluster, but on different JVM environments.

```
hduser@test-VirtualBox:~$ jps
15280 NodeManager
14130 NameNode
16259 HQuorumPeer
14467 SecondaryNameNode
15156 ResourceManager
16324 HMaster
14279 DataNode
16599 ThriftServer
16474 HRegionServer
16843 Jps
```

Figure 12. Java Virtual Machine Process Status Tool (JPS)

Hadoop's daemons make usage of a small quantity of ports over TCP protocol, to interact among themselves, whilst thers ports are directly listened to users, via HTTP or Java client.

Initially, by hitting the URL <http://<IP>:50070> , where the <IP> equals to IP of the Hadoop cluster's NameNode, cluster's overview is illustrated on the screen, where choosing the "DataNodes" tab results the outputs of them. In our occasion, NameNode's IP set to 192.168.0.10, during configuration process.

Figure 13 shows NameNode's overview using Web User Interface communication from a remote client.

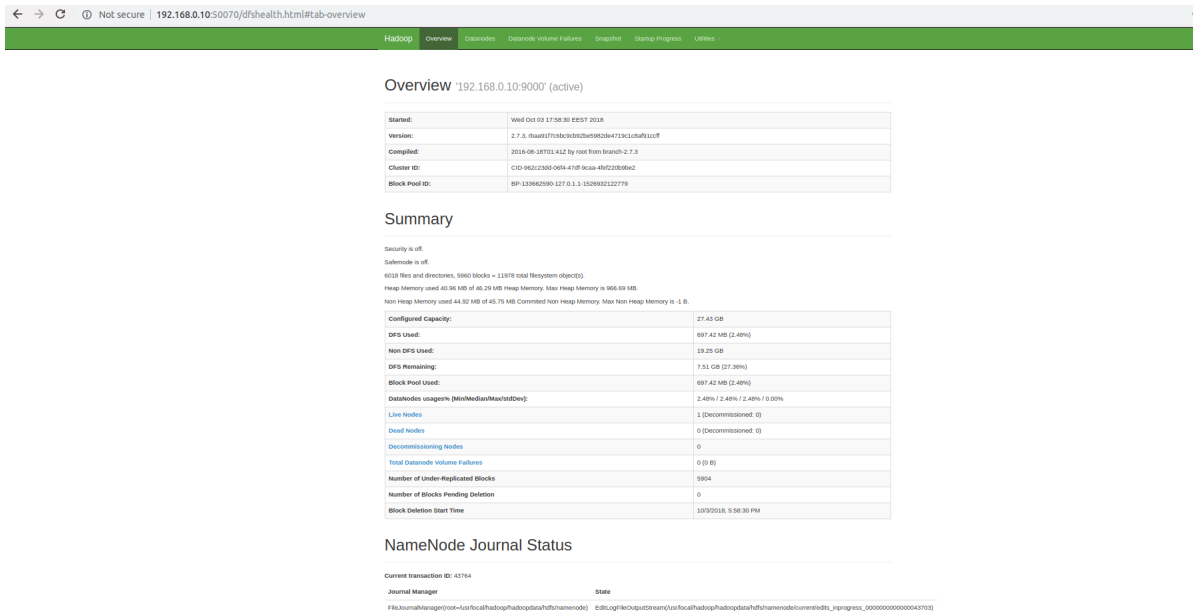


Figure 13. NameNode Overview Using Web User Interface Communication

The default Hadoop ports are as follows:

	Daemon	Default Port	Configuration Parameter
HDFS	Namenode	50070	dfs.http.address
	Datanodes	50075	dfs.datanode.http.address
	Secondarynamenode	50090	dfs.secondary.http.address
	Backup/Checkpoint node ²	50105	dfs.backup.http.address
MR	Jobtracker	50030	mapred.job.tracker.http.address
	Tasktrackers	50060	mapred.task.tracker.http.address

Figure 14. Hadoop UI Defaults TCP Ports

3.4 Working with HDFS using Python Programming Language

Interaction with HDFS is primarily performed through a command-line interface who have used POSIX interfaces on Unix or Linux, using command process called ‘hdfs’. The ‘hdfs’ command follows the below syntax:

`$ hdfs COMMAND [-option <arg>]`

,where the functionality usage of HDFS will be instructed from the “COMMAND” argument, the option argument is named from the specific option field and the closing “arg” field can receive one or more arguments that are specified for the selected option.

Additionally, there is an HTTP interface to HDFS, as well as a programmatic interface written in Java.

The main scope of this Master Thesis is the contribution to the current state of interaction between Hadoop Distributed File System (HDFS) and the average user. In order to achieve

the above approach, we created an interface using Python Programming Language, that includes the basics interactions with HDFS.

Nowadays, python includes variant deployments such Jython, scripted in Java language for Java Virtual Machine: IronPython scripted in C# , and PyPy version scripted in RPython and interpreted into C. Most of these modules are free, open-source software and most of them are implemented on community development models.

Some important advantages while programming and executing in Python involve:

- Appearance of 3rd Party Modules
- Wide Range of Supported Libraries
- Open Source Development and Huge Community
- Easy Learning
- Available Foundation Support
- Suitable Data Structures
- Speed and Productiviness

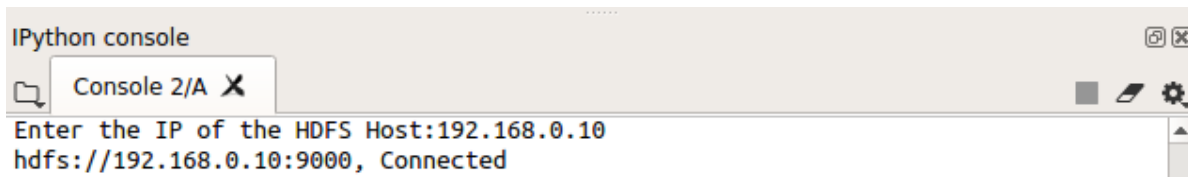
In the direction of creating a Deep Learning API, which detects possible ships positions into satellite images (more details in chapter 5) and highlight usage and availability of a Big Data Ecosystem, such as Hadoop, we created a (data) warehouse of unstructured data, which contains images of ships to train the model, json files with all the essential informations and also port images which they has to be scanned, using deep-learning API, and lead us to the desired results [14].

All of the usual file system operations are available to the user, such as creating directories, moving, removing and copying files, listing directories and modifying permissions of files on the cluster. The required script development took place using Spyder environment an open source cross-platform Integrated Development Enviroment (IDE) for scientific programming executions and implementations using Python language. Spyder integrates an amount of conspicuous packages in the scientific Python stack, inclusive of NumPy, SciPy, Matplotlib, Pandas, Ipython, SymPy and Cython, as well as other open source packages. It is released under the MIT license.

Many of the familiar commands for interaction with the file system show up, specified as arguments to the `hadoop fs` command as flag arguments in the Python style—that is, as a single dash (`-`) supplied to the command. Secondary flags or options to the command are specified with additional Python style defined functions delimited by including the initial command.

An important note is that the `hdfs` command execution occurs by adopting the permissions of the specific system user, running under a (specific) command, each time. The subsequent instances are exexuted run from a user named 'hduser' on a group named 'hadoopgroup'.

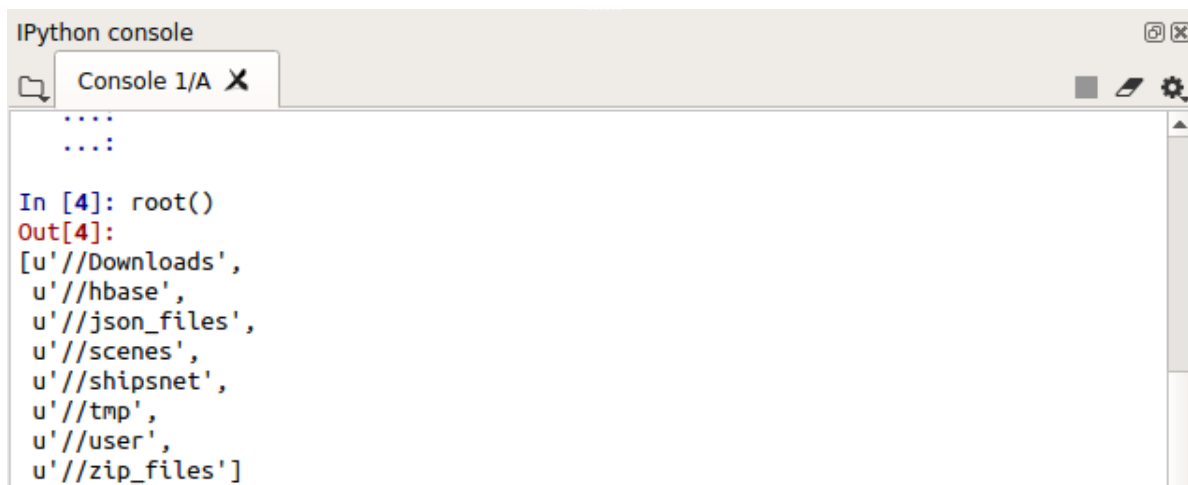
The initial step is the (remote) connection to the HDFS cluster, which in our occasion obtained IP 192.168.0.10.



```
IPython console
Console 2/A X
Enter the IP of the HDFS Host:192.168.0.10
hdfs://192.168.0.10:9000, Connected
```

Figure 15: Connection to HDFS Cluster using IP 192.168.0.10

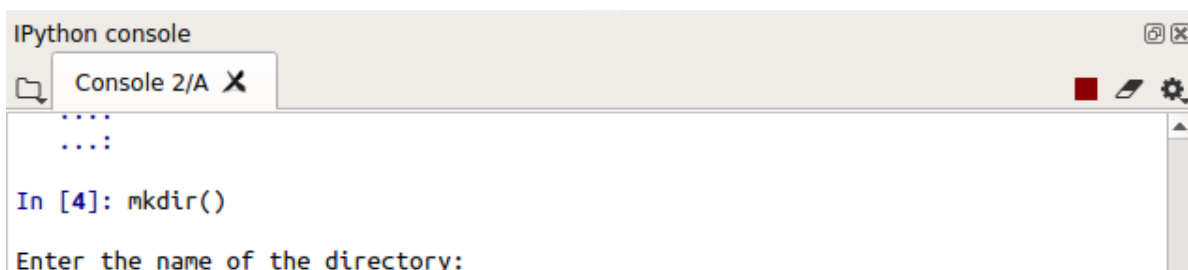
Next step is the usage of a function, called 'root()', for listing the contents and folder destinations of the user's home directory on HDFS equivalent to 'ls' argument in linux terminal. The output result of the called function is a list with all contents in a uni-code string format.



```
IPython console
Console 1/A X
...:
...:
In [4]: root()
Out[4]:
[u'//Downloads',
 u'//hbase',
 u'//json_files',
 u'//scenes',
 u'//shipsnet',
 u'//tmp',
 u'//user',
 u'//zip_files']
```

Figure 16. List of the Contents Files and Destination Folders in Root of HDFS

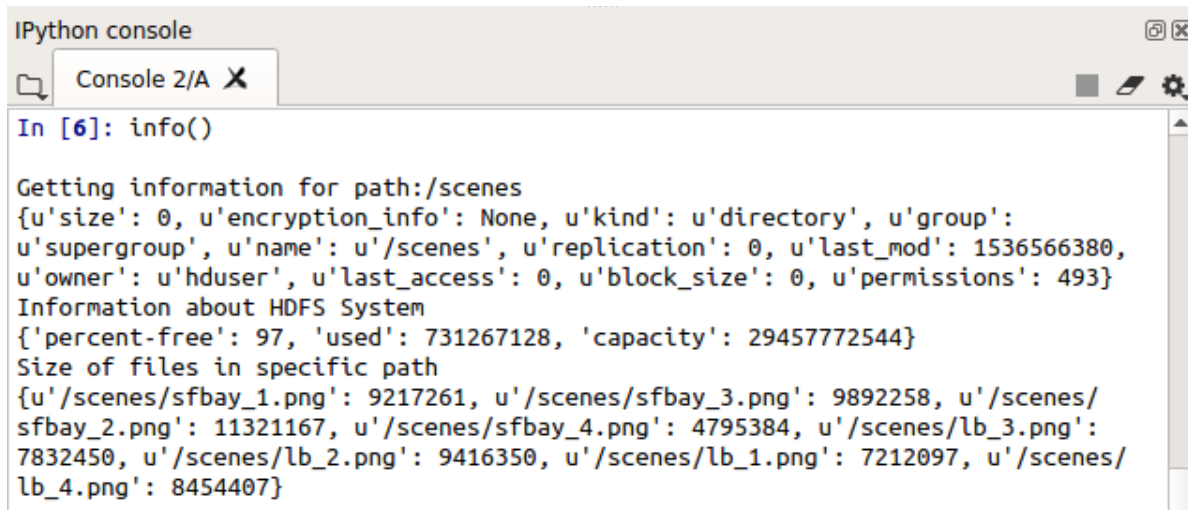
Home directories within HDFS are stored in /user/\$HOME directory, where \$HOME define during installation process. From the previous example with listing root's contents , can be recognized that any /user_type directory does not currently exist. To create any type of a /user_type directory within HDFS, we use a function called mkdir that includes 'mkdir()' command. Giving the direction name, the pythonic function place it on the desired destination (root or any other that already exists).



```
IPython console
Console 2/A X
...:
...:
In [4]: mkdir()
Enter the name of the directory:
```

Figure 17: Create New Direction in HDFS

As can be seen in figure 16, HDFS home direction (root) contains a number destination folders, which created using mkdir function. Information about files or destination folders can be received using a function called 'info()', with output results in dictionary format.

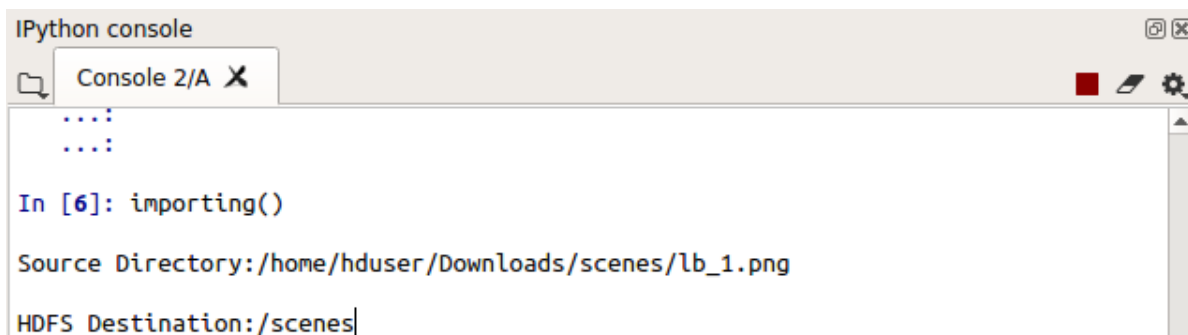


```
IPython console
Console 2/A X
In [6]: info()

Getting information for path:/scenes
{'u'size': 0, u'encryption_info': None, u'kind': u'directory', u'group':
u'supergroup', u'name': u'/scenes', u'replication': 0, u'last_mod': 1536566380,
u'owner': u'hduser', u'last_access': 0, u'block_size': 0, u'permissions': 493}
Information about HDFS System
{'percent-free': 97, 'used': 731267128, 'capacity': 29457772544}
Size of files in specific path
{'u'/scenes/sfbay_1.png': 9217261, u'/scenes/sfbay_3.png': 9892258, u'/scenes/
sfbay_2.png': 11321167, u'/scenes/sfbay_4.png': 4795384, u'/scenes/lb_3.png':
7832450, u'/scenes/lb_2.png': 9416350, u'/scenes/lb_1.png': 7212097, u'/scenes/
lb_4.png': 8454407}
```

Figure 19. Getting Information for Path called '/scenes'

Following directory creation for the current user, data is available for transmission through the user's HDFS home directory with the usage of put command. This command copies a specified from the local file system on HDFS. During interface development, a function called 'importing()' created for writing files to the distributed file system without removing the local copy. Current command asks for the (complete) local and distributed file destination.



```
IPython console
Console 2/A X
...:
...:
In [6]: importing()

Source Directory:/home/hduser/Downloads/scenes/lb_1.png
HDFS Destination:/scenes|
```

Figure 20. Import Unstructured Data from Local to HDFS

On the opposite, data can also be replicated from HDFS to the local filesystem using the get command. This command copies existing file from HDFS destination to the local filesystem. Function called 'retrieve()'.

```

IPython console
Console 2/A X
...:     dst = str(raw_input('Destination: '))
...:     hdfs.get(src, dst)

In [8]: retrieve()

HDFS Directory:/scenes

Destination:/home/hduser/Downloads/scenes/lb_1.png

```

Figure 21. Retrieve Unstructured Data from HDFS to Local

On many occasions, such as during the current Deep Learning API development, observed the need of batch processing. During this process, large datasets get as inputs all at once, resulting in large process and write outputs between two destinations.

Batch processing is the execution of a series of jobs in a program on a computer without manual intervention (non-interactive). Strictly speaking, it is a processing mode: the execution of a series of programs each on a set or "batch" of inputs, rather than a single input. Hadoop's MapReduce is the best framework for processing data in batches.

As long as, scope of the thesis is the development of a Pythonic interface between local system and HDFS, two functions that import and retrieve data in batches were created. Along these lines, the successful import of a whole dataset of 4000 satellite images of ships, from local to HDFS and vice versa, took only a few milliseconds.

Figure 22 demonstrates the function that implements batch imports, called 'batch_im_cmd()', whilst the opposite function, called 'batch_retr_cmd()'. Both of them works in the same way, as 'importing' and 'retrieve' functions, which mentioned above.

```

IPython console
Console 2/A X
...:
...:

In [10]: batch_im_cmd()

Local Directory:/home/hduser/Downloads/scenes

HDFS Destination:/scenes|

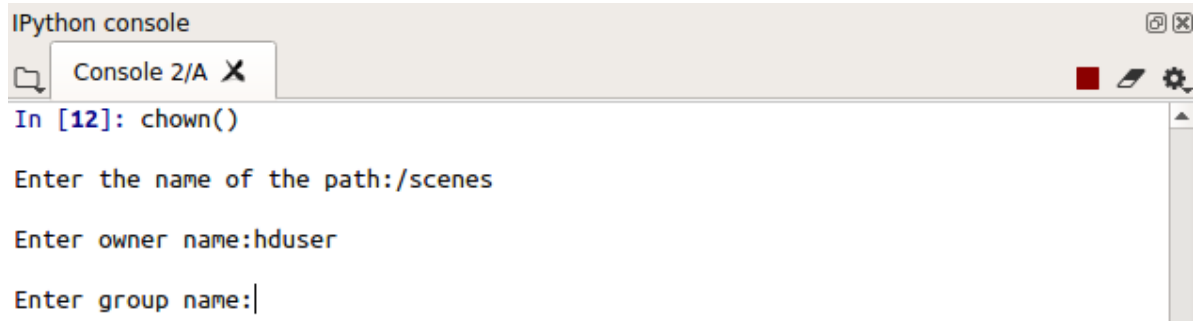
```

Figure 22. Batch Data Import from Local to HDFS

As mentioned earlier, HDFS has POSIX-like file permissions. Three types of permissions are defined: read (r), write (w), and execute (x). These permissions indicate the access levels for the owner, the group, and any other system users. For directories, the execute permission allows access to the contents of the directory, however, execute permissions are ignored on HDFS for files. Read and write permissions in the context of HDFS specify who can access the data and who can append to the file. Permissions are expressed during the directory listing command `ls`. Each mode has 10 slots. The first slot is a `d` for directories, otherwise a `-` for files. Each of the following groups of three indicates the 'rwx' permissions for the owner, group, and other users, respectively. There are several HDFS shell commands

that allow to manage the permissions of files and directories, namely the familiar `chmod` , `chgrp` ,and `chown` commands.

Function called '`chown()`' changes the owner and the group of a path or file.



```
IPython console
Console 2/A X
In [12]: chown()

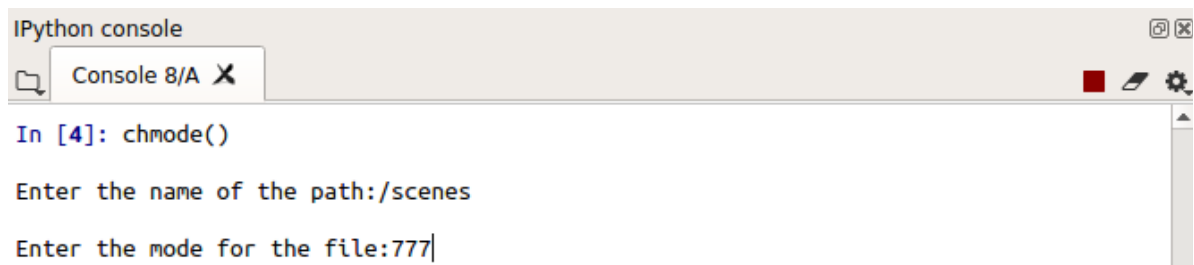
Enter the name of the path: /scenes

Enter owner name: hduser

Enter group name: |
```

Figure 23. Change of Owner and Group on a Path or a File

Function called '`chmod()`' changes the permissions of a path or file, using an octal representation of the flags to set for the permission triple.



```
IPython console
Console 8/A X
In [4]: chmod()

Enter the name of the path: /scenes

Enter the mode for the file: 777|
```

Figure 24. Change of Permissions on a Path or a File

An important issue with file permissions on HDFS consists the identity definition of the client, that is created by the username and groups of the process operating across HDFS, which means that remote clients can create arbitrary users on the system. These permissions, therefore, should only be used to prevent accidental data loss and to share file system resources between acknowledged users, not as a security mechanism.

These are the most important functions that were created during the development of the pythonic interface between an average user and HDFS environment. Besides of these, interface includes a number of additional functions that provides more interaction capabilities and operations such as:

- '`remove()`' function that deletes directories and/or contents
- '`move()`' function that move a file from a path to another
- '`read()`' function that returns the contents of a stored file on HDFS
- '`du()`' which represents disk usage for the files on a path
- '`set_repl()`' function that instructs HDFS to set the replication for the given file or path

Chapter 4

HBase Infrastructure Development

4. HBase Infrastructure Development

HBase, actually, consists Hadoop's database where data access on real-time along with scalability capabilities are implemented. HBase designation was positioned on the BigTable architecture, a database was dispatched by Google. HBase scopes in the deployment of an environment which stores and process Big Data with ease. It is an open source, distributed with a numerous versions database model that adopts NoSQL (Not Only SQL) architecture. It can be applied on the local file systems and on HDFS. Furthermore, parallel process of Big Data among Hadoop clusters can be implemented using MapReduce. An additional important feature consists the combination of storage in conjunction with parallel computing , by using a specified configuration process to manipulate [15].

4.1 Limitations of the Traditional Databases

As Bloor mentioned: «With the development of the Internet technology, especially the Web 2.0 applications, like Facebook, and Twitter, the data processing technology has to face the problem of the changes in data amount, data structures, and the processing requirements. All these changes and problems have brought great challenges to the traditional RDBMS, mainly reflected in three respects» (Bloor, 2003).

Nowadays, conventional databases cannot adjust to the various types of data structures due to the large amounts of semi-structured and unstructured data, for instance, the emails, web pages, images and videos, which cataclysm network. RDBMS are initially composed for structured data, where is not easy to achieve manipulation processes between derived data in an efficient way. Furthermore traditional databases are unable to manage the highly coexisting writing operations. It is commonly approved between new websites formats the necessity to produce dynamic web pages to display the data, like the social updates, corresponding to the customized features, when the users activities as the output result data into the database. There is a huge differential point between the traditional static pages and the modern pages. The conventional relational database is not good at the high concurrency writing operation as well as are inadequate to handle rapid alterations on network traffic and data types[16].

Pre-mentioned alterations require the ability of a powerful extensibility in the hardware and data structure construction of the database recognized as one of the major vulnerabilities of the RDBMS.

4.2 Architecture of HBase

Hbase positioned on the primary storage of HDFS, implementing the MapReduce model to for data processes, and collaboration with the ZooKeeper [17],[18].

According to Figure 25, HBase Architecture contains the following four key components:

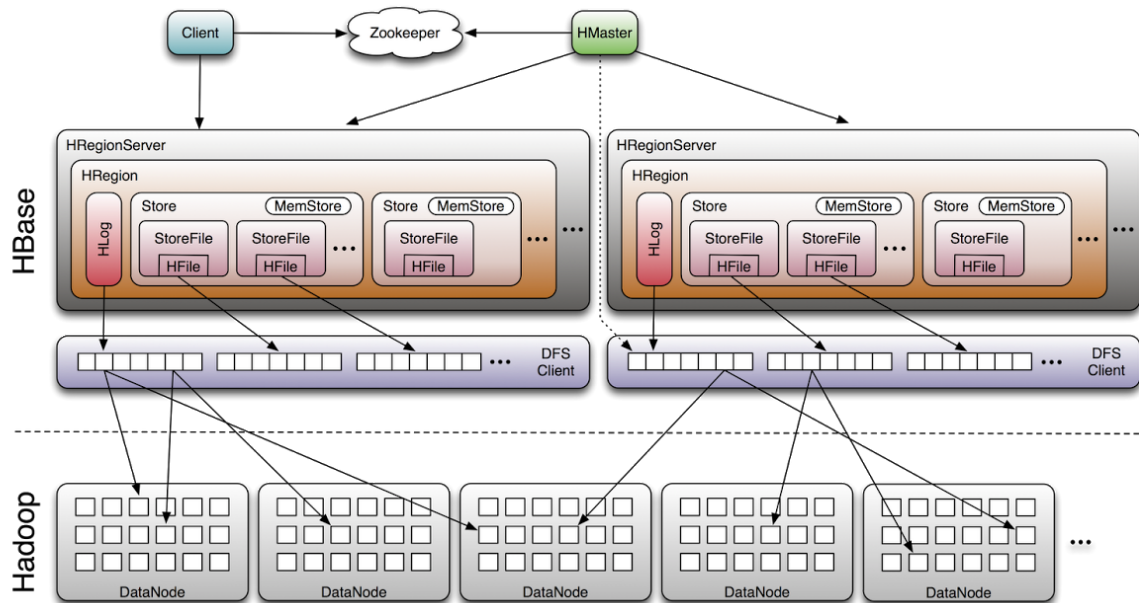


Figure 25. HBase Architecture
(Copyright 2015, Lars George, HBase: The Definitive Guide)

- HBase Client: Main user of the HBase, which handles operations side by side HMaster and read/write processes with HRegionServer.
- HMaster: Is responsible for importing, deleting, and quering the data. It adapts the HRegionServer load balance and the Region distribution to confirm that the Region will go ahead to the next available Region when any HRegionServer failure may occurs. An HBase environment can initiate a mechanism with a backup Hmaster to avoid possible failures.
- ZooKeeper: Can supply distributed colaboration, configuration functions as long as synchronization. The ZooKeeper regulates all the clusters of HBase that includes the HMaster position and HRegionServer condition information by manipulating the existing data.
- HRegionServer: Is responsible for reading and writing queries administration and executing the equivalent processes on HDFS for the users.

4.3 Comparison Between HBase and RDBMS

HBase databases are frequently in correlation with the conventional RDBMS due to the different approach on implementation structure and execution results. HBase and RDBMS can put in place of each other in some specific circumstances[15].

HBase composed as a distributed database system where the fundamental warehouse storage deposition employs the Hadoop Distributed File System and does not require severe obligations concerning the hardware platform.

The major differences between these two contrasting types of databases consists the operation structure alongside with the designation purpose [19].

- Hardware Requirements: RDBMS is row-oriented meaning that during the reading process, the users are required to pass the whole data even if they examine only few

columns revealing the necessity of greater, higher performed and more expensive hardware to encounter the desired results. Oppositely, HBase, is a new column-oriented database format which grants easier access to the data with equivalent aspects following in better data connection speed results. HBase perform higher proceeding results due to its column-oriented construction. Meantime, HBase can be implemented on a broad amount of low-cost hardware machines whilst can also maintain high performance levels.

- **Extensibility:** HBase due to the parallel processing capability of HDFS, can expand the extensibility by simply running the RegionServer. On the other hand RDBMS is capable of a limited extensibility at the time that does not hold up the architecture ability of importing node.
- **Reliability:** During possible failure on storage nodes of RDBMS data could completely lost, even if master-slave model can provide a degree of safety. Meanwhile, in HBase as a result of the distributed architecture among MasterNode and SlaveNodes and the existence of (distributed) replicas, highly reliability is provided.
- **Difficulty in Use:** Compared to RMDBS, HBase interaction is still at an early boost stage where the advanced developments processes are more rare resulting in high HBase interaction difficulty. Nevertheless, the evolution of Hadoop technology, highlights hot-spots of HBase during data processing which could subscribe to a possible popularity increment of HBase advantages.

Situated on the above comparison between these two database systems, it becomes clear the RDBMS is applicable for the plurality of small-scale data operation cases. At the other hand HBase can be acknowledged as an ideal solution when the data production approaches a really huge or giant amount of information.

4.4 Interaction with HBase by Developing a Pythonic Interface

Chapter 4 focuses on the representation of HBase environment, one of the core parts of Hadoop ecosystem environment. As Gardner's analyst Merv Adrian said «Anyone who want to keep data within HDFS environment and want to do anything other than brute-force reading of the entire file system (with MapReduce) needs to look at HBase. For random access, you need to have Hbase». HBase offers rapid reads and writes, randomly, that are impossible to be manipulated by Hadoop, as long as RDBMS. In such a manner, finds extensive application capability in commercial enterprise.

HBase is a column-oriented, distributed as well as NoSQL database sits on top of HDFS. All rows in HBase are always sorted in lexicographical way by using their row key. Placing in order in lexicographical sequence, each key related on a binary level, from left to right, byte by byte.

Data stored in HBase is grouped into tables. Conceptually, a table consists of a collection of rows and columns. Each row in HBase database has an exclusive row key and multiple column keys. The values are correlated with column keys. Client can create an arbitrary number of columns using new column attribute on the fly. As Columns in HBase defined the correlation between the column family name and column qualifier (called also column key or attribute), separated by colon: column family: column qualifier. Figure 26 illustrates the basic schema of HBase table.

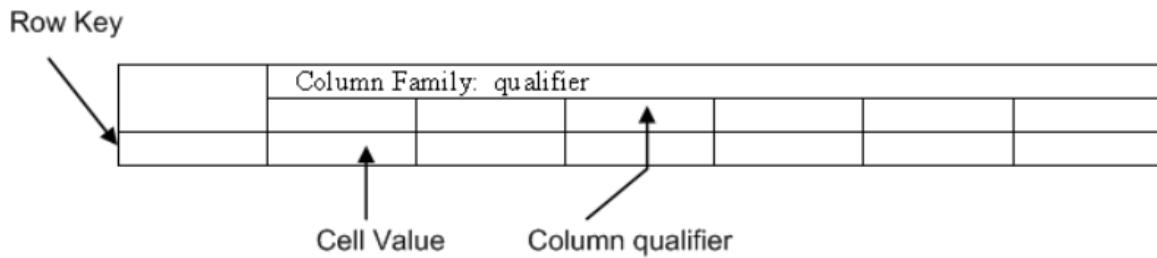


Figure 26. HBase Table Schema
 (Copyright 2015, Bikash Agrawal, Analysis of Large Time-Series in Open-TSDB)

Creating an HBase table instance is time consuming. Because each instance of HTable includes examination of the META-table to confirm if the table already exists and is enabled. So, it is always better to reuse the HTable instance and close the HTable instance after completion of the task. The META and ROOT tables are internal system tables. The ROOT table keeps list of all regions in the META-table whereas the META-table maintains a record of the whole regions in the system.

HBase provides a Java API for client interaction. In combination with the aid of Thrift server and Python language bindings, HBase can be accessed in web services, quite easily and in a user-friendly way.

The development of a user-friendly interface that embeds all the basics-read, write and delete-operations of HBase, as of any common (relational) database, and manipulates the input data was held by using Python. During a data manipulation process, we used all the required information about Chapter's 5 API, such as labels of images, scene id, longitude and latitude coordinates, as well as API's prediction results, such as possible coordinates positions of the ships, on the satellite images.

HappyBase is a suitable Python development library to communicate with Apache Hbase. It is appropriate for HBase setups, and comes along with an application to interact with HBase database. HappyBase includes the Python Thrift library to interact with HBase by making usage of HBase's Thrift gateway, that is pre-installed into newest releases of HBase [20].

HBase installation process was also implemented in Pseudo-Distributed mode, where all the master(HBase Master) and slave (Regionservers) daemons run on the same machine, using Hadoop Distributed File System (HDFS), at 192.168.0.10.

```
starting master, logging to /usr/local/hbase/logs/hbase-hduser-master-test-VirtualBox.out
localhost: starting regionserver, logging to /usr/local/hbase/bin/./logs/hbase-hduser-regionserver-test-VirtualBox.out
```

Figure 27. HBase Master and Slave Initialization

HBase still initializes a web user interface (UI) giving a lot crucial attributes and informations. By default, a UI is implemented on the master host at port 60010. After HBase configuration, we set the browser at <http://192.168.0.10:60010>, to display master's home summary page to connect and retrieve a diversity of informations related to HBase cluster, such as list of tables, running operations or informations about the group of nodes.

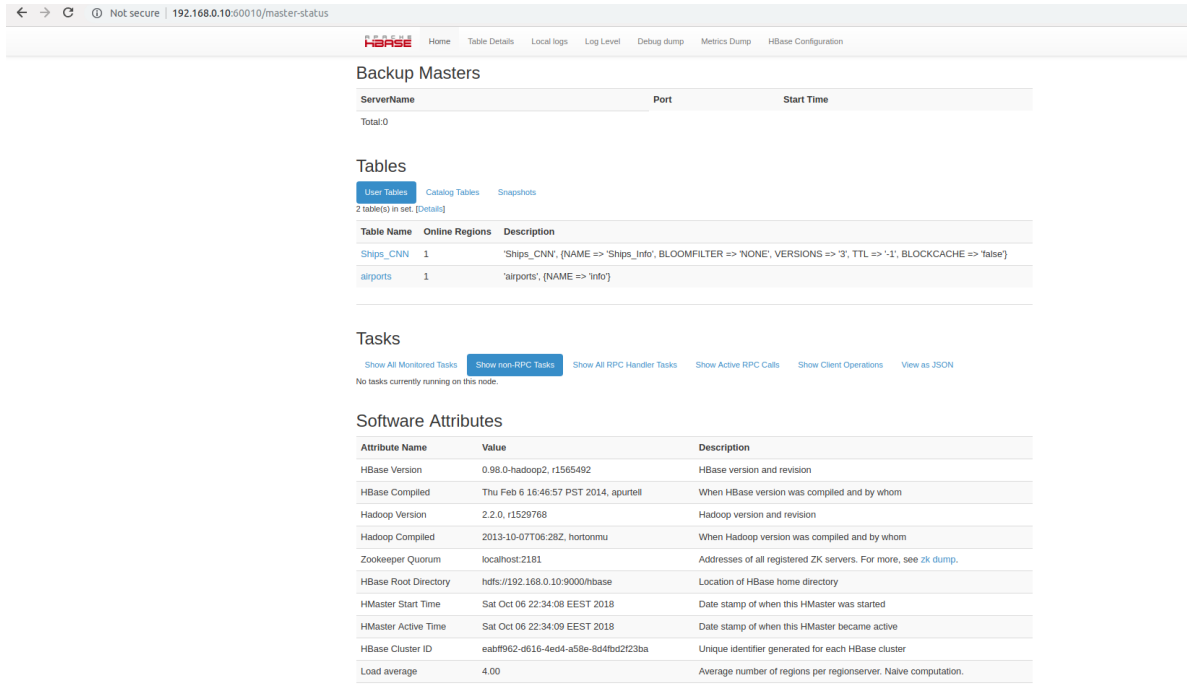


Figure 28. HBase Master Web User-Interface

HBase region servers UI is initialized at 60030, giving a lot of crucial informations, as grants permissions for currently used servers, existence of region server tables even report for the active regions. Following cluster's initialization, must be confirmed the enrolment among all the region and the master server, as well as that HBase and Hadoop are indeed running the correct version.

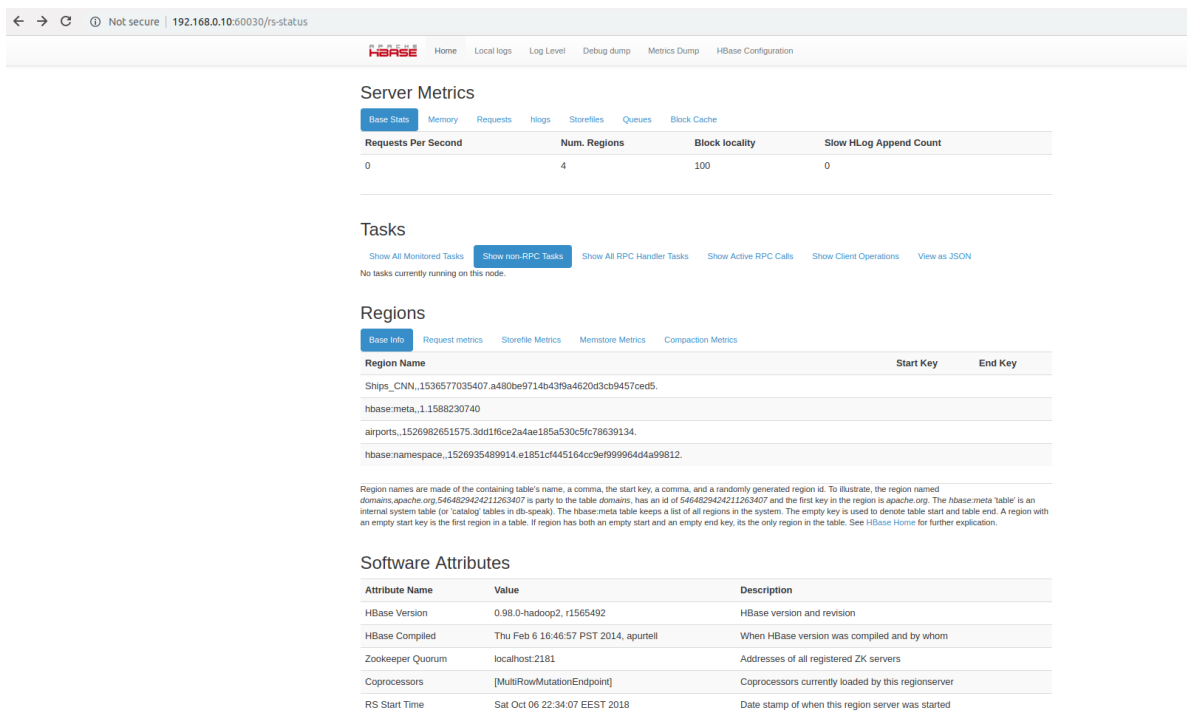


Figure 29. HBase Slave Web User-Interface

The initial step is the (remote) connection to the HBase, which in our occasion obtained IP 192.168.0.10, by creating a new Connection instance. Meanwhile, a new socket connection to the Thrift server (which is located into HBase configuration) established, supplying by this way the main entry point to interact with HBase.

```

IPython console
Console 3/A X
Enter the IP of HBase Host:192.168.0.10
  
```

Figure 30. Remote Connection to HBase Host

After the creation of a connection instance, we created a function called 'main()', to list the available tables. Figure 31 displays the results made from the usage of the above function. The HBase filesystem includes a table called 'Ships_CNN', which accommodates all the necessary informations, before and after the creation of API in Chapter 5.

```

IPython console
Console 1/A X
In [3]: main()
Out[3]: ['Ships_CNN', 'airports']
  
```

Figure 31. List of All Available Tables in HBase

Using the 'table_info()' function, a lot informations about regions and column families of a specific table can be retrieved.

```

IPython console
Console 3/A X
...:
...:
In [3]: table_info()

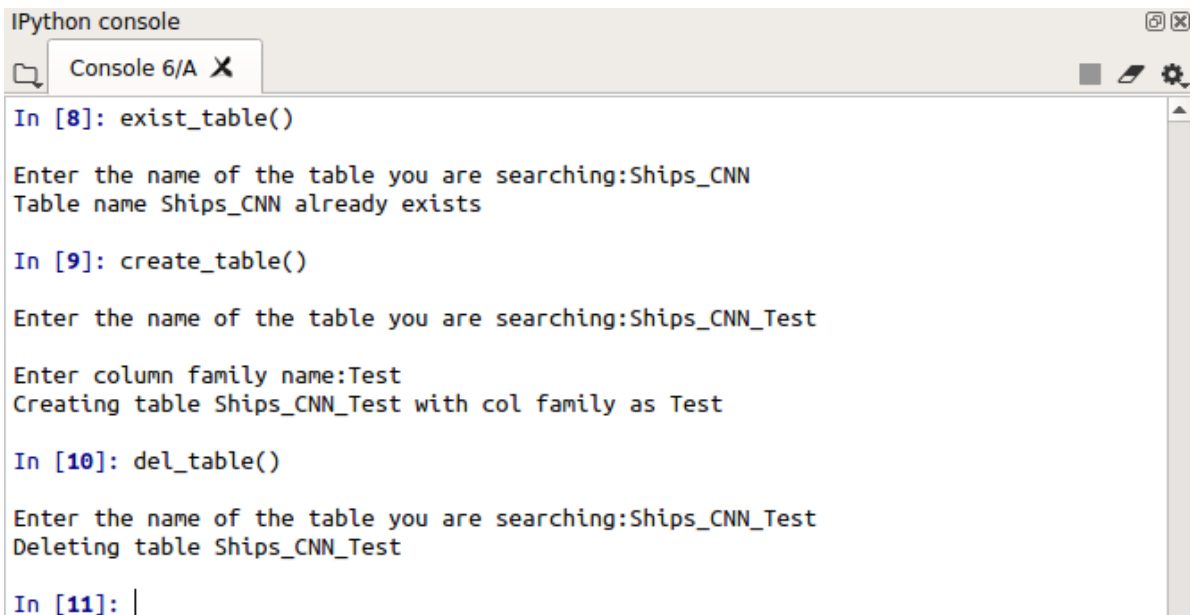
Enter the name of the table you are searching:Ships_CNN
{'Ships_Info': {'max_versions': 3, 'bloom_filter_vector_size': 0, 'name':
'Ships_Info:', 'bloom_filter_type': 'NONE', 'bloom_filter_nb_hashes': 0,
'time_to_live': -1, 'in_memory': False, 'block_cache_enabled': False,
'compression': 'NONE'}}
[{'name': 'Ships_CNN,,1536577035407.a480be9714b43f9a4620d3cb9457ced5.',
'server_name': 'test-VirtualBox', 'port': 60020, 'end_key': '', 'version': 1,
'start_key': '', 'id': 1536577035407}]
  
```

Figure 32. Regions and Column Families Infos for a Specific Table

The most usually applied methods for database system administration tasks like creating, dropping, enabling and disabling tables, were also implemented as part of the interface development.

Many times, an instance creation between the client and the thrift server may close, resulting in the demand of a new connection establishment of a table. Interaction with non-existing tables later may return errors. By using 'exist_table()' we can check if a table

already exists or not, while the 'create_table()' function creates a new one. On the other hand 'del_table()' completely deletes an existing table.

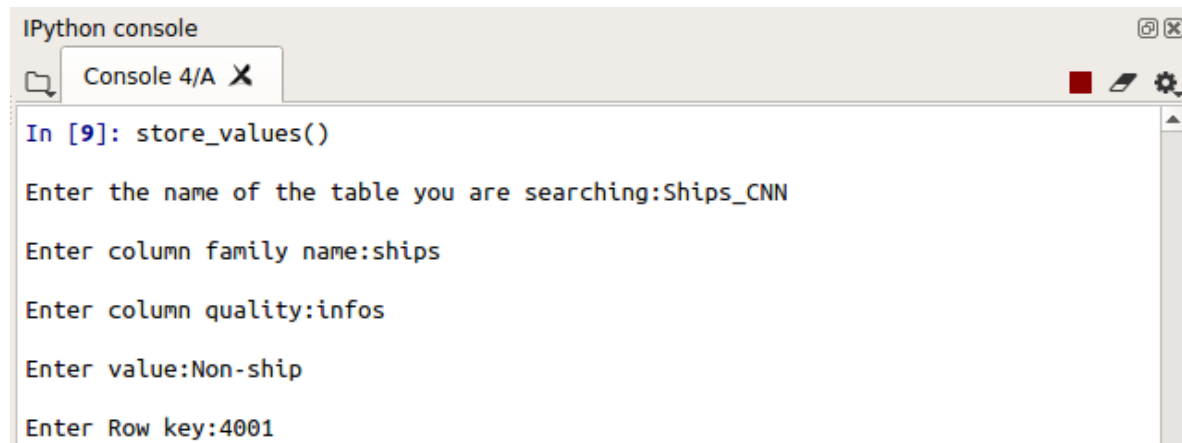


```
IPython console
Console 6/A X
In [8]: exist_table()
Enter the name of the table you are searching:Ships_CNN
Table name Ships_CNN already exists
In [9]: create_table()
Enter the name of the table you are searching:Ships_CNN_Test
Enter column family name:Test
Creating table Ships_CNN_Test with col family as Test
In [10]: del_table()
Enter the name of the table you are searching:Ships_CNN_Test
Deleting table Ships_CNN_Test
In [11]: |
```

Figure 33. Existence, Creation and Delete of Tables into HBase

The structure of a HBase table includes column families with column qualifiers, also consisting of a value and a timestamp. Meanwhile column families and qualifiers are specific approaches in the HBase data model, are most commonly implemented all at once, during interaction and manipulation processes alongside the data.

To store a single cell of data in a table, the 'store_values()' function was created. For an existing table connection, this function asks for family name and quality, row key number and value. Function 'del_row()' implements the opposite operation: given a preferred (existing) row key number, it deletes the corresponding registration.



```
IPython console
Console 4/A X
In [9]: store_values()
Enter the name of the table you are searching:Ships_CNN
Enter column family name:ships
Enter column quality:infos
Enter value:Non-ship
Enter Row key:4001
```


```
In [11]: del_row()
```

```
Enter the name of the table you are searching:Ships_CNN
```

```
Enter Row key:4001
```

Figure 34. Storing and Deleting Data in HBase Table

The function to retrieve data from a table is called 'retrieve_row()', while 'retrieve_rows' returns specific rows, selected by the user. In the example below, we used 'retrieve_row()', with row key number equals to 56. Ships_Info is the column family, Longitude, Latitude, Labels and Scenes ID column qualities in addition to the corresponding values.



```
IPython console
Console 4/A X

...:
...:

In [13]: retrieve_row()

Enter the name of the table you are searching:Ships_CNN

Enter Row Key:56
{'Ships_Info:Longitude': '37.7283639541', 'Ships_Info:Latitude': '-122.333777341', 'Ships_Info:Labels': '1',
'Ships_Info:Scene_IDs': '20171217_181637_1032'}

In [14]: retrieve_rows()

Enter the name of the table you are searching:Ships_CNN

Enter Row number:1510

Enter Row number:15

Enter Row number:1000

Enter Row number:1

Enter Row number:3999
('1510', {'Ships_Info:Longitude': '37.8077278072', 'Ships_Info:Latitude': '-122.334394063', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20170716_180815_103a'})
('15', {'Ships_Info:Longitude': '37.7580383438', 'Ships_Info:Latitude': '-122.335929375', 'Ships_Info:Labels': '1',
'Ships_Info:Scene_IDs': '20170903_181304_1041'})
('1000', {'Ships_Info:Longitude': '37.8114062888', 'Ships_Info:Latitude': '-122.334599614', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20161218_180844_0e26'})
('1', {'Ships_Info:Longitude': '37.7491755587', 'Ships_Info:Latitude': '-122.332228663', 'Ships_Info:Labels': '1',
'Ships_Info:Scene_IDs': '20170705_180816_103e'})
('3999', {'Ships_Info:Longitude': '37.6985572101', 'Ships_Info:Latitude': '-122.495313877', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20180206_184438_1043'})
```

Figure 35. Basic Functions for Retrieving Specific Values from HBase Table

Additionally, due to the purpose of getting data informations from specific rows ids (or keys) all rows in the table executing a full table scanner, implemented through the 'scan()' function. Full table scans are prohibitively expensive in practice, so, using more restricted scan processing, such as using 'retrieve_rows()' function that pre-mentioned, to make more selective range queries, are always preferred.

```
IPython console
Console 4/A X
...:
In [16]: scan()

Enter the name of the table you are searching:Ships_CNN
('0', {'Ships_Info:Longitude': '33.7380372592', 'Ships_Info:Latitude': '-118.225469433', 'Ships_Info:Labels': '1',
'Ships_Info:Scene_IDs': '20180708_180909_0f47'})
('1', {'Ships_Info:Longitude': '37.7491755587', 'Ships_Info:Latitude': '-122.332228663', 'Ships_Info:Labels': '1',
'Ships_Info:Scene_IDs': '20170705_180816_103e'})
('10', {'Ships_Info:Longitude': '37.7286473953', 'Ships_Info:Latitude': '-122.336089791', 'Ships_Info:Labels': '1',
'Ships_Info:Scene_IDs': '20170106_180851_0e30'})
('100', {'Ships_Info:Longitude': '37.7356394938', 'Ships_Info:Latitude': '-122.338921365', 'Ships_Info:Labels': '1',
'Ships_Info:Scene_IDs': '20170910_181216_1010'})
('1000', {'Ships_Info:Longitude': '37.8114062888', 'Ships_Info:Latitude': '-122.334599614', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20161218_180844_0e26'})
('1001', {'Ships_Info:Longitude': '37.7473208549', 'Ships_Info:Latitude': '-122.134401353', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20170505_181257_0e2f'})
('1002', {'Ships_Info:Longitude': '37.7080306968', 'Ships_Info:Latitude': '-122.137785501', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20170505_181258_0e2f'})
('1003', {'Ships_Info:Longitude': '37.6492024666', 'Ships_Info:Latitude': '-122.095719038', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20170905_181215_0f12'})
('1004', {'Ships_Info:Longitude': '37.8206724662', 'Ships_Info:Latitude': '-122.387929566', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20170917_190616_0f3c'})
('1005', {'Ships_Info:Longitude': '37.7292356283', 'Ships_Info:Latitude': '-122.395151369', 'Ships_Info:Labels': '0',
'Ships_Info:Scene_IDs': '20170910_181216_1010'})
```

Figure 36. Full Table Scan Iteration

Methods such as ‘store_values()’ or ‘del_row()’ are not adequate when the import or delete operations includes a numerous size of dataset. The ‘batch_hbase()’ function does creates a group of instances involving put (and/or delete) method, where the total alterations number are directed back to the server in a distinct round-trip. Executing this function, a number of 4000 information row key numbers were stored into Ships_CNN table in one go. Figure 37 illustrates function’s usage, where column-families and qualities were asked, while values retrieved from a specific source (numpy table, json file).

```
IPython console
Console 4/A X
In [18]: batch_hbase()

Enter the name of the table you are searching:Ships_CNN

Enter column family name:Ships_Info

Enter column quality:Labels|
```

Figure 37. Batch Storage in HBase Table

Chapter 5

Ship Detection In Satellite Imagery Using Deep Learning

5. Ship Detection In Planet Satellite Imagery Using Deep Learning

Chapter 5 highlights the Hadoop Distributed File System (HDFS) functionality in conjunction with the development of a Deep Learning Application that classifies and detects ships using planet satellite imagery, which captured at San Francisco Bay.

The Hadoop Distributed File System (HDFS) is used as a storage warehouse, by creating specific directories, where the dataset includes ships exported from planet satellite imagery, essentially for using and testing the API's functionality, as long as the extracted detection results, such as ships detection on satellite images, predictions results, are stored. The Pythonic interfaces that were presented in chapters 3 and 4 are used for the specific purpose of interaction.

5.1 Application Description

Satellite imagery provides unique insights into various markets, including agriculture, defence and intelligence, energy, and finance. New commercial imagery providers, such as Planet, are using constellations of small satellites to capture images of the whole Earth every day.

This overwhelming growth of imaginary datasets increases the efficiency on visual examinations, where the necessity for machine learning and computer vision algorithms become more important to make automation and analysis processes more suitable. Dataset created in the point of detecting the location of large-sized ships in satellite images.

5.2 Ship Detection Dataset Parameters

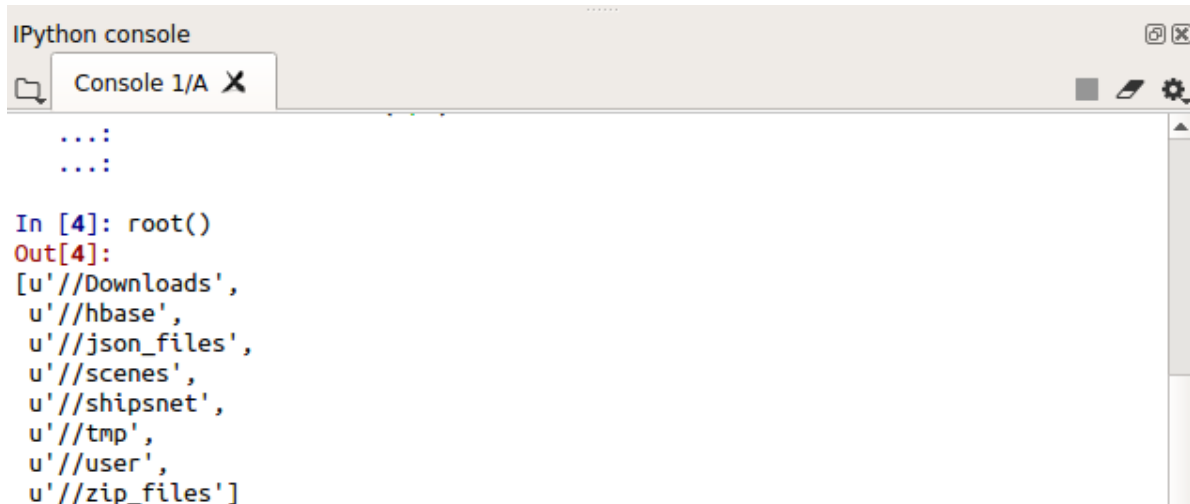
The dataset is comprised of images of ships exported from Planet satellite imagery collected over the San Francisco and San Pedro Bays areas of California. There, 4000 images, 80x80, 3-channel format (RGB images), labelled with either a "ship" or "no-ship" classification, are contained. Ships images were collected from the Planet.com along with full-frame visual products.

By providing a zip-format directory called shipsnet.zip that contains the entire dataset as .png ship image. Each individual image filename follows a specific format: 'label-scene id-longitude-latitude.png' format, where:

- **label:** Representation values of one (1) or zero (0) corresponds to the "ship" or "no-ship" class.
- **Scene id:** Combined with the Planet application helps to find the entire scene, as the one and only identifier of the Planet Scope.
- **Longitude & latitude:** Coordinates of the image. Their values are split by the usage of an underscore.

Additionally, the dataset is formatted as a JSON text file called shipsnet.json where data, label, scene_id, and location lists are included.

The whole dataset was stored and manipulated using the Hadoop Distributed File System (HDFS) pythonic interface, which was introduced in chapter 3. Using the pythonic interface application three new storage directories were created-called-‘shipsnet’, ‘scenes’ and ‘json_files’, where images of ships or non ships images, San Francisco and San Pedro Bay images and JSON formatted file were stored,respectively. Furthermore, the ‘zip_files’ directory, includes the entire dataset. Using the ‘retrieve()’ function, which was also mentioned in chapter 3, we download the data from HDFS, speeding up the process, since there in no longer necessity for data loading in local computer, in order to train and validate our model (Reference on Appendix A).



```
IPython console
Console 1/A X
...:
...:
In [4]: root()
Out[4]:
[u'//Downloads',
 u'//hbase',
 u'//json_files',
 u'//scenes',
 u'//shipsnet',
 u'//tmp',
 u'//user',
 u'//zip_files']
```

Figure 38. Storage Direction of Dataset on Hadoop Distributed File System

Every pixel among 4000 images is listed as 19200 integers inside the data. The first 6400 entries includes the red channel, following 6400 the green, and last 6400 the blue channel. Image storage set up in row-oriented sequence where the first 80 entries of the array consists the red channel values of the first row of the image.

The ‘ship’ class includes 1000 images that are center-oriented on the body of a single ship with different sizes, and atmospheric conditions. Example images from this class are illustrated in figure 39.

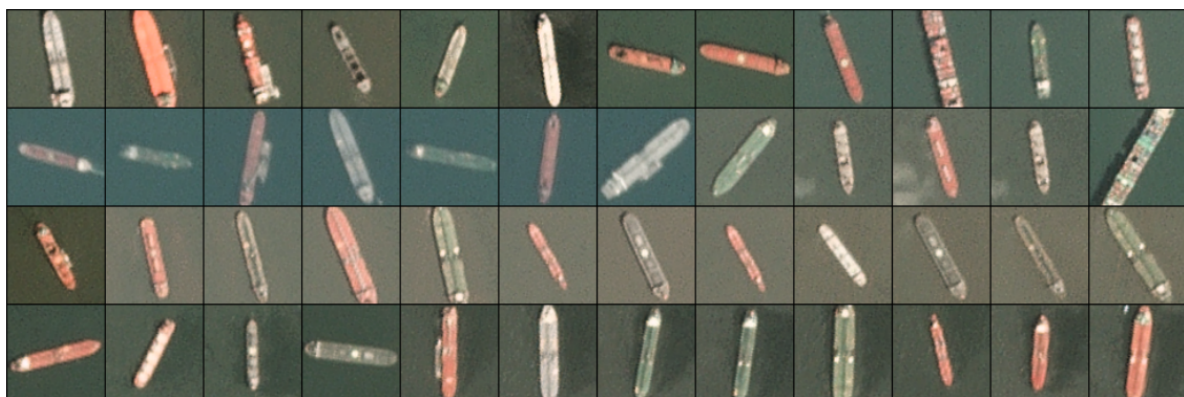


Figure 39. ‘Ship’ Class Sample Images

The "non-ship" class includes 3000 images: first 1000 entries consists of a randomly collected samples of different land displays (such as water, buildings) that do not contains any piece or part of a ship. Following 1000 includes just a piece of ship, when, the last 1000 images have previously been mislabelled by machine learning models, typically caused by bright pixels or strong linear features. Example images from this class are shown below.



Figure 40. 'Non-Ship' Class Sample Images

5.3 Convolutional Neural Network Architecture

The Keras library is a high-level module for building neural networks. Specifically, Keras is a powerful as well as simple Python library for deep learning construction. During ship detection network configuration, a mass number of libraries were used: numpy, a library responsible for elements storage into arrays for specific processes, matplotlib for graphics display, PIL for images manipulation and some also Kera's included, appropriate for model compilation, such as Sequential for model initialization in a sequence way between each layer, Conv2D, MaxPooling2D, Flatten, Dropout and Dense for the linear stack creation, which is referred below.

```

14 import numpy as np
15 from matplotlib import pyplot as plt
16 from keras.models import Sequential
17 from keras.layers.core import Flatten, Dense, Dropout, Lambda
18 from keras.layers.convolutional import Conv2D, MaxPooling2D
19 from keras.optimizers import Adam, SGD
20 from PIL import Image, ImageDraw

```

Figure 41. Appropriate Libraries for Ship Detection Network Configuration

The Keras Sequential Model usually consists of a linear stack of layers. The most commonly used structure of Convolutional Neural Networks (CNN) is composed of three different types of layers.: Convolutional, Pooling or fully connected. Each layer type has different rules with respect to forward and error backward signal propagation [23]. CNNs typically uses multilayer perceptron structures: an input layer, some hidden layers as well as an output layer.

Figure 42 illustrates the above structure: Feature extraction part, is used for combinations of convolutional and pooling layers, when classification part is using fully connected layers, to output the results.

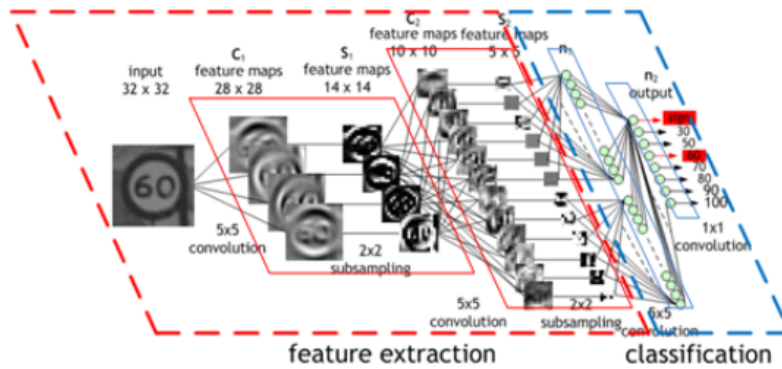


Figure 42. Typical Structure of Convolutional Neural Network

(Copyright, Piyush Rai, Deep Learning: Models for Sequence Data)

For the present ‘Ship Detection Application’, a usual Convolutional Neural Network that (CNN) was used, involving four major steps:

- Convolution step
- Pooling step
- Flattening step
- Full Connection step

Convolutional Neural Network needs to bear in mind the exact input shape that have to expect, which in our occasion equals to 80X80X3, as our images are of size 80X80 pixels and uses 3 channels (RGB format). A ‘relu()’ function was selected as the activation function for the first four (4) layer, which is suitable to repair problems that are appeared with dying rectified linear units, by helping network learn into new decision edges.

Convolution implement on the training images takes place by the convolutional layers. As pre-mentioned above, CNN’s Model hidden layers are composed from 4 convolutional layers, approved to learn more complex representations, prevented from data under-fitting. Convolutional layers uses 32 filters, where every filter is in the shape of 3X3 due to the intake defined as a 80X80 pixel coloured image in a RGB format and the layer used rectifier function for handling and manipulating processes. Convolution layers are two-dimensional (2d), as the images are two-dimensional pixel data arrays [24],[25].

Following to each convolutional layer, there are four max-pooling layers, which performs the pooling operation using a max-pooling function, because of the necessity to distinguish the maximum pixel for each district of interest.

The pooling layer performs a pooling process, where following convolutional operation, outputs multiple feature maps per image and pooling operation runs on this output, collecting these by the usage of a 2X2 matrix to minimize the pixel loss while getting a precise region around feature locations.

The output from the last pooling layer was flattened from the two-dimensional (2d) array into a one-dimensional (1d) array, which-after that-was fed into the feed-forward neural network accepting 4096 values array.

The output from pooling layer was finally flattened to get a one-dimensional (1d) single vector, which was then fed to the hidden layer just like in simple feed-forward network introduced before, needed for the two-class classification. Output layer consists of a dense layer configured with a sigmoid function for binary classification process.

Furthermore, a dropout layer was added to overcome possible over-fitting problems. Dropout randomly turns off a fraction of neurons during the training process, causing reductions of the reliance on the training set by some amount. The specific number of neurons which are important to be deactivated is determined by a hyper-parameter equalled to 0.25 that was set during configuration process. Through this process the model keeps in memory the fitting (working) data without taking account of unused neurons.

Figure 43 illustrates the compilation process for the current ship detection model, using ‘add()’ function to import the beyond mentioned hidden layers.

```
#Convolutional Neural Network Design Architecture
model=Sequential()

# 1st Hidden Layer
model.add(Conv2D(kernel_size=(5,5),activation="relu",padding="same",filters=100,input_shape=(80,80,3)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 2nd Hidden Layer
model.add(Conv2D(kernel_size=(5,5),activation="relu",padding="same",filters=100))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 3rd Hidden Layer
model.add(Conv2D(kernel_size=(5,5),activation="relu",padding="same",filters=100))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 4rth Hidden Layer
model.add(Conv2D(kernel_size=(5,5),activation="relu",padding="same",filters=100))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Output Layer
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(1,activation="sigmoid"))
```

Figure 43. Model's Hidden Layer Configuration Output

Subsequently, model’s compilation takes place, using a function called ‘compile()’. Some important configuration informations during model compilation (training) consists the learning process parameters configuration: The batch size, which is responsible for the division of the model in a fixed (batch) size, anyone of them receiving a steady number of images to train. For the avoidance of memory errors, usage of batch sizes in the power of two (2) in addition to specified sizes affordable by computer’s RAM can lead to better results:during our model compilation process, batch size number which was selected was 128 (7th power of 2). As Optimizer, for maximization activation of the model , Stochastic Gradient Descent (SGD) was used in order to train the CNN, due to its indication for this kind of networks. Validation split was used was 0.2, which means that 20% of the total dataset, was randomly picked as validation data, for validation processes usage. As loss function, ‘categorical crossentropy’ was selected, in the way that one-hot encoded used, during images labels conversion. Finally, training epochs (or iterations) indicates the total number that the network is trained. Network was trained for 12 epochs, meaning the specific

number of times iterations go through the training set. Usage of ‘fit()’ function outputs (the possibility to use) history objects, while model suits to the data. Calling ‘summary()’ function offers the ability to visualize important informations about pre-constructed model.

```
#Model Compilation
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#Model summary
model.summary()

#Train model with 20 % of data used for validation
history = model.fit(x, y, batch_size=128, epochs=12, validation_split=0.2)
```

Figure 44. Ship Detection Learning Process Configuration

5.5 Prediction Results

After the implementation of learning/training process, using ‘fit()’ function, the model on ships satellite images, by using pre-mentioned parameters, as long as observing the training accuracy and loss functions results, concluded that the model output, for the previous mentioned, number of epochs the training accuracy is 98,52% and the training loss is enough low, almost 4,38%, while validation accuracy is 99,14% and validation loss 3,26%.

```
....
Train on 17280 samples, validate on 4320 samples
Epoch 1/12
17280/17280 [=====] - 173s 10ms/step - loss: 0.1759 - acc: 0.9315 - val_loss: 0.1420 -
val_acc: 0.9519
Epoch 2/12
17280/17280 [=====] - 175s 10ms/step - loss: 0.1005 - acc: 0.9628 - val_loss: 0.1464 -
val_acc: 0.9491
Epoch 3/12
17280/17280 [=====] - 203s 12ms/step - loss: 0.0892 - acc: 0.9668 - val_loss: 0.0932 -
val_acc: 0.9694
Epoch 4/12
17280/17280 [=====] - 196s 11ms/step - loss: 0.0794 - acc: 0.9709 - val_loss: 0.0964 -
val_acc: 0.9688
Epoch 5/12
17280/17280 [=====] - 192s 11ms/step - loss: 0.0728 - acc: 0.9737 - val_loss: 0.0933 -
val_acc: 0.9699
Epoch 6/12
17280/17280 [=====] - 184s 11ms/step - loss: 0.0660 - acc: 0.9771 - val_loss: 0.0993 -
val_acc: 0.9683
Epoch 7/12
17280/17280 [=====] - 199s 12ms/step - loss: 0.0606 - acc: 0.9795 - val_loss: 0.0474 -
val_acc: 0.9850
Epoch 8/12
17280/17280 [=====] - 195s 11ms/step - loss: 0.0584 - acc: 0.9794 - val_loss: 0.0645 -
val_acc: 0.9782
Epoch 9/12
17280/17280 [=====] - 195s 11ms/step - loss: 0.0553 - acc: 0.9800 - val_loss: 0.0560 -
val_acc: 0.9819
Epoch 10/12
17280/17280 [=====] - 187s 11ms/step - loss: 0.0511 - acc: 0.9826 - val_loss: 0.0621 -
val_acc: 0.9782
Epoch 11/12
17280/17280 [=====] - 213s 12ms/step - loss: 0.0476 - acc: 0.9836 - val_loss: 0.0542 -
val_acc: 0.9806
Epoch 12/12
17280/17280 [=====] - 207s 12ms/step - loss: 0.0438 - acc: 0.9852 - val_loss: 0.0326 -
val_acc: 0.9914
```

Figure 45. Model Compilation & Prediction Results of Ships Detection Application

Evaluating the performance of the model on the test set, it is obvious that both validation loss and accuracy are in harmonic synchronization with the corresponding training

characteristics leading to the conclusion that the model was not over-fitted. It is approved that model's efficiency improved by the time that the loss test process was almost equal compared to training loss, as the first (validation) is reduced throughout the epochs, while the gap between training and validation accuracy was eliminated.

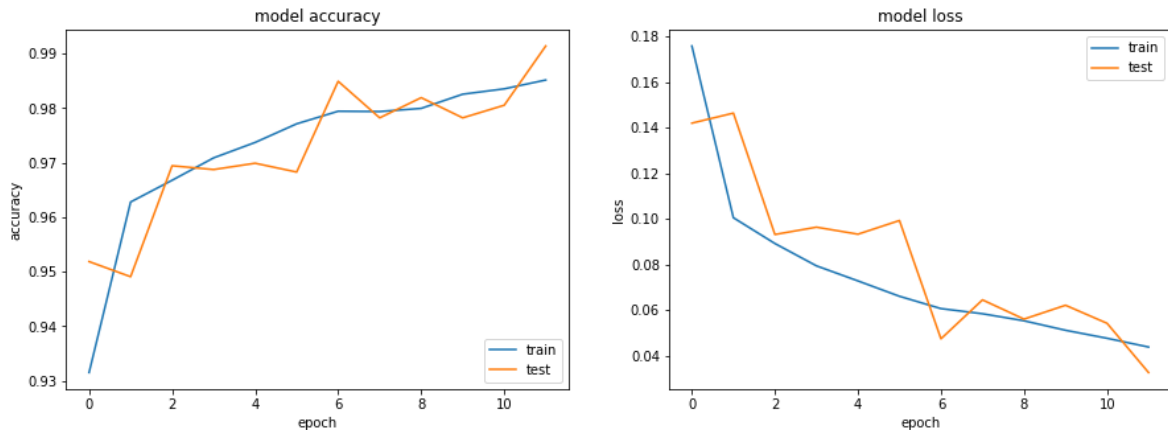


Figure 46. Accuracy and Loss Plots Between Training and Validation Data

Accuracy and Loss Plots (Figure 46) along with Predictions results are stored in 'Predictions Weights' directory into HDFS, in order to be available at any moment a new prediction process needs to be made, without necessity of model retrain. Saving Keras models in HDF5 (.h5) format-optimal for saving multidimensional arrays of numbers-files economize days, even weeks of model (re)training.

```

IPython console
Console 1/A X
In [11]:
In [11]: importing()
Source Directory:/ship_detection_weigths.h5
HDFS Destination:/Prediction Weights/ship_detection_weigths.h5
In [12]: hdfs.ls('/Prediction Weights')
Out[12]: [u'/Prediction Weights/ship_detection_weigths.h5']

```

Figure 47. Save Keras Models into HDFS for Future Load

5.6 Visualization of Prediction Results

After the completion of training process, evaluating neural network consists the last step. In this case, we tried to get a glimpse of well your model performs by picking 10 random images from validation data and receiving, as label, the predicted result.

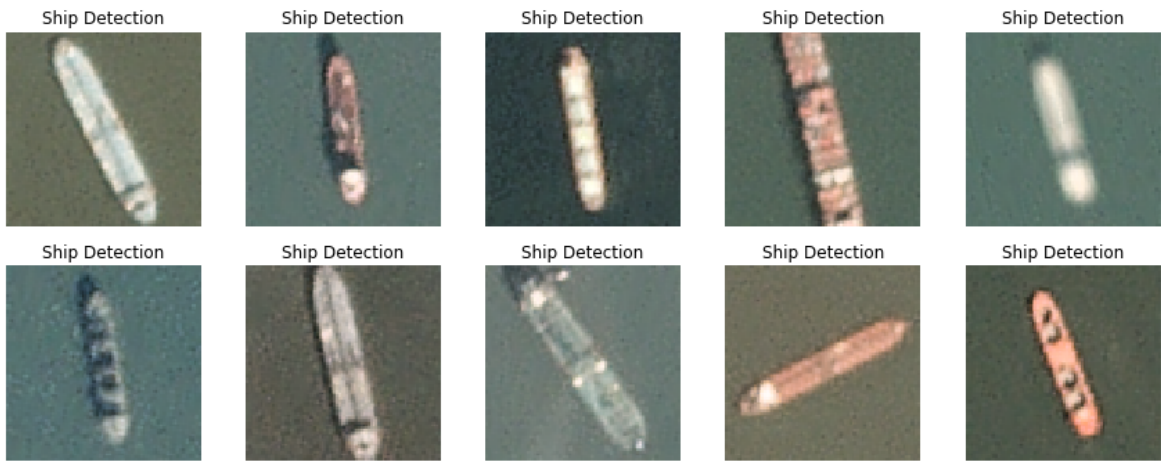


Figure 48. Visualizations of 'Ships' Labels Predictions

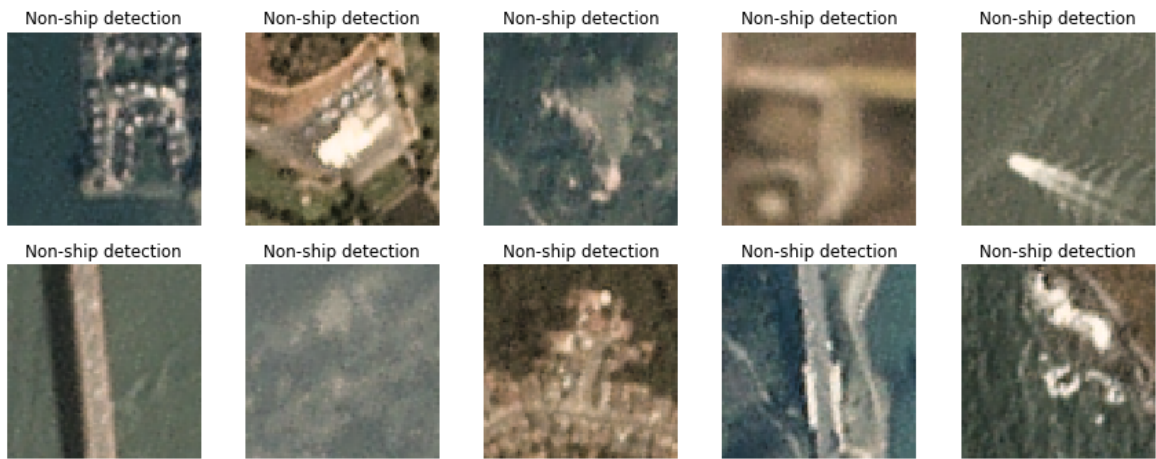


Figure 49. Visualizations of 'Non-Ships' Labels Predictions

Looking at a random sample of twenty (20) images into the validation data, we assured that predicted labels of the earlier trained model are quite close to the real labels, confirming by this way, the accuracy of the current model.

Except of making predictions on ships or non-ships labels, on images, current algorithm also searches on real captured bays satellite images for ship detection into these. HDFS storage directory called 'scenes' includes 8 Planet satellite images, from bays areas, captured from San Francisco and San Pedro. Figure 50 illustrates the prediction results during satellite image scanning, using previous model prediction results. White, rectangle patches used for highlighting possible hotspots of ships detection.



Figure 50. Searching on Satellite Images of Bays for Ships Existence

Contributing with Hadoop Distributed File System, by using pythonic interface application, a newly storage direction, called 'Prediction Output Results', was created.

```

IPython console
Console 5/A X
...
In [4]: root()
Out[4]:
[u'//Downloads',
 u'//Prediction Output Results',
 u'//hbase',
 u'//json_files',
 u'//scenes',
 u'//shipsnet',
 u'//tmp',
 u'//user',
 u'//zip_files']

In [5]: mkdir()

Enter the name of the directory:/Prediction Output Results|
  
```

Figure 51. Creation of Output Storage Direction into HDFS

Output results brought out from the previous model were stored, from local to HDFS, using 'importing function()'.

```
IPython console
Console 5/A X
In [8]:
In [8]: importing()
Source Directory:/Downloads/Ships Detection Results on sfbay3.png
HDFS Destination:/Prediction Output Results/Ships Detection Results on sfbay.png
In [9]: importing()
Source Directory:/Downloads/Ships Detection Results.png
HDFS Destination:/Prediction Output Results/Ships Detection Results.png
In [10]: importing()
Source Directory:/Downloads/Ships Detection Results 2.png
HDFS Destination:/Prediction Output Results/Ships Detection Results 2.png
```

Figure 52. Inporting of Output Data into HDFS

```
IPython console
Console 5/A X
In [13]:
In [13]: dir_cont()
Choose Directory:/Prediction Output Results
Out[13]:
[u'/Prediction Output Results/Ships Detection Results 2.png',
 u'/Prediction Output Results/Ships Detection Results on sfbay.png',
 u'/Prediction Output Results/Ships Detection Results on sfbay3.png',
 u'/Prediction Output Results/Ships Detection Results.png']
```

Figure 53. Searching for Directory Contents in HDFS

The above figures depicts the HDFS pythonic interface application in combination with ‘Ships Detection’ application deployment. For the sake of the example, HDFS storage application inputs necessary datasets for the development of the model, while output prediction results were also saved in HDFS, to an additional specified directory, showing the interaction process between a big data platform and deep-learning technology.

Figure 54 gives an explanatory illustration schema where Hadoop Distributed File System (HDFS) and HBase Database Ecosystem interact both each other, as long as with Ship Detection (Deep Learning) Application, which was developed and implemented.

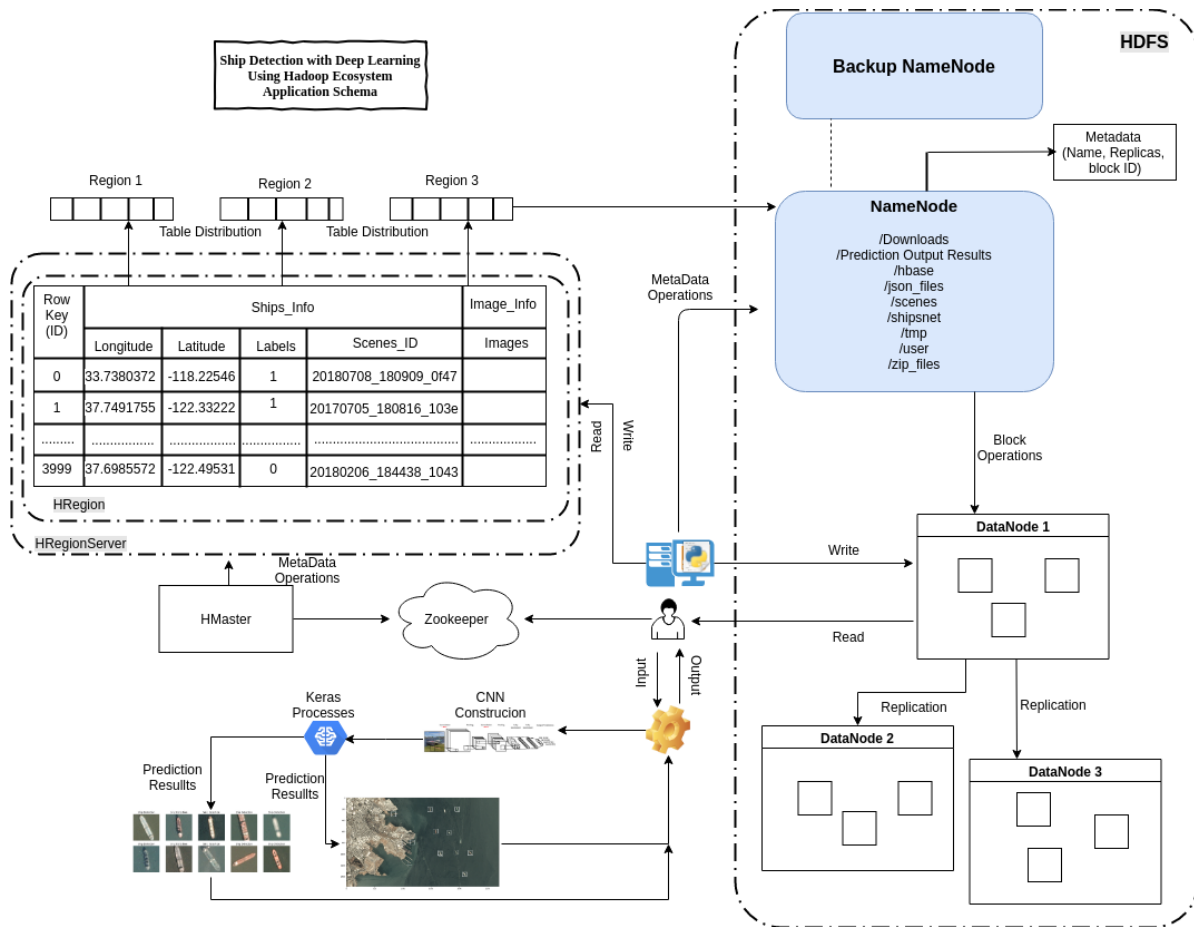


Figure 54. Ship Detection Model using Hadoop Ecosystem Interaction Schema

Chapter 6

Conclusions

6. Conclusions

Big data is becoming extremely widespread among IT Research and Development projects. Big data technology is already present also in a wide range of enterprises, infrastructures and organizations for solving practical every-day problems. At the same time, the rate of big data generation and development increases in a tremendous manner.

The current master thesis, after an overview of the big data landscape, introduced the core technology of the Hadoop Ecosystem where a massive number of applications and development processes already exist. Hadoop is the most commonly used and acknowledged framework for big data processings in an efficient and expandable way. It's a reliable, exceptionally scalable, error tolerant as well as profitable solution that supports cluster distribution computing and the manipulation of petabytes of data on a massive amount of nodes. Hadoop is composed of two main components, HDFS and MapReduce, and it is a recommended method for storing and processing of semi-structured or unstructured data. Industry leading organizations such as Google, Yahoo, and Facebook approved and selected the Hadoop's environment framework.

HBase is a column-oriented database constructed on the Hadoop Platform, capable of offering several advantages in comparison with conventional row-oriented databases. RDBMS fails in scaling data efficiently as well in a cost effective way, despite their partitioning and parallel processing capabilities. At the same time, RDBMS comes along with structured data and is proven inadequate for handling unstructured one, which is commonly used by high-tech applications as smart-phones, and social networking websites.

The Python programming language comes along with a high number of implementation examples and a large amount of standard libraries covering a wide range of technology fields. Using Python, the interaction with the Hadoop Distributed File System and the HBase environment can be more efficient and comfortable for the decent user, by creating appropriate programming interfaces, in this specific easy to learn and powerful programming language, which is used in many scientific projects, as well as in machine learning, hacking and web developing fields.

The main scope of this Thesis is the contribution to the current state of interaction between the Hadoop Distributed File System (HDFS) and the average user, by creating an interface for the basic interaction with HDFS, using the Python Programming Language. Towards this end, a user-friendly interface that embeds all the basic operations of HBase was developed and showcased for handling the data required by a Deep Learning application. More specifically, the application is about detecting possible ship positions within satellite images by using a deep neural network of layers. Training the network requires a pretty large amount of imagery data, labelled as 'ship' or 'non-ship', while its full deployment for real use purposes meets the Big Data characteristics. In our thesis, we demonstrated how to store, retrieve and use this data in the Hadoop-HBase ecosystem. Using our experimentation results, we conclude that Hadoop and HBase offer adequate stability, expandability and latency in a huge amount of data manipulation. The matter of contention relevant to stability of Hadoop and HBase is being investigated by many developers in more recent releases.

Future work includes construction and benchmarking of machine learning algorithms on infrastructures that tend to or already manipulate artificial intelligence applications fed by big data.

APPENDIX A. ΠΕΡΙΛΗΨΗ

Τα μεγάλα δεδομένα έχουν γίνει ιδιαίτερα διαδεδομένα στις καθημερινές δραστηριότητες μεγάλων οργανισμών κι επιχειρήσεων. Το μέγεθος των μεγάλων δεδομένων και του ρυθμού αύξησής τους είναι τεράστιο. Η τεχνολογία μεγάλων δεδομένων είναι βέβαιο ότι θα χτυπήσει σύντομα την πόρτα κάθε επιχείρησης και οργανισμού σε κάθε τομέα.

Η ορολογία των μεγάλων δεδομένων αφορά μεγάλες ή πολύπλοκες δομές δεδομένων τέτοιες ώστε οι παραδοσιακές εφαρμογές επεξεργασίας δεδομένων είναι ανεπαρκείς για την αντιμετώπισή τους. Τα οικοσυστήματα μεγάλων δεδομένων δύναται να βοηθήσουν στην ακριβέστερη ανάλυση, τη λήψη καλύτερων αποφάσεων, στη βελτίωση της λειτουργικής αποτελεσματικότητας, την ελαχιστοποίηση του κόστους και των απειλών για την εκάστοτε μορφή επιχείρησης. Περιλαμβάνουν τεράστιο όγκο, υψηλή ταχύτητα και εκτάσιμη ποικιλία δεδομένων και μορφών.

Τα Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks ή CNN) είναι η κορυφαία αρχιτεκτονική στη βαθιά μάθηση που χρησιμοποιείται για τις τεχνικές επεξεργασίας εικόνων. Δουλεύουν σε εικόνες με τρόπο παρόμοιο με τον ανθρώπινο εγκέφαλο: με την εξεύρεση μικρότερων λεπτομερειών και έπειτα σε υψηλότερα επίπεδα προς πιο αφηρημένα χαρακτηριστικά.

Σκοπός της παρούσας διπλωματικής εργασίας για το ‘Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών (ΔΠΜΣ) Γεωπληροφορικής’ αποτελεί η εγκατάσταση και η ανάπτυξη ενός συστήματος NoSQL (Not Only SQL) βάσεων δεδομένων. Το σύστημα αυτό είναι ένα διασυνδεδεμένο σύστημα αρχείων, με την ονομασία Hadoop Ecosystem, που αποσκοπεί στην ανάδειξη ενός framework ανοιχτού λογισμικού (open source) το οποίο διαχειρίζεται την επεξεργασία και την αποθήκευση εφαρμογών μεγάλων δεδομένων (big data applications) σε συστήματα τύπου συμπλέγματος (cluster). Η Hadoop μπορεί να διαχειριστεί πολυποίκιλους τύπους-δομημένων και μη-δεδομένων-και αποτελεί το κέντρο των τεχνολογιών μεγάλων δεδομένων, οι οποίες υποστηρίζουν πληθώρα εφαρμογών συμπεριλαμβανομένων της εξόρυξης δεδομένων, αναλύσεων με πρόβλεψη και μηχανικής μάθησης (data mining, predictive analytics and machine learning).

Το οικοσύστημα Hadoop είναι το πιο ευρέως αποδεκτό και χρησιμοποιούμενο πλαίσιο ανοιχτού κώδικα για τον υπολογισμό μεγάλων αναλυτικών δεδομένων σε ένα εύκολα κλιμακώσιμο περιβάλλον, που επιτρέπει την αποθήκευση και επεξεργασία μεγάλων δεδομένων σε ένα κατανεμημένο περιβάλλον συμπλεγμάτων υπολογιστών χρησιμοποιώντας απλά μοντέλα προγραμματισμού. Έχει σχεδιαστεί για να επεκτείνεται από ένα διακομιστή σε χιλιάδες μηχανές, το καθένα από τα οποία προσφέρει τοπικούς υπολογισμούς και αποθήκευση. Το Hadoop είναι ένα λογισμικό Java που υποστηρίζει κατανεμημένες εφαρμογές με μεγάλη ένταση δεδομένων και λειτουργεί με χιλιάδες κόμβους έως και petabytes δεδομένων. Τα δύο μεγάλα κομμάτια του Hadoop είναι το HDFS, το σύστημα του Hadoop που τρέχει πάνω από τα συστήματα αρχείων των υποκείμενων λειτουργικών συστημάτων και το MapReduce, ένα κατανεμημένο πλαίσιο επεξεργασίας, όπου η εφαρμογή διαιρείται σε πολλά μικρά κομμάτια εργασίας, καθένα από τα οποία μπορεί να εκτελείται ή εκτελείται εκ νέου σε οποιονδήποτε κόμβο του συμπλέγματος. Διαχειρίζεται πολύ καλά την αποθήκευση και την ανάλυση μη δομημένων δεδομένων. Το Hadoop είναι μια δοκιμασμένη λύση στο περιβάλλον παραγωγής και έχει εγκριθεί από κορυφαίους οργανισμούς όπως το Google, το Yahoo και το Facebook.

Για την ανάπτυξη της απαραίτητης διεπαφής γίνεται χρήση της γλώσσας προγραμματισμού Python προκειμένου να αναδειχθούν τα πλεονεκτήματα της Τεχνολογίας Μεγάλης Κλίμακας (Big Data Technology). Στη συνέχεια, αναπτύσσουμε μια εφαρμογή Τεχνητής Νοημοσύνης που χρησιμοποιεί τεχνολογία Βαθιάς Μάθησης (Deep Learning), η οποία ανιχνεύει πλοία σε δορυφορικές εικόνες, χρησιμοποιώντας όλα τα βασικά εργαλεία του οικοσυστήματος Hadoop (Hadoop Ecosystem), με φιλικό προς το χρήστη τρόπο.

Στην παρούσα εργασία έγινε ανάπτυξη ενός αλγορίθμου βαθιάς μάθησης (deep learning), που ανήκει στην κατηγορία των εφαρμογών μηχανικής μάθησης (machine learning) και αναγνωρίζει την ύπαρξη πλοίων από δορυφορικές εικόνες λιμανιών. Προς το σκοπό αυτό έγινε χρήση μιας βιβλιοθήκης υψηλού επιπέδου γραμμένη σε Python, την Keras.

Οι πηγές δεδομένων (data sources), για την εφαρμογή του αλγορίθμου, αποθηκεύονται σε ένα καταναμημένο σύστημα αρχείων του Hadoop, γνωστό ως HDFS (Hadoop Distributed File System), το οποίο αποτελεί υποπρόγραμμα αυτού. Το HDFS δύναται να διατηρεί πολύ μεγάλο όγκο δεδομένων προσφέροντας ευκολότερη πρόσβαση. Η αποθήκευση αυτού του όγκου γίνεται σε πολλές μηχανές (servers) 'in redundant fashion', ώστε να μπορεί να γίνει η διάσωση του συστήματος από ενδεχόμενες απώλειες σε περίπτωση βλάβης.

Επιπλέον γίνεται χρήση μιας ανοιχτής, μη σχεσιακής, καταναμημένης βάσης δεδομένων γραμμένης σε Java, η οποία διαμορφώθηκε μετά το Bigtable της Google, αναπτύχθηκε ως μέρος του έργου Apache Software Foundation, του Apache Hadoop, και λειτουργεί πάνω από το HDFS παρέχοντας δυνατότητες Bigtable για Hadoop. Η βάση αυτή είναι η Hbase. Σε αυτήν, αποθηκεύονται πολλά δεδομένα των data sources, που χρησιμοποιούνται πριν και κατά την ανάπτυξη του προαναφερθέντος Deep Learning Algorithm, όπως labels infos, longitudes, latitudes, scene id's των εικόνων, καθώς και κατόπιν της υλοποίησης αυτού, όπως πίνακες που περιέχουν τα αποτελέσματα της πρόβλεψης και τα βάρη κατανομής για κάθε pixel των εικόνων.

Πιο συγκεκριμένα, η διπλωματική εργασία αποτελείται από τα παρακάτω μέρη: Κατόπιν της εγκατάστασης του Hadoop Ecosystem, σε ψευδο-καταναμημένη λειτουργία (pseudo-distributed mode), σε έναν εικονικό διακομιστή (server), που τρέχει σε μια εικονική μηχανή (virtualbox), με λειτουργικό σύστημα Ubuntu 16.04, κάνουμε τις κατάλληλες παραμετροποιήσεις και την αρχικοποιούμε θέτοντας το σε λειτουργία με IP 192.168.0.10.

Στη συνέχεια έχει γίνει ανάπτυξη κατάλληλων συναρτήσεων προγράμματος (scripts), με σκοπό την διεπαφή, τόσο με το HDFS, όσο και την Hbase. Η ανάπτυξη των scripts έγινε σε μια από τις πλέον ανερχόμενες γλώσσες προγραμματισμού, την Python, με την χρήση κατάλληλων βιβλιοθηκών. Πιο συγκεκριμένα: Για την αλληλεπίδραση και την διαχείριση του HDFS, τόσο μέσω του διακομιστή (server), όσο κι απομακρυσμένα μέσω ενός χρήστη-πελάτη (client), έγινε χρήση της βιβλιοθήκης hdfs3 (open source), με την δημιουργηθείσα εφαρμογή να δίνει πληθώρα επιλογών, όπως:

- Σύνδεση στο HDFS, με χρήση της κατάλληλης IP και Port Απεικόνιση των περιεχομένων του root (/), των φακέλων (destination folders) αυτού και των περιεχομένων τους
- Απόδοση συγκεκριμένων δικαιωμάτων (permissions) σε χρήστες (users) κι ομάδες (groupusers), ώστε να έχουν συγκεκριμένες δυνατότητες ανάγνωσης, εγγραφής ή εκτέλεσης (read, write, execute)

- Δυνατότητα δημιουργίας νέων διαδρομών/φακέλων εντός του HDFS, αμφίδρομη μεταφορά ολόκληρων φακέλων ή αρχείων από τον server στον client (κι αντιστρόφως), ώστε να πραγματοποιηθούν τα ζητούμενα tasks (πάντα λαμβάνονται υπόψη τα permissions που έχουν δοθεί στον εκάστοτε user)
- Δυνατότητα ανάγνωσης αρχείων, απευθείας από την HDFS, με σκοπό την πραγματοποίηση συγκεκριμένων αναλύσεων

Εντός του HDFS δημιουργούνται ξεχωριστοί φάκελοι προορισμού, που περιέχουν:

- Τις εικόνες που θα χρησιμοποιηθούν για την εκπαίδευση (training) του μοντέλου αναγνώρισης πλοίων σε δορυφορικές εικόνες
- Αρχεία JSON (JSON Files), που περιέχουν τις μαζικές πληροφορίες του μοντέλου, τα οποία αποθηκεύονται στην συνέχεια σε πίνακες εντός της Hbase
- Αποτελέσματα της ταξινόμησης (classification) των εικόνων, που προκύπτουν ως απόρροια του αλγόριθμου (εικόνες που εισάγουμε στον αλγόριθμο κι αποφαίνεται εάν είναι ή όχι πλοίο, εν προκειμένω)
- Δορυφορικές εικόνες λιμανιών, που έχουν ‘σαρωθεί’ ανά pixel κι έχουν αποτυπωθεί τα πιθανά σημεία, που βρίσκονται πλοία, εντός αυτών, με την εφαρμογή του αλγόριθμου

Αντίστοιχα, με το προηγούμενο, έγινε ανάπτυξη ενός application script, με σκοπό την αλληλεπίδραση και την διαχείριση με την NoSQL βάση δεδομένων Hbase. Χρησιμοποιήθηκε ξανά μια open source library, η happybase v0.98, η οποία δίνει αντίστοιχα:

- Δυνατότητα σύνδεσης στην Hbase, που βρίσκεται εγκατεστημένη στην πλευρά του διακομιστή (server)
- Απεικόνιση των περιεχομένων πινάκων (tables) αυτής
- Δυνατότητα απόδοσης συγκεκριμένων δικαιωμάτων, στον εκάστοτε χρήστη, για τον εκάστοτε πίνακα
- Δημιουργία και διαγραφή πινάκων, εισαγωγή και διαγραφή τιμών σε αυτούς, καθώς και η δυνατότητα μαζικής απόδοσης (batches) τιμών δεδομένων, για διαφορετικά column families, ακόμα κι εντός του ίδιου πίνακα
- Σάρωση και δυνατότητα ανάκτησης των δεδομένων, βάση συγκεκριμένων κριτηρίων (rows, columns, column families, timestamps)

Το τελευταίο τμήμα της διπλωματικής εργασίας, όπως αναφέρθηκε και προηγουμένως, περιέχει έναν αλγόριθμο αναγνώρισης εικόνων πλοίων σε δορυφορικές εικόνες, καθώς και την πιθανότητα ύπαρξης πλοίων σε εικόνες λιμανιών. Για την υλοποίηση του μοντέλου έγινε χρήση κατάλληλων datasets από εικόνες που έχουν ληφθεί από δορυφόρους κι απεικονίζουν πλοία εντός λιμανιών και κόλπων.

Κατόπιν της δημιουργίας του μοντέλου Βαθιάς Μάθησης (deep learning) αποθηκεύουμε τα αποτελέσματα εντός της Hbase, για πιθανή περαιτέρω ανάλυση και χρησιμοποιούμε εικόνες πλοίων, ώστε να γίνει επαλήθευση των αποτελεσμάτων. Τα αποτελέσματα δύναται να αποθηκευθούν σε ειδικές διαδρομές, εντός του HDFS. Τέλος, κάνουμε εισαγωγή κάποιων δορυφορικών εικόνων λιμανιών και κόλπων, όπου με τη χρήση κατάλληλων αλγορίθμων σάρωσης της εικόνας, εφαρμόζουμε τα αποτελέσματα του Αλγόριθμου Βαθιάς Μάθησης (deep learning algorithm) και απεικονίζουμε πιθανές προβλέψεις.

Το κεφάλαιο 1 παρέχει μια εισαγωγή στα Μεγάλα Δεδομένα και στο Οικοσύστημα της Hadoop. **Στο 2^ο κεφάλαιο** αναπτύσσεται μια επισκόπηση του θεωρητικού υποβάθρου που χρησιμοποιείται για τις διεργασίες των εφαρμογών των Hadoop, HBase και Βαθιάς Μάθησης (Deep Learning). **Στα κεφάλαια 3 και 4** περιγράφονται οι διεργασίες ανάπτυξης μια πλατφόρμας διεπαφής με τα περιβάλλοντα των Hadoop και HBase αντίστοιχα, με τη χρήση της γλώσσας προγραμματισμού Python. **Το κεφάλαιο 5** προτείνει την δημιουργία ενός πλαισίου βαθιάς μάθησης, με την δυνατότητα εντοπισμού πλοίων σε δορυφορικές εικόνες, με σκοπό την αλληλεπίδραση και την ανάδειξη της εφαρμοστικότητας των διεπαφών που αναπτύχθηκαν στα κεφάλαια 4 και 5. Η παρούσα μεταπτυχιακή διπλωματική εργασία συμπυκνώνεται στο **κεφάλαιο 6**, όπου δίνονται κάποιες προτάσεις για μελλοντική έρευνα, βασισμένες στην εξέλιξη των τεχνολογιών, στους άνωθεν τομείς.

Λέξεις Κλειδιά

Hadoop, Hbase, HDFS, MapReduce, Τεχνητή Νοημοσύνη, Βαθιά Μάθηση, Μεγάλα Δεδομένα, Νευρωνικά Δίκτυα, Αναγνώριση Εικόνας, Δορυφορικές εικόνες, Συνεργατικά Νευρωνικά Δίκτυα, Python, hdfs3, Happybase.

APPENDIX B. Interaction Between HDFS and DL Model

The specific appendix is devoted to the implementation part where the interaction between Ship Detection Algorithm and HDFS occurs.

As pre-mentioned in chapter 5, Ship Detection Application interact with HDFS, in order to retrieve datasets, such as during configuration and compilation process, or store data for future usage such as weights data, indispensable, when model needs to be loaded to avoid a new-time consuming-re-train process.

Figure 55 displays the interaction process, where 'retrieve()' function downloads model's dataset, which is converted in JSON format, that includes the whole requisite data for training process. After dataset is fetched from HDFS directory, data is converted from list to array format, using numpy library, to pass the whole data into model's input for train and validation process intentions in the most suitable way, using the architecture that constructed and displayed in previous chapters. Images reshaped at 80X80 pixels (as referred in chapter 5.2), a sample of these are figured out into three (3) different channels and finally are imported into network.

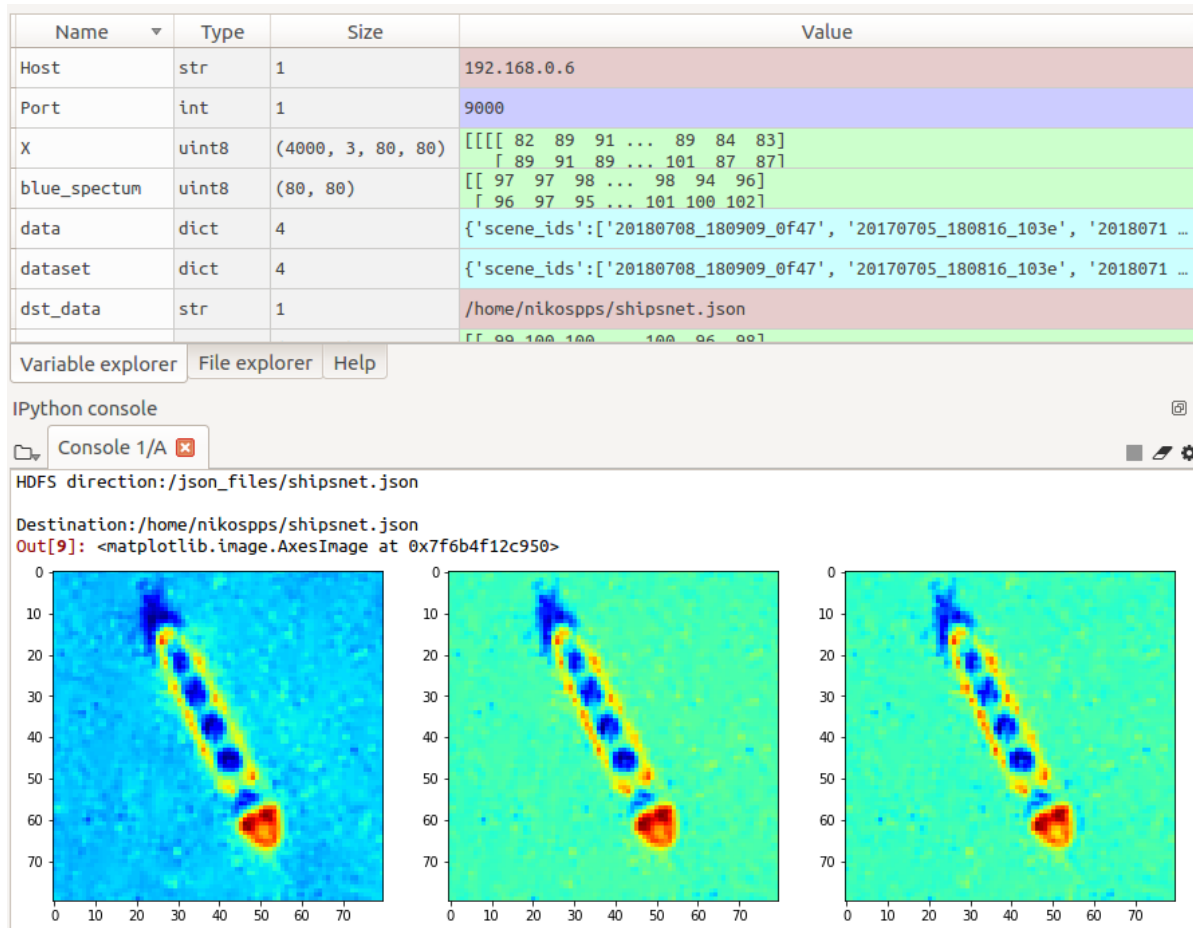


Figure 55. Interaction Between HDFS and Compilation Process

With regard to use the model repeatedly, without spending time during training process, we saved the weights into HDFS directory called ‘Prediction Weights’ (chapter 5). Applying the same function, we get back hdf5 format file and load the weights, the architecture, training configuration as well as the results from the optimizer. As follows, we return to the training state where we stopped. Retrieving back, loading and displaying the summary of the architecture of the model which was pre-trained exposed in figure 56.

The screenshot shows an IPython console window with the following content:

```
HDFS direction:/Prediction Weights/ship_detection_weights.h5
Destination:/home/nikospps/ship_detection_weights.h5
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 80, 80, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 40, 40, 32)	0
dropout_1 (Dropout)	(None, 40, 40, 32)	0
conv2d_2 (Conv2D)	(None, 40, 40, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 32)	0
dropout_2 (Dropout)	(None, 20, 20, 32)	0
conv2d_3 (Conv2D)	(None, 20, 20, 32)	82976
max_pooling2d_3 (MaxPooling2D)	(None, 10, 10, 32)	0
dropout_3 (Dropout)	(None, 10, 10, 32)	0
conv2d_4 (Conv2D)	(None, 10, 10, 32)	82976
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 32)	0
dropout_4 (Dropout)	(None, 5, 5, 32)	0
flatten_1 (Flatten)	(None, 800)	0
dense_1 (Dense)	(None, 2)	1602

```
Total params: 177,698
Trainable params: 177,698
Non-trainable params: 0
```

Figure 56. Model Weights Re-Load from HDFS

PIL library (reference to chapter 5) is used for images loading and manipulation. During ship detection process over satellite images, the necessity of loading from HDFS, converting into array format and scanning for further validation process and prediction results is important. Using the above function, we download a random image from ‘scene’ direction called ‘sfbay.png’, save it in an array format and scanning over it, making predictions using model’s weights that reloaded above. Figure 57 illustrates the specific operation, which after image download bay satellite image from HDFS directory makes the appropriate predictions and defines possible ship positions, point them out using white patches.

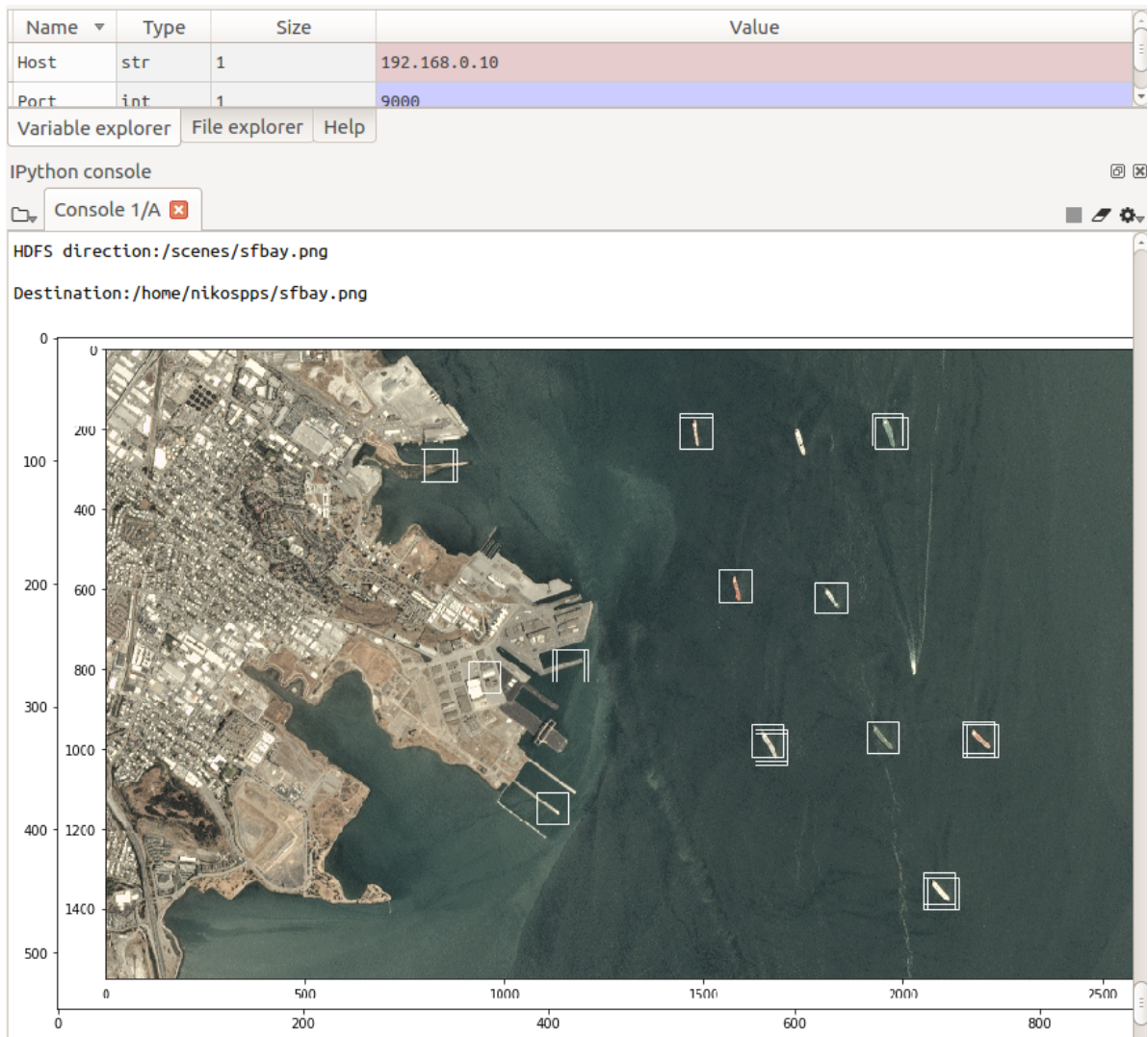


Figure 57. Retrieve Satellite Image and Make Prediction Process

Apart from download processes, pythonic interaction platform also offers storage capabilities for output results, for further manipulation and analysis. With refer to figures 48-53 on chapter 5, prediction results were stored to a directory called ‘Prediction Output Results’.

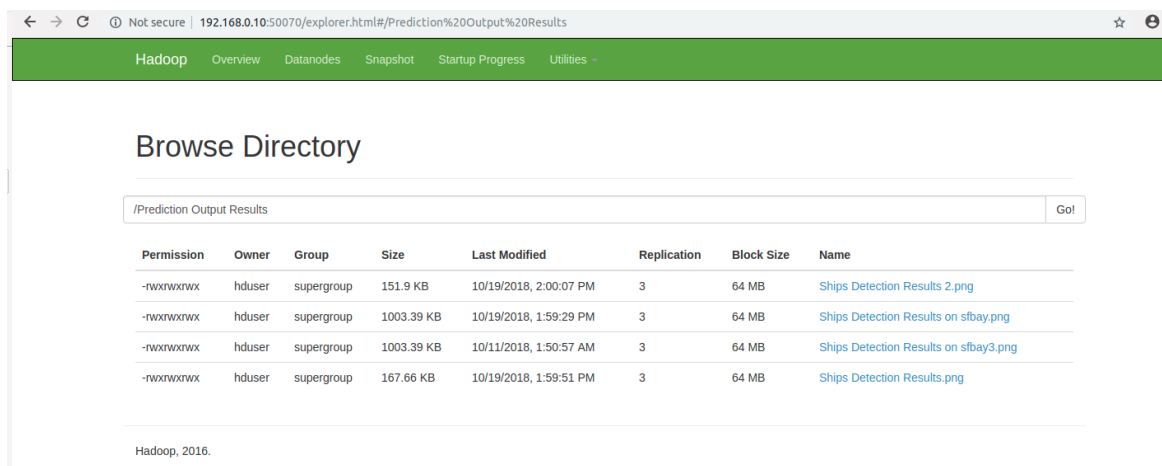


Figure 58. Output Results Stored into HDFS Directory

REFERENCES

- [1] Xiaomeng,S (2012). *Introduction to Big Data*. Norway: NTNU. 2-11.
- [2] Padberg, M (2015). *Big Data and Business Intelligence: a data-driven strategy for e-commerce organizations in the hotel industry*. Twente: University of Twente. 57.
- [3] Wu, S (2015). *Big Data Processing With Hadoop*. Finland: Turku University. 45.
- [4] Prajapati, V (2013). *Big Data Analytics with R and Hadoop*. Birmingham: Packt Publishing. 222.
- [5] White, T (2009). *Hadoop:The Definitive Guide*. USA: O'Reilly Media, Inc. 503.
- [6] Bengfort, B.; Kim, J (2016). *Data Analytics with Hadoop*. USA: O'Reilly Media, Inc. 270.
- [7] Radtka, Z,; Miner, D (2016). *Hadoop With Python*. USA: O'Reilly Media, Inc. 63.
- [8] Kalický, A (2013). *High Performance Analytics*. Prague: Charles University in Prague. 73.
- [9] Jia, B (2010). *Data Acquisition in Hadoop System*. Norway: University of Stavanger. 45.
- [10] Petsas, K (2016). *Study of technologies/research systems for big scientific data analytics*. Piraeus: University of Piraeus. 74.
- [11] Subhash Raste, K (2014). *Big Data Analytics - Hadoop Performance Analysis*. USA: San Diego University. 55.
- [12] Suurna, E (2014). *Data analytics on the example of cluster computing framework Apache Spark*. Tallinn: Tallinn University of Technology. 65.
- [13] Fox, E (2017). *Clustering and Topic Analysis Final Report*. USA: Virginia Polytechnic Institute and State University. 50.
- [14] Continuum Analytics (2018). *hdfs3 Documentation*. 3rd ed. USA: Continuum Analytics. 17.
- [15] George, L (2011). *HBase: The Definitive Guide*. USA: O'Reilly Media, Inc. 523.
- [16] Dimiduk, N,; Khurana, A (2013). *HBase in Action*. USA: Manning Publications Co. 334.
- [17] Spaggiari, JM,; O'Dell, K (2016). *Architecting HBase Applications: A Guidebook for Successful Development and Design*. USA: O'Reilly Media, Inc. 231.
- [18] Agrawal, B (2013). *Analysis of large time-series data in OpenTSDB*. Norway: University of Stavanger. 62.
- [19] Bolsterlee, W (2016). *HappyBase*. 3rd ed. USA: MIT License. 29.
- [20] Chollet, F (2018). *Deep Learning with Python*. USA: Manning Publications Co. 362.
- [21] Roig Mari, C (2016). *Deep Learning Architectures For Computer Vision*. Barcelona: Universitat Politècnica de Catalunya. 40.
- [22] Claesson, L,; and Hansson, G (2017). *Deep Learning Methods and Applications*. Sweden: Chalmers University of Technology. 94.

- [23] Luberg, KK (2018). *Human Body Poses Recognition Using Neural Networks with Class Based Data Augmentation*. Tartu: University of Tartu. 47
- [24] Serra, X (2017). *Face recognition using Deep Learning*. Catalonia: Polytechnic University of Catalonia. 78.