Image Object Recognition Using Apache Hadoop and Python

Jaileen Del Valle Maldonado
Master in Computer Science
Nelliud Torres Batista Ph.D.
Electrical & Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— The amount of data generated by people each day on social media platforms is increasing at an alarming rate. Studies performed show that approximately 1.5 billion images are uploaded to the internet each day. Applications that can use and analyze this data are not available to all users due to limitations in processing power or storage space required for the analysis of these large datasets. Apache Hadoop is an open-source framework that allows distributed processing and fault tolerance of Big Data with the use of commodity hardware using Hadoop Distributed File System (HDFS) and MapReduce. Using HDFS data is stored in a distributed manner across different machines (datanotes). The use of the MapReduce framework parallelized computing is available and manageable to be able to mine and analyze the image data available created by users. The focus of this article will be the analysis of image data in large datasets to create feature vectors using the k-means algorithm to group together images that contain similar objects inside them using Apache Hadoop, MapReduce, Apache Spark, Computer Vision, and the Python programming language.*

Key Terms *— K-Means Clustering, Map Reduce, Sequence File, Scale Invariant Feature Transform (SIFT)*

INTRODUCTION

The term Big Data is used to describe the huge volumes of data generated by digital processes. Currently, the use of social media sites has increased the amount of image data being uploaded every day on sites like Facebook, WhatsApp, and Twitter. This increasing amount of information growth has created a new kind of problem for data analysts. Analyzing and processing such huge amounts of data could create bottlenecks caused by the use of a single computer, power concerns, and the amount of storage space needed. The present-day computer architectures are reaching their physical limitations and with this, the implementation of distributed systems is becoming more widespread. The main reason to which the popularity of distributed systems can be attributed are: i) physical limitations of processors, ii) scalability, iii) fault tolerance iv) latency. [1] With the use of distributed systems tasks are completed by dividing a task into multiple subtasks. Dividing system tasks is known as parallelization, this makes applications running on a distributed system more scalable and efficient. A widely known distributed system platform is Apache Hadoop and the Hadoop MapReduce algorithm.

Hadoop is being used as a system for processing huge datasets by using parallel and distributed computing. In addition, various studies have been performed using Apache Hadoop an example of these are: conducting analysis of text files, examining DNA sequencing data, converting images to PDF, and feature extraction and selection. These studies are performed by dividing the data across multiple features like algorithm parameters, images, or pixels. The k-means algorithm has been implemented within the MapReduce programming framework to analyze images and classify them based on their color.

BACKGROUND AND RELATED WORK

The following section will explain in detail the tools, concepts, and previous knowledge required for the implementation of image object recognition using Apache Hadoop and Python.

Apache Hadoop

Hadoop is an open-source framework that works as a distributed system with a scalable, fault-tolerant design. It is used for data storage and processing. The creation of Apache Hadoop was inspired by Google's articles on Map Reduce and Google Distributed File System. Apache Hadoop was developed and written in the java programming language. Currently, it is being used for the analysis of Big Data at companies such as Facebook and Yahoo!. Since the Hadoop framework is best known for its scalability and fault tolerance it can be used for data analysis. The main components of Apache Hadoop are HDFS and MapReduce [2].

Apache Hadoop Distributed File System

HDFS is a filesystem designed for storing large files in a distributed manner with streaming data access patterns. These are usually run-on commodity hardware. The definition of "very large files" in the Big Data environment means files that could span from hundreds of GB, TB, or PB. The architecture of HDFS makes it scalable but there are a few drawbacks: 1. HDFS works better performing long sequential reads from files, but it is not used for random reads 2. Caching is not the best since files contain a big overhead and data would be re-read from the source. 3. HDFS only performs appends to files there is no updating functionality [2].

The files stored within HDFS are stored as metadata files and a collection of blocks. The standard block size for an HDFS file is 64 MB. Block size is important regarding image processing for example if a file is bigger than the default block size it could be stored in different physical locations and if a file is too small more than one file will be stored inside a block or one file of KB will be stored in the block causing an impact to HDFS performance.

The master component or master node in the HDFS architecture is called the NameNode. The NameNode stores metadata information like where each block is stored, and how many times the file is replicated within the system and tracks the DataNodes. The DataNode is where the files and data are stored in the system. The NameNode is the one that administers all the DataNodes in the cluster. This includes DataNode failure and heartbeat messages. A heartbeat is a message that includes information about activity within the cluster and DataNode failures. These messages are configured to be sent every three seconds [3].

HDFS has implemented a checksum to track faults in storage, network, or data that arrives corrupted. Each DataNode block has its hidden checksum. When a client using HDFS requests a block of data; it receives both the content and the checksum file. If the checksum does not match the locally calculated checksum the system will look in another DataNode for the replica of the data. See Figure 1 below.

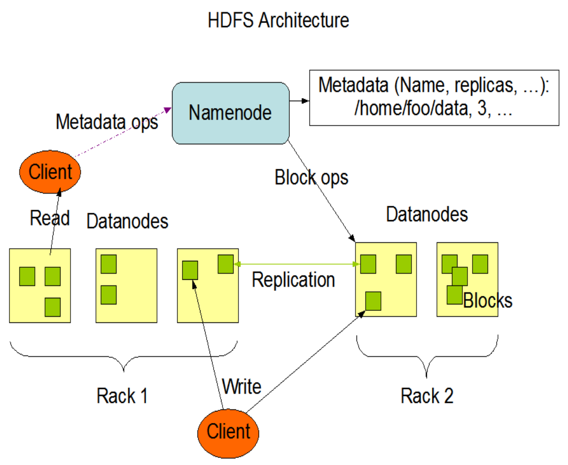


Figure 1
HDFS Architecture

Apache Hadoop MapReduce

MapReduce was originally developed by Google and detailed in the article; MapReduce: Simplified Data Processing on Large Clusters. Apache Hadoop implemented its own open-source version of MapReduce considering the design described in Google's article. MapReduce can be defined as a programming model used for processing data. MapReduce considers the problems of distributing the data in a network of computers to always assure that all available memory, processor, and storage are used in the most optimized manner. MapReduce works with parallel data processing using the map phase and the reduce phase. Both phases have input and output key-value pairs. The type of input and output is chosen by the programmer or developer using this programming paradigm.

Additionally, the programmer must specify what will the map and reduce function do. Hadoop divides the input into fixed-size splits, for each split created in the map and reduce phases are performed using the client-specific code in both functions. Hadoop performs exceptionally well when all the data being processed is contained within a single DataNode in HDFS [3]. The Map function writes its output to the local disk since it is an intermediate output received as the reduce function's input. The reduce function is the one that writes its output directly to HDFS because the output from the phase is the final output of this programming model.

The execution of a MapReduce program or job can be summed up in the following steps:

1. The user uploads input data to HDFS, which in turn distributes and stores it in the computing nodes.
2. The user starts the job by specifying the MapReduce program to execute along with the input-output paths and other parameters.
3. The master node sends a copy of the program along with its parameters to every computing node and starts the job.
4. Computing nodes start the Map phase by first processing data on their local storage, fetching more data from other nodes if necessary and possible.
5. After all Map tasks are finished, their output is sorted in a way, that for every Key, a Reduce task processes all the pairs with the Key.
6. Once the Reduce phase is finished and its output has been written back to HDFS, the user retrieves the data. [4]. See Figure 2.

Also, the programmer must specify various things to be able to write a MapReduce program: InputFormatClass, Mapper class, Reducer class, and OutputFormat. Hadoop MapReduce performs better when the input is a small number of large files. When faced with a large amount of small files MapReduce is faced with an increasing amount of seeks that are needed to run a job. One way to avoid this problem is to merge all the small files by turning them into Sequence Files. In Sequence Files the key is the filename and the value, the file content.

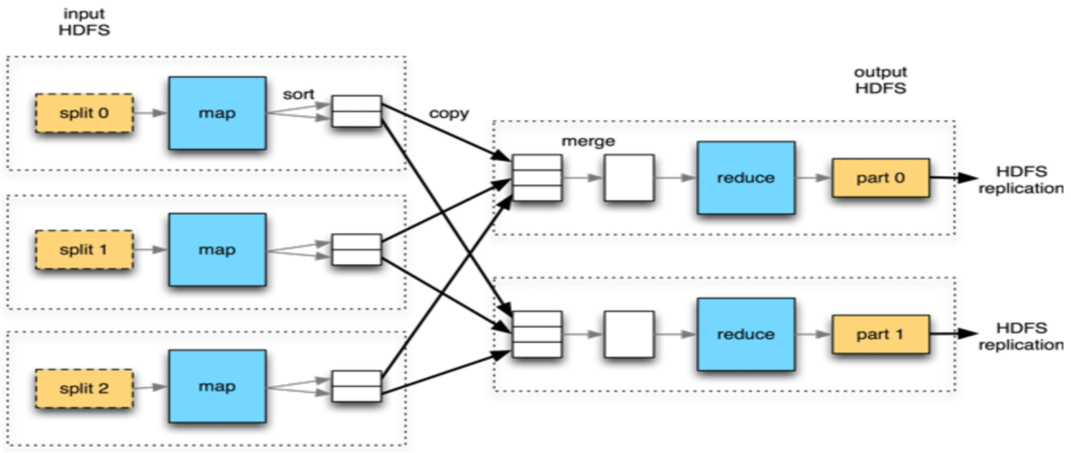
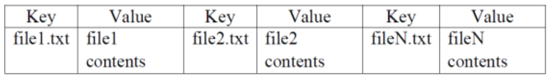


Figure 2
MapReduce Program Flow

Sequence Files

Sequence files are a Hadoop file format that stores binary key-value pairs in a sequential form. This type of file is splitable, supports compression, and can store arbitrary types using serialization frameworks [3]. In addition, it is possible to create multiple sequence files in parallel and merge them into a bigger, more robust file. Hadoop was designed to work with files that are large, and larger than the default block size of 64 MB or 128 MB, depending on the version of Hadoop implemented.

Sequence files can be visualized as a container for multiple small files. In the case of images, the file name would be converted into the key of the sequence file and the value will be the binary content of the image or file. The creation of a sequence file must be performed using a MapReduce job. This job would receive as an input the large number of small images which are already stored within different blocks in HDFS, and it will output the sequence file in a <key, value> format. See Figure 3 below.



Key	Value	Key	Value	Key	Value
file1.txt	file1 contents	file2.txt	file2 contents	fileN.txt	fileN contents

Figure 3
Visualization of a Sequence File

The use of sequence files is required for the objective discussed in this article; since most of the files generated by social media or found across the internet are small in file size. Hadoop has some limitations when dealing with many small files. The performance of Apache Hadoop's HDFS is severely affected by this for the following reasons: DataNode memory, NameNode memory, and CPU time consumption.

The DataNode memory is affected because each file will be stored in a single HDFS block if the size is smaller than the default block size and having many blocks with such small files consume too much of the DataNode memory. NameNode memory is affected because files, directories, and blocks are objects that consume a lot of the system's memory. Finally, the CPU time is spent more slowly and less efficiently since the MapReduce job will have the same number of mappers as the number of files being converted [5]. These deficiencies in the Hadoop system are mitigated when converting the small files into a sequence file.

Feature Extraction using Descriptors

The SIFT was created by D.Lowe in 2004. The main goal of this algorithm is to extract features or descriptors from keypoints [6]. The features will be extracted and stored in HDFS as a sequence file having as the key the image name and the features as the value. The dimensions of the feature are separated by commas for simplicity of use. SIFT is a reliable algorithm since it is invariant to image scale and rotation. The descriptors of this algorithm were created for the sole purpose of image matching. Each feature vector descriptor is highly different which facilitates the process of matching with another feature vector within the file system.

The properties of an image that are commonly used for feature extraction are intensity, color, and texture. Image feature building requires a feature vector to have the repeatability property. When a feature vector has a proper repeatable factor when feature extraction is done it will bring back many of the same features that were detected from the images being compared. Consistency is also an important factor in feature detection and extraction since features must be detected even while an image has suffered changes like blurring, re-orientation, and re-escalation [7].

The SIFT algorithm can be divided into four steps: (1) keypoint localization, (2) orientation assignment, (3) keypoint descriptor, and (4) keypoint matching. In keypoint localization, an image is scaled, and then with this scaling, the Difference of Gauss is calculated. The result of the Difference of Gauss is then used as the input to calculate the Laplacian of Gaussian. Afterward, a pixel is compared to its 8 neighboring pixels and 9 pixels in the previous and next scale. If it is determined that the pixel is a local maximum, then it can be considered a potential keypoint. In the orientation assignment, a direction is given to each keypoint. A histogram that covers all 360 degrees is created with 36 bins. The highest point in the histogram is used and any other peak that is above 80 percent is also considered.

In the keypoint descriptor stage, a window of 16x16 is crated around the selected keypoint. This window is further divided into 16 sub-blocks of 4x4. In each of these 16 sub-blocks, an orientation histogram of 8 bins is created. This makes for a total of 128 bins that are transformed into a vector of descriptors [7]. The recently created feature vector still has a few problems; it is rotation dependent and illumination dependent. These are mitigated by removing keypoint orientation and thresholding the big numbers. Resulting in a normalized vector. The final step in the SIFT algorithm is keypoint matching. Keypoint matching is achieved when two images are matched by the identification of the nearest neighbor.

Feature Vector Clustering

One of the most important tasks in data mining is the use of clustering techniques. Clustering allows the user to extract significant knowledge from the dataset. Clustering consists of partitioning the data of a dataset into different amounts of subsets or groups in a way that all the data that is similar end up together and the other unrelated data is grouped in other subgroups. One of the most known and most used clustering techniques in data mining is the K-Means clustering algorithm. K-means is defined as an unsupervised clustering algorithm [8]. The main purpose of K-Means is to classify the data in a dataset into multiple groups based on the patterns found in the data. This is achieved by looking for several clusters (k) in a dataset. The time complexity of the K-Means algorithm is O (nkt), where n refers to the number of datapoints, k is the number of clusters and t is the number of iterations [9].

The K-Means clustering algorithm can be divided into two main steps: (1) Cluster Assignment and (2) Move Centroid Step. These steps are repeated iteratively until one of the following conditions is reached; the centroids will not change their positions anymore or the iterations have gone through the maximum number. The centroid of the clusters (k) can be chosen randomly at the start, or several centroids can be assigned initially using the elbow method. See Figure 4 below.

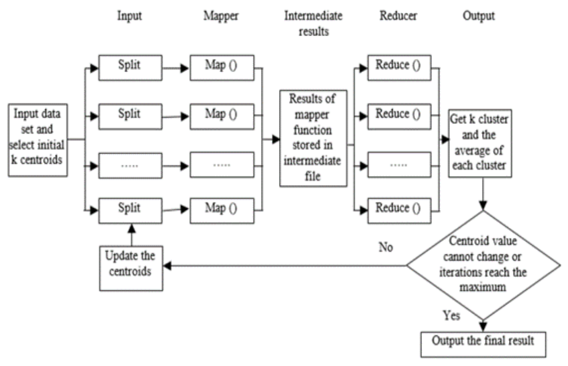

Input
Mapper
Intermediate results
Reducer
Output
Input data set and select initial k centroids
Split
Split
.....
Split
Map ()
Map ()
.....
Map ()
Results of mapper function stored in intermediate file
Reduce ()
Reduce ()
.....
Reduce ()
Get k cluster and the average of each cluster
Centroid value cannot change or iterations reach the maximum
No
Yes
Update the centroids
Output the final result

Figure 4
Visualization of the K-Means algorithm in MapReduce

The Elbow Method consists of plotting values with a different number of clusters (k). The graph will show a tendency where the number of clusters will grow while the amount of data points per cluster diminishes. Hence, the optimal number of clusters will be at the inflection point. The elbow point can be easily chosen as observed in the graph. See Figure 5 below.

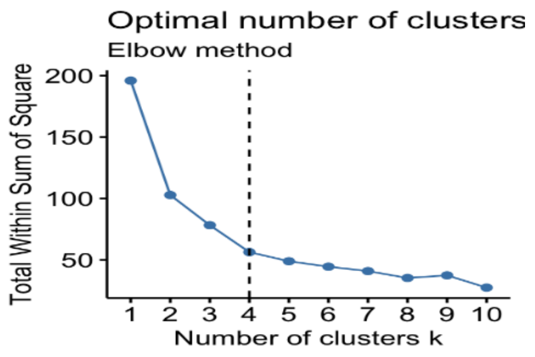


Figure 5
Elbow Method Example

After having a set number of clusters, the next step is to calculate the Euclidean distance. The Euclidean distance is the distance calculation between datapoints and the established centroid or cluster centers [9]. The K-Means algorithm can be implemented on the terms of Hadoop MapReduce, where the Map function will assign each data point to the nearest cluster center or centroid and the Reduce function will be constantly updating the cluster centers until they remain unchanged.

METHODOLOGY

Hadoop's principal components: Hadoop Common, HDFS, MapReduce, and YARN will be installed using a binary tarball which can be found on the Apache Software Foundation. Before installing Hadoop, the user must make sure they have Java installed on the computer. The Hadoop environment can run on Unix and Windows, but only Linux is given support, and Windows is mostly used for development purposes. A stable release in form of a gzipped tar file can be downloaded from the releases page of Apache Hadoop. After downloading the stable version of Apache Hadoop various environment variables must be set, this file should be a shell startup file, for example, JAVA_HOME and HADOOP_HOME. All the individual components in Hadoop are configured via XML files. Hadoop can be configured to run in one of three modes: standalone or local mode, pseudo-distributed mode, and fully distributed mode [3].

In standalone mode, no daemons are running, and everything runs in a local JVM. In pseudo-distributed mode, the daemons run locally creating a simulated virtual cluster and in fully distributed mode the daemons run on a cluster of PCs or machines. Configuration for pseudo-distributed mode and fully distributed mode require to have SSH installed and configured to be a password-less login. After this step, the HDFS installation must be formatted to create the storage directories and the namenode's data structures on the system. When all these steps are completed the only left to do is to start the Hadoop daemons on the machine.

Additionally, another tool from the Hadoop ecosystem that needs to be installed is Apache Spark. First, the correct Spark version must be downloaded which should be the correct version for the Hadoop installation that's already configured in the machine. After this step, two Hadoop configuration files must be included inside Apache Spark's classpath, these are hdfs-site.xml and core-site.xml. These configuration files will provide behaviors for the HDFS client and will set the default filesystem name. Inside Hadoop, the files can be found within the /etc/Hadoop/conf directory. Making these files visible requires including an environment variable located in spark-env.sh. The variable is HADOOP_CONF_DIR. Enabling the use of Python with Spark requires installing Pyspark, this can be done using the command pip install pyspark [10]. Last, when the use of applications that use spark a SparkSession must be created. The SparkSession allows the use of Spark across the cluster.

Now that HDFS, MapReduce, and Apache Spark have been installed on the machine we can begin the process of processing the images and turn them into one sequence file. The first step is to store in HDFS a text file that contains the paths of all the images that are going to be used for object recognition purposes. This file is stored within HDFS using the following command: -put /home/file.txt /user/input, the user must be aware that the target path must be created beforehand using the command -mkdir /user/input.

After this file is stored in the distributed file system a MapReduce job must be performed to convert the images in the path into one SequenceFile. This MapReduce job can be completed without the use of a reduce function since the output of the map function does not have to be combined. The output of the map function will be a sequence file in the <key,value> format in which the key will be the path of the image and the value will be the contents of the image in a binary format. Afterward, another MapReduce job must be performed to read the image bytes from the images and save the image itself in HDFS. This job can also be performed using a map function since the individual results of the map function are the information needed to be stored in the file system.

Now that the sequencefile has been created, the features from the different images stored in HDFS must be extracted. The SIFT technique will the applied feature extraction algorithm. This functionality is included within the opencv library included in the pyspark script. SIFT is used to detect the different features in the images and generates feature vectors of the descriptors in a 128-dimensional array. Additionally, the pyspark script includes the numpy and pyspark libraries.

The input parameters of the pyspark script are the following: feature extraction name, the path where the sequence file is located, the output path where the file will be stored, and the number of partitions. First, the instance of Apache Spark is created by generating the spark context. The sequencefile that was specified in the input is transformed into a flatmap, which arranges the file in a dataframe. The images received will be converted into a 128-dimensional array. The resulting array is then decoded by using the computer vision function imdecode which converts the image data in the cache to an image format. After this process, the images are inputted to another function which creates and computes the keypoints and descriptors using the SIFT algorithm available in the computer vision library. Finally, the keypoints and descriptors created from the SIFT algorithm are filtered using a map function that groups the filenames with the feature extracted.

After having created the feature vectors using the SIFT algorithm, these vectors are going to be inputted into a python script that uses Apache-Spark library MlLib and OpenCV. This will cluster feature vectors by their similar properties using the K-Means Algorithm. Before performing the actual clustering of the feature vectors, the k-means model must be trained with the existing feature vectors. To begin an Apache-Spark context is created.

This script has the following input parameters, the number of centroids, the path where the file containing the feature vectors is located, and the path where the results of the script will be outputted. The script reads the file containing the feature vectors of all the images, flattens the data frames, and returns a new resilient distributed dataset.

This means that every item, image, on the file will transform from a 128-dimensional array to one-dimensional vectors of 128 in length. This new resilient data set is given as input to the k-means training model which in turn returns the clusters and final centroids in a plain text file. Another Apache-Spark application then takes the k-means model dictionary and proceeds to encode it to a single cluster. The process is performed by taking every row of the original 128-dimensional array and assigning it to a single cluster of the k-means dictionary.

CONCLUSION

In conclusion, computer vision techniques and processes have been integrated with the use of the Hadoop environment to achieve a scalable algorithm capable of performing parallel tasks to manage Big Data applications such as image classification. This approach has been based on the Vector for Locally Aggregated Descriptors (VLAD) technique to generate an algorithm capable of recognizing image features. With the use of clustering algorithms like K-means through the Hadoop MapReduce function, the system can generate a dictionary of features or Bag of Visual Words (BoW) and then classify images based on this trained set for desired features.

The use of the Hadoop environment for this type of application over the years has been developed with the creation of analytics engines such as Apache Spark. This robust engine has simplified the process for users to develop algorithms without the need to learn the complex parallel programming behind a classical MapReduce function.

Having a programming base such as Python makes the proposed algorithm an accessible means for implementation, given that all tools used for this design are considered open source. Making this type of solution to even be distributable for commercial use. Given the hardware limitations required for a Hadoop application of this nature, applying it on a small scale was not possible during the time frame of this investigation. Since this environment platform performs efficiently through a dedicated networking system, proportional to the number of datanodes associated and dedicated to such type of analysis.

FUTURE WORK

The proposed solution to the problem should be implemented on a server that has Hadoop installed in fully distributed mode. The server should have at the bare minimum 6 datanodes to at least 10 datanodes with a principal namenode. Also, the solution could be implemented using cloud services like the ones offered by Amazon Web Services (AWS). AWS is a service that allows users to process and study large datasets using all the latest versions of big data frameworks. Hadoop installed in standalone mode does not have the memory or processing requirements necessary to perform the computations needed for content-based image recognition. Additionally, to complete the process of content-based image recognition the inclusion of a supervised algorithm for classifying the images. The process of classification is to label each image according to the feature vector for the main purpose of querying or fetching an image from a large dataset which includes a feature vector that represents the object.

REFERENCES

[1] N. Kumar, "Content Based Image Retrieval for Big Visual Data using Map Reduce - raiith", Raiith.iith.ac.in, 2015. [Online]. Available: https://raiith.iith.ac.in/1609/

[2] C. Reggiani, "Scaling feature selection algorithms using MapReduce on Apache Hadoop", Claudioreggiani.com, 2013. [Online]. Available: http://claudioreggiani.com/pdf/masterthesis.pdf

[3] T. White., Hadoop: The Definitive Guide, Second Edition. O'Reilly Media, Inc., 2010.

[4] K. Potisepp, "Large-scale Image Processing Using MapReduce", semanticscholar, 2013. [Online]. Available: https://www.semanticscholar.org/paper/Large-scale-Image-Processing-Using-MapReduce-Potisepp/bb7ce436cc9e2b1c4c64516ae9f600dc517b7353

[5] T. El-Sayed, A. El-Sayed and M. Badawy, "Impact of Small Files on Hadoop Performance: Literature Survey and Open Points", researchgate, 2019. [Online]. Available: https://www.researchgate.net/publication/337677872_Impact_of_Small_Files_on_Hadoop_Performance_Literature_Survey_and_Open_Points#:~:text=Hadoop%20performs%20well%20with%20files,of%20the%20MapReduce%20applications%20

[6] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", Cs.ubc.ca, 2004. [Online]. Available: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[7] W. Murphy, "Large Scale Hierarchical K-Means Based Image Retrieval With MapReduce", AFIT Scholar, 2014. [Online]. Available: https://scholar.afit.edu/etd/616/

[8] S. Vemula and C. Crick, "Hadoop Image Processing Framework", ieeexplore, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7207264

[9] T. Habib and Z. Ansari, "An Analysis of MapReduce Efficiency in Document Clustering using Parallel K-Means Algorithm", researchgate, 2018. [Online]. Available: https://www.researchgate.net/publication/325208173_An_Analysis_of_MapReduce_Efficiency_in_Document_Clustering_using_Parallel_K-Means_Algorithm

[10] B. Chambers and M. Zaharia, Spark: The Definitive Guide: Big Data Processing Made Simple, 1st ed. Seoul: Hanbit Midieo, 2018.