COMPARISON OF LINEAR AND NON-LINEAR FEATURE EXTRACTION ON VEGETATION AND OIL SPILL HYPERSPECTRAL IMAGES

A Thesis

by

ANDRES RAMIREZ-AGUILAR

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Texas A&M University - Corpus Christi

Corpus Christi, Texas

August 2015

Major Subject: Computer Science

COMPARISON OF LINEAR AND NON-LINEAR FEATURE EXTRACTION ON VEGETATION AND OIL SPILL HYPERSPECTRAL IMAGES

A Thesis

by

ANDRES RAMIREZ-AGUILAR

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Computer Science Graduate Program
School of Engineering & Computing Sciences
Texas A&M University - Corpus Christi

Approved by:

Maryam Rahnemoonfar, Committee Chair	Scott A. King, Committee Member
Li Longzhuang, Committee Member	

August 2015

ABSTRACT

A hyperspectral image provides a multidimensional figure rich in data consisting of hundreds of spectral dimensions. For this research, the method of analysis for a hyperspectral image will consist of two different feature extraction algorithms: principal component analysis locally linear embedding. Analyzing the spectral and spatial information of such image with linear and non-linear algorithms will result in high computational time. In order to overcome this problem, this research proposes a system using a *MapReduce*-Graphics Processing Unit (GPU) model that can help analyze a hyperspectral image through the usage of parallel hardware and a parallel programming model, which will be simpler to handle compared to other low level parallel programming models. Additionally, Hadoop will be used as an open-source version of the *MapReduce* parallel programming model. The ultimate goal of this research is to provide a foundation for a simple and powerful system that is scalable and easily extend-able.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

Acronyms

ECHO extraction and classification of homogeneous objects. 58, 59

GPU graphics processing unit. 12, 38, 39

HDFS Hadoop distributed file system. 39, 44

K-NN K-Nearest Neighbors. 49

LLE locally linear embedding. 3, 38, 44, 49, 54

ML maximum likelihood. 59

MVN multi-variate normal. 58

PCA principal component analysis. 3, 38, 44, 48, 49, 54

CHAPTER 1

INTRODUCTION

1.1 Motivation

A general term known as the *curse of dimensionality* refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces. In this case the data refers to hyperspectral images and the dimensions can range between (50 to 500 bands) 5 to 10 nm spectral bandwidths[34]. Therefore, many different problems arise when using these many dimensions in the domain of image classification. As we keep adding more dimensions to the dataset, the dimensionality of the feature space also grows and the space would also become sparse. The problem would not be to find a separable hyperplane between the different classes of an image with a high number of dimensions, the problem lies when we attempt to project the highly dimensional classification result back to a lower dimensional space. This idea refers to overfitting[10] which is a direct result of the curse of dimensionality. Therefore, the performance of a classifier decreases when the dimensionality of the problem becomes too large. The curse of dimensionality when classifying hyperspectral data is the main motivator for this research.

1.2 Problem Domain

The difficulty of using hyperspectral data is that they consist of redundant and strongly correlated data which leads to one problem when classifying the data[27].

The classification accuracy of hyperspectral images increases gradually in the beginning as the number of spectral bands increases, but accuracy decreases dramatically when the number of bands reaches a critical value. Additionally, a second problem that arises with the classification of hyperspectral data is that as the number of bands increases, then more training samples are required for each class in order to obtain accurate classification of objects. The second problem is more prominent especially when a class in the HSI data is fairly small. If the pixel size of the class is much less than the number of dimensions in the hyperspectral image, then it is nearly impossible to correctly detect small classes due to the lack of training samples. Therefore, we need to reduce the number of dimensions of HSI data in order to improve classification accuracy even when the class size is small. The two solutions to reduce dimensions are principal component analysis (PCA) and locally linear embedding (LLE). PCA refers to linear feature extraction and LLE refers to non-linear feature extraction. We investigate and compare results obtained by PCA and LLE in order to detect which feature extraction algorithm works best under certain circumstances.

1.3 Contribution

My contribution consists of two distinct parts. The first part is the creation of two programs, an implementation of PCA and LLE, and their ability to perform faster than their non-optimized approach with the use of Lanczos and Arnoldi's iterative algorithms to obtain eigenvectors of the smallest and highest magnitude. The second part is a thorough comparison of PCA and LLE involving the time it took to run the feature extraction classification accuracy by implementing programs that run on

the three following systems: Hadoop and GPU, GPU and CPU.

1.4 Literature Review

1.4.1 MapReduce

MapReduce can be considered a programming model and a framework. The main purpose of *MapReduce* is to allow users to focus solely on data processing strategies and to abstract the details of parallel execution. The two primitive functions that this model uses are *Map* and *Reduce*[30]. The main input for *MapReduce* is a list of key-value pairs and the Map function is applied to each pair to compute intermediate key-value pairs. The intermediate key-value pairs are then grouped together on the key-equality basis. For each key, within the grouped keys, the function Reduce works on the list of all values, then produces zero or more aggregated results.

Furthermore, the *MapReduce* model is utilized in Google and communicates with the Google File System (GFS) as an underlying storage layer to read input and store output[20]. Apache Hadoop is an open-source Java implementation of *MapReduce*, so we will continue with the explanation with Hadoop since the GFS is not open to the public for its proprietary use[30]. Hadoop is a framework that is used both commercially and within research. It attempts to solve efficiency problems by running tasks in parallel. A reason that Hadoop is becoming popular deals with the use of commodity hardware which means that Hadoop can easily run on inexpensive hardware and allows for high scalability. Also note that Phoenix is another example of the state-of-the-art *MapReduce* framework on multi-core CPUs[41]. Furthermore

there are successful versions of *MapReduce* that can run on multiple clusters, on multi-core CPU[30] and even on Graphical Processing Units (GPU)[17, 5]. Amazon has also launched Elastic *MapReduce* (Amazon EMR) which enables Hadoop on its cloud servers with GPU's, which shows the advancement of *MapReduce* within industry.

GPU's have been utilized as graphic processing units with the primary intention of rendering images close to realism and have transitioned to general purpose engines for high throughput floating-point computation. Due to the massively parallel architecture that GPU's provide, it has also been utilized in other domains such as high-performance computing. It is difficult to implement a program for the GPU because global communication and synchronization is complicated, error prone and specific to the exact GPU which is being targeted. A successful implementation of *MapReduce* on a GPU is called Mars[17] where it reached a performance gain of 1.5x-16x when analyzing Web data (search/logs) and processing Web documents (including matrix multiplication). Other examples that implemented *MapReduce* on GPUs are: Grex[5], GPMR[50], StreamMR[16], and MITHRA[18] . However most of them are no longer supported and were built for a particular scientific project.

1.4.1.1 Parallel Processing

HSI is important for oil spill detection in order to obtain an estimate of the amount of oil that should be collected to reduce the amount of damage to the environment. One particular research[39] implemented oil spill detection with parallel techniques by using the following analysis approaches: clustering, classification and spectral

mixture analysis on hyperspectral images. The computer systems were homogeneous and heterogeneous parallel systems at NASA's Goddard Space Flight Center and in the University of Maryland. They applied computing on heterogeneous networks of workstations (NOWs) since they provide an economical alternative for parallel computation. In the end, the project was able to effectively create parallel algorithms on a heterogeneous NOW since they were compared to their homogenous counterparts which is done by applying load balancing. This is important because heterogeneous NOW's are more cost effective then their homogenous counterparts.

Additionally, Bernabe et al. have worked on target and anomaly detection for HSI analysis implemented on GPU[7]. They implement parallel versions of automatic target detection and classification algorithm (ATDCA) and Reed and Xiaoli (RX). The ATDCA finds a set of spectrally distinct target pixels vectors using the concept of orthogonal subspace projection in the spectral domain. The RX algorithm is based on the application of a so-called RXD filter given by the well known Mahalanobis distance. They concluded that GPU hardware devices may offer important advantages in defense and security applications that demand a response in real-time.

Paz et al[38] investigated work between a cluster and GPU for HSI with a focus on orthogonal target detection algorithms.The parallel implementation of the algorithms were tested in two types of parallel computing architectures: a massively parallel cluster, and a commodity GPU. Their final assessment stated that commodity clusters represent a source of computational power that is accessible and applicable to obtain results quickly for target detection.

1.4.1.2 Oil Spill Detection

Plaza et al.[40] present a novel automated hyperspectral target detection technique for determining the level of oil contamination, of polluted areas in the shoreline, by using spatial and spectral information of polluted areas in the shoreline. The method used was mathematical morphology operations. Additionally, the data set used dealt with real hyperspectral data and simulated data. Here they extended morphological operations to hyperspectral image data such as erosion and dilation. A standoff detection system by applying morphology was used which is named Automated Determination of Morphological Profiles (ADMP). A drawback of the proposed approach concerns the necessity of looking at a range of increasing opening and closing by reconstruction operations, which may result in a heavy computational burden when processing high dimensional data. The usage of a parallel implementation is being tested on low-cost cluster architecture.

Alam et al. [4] investigate and compare the performance of the five most widely used target detection algorithms for the identification and tracking of surface and subsurface oil spills in ocean environment. Algorithms that were used are: Spectral information divergence (SID), Adaptive coherence estimator (ACE), constrained energy minimization (CEM), spectral fringe adjusted joint transform correlator (SFJTC), and Reed-Xiaoli Uniform Target Detector (RX-UTD) Anomaly detection. Techniques that gave the best results are CEM and SFJTC, while some techniques such as SID contain misclassification's and ACE contains large amount of noise. Also, SID and ACE fail to detect one test region even though they present good result

for an alternate test region. Overall SFJTC and the CEM partial detected the test target regions.

Kokaly et al. [29] propose a spectroscopic approach to detect oil which was applied to AVIRIS data collected over the marshes in southern Louisiana. Their novel detection method is a spectral feature analysis of two hydrocarbon absorption features. They used Material Identification and Characterization Algorithm (MICA) which is a module within the interactive data language(IDL) program. MICA is a method of identifying and mapping materials using spectral-feature based analysis of reflectance data in an expert-system framework. Overall the accuracy assessment of this paper showed that a spectroscopic approach to remote sensing was a useful method for delineating oil contamination in marshes impacted during the Deepwater Horizon oilspill. Spectral feature analysis applied to the high spatial resolution AVIRIS data were shown to be very accurate in detecting the hydrocarbon absorption features of oil and in mapping oiled marsh.

Roger et al.[12] present a method to derive oil thickness and the oil to water ratio from remotely sensed spectral data of near-infrared (NIR) features by applying a NIR spectroscopy. The data used was collected with the NASA AVIRIS sensor obtaining information from the Deepwater Horizon spill. In the end, since they demonstrated that NIR spectroscopy can probe several millimeters into oil and that the spectral properties can be used to obtain: oil to water ratio, oil thickness up to the optically thick limit (which varies with the oil to water ratio), and the sub-pixel areal fraction of oil resulting in the total volume of oil for any given area.

Sykas et al. [52] presents a new method for estimating oil spill thickness by using

abundance fractions provided by spectral unmixing methods. The unmixing methods used are: fully constrained network based method (F-NBM), fully constrained least square method (FCLS), orthogonal subspace projection and constrained energy minimization (CEM). In the end, the first unmixing method algorithm, F-NBM, gave the best results for estimating oil spill thickness.

1.4.1.3 Agriculture and Vegetation

Thenkabail et al.[56] investigated the best hyperspectral wavebands in the study of vegetation and agricultural crops between the spectral range 400-2500 nm. They applied principal component analysis (PCA), lambda-lambda R2 models (LL R2M), stepwise discriminant analysis (SDA), and derivative greenness vegetation indices (DGVI) to concluded that 22 optimal bands best characterize and classify vegetation and agricultural crops. Using more than 22 bands would only increase accuracy marginally up to 30 bands. Beyond 30 bands, accuracies would become asymptotic or near zero.

Additionally, Stefan et al.[43] did an investigation in search of the best band selection and assesed their work with a multicore environment and MPI. They developed a parallel best band selection (PBBS) algorithm which distributes the data to all available nodes and then computing k equally sized intervals of the search space; the investigation was also implemented on a high performance cluster. Their experimental results show that PBBS resulted in an optimal solution to the feature selection problem.

1.4.2 Feature Extraction

1.4.2.1 Linear

Furthermore, several researchers have used PCA[55, 25, 59]. Overall, the use of PCA results in high computational analysis time and can inaccurately represent non-linear data which can reside within a hyperspectral image. Even though PCA is a time consuming algorithm, researchers have attempted to improve the time complexity by creating algorithms such as folded or segmented PCA[59] resulting in a 10% efficiency increase.

Regardless of the PCA performance stated, it does contain advantages to manifold learning. Due to the extraction of linear patterns, the amount taken to compute the principal components is much less than manifold learning methods. Additionally, the amount of computer memory required to handle the computation of PCA is much less than the allocation of memory that manifold learning demands.

1.4.2.2 Non-linear

Additionally, the non-linear implementations of feature extraction are manifold learning techniques. There are both local and global manifold learning techniques where the former tends to be significantly faster compared to the latter[27, 32, 11, 57, 44]. Global manifold learning techniques include:multidimensional scaling, maximum variance unfolding, and isometric mapping (ISOMAP), Kernel-PCA (KPCA). Localized manifold learning include: Laplacian Eigenmaps (LE), Local Linear Embedding (LLE) and Local Tangent Space Alignment (LTSA).

In the dissertation presented by K. Wonkook [27] manifold learning was used in order to robustly classify hyperspectral images. This dissertation focused on the global manifold techniques and compared the results to local manifold techniques, tested by another researcher. It was noted that global manifold learning methods capture the overall structure of the data with fewer bands than local manifold learning. Typically, when dealing with complex geometry, local manifold represent the data better than global manifold methods. Additionally, Isomap generally outperforms KPCA and KPCA offers little advantage relative to PCA. Among the local manifold learning techniques, LLE and LTSA consistently yield better results than LE, while Isomap outperformed LE.

CHAPTER 2

BACKGROUND

The following topics have been mentioned within the literature review but will now be described in more detail in order to obtain a much richer knowledge over the system design implemented for this research. This chapter will go over the MapReduce system named Hadoop, followed by a general overview of the graphics processing unit (GPU), hyperspectral data imaging, linear and non-linear feature extraction along with classification algorithms.

2.1 Hyperspectral Imaging (HSI)

Hyperspectral imaging is an emerging and fast-growing area in remote sensing. This area is concerned with the measurement, analysis, and interpretation of spectra acquired from a given scene, or object, at a short, medium, or long distance by an airborne or satellite sensor. The sensors are able to obtain an abundance of spectral and spatial information, and hyperspectral images can become uncomfortably large, which introduces new processing challenges. To illustrate the amount of information that can be obtained consider NASA's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) which is an instrument for Earth Remote Sensing flown aboard various aircraft types. It is an optical and infrared sensor that delivers calibrated spectral channels with wavelengths from 360 to 2,500 nanometers using 224 spectral bands with a spatial resolution of 20 meters at an altitude of 20 kilometers[36]. AVIRIS can also acquire images from a low-altitude aircraft with spatial resolutions ranging from

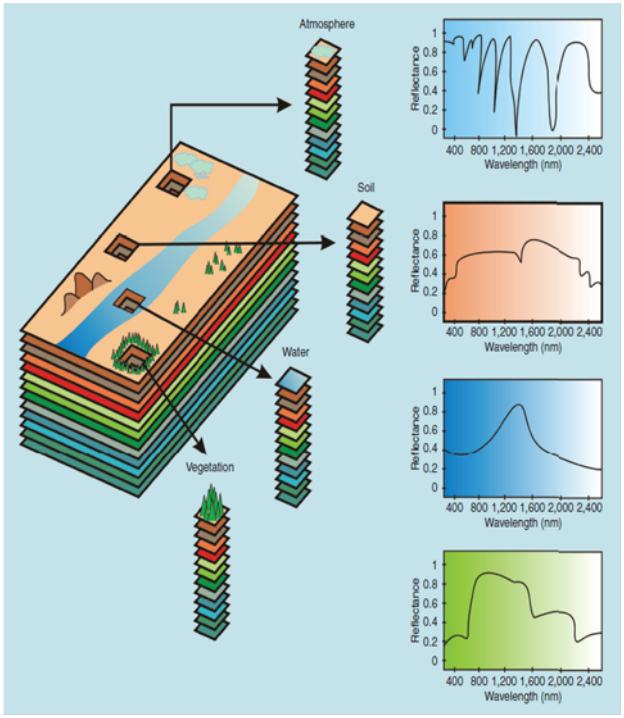


Figure 1.: Bands of a Hyperspectral Image at Different Wavelengths.

1-4 meters. The produced image consists of two spatial and one spectral dimension as noted in figure 1, where each visible layer represents one particular band ranging between 360-2,500nm. Furthermore, each band has a different reflectance value as shown in the reflectance vs. wavelength graphs.

2.2 Apache Hadoop

The MapReduce paradigm is widely used in academia, research and industrial facilities[54, 58]; for this research we use Apache Hadoop MapReduce since it is an open source implementation of MapReduce. Apache Hadoop allows for distributed processing of large data sets across clusters of computer through the use of simple programming

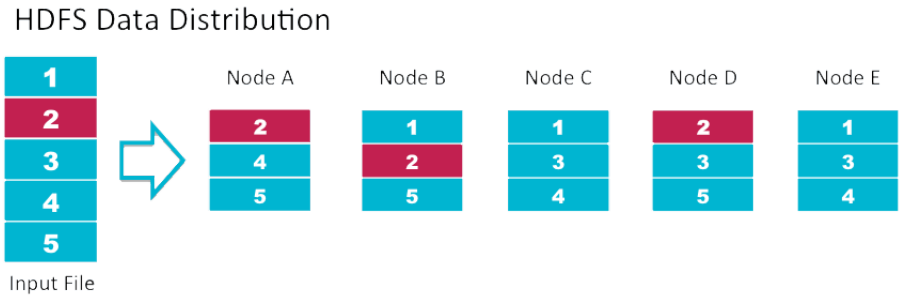


Figure 2.: HDFS Data Distribution.

models[54].

In order to have files within Hadoop, they have to be stored within the Hadoop Distributed File system (HDFS). The HDFS is very similar to a regular file system that can be seen on a standard machine. Whenever we store files into the HDFS, the file is split into pieces called chunks, or blocks, among the total number of nodes in a cluster. The default size of each chunk is 64MB but can be changed when initially setting up Hadoop on a cluster or in a single node machine. In Figure 2 we see an illustration of how data are stored in the HDFS. In this simple example, we first divide the entire input file into 5 different blocks and they are distributed among 5 nodes (node A through E). Each node within the cluster is also running a daemon called the datanode, which runs as a background process.

Clearly we have to know which blocks make up the original file, which is handled by a different machine that is running another daemon called the namenode and the information stored on the namenode is called the metadata. This typical setup is known as the master slave architecture where the slaves are told specific instructions by the master node. An example of this architecture can be seen in Figure 3.

There are three main issues that arise when storing data within the HDFS. The

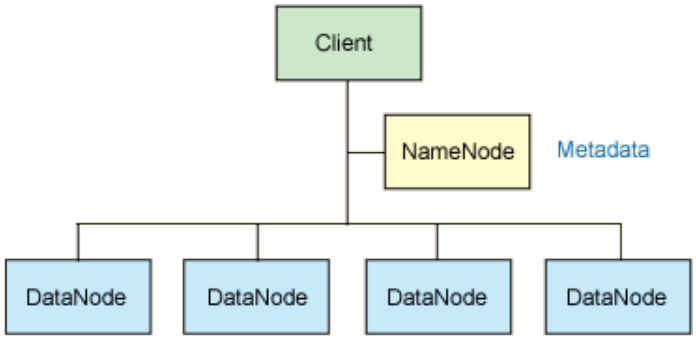


Figure 3.: HDFS Datanode and Namenode Representation.

first problem that can occur is if there is a network failure between the nodes which would not allow the nodes to talk to one another. The second problem that can arise is if a disk failure occurs on a datanode which would lead to losing all the data stored. The third issue that can occur is if there is a disk failure on the namenode, which leads to a single point of failure. If there is an issue with the namenode, then the data will be inaccessible and the data can be lost forever. In order to overcome the second issue, Hadoop implements data redundancy by replicating data blocks 3 times, illustrated in Figure 2. In order to avoid the single point of failure previously stated, the first alternative is to store the metadata from namenode on the network file system (NFS). A second alternative to avoid the single point of failure is to configure two namenodes, the active namenode and the standby namenode. The active namenode would work normally while the standby namenode would take over in the case that the active namenode fails.

After storing data within HDFS, then MapReduce is used in order to process the data. When dealing with a file, the HDFS will break it down into default blocks

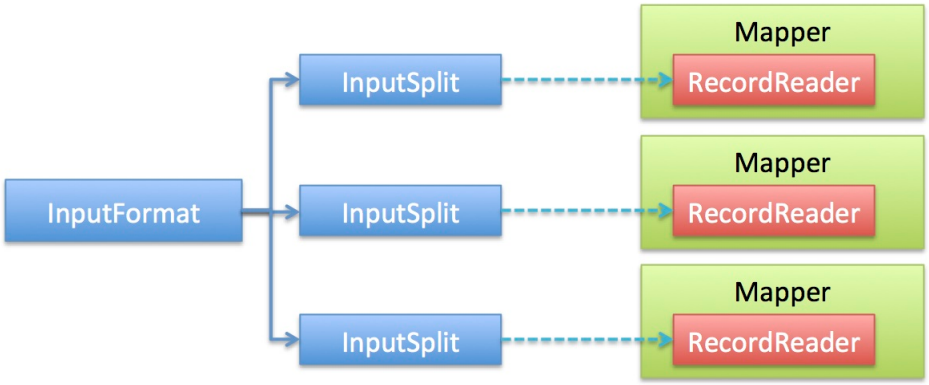


Figure 4.: Inputsplit Illustration.

of 64MB which means that the light data files are broken into chunks of exactly 64 MB. Since the file blocks are exactly 64MB and because HDFS does not know what is inside the file blocks, it does not recognize when a record might spill over into another block. This issue is solved by using a logical representation of data known as input splits. When a MapReduce job client calculates the input splits, it obtains the starting point of the first whole record and where the last record in the block ends. Furthermore, the number of input splits that are calculated for a specific application determines the number of mapper tasks as illustrated in Figure 4. Then each mapper task is assigned to a slave node where the input split is stored. The Jobtracker does its best to ensure that input splits are processed locally.

After calculating the input splits and allocating the appropriate number of mappers, each mapper outputs intermediate records in the (key,value) pair format. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. When the

mappers finish, the next phase involves shuffling and sorting the data. The shuffle portion is the moving of intermediate record from the mappers to the reduces. The sorting part is the fact that the reduces will organize the set of records into sorted order. Each reducer will always get one key and the list of all the values pertaining to that particular key. Overall reducer implementations are passed to JobConf for the job and can override it to initialize themselves. The reduce has 3 primary phases: shuffle, sort and reduce. Additionally, increasing the number of reduces increases the framework overhead but increases load balancing and lowers the cost of failures.

2.3 Graphics Processing Unit (GPU)

With the help of the Compute Unified Device Architecture (CUDA) we are able to control the CPU and the GPU with one program in order to harness the power provided by the GPU. When dealing with CPU and GPU programming, the CPU is referred to as the "host" and the GPU is referred to as the "device". CUDA assumes that the device is a coprocessor to the host and that the host and device have their own separate memories where they store data. The CPU will always be the main processor that tells the GPU which commands to execute. Some responsibilities of the CPU involve: copying data from the CPU to the GPU, copying the data back from the GPU to the CPU, allocating memory on the GPU, and launching kernels on the GPU.

A kernel appears as a serial program, and each kernel has to be written as if it will run on a single thread. When the time comes to launch a kernel, the developer should define the total number of threads that will execute the kernel.

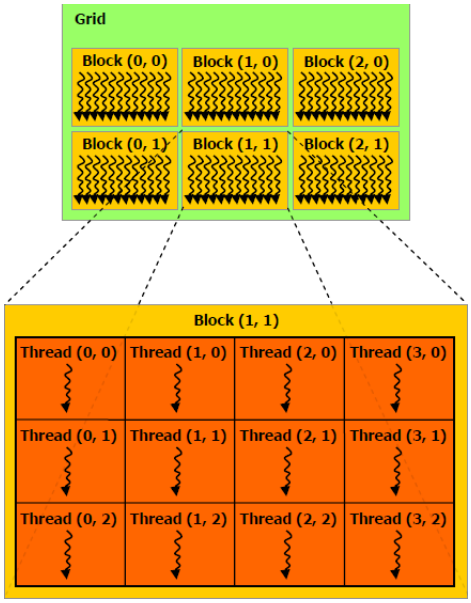


Figure 5.: A grid of threadblocks.

Apart from having a high level knowledge of how the CUDA model works, it is also important to have a basic understanding of how the GPU hardware is setup. This is necessary because when a developer launches kernels, they have to define the number of blocks and the number of threads per block that the GPU will execute. In Figure 5, we see how blocks are organized within the GPU. Each block also has a finite number of threads that can execute, and is illustrated in Figure 5. Upon the execution of the parallel program, each thread knows its own index within the threadblock and in the grid of threadblocks. Depending on the compute capability of the GPU, it will have a default number of blocks and threads per blocks available. For example, a GeForce GTX 660M GPU has a 3.0 compute capability. Therefore, the maximum number of resident blocks per multiprocessor is 16 and the maximum number of threads per block is 1024. If the computer capability of the GPU is much

older, like a 2.0 compute capability, it has a default of 8 maximum number of resident blocks per multiprocessor with 512 threads per block. Also, the number of blocks and threads can be specified as 1, 2 or 3 dimensions. Additionally, a developer can also specify the total amount of shared memory in bytes allocated per thread block.

Next we will go over dimensional reduction algorithms, feature extraction, that extract linear and non-linear data from hyperspectral images.

2.4 Feature Extraction

Feature extraction or dimension reduction is used to mitigate the issues of high interband spectral correlation and to overcome Hughes phenomenon for the classification of hyperspectral data as stated in the motivation and problem domain of this research. After the reduction, the results require reduced data storage and computation and can potentially produce more robust and accurate classifer. In order to reduce dimensions, two known approaches are linear and non-linear methods.

Non-linear extraction methods, namely manifold learning, have been investigated in order to extract the nonlinear characteristics embedded within hyperspectral data. The linear dimensional reduction algorithms include Principal Component Analysis (PCA), Singular Value Decomposition (SVD) and Independent Component Analysis (ICA). Even though linear dimensionality reduction is useful, it is sometimes not enough to extract the linear variance among the data because the data can potentially have a non-linear variance[27]. An example of non-linear variance will be explained under the Isometric Feature Mapping sub-section.

2.4.1 Principal Component Analysis

PCA is a statistical technique used to emphasize variation and extract strong patterns in a dataset that are linearly correlated. The number of principal components is less than or equal to the number of original variables. Once the transformation is to be implemented, the first principal component has the largest possible variance and each succeeding component has the highest variance possible if it is orthogonal to the preceding components. A descriptive mathematical derivation of PCA will be given in chapter 3 under the system design.

Additionally, the extraction of linear variance is sometimes not enough when dealing with hyperspectral data since it the correlation can be non-linear. The extraction of non-linear variance can be achieved through different manifold learning methods.

2.4.2 Global Manifold Learning

Manifold learning can be classified as global or local approaches in order to extract non-linear variance from high dimensional datasets. The global manifold methods simply keep the fidelity of the overall data set but unfortunately have a greater computational overhead for large data sets, while local methods preserve local geometry and are computationally efficient because they simply require sparse matrix computations as opposed to dense matrix computations.

Three global manifold learning methods that will be reviewed are: Kernel Principal Component Analysis (KPCA), Isometric Feature Mapping (Isomap), and

Landmark-Isomap (L-Isomap). Additionally, three local manifold learning methods that will be reviewed are: Locally Linear Embedding (LLE), Local Tangent Space Alignment (LTSA) and Laplacian Eigenmaps (LE).

The local manifold learning methods seem to outperform the global manifold algorithms for some particular cases[27], and the Isomap algorithm generally outperforms the KPCA method[27]. Although the KPCA method seems to be a fairly low performance algorithm, certain optimizations can potentially help improve the performance.

2.4.2.1 Kernel Principal Component Analysis (KPCA)

KPCA is a nonlinear extension of linear PCA in a feature space induced by a kernel function[47]. As PCA performs a linear separation of the data in the original space, KPCA embeds the data into a high dimensional space called the feature space $\mathscr{H}$ and the input space is denoted as $\mathbb{R}^n$. We also have a mapping function denoted as Φ:

$$\begin{aligned} \Phi : \mathbb{R}^n &\longrightarrow \mathscr{H} \\ \mathbf{x} &\longrightarrow \Phi(x) \end{aligned} \tag{2.1}$$

where Φ can be non-linear. If we want to perform PCA in $\mathscr{H}$ with a mapping function Φ we do not need to know the explicit mapping function if the "kernel trick" is applied. One advantage about KPCA is that it does not require the projection of the data points into the higher dimension, and instead we simply apply the "kernel

trick "[27] on the data points within the input space. KPCA solves the following eigenvalue problem:

$$\lambda\alpha = K\alpha, \quad \text{subject to } ||\alpha||_2 = \frac{1}{\lambda} \tag{2.2}$$

Where K is the kernel matrix constructed as follows:

$$\mathbf{K} = \begin{pmatrix} k(x^1, x^1) & \dots & k(x^1, x^\ell) \\ k(x^2, x^1) & \dots & k(x^2, x^\ell) \\ \vdots & \ddots & \vdots \\ k(x^\ell, x^1) & \dots & k(x^\ell, x^\ell) \end{pmatrix} \tag{2.3}$$

The function k is referred to a positive semidefinite function, or the *kernel* on $\mathbb{R}^n$ which so happens to introduce non-linearity into the process. The k function is the core of KPCA. For instance, going back to PCA, we can apply a linear kernel that would produce the principal components with the following linear kernel (c is an optional constant):

$$\mathbf{k(x,y)} = x^T y + c \tag{2.4}$$

Other classic kernels[19] are the polynomial kernel, $q \in \mathbb{R}^+$ and $p \in \mathbb{N}^+$

$$\mathbf{k(x,y)} = (\langle x, y\rangle_{\mathbb{R}} + q)^P \tag{2.5}$$

and the Gaussian kernel, $\sigma \in \mathbb{R}^+$

$$\mathbf{k(x,y)} = exp\left(-\frac{||x-y||^2}{2\sigma^2}\right) \tag{2.6}$$

The application of an appropriate kernel is a non-trivial task[19].The use of a polynomial kernel would be appropriate if higher-order statistics are relevant to discriminate samples[19]. The usage of the Gaussian kernel would be applied under that Gaussian cluster assumption. Hyperspectral remote sensing data are known to be well approximated by a Gaussian distribution[42].

2.4.2.2 Isometric Feature Mapping

Isomap assumes that the local feature space formed by the nearest neighbors are linear, and the global nonlinear transformation can be found by connecting the piecewise linear space[53]. Additionally, Isomap uses the core principles used in the multidimensional scaling (MDS) algorithm. MDS is a dimensionality reduction tehcnique that places a set of samples in a lower dimensional space that exlains the similarity between samples. This approach works by preserving distances between points in the data which results in a spatial arrangement that preserves distances. MDS is implemented by first obtaining the distances between points in a dataset by creating a distance matrix. The distance matrix is a matrix of inner products. Afterwards, an eigen decomposition of the distance matrix returns the lower dimension embedding. The only difference between ISOMAP and MDS is the construction of the distance matrix. MDS obtains the distance between two points by using the Euclidean distance. In ISOMAP, the distances between the points are the weight of the shortest

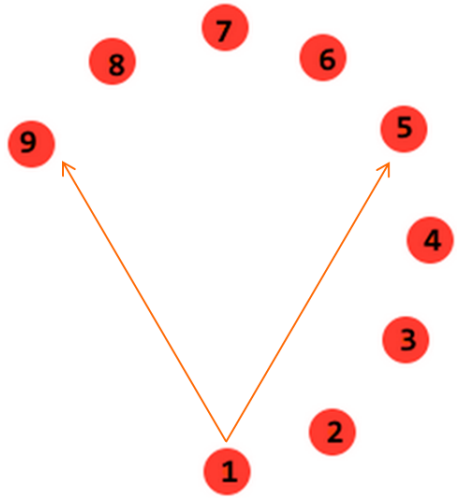


Figure 6.: Distances between data points with Euclidean distance.

path in an undirected graph. The graph is created by placing an edge between two sample points if the euclidean distance between them falls between a point and its top K-nearest neighbors (K-NN). This concept is very useful because it allows us to obtain certain manifold within the data that could not be obtained if euclidean distances were measure between samples. To illustrate this point, refer to Figure 6.

If we obtain the Euclidean distance between point 1 and 5 and between point 1 and 9, we would obtain the same distances. Our main goal is to capture the curved nature of the data points, which cannot be accomplished by using the Euclidean distance. By creating a graph K-NN, in this case $K = 1$, we can obtain a much better representation of the manifold within the dataset illustrated in Figure 7.

Furthermore, in order to compute the weights on the shortest paths is to use Dijkstra's shortest path algorithm[53, 49, 51] or the Floyd Warshall algorithm. Furthermore, the distance matrix constructed under the Isomap algorithm captures the

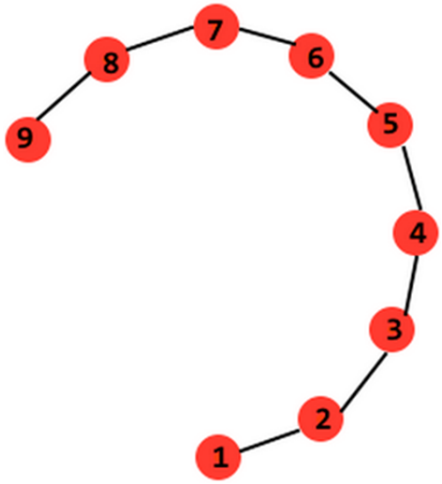


Figure 7.: Distances between data points with K-NN.

underlying manifold more accurately than the distance matrix constructed using Euclidian distances. The downside to Isomap is that the shortest path algorithm is computationally demanding which could be mitigated by using a divide and conquer algorithm or by using landmarks (L-Isomap) with embedding of non-landmark points via the derived embedding vectors reducing the complexity of computing the geodesic distance matrix.

2.4.2.3 Landmark Isometric Feature Mapping

As previously stated, ISOMAP is a computationally demanding algorithm. Essentially there are two computational bottlenecks[49]. The first is calculating the NxN distance matrix D_N where N is the number of samples. The application of Floyd Warshall's algorithm returns a complexity of $O(N^3)$. We can improve the complexity returned by Floyd Warshall's with the application of Dijkstra's algorithm with

Fibonacci heaps to $(KN^2 \log N)$ where K is the neighborhood size[49]. The second bottleneck resides in the MDS eigenvalue calculation which involves an NxN matrix with a complexity of $O(N^3)$.

Therefore, L-ISOMAP addresses both bottlenecks by first designating n data points to be "landmarks" where $n << N$ (n is much less than N). Then instead of computing D_n, we compute a new distance matrix of size nxN ($D_{n,N}$). Next, Landmark MDS (L-MDS) would be applied in order to find a Euclidean embedding of the data by using D_n, N instead of D_N. By applying Dijkstra's algorithm when n is significantly smaller than N, we can run Dijkstra's algorithm in $O(KnN \log N)$ and run L-MDS in $O(n^2N)$.

Even though L-ISOMAP is a promising improvement on the standard ISOMAP algorithm, due to irregular distribution of pixel points in spectral space, the usual random selected landmarks do not perform well within hyperspectral image data[51]. One method to improve this problem is to apply Vector Qunatization to mitigate the landmark selection problem and icrease the classification accuracy[51].

Even though global manifold learning can be changed to become more computationally efficient, there are also local manifold learning algorithms that tend to be computationally less intensive than global manifold learning method.

2.4.3 Local Manifold Learning

The initial step of the local manifold algorithms (LLE, LTSA, LE) include the construction of the K-NN's for each data point, and the local structures are then used to obtain a global manifold. Three different local manifold learning methods were

previously stated. After constructing the nearest neighborhood, LLE would then obtain the local properties of each neighborhood by the linear coefficients that can best reconstruct each data point from its neighbors[44]. A more descriptive mathematical derivation of LLE will be given in chapter 3 under the system design. With the LTSA, the geometry in the local region is described by the local tangent space of each data point, and then the global manifold is determined by aligning the overlapping local tangent space. Lastly, the LE algorithm constructs a weighted neighbor graph of each data point by calculating the pair wise distances between neighbors. Typically the distance is calculated using a Gaussian kernel function with parameter σ.

2.4.3.1 Local Tangent Space Alignment (LTSA)

In LTSA[60, 27], the local geometry is described by the local tangent space of each data point, and by overlapping local spaces we obtain a global manifold. Let X_i be the K-NN's of point x_i, Θ_i be the local tangent coordinates with a dimensionality value of p, H is the centering matrix, E is the reconstruction matrix, T is the transformation matrix and let Y_i be the global coordinates related to Θ_i with the following affine transformation:

$$Y_i H = T_i \Theta_i + E_i \tag{2.7}$$

where $T_i \in \mathbb{R}^{p \text{x} p'}$, $H \in \mathbb{R}^{k \text{x} k}$ and $E_i \in \mathbb{R}^{p \text{x} k}$. The value p is the target dimension of manifold coordinates, the value p' is the dimension of the local tangent space and

k is the number of neighborhood samples. Then we can obtain the local geometry in the embedded space by minimizing E_i according to the following expression:

$$\Theta(Y) = \sum_{i=1}^{n} (\|Y_i U_i\|_F^2) \tag{2.8}$$

such that $YY^T = I$ and where $U_i = H(I - \Theta_i^T(\Theta_i \Theta_i^T)^{-1}\Theta_i)$. Then we can obtain the embedding of Y by solving:

$$\underset{YBY^T=I}{\operatorname{argmin}}\, tr(YLY^T) \tag{2.9}$$

where I is the identity matrix, B is the constraint matrix and L is the alignment matrix. Furthermore, *argmin* stands for argument of the minimum where we have points $YBY^T = I$ for which $tr(YLY^T)$ attains its smallest values and the tr function represents the trace operator defined to be the sum of the elements on the main diagonal of the given matrix, in this case matrix YLY^T.

2.4.3.2 Laplacian Eigenmaps (LE)

In LE[6, 27] we can obtain the weighted neighborhood graph of each data point by calculating the pairwise distance between neighbors by using a Gaussian kernel function as stated in equation 2.6. Let $W \in \mathbb{R}^{nxn}$ represent the adjacency containing the neighborhood relations. Then we can continue with the embedding of Y, the manifold coordinates, into a lower dimension by minimizing the distances between each data points and its neighbors with the following expression:

$$\Theta(Y) = \sum_{i,j} \|y_i - y_j\|^2 W_{ij} \tag{2.10}$$

such that $YDY^T = I$ where D is the degree matrix. Additionally, equation 2.10 is similar to equation 2.9 were $L = D - W$ and $B = D$.

2.5 Classification

The classification of a hyperspectral image pertains to the grouping of pixels in order to represent certain features such as urban, agricultural, water and many other type of areas. Two general techniques in order to classify images are unsupervised and supervised classification methods[9]. Classification can be applied to standard color image as well as multiband raster images. Generally, the time to classify an multiband image takes longer as the number of channels/bands increases which can be mitigated with the usage of feature extraction through PCA or manifold learning techniques.

2.5.1 Unsupervised

The application of unsupervised classification is rather simple due to its unsupervised nature albeit it is a very powerful classification tool. Among this method, pixels are grouped based on their common characteristics. The groups created are referred to as clusters. In order to begin this process of classification, a user first identifies the number of clusters desired. Afterwards, the user would select the type of algorithm in order to cluster the pixels such as K-means clustering or ISODATA[21].

The following will be only be a description of the K-means algorithm. The core idea is to define k centroids, one for each cluster. Selecting appropriate centroids can be non-trivial task because placing the points within incorrect regions can result in non-optimal results. Typically, centroid locations can also be picked in a random manner. Then the algorithm proceeds by taking each pixel and associate it to the nearest centroid and then after all pixels have been assigned, then the algorithm recalculates the positions of the K centroids. Then we repeat the assigning of pixels to the closest centroid and then the recalculation of the positions of the K centroids until the centroids no longer move.

2.5.2 Supervised

Supervised classification is based on the user selecting sample pixels in an image that represent specific classes. More work is required with this technique, but the classification results tend to better than unsupervised classifiers[35]. Therefore we first need to create a training dataset as references for the classification of all other pixels in the image. The training data is selected by the user based on a set of images pertaining to particular classes or images that are based on the ground-truth. The user also sets bounds for how similar other pixels must be in order to group them together. The bounds are usually set based on spectral characteristics of the training area. A widely used supervised classification method is Suport Vector Machine (SVM)[35]. SVMs have often been found to provide higher classification accuracies and are especially advantageous in the presence of heterogeneous classes for which only a few training samples are available. To formally define what an SVM

is, we can state that it is a discriminative classifier defined by a separating hyperplane. Futhermore, given labeled training data supplied by the user (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In order to obtain an optimal hyperplane, SVM needs to find the hyperplane that gives the minimum distance to the training examples. Additionally, SVM can perform linear and non-linear classification through the kernel-trick as described with KPCA. Therefore, the selection of a proper SVM kernel for a given dataset is important in order to obtain optimal classified results. The selection of a proper kernel is described under the KPCA sub-section in the manifold learning section.

CHAPTER 3

SYSTEM DESIGN

The description of the system design has the general flow stated in Figure 8. First this system begins with the reading of a hyperspectral image, and the writing of the dimensionally reduced data which is applied solely after applying PCA or LLE. Then, a description of the spectral partitioning of the HSI data will be described. Next, the MapReduce parallelism will be described and its association with the analysis of the spectral levels of the hyperspectral image. Additionally, a detailed description will be given according to the Hadoop+GPU implementation that was used for this research. Then, a theoretical description will be given concerning the linear and non-linear dimensional reduction algorithms followed by the implementation and enhancements done to the initial algorithm in order to provide speedup in the time domain. Lastly, the two classifiers that were used will also be described theoretically along with the tools that were used to implement the classification process.

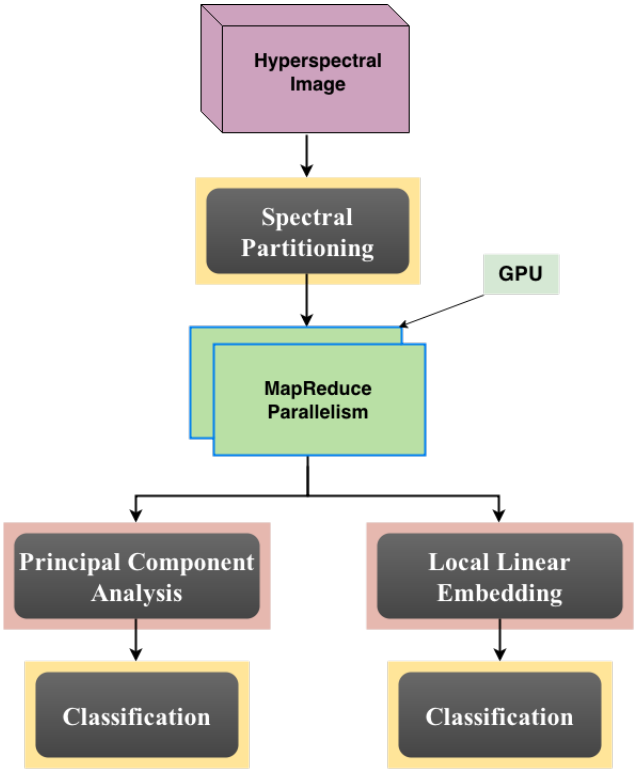


Figure 8.: High Level Overview of System.

3.1 Reading and Writing Hyperspectral Data

Hyperspectral data can be written in three different formats: band interleaved by line (BIL), band interleaved by pixel (BIP), and band sequential (BSQ). The aforementioned methods are three common approaches of organizing image data for multiband images. BIL data stores pixel information band by band for each line, or row, of the image and an illustration of BIL is shown in Figure 9. BIP data is similar to BIL data except that the data for each pixel is written band by band, and the difference between BIL and BIP can be seen in Figure 10. Lastly, BSQ stores information for the image one band at a time as depicted in Figure 11. Additionally, the BIL, BIP

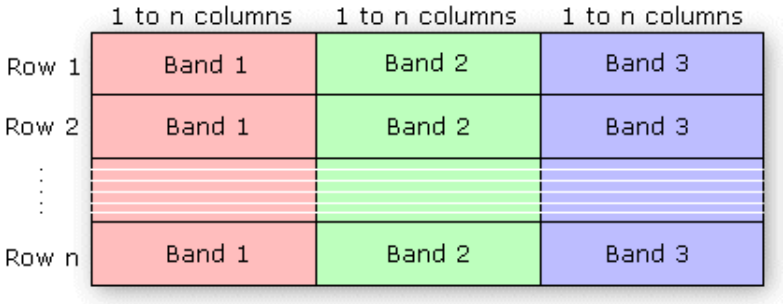


Figure 9.: BIL Interleave Example.

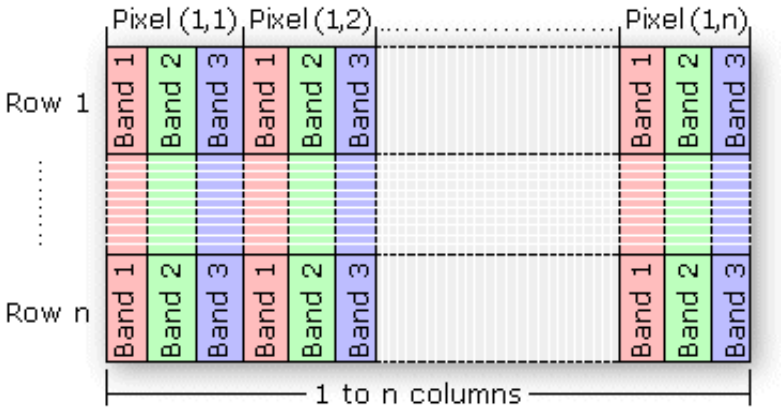


Figure 10.: BIP Interleave Example.

and BSQ files are binary files and must have an associated ACII file header to be interpreted properly. The header file describes the layout of the image pixel data and contains a set of entries, each of which describes a particular attribute of the image.

There are software available that allows for the reading and writing of HSI data such as ENVI and ArcGIS but they are both commercial products, which is not ideal for this research. Therefore, the tool used is the Geospatial Data Abstraction Library (GDAL)[2]. Fundamentally, GDAL is an open-source translator library for raster and vector geospatial data formats. It has support for single raster abstract

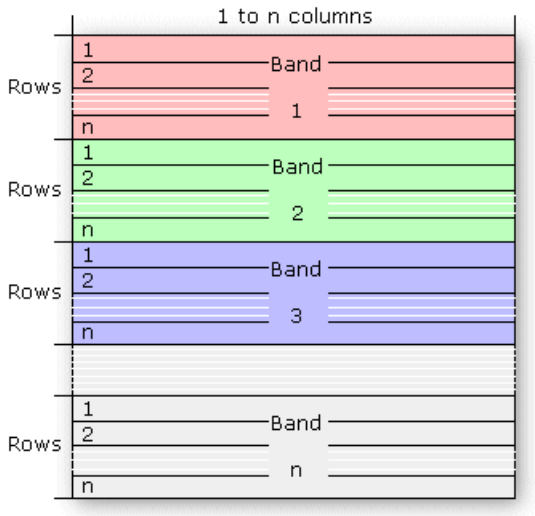


Figure 11.: BSQ Interleave Example.

data and vector abstract data model. As a library, it presents a single abstract data model to the calling application for all supported formats. It can also be built with a variety of useful command-line utilities for data translation and processing.

Now that GDAL has been described as the main tool to read/write HSI data, the rest of this subsection will explain how GDAL was used to read the hyperspectral images and to write the dimensionally reduced data. The first step to read in the data with GDAL is to provide a correct path to the hyperspectral binary file, and not the header file. Some software products require the header file first, but GDAL is smart enough to automatically find the respective header file for the binary file. After the dataset has been successfully opened, we will extract the data on a per band basis. In other words, we will read each band from the image first since it is faster than reading each pixel value vector where each pixel vector contains the intensity values for each band. As soon as each band is read, the intensity values of each band are

stored in a matrix. Note that GDAL reads in a 2-dimensional band and transforms it into a 1-dimensional array where the n-th element of the array corresponds to the intensity value of the n-th pixel. Each matrix row will contain the information for each band. For example, suppose that we have an image X with n bands. Let $X = B_1, B_2, \ldots, B_n$ where B_n is the n-th band and let $B_n = (a_1, a_2, \ldots, a_i)$ where a_i is an intensity value of a pixel. Let A be the matrix that will contain all the data from image X. Then we create the following dataset matrix:

$$\mathbf{A} = \begin{pmatrix} B_1 = \{a_1 & a_2 & \ldots & a_i\} \\ B_2 = \{a_1 & a_2 & \ldots & a_i\} \\ \vdots & & \ddots & \vdots \\ B_n = \{a_i & a_i & \ldots & a_i\} \end{pmatrix} \tag{3.1}$$

In the matrix described in equation 3.1, the first row of matrix A corresponds to the first band of the image, the second row of matrix A corresponds to the second band of the image, and so on and so forth. Then a_1 corresponds to the intensity value of the first pixel of its respective band, so a_i corresponds to the intensity value of the i-th pixel for the n-th band. With this approach, the total number of rows of matrix A is equivalent to the number of bands in the image and the total number of columns is equivalent to the total number of pixels. Additionally, by setting up the matrix in this format, we have access to each pixel vector by simply calling each column from the matrix which is a useful property when we apply dimensionality

reduction, which will be shown in the later sub section.

After we create matrix A we can then manipulate the matrix as needed in order to apply PCA and LLE. After the reduction of dimensions has been successfully applied, we then result in a matrix similar to matrix A in equation 3.1 but with less rows since the dimensions have been reduced. Let A_r be the data matrix with the reduced dimensions, d is the number of dimensions kept, b is the new pixel value, and R_d is the new band to be written.

$$A_r = \begin{pmatrix} R_1 = \{b_1 & b_2 & \dots & b_i\} \\ R_2 = \{b_1 & b_2 & \dots & b_i\} \\ \vdots & & \ddots & \vdots \\ R_d = \{b_i & b_i & \dots & b_i\} \end{pmatrix} \tag{3.2}$$

In order to write the data from matrix A_r, we approach it in a similar manner when the image was read into matrix A. We will read each row from matrix A_r and then use GDAL to write each matrix row as a band where the d-th row of matrix A_r corresponds to the d-th band in a an image. This is the general format in which data was read and written. The following subsection will describe the program development aspect for each different system and will describe any difference concerning the reading and writting of data within each system.

Now that we have described the general form of reading and writing hyperspectral data, we will now continue on the partitioning of the hyperspectral data before it gets submitted into the Hadoop and GPU system.

3.2 Spectral Partitioning

The spectral partitioning of hyperspectral data pertains to the separation of each spectral layer of the image; this is also analogous to the separation of each band. Before the hyperspectral data gets read into the Hadoop+GPU system, we first split the image spectrally by writing each band to one single file where the number of lines in the file will consist of the total number of bands that the image contains. The process of separating the bands is similar to the section where we explain the reading and writing of HSI data. The only difference is that we store the spectral data into the aforementioned dataset matrix after we submit the text file, with the spectral information, into the Hadoop +GPU system. After we prepare the text file containing the spectral information, then we can begin the MapReduce parallelism level described in Figure 8.

3.3 MapReduce Parallelism

For this research, the MapReduce parallelism level will utilize the GPU in order to speed up computations for PCA and LLE. We can see the interaction between the MapReduce level and the GPU in Figure 12. Each reduction that occurs will directly interact with the GPU. Once the communication becomes accessed, then the general workflow to access the GPU is depicted in Figure 13. Since the GPU is only a device, it must be used together with the CPU in order to do anything meaninful. That is why it is required to copy the data from the matrix into the GPU, apply some calculation, then the final results are returned and need to be copied back to the

main CPU memory. All of those steps are occuring once the GPU level parallelism is initialized. After the GPU parallelism and MapReduce parallelism are done, the system will then return a set of dimensionally reduced points and those points would then become emitted by the reductions and then written to an external file. The writing of the external file is taken care of by the Hadoop distributed file system (HDFS), then in order to convert the written file of raw data points into an image we have to "get" the output file from the HDFS and run a script, written for this research, that converts a comma delimited file of floating points values into a "*.tiff" image. After discussing the communication process between MapReduce and the GPU hardware, we will now focus on the actual Hadoop+GPU system that uses the GPU to speed up computations and to create a scheduling algorithm that speeds up the native Hadoop system.

3.4 Hadoop and GPU

In this section we will discuss the Hadoop+GPU base for the proposed system. This section will cover the first approach that was initially used followed by the final implementation of the system. For the first and final approach, the same Hadoop+GPU system was investigated and the only difference was the programming language that was to be used. Therefore, the final approach will discuss the Hadoop+GPU system in more detail.

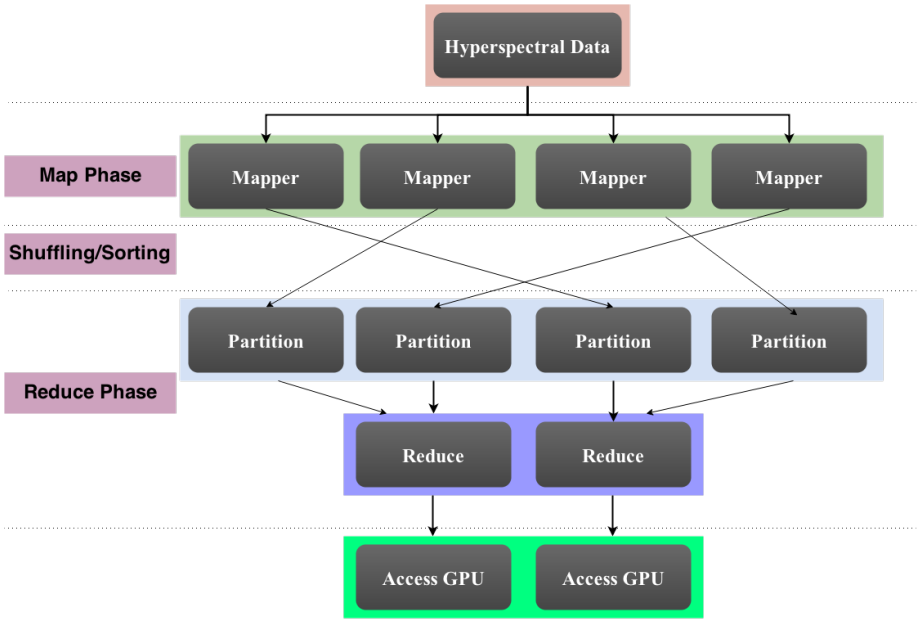


Figure 12.: MapReduce and GPU communication.

3.4.1 First Approach

The first alternative approach involves the usage of an interpretive language called Python. Regarding the HSI aspect, it is possible to use Spectral Python (SPy)[8], a Python package for reading, viewing, manipulating and classifying HSI data. Furthermore, one can run a Python based MapReduce program on Hadoop through Hadoop Streaming. Hadoop Streaming is a generic API that allows Map and Reduce functions to be written in virtually any language. The Map and reduce functions are necessities when dealing with Hadoop because the entire model is based on two primitives: Map and Reduce. The Map and Reduce functions are natively written in Java, so the way that other languages can apply the Map and Reduce paradigm is the

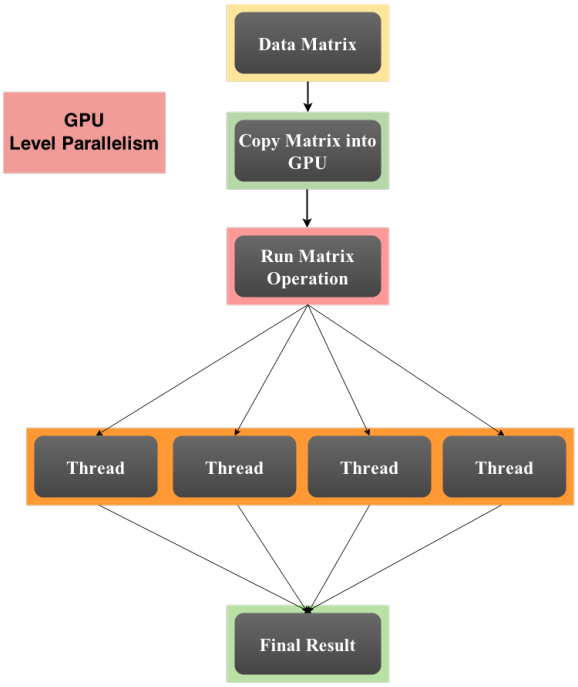


Figure 13.: Overall work distribution on GPU.

receive input from "stdin" and emit output to "stdout". The input pairs are written to stdin for a mapper or reduce with a tab character separating a key from its value. The inputs to the reducer are sorted so that while each line contains only a single key/value pair, all the values for the same key are adjacent to one another. When using streaming, Map and Reduce functions have to be written as two independent files and the command line interface is used to redirect the data flow. Additionally, the purpose of streaming is to reduce development complexity since programmers do not have to follow Hadoop APIs.

In respect to the GPU, Python programs can also be developed using CUDA. NVIDIA has released the NumbaPro[13] Python compiler, a Python CUDA compiler, which can be used with Anaconda Accelerate from Continuum analytics which

contains packages for science, math, engineering and data analysis. Another way to use the GPU with Python is to use PyCUDA[28] which maps all of CUDA into Python and has near-zero wrapping overhead which makes it a fast package. Since PyCUDA's base is written in C++, some users might prefer this method since it has the ability to include CUDA C or C++ code within the Python program.

If users would not like to use Hadoop they can use the Disco Project[37], a lightweight, open-source framework for distributed computing based on the MapReduce paradigm for Python. Disco was created in the Nokia Research Center and this system is much easier to work with due to its native language being Python (Hadoop's native language is Java). Disco distributes and replicates user's data and schedules jobs efficiently. Hadoop and Disco might have different native languages but they are very similar. For instance, Hadoop and Disco can both run on a distributed cluster or on a single node machine. Additionally, they are both based on the master-slave architecture and they also have their own Distributed Filesystem (i.e. Hadoop Distributed FileSystem, Disco Distributed Filesystem). Even though Disco sounds like a great MapReduce implementation model due to the ease of Python, Hadoop is still a much better model to pursue mainly because of the HDFS and due to its popularity among industry and academia. The HDFS gives users the ability to dump large datasets to this distributed filesystem and easily access it with tools that Hadoop provides. Also, the HDFS is very fault tolerant which means that losing a disk or a machine typically does not relate to a disastrous loss of data. The HDFS has evolved into a reliable form to store data and share it with other open-source data analysis tools.

3.4.2 Final Approach

The final approach is to use the C++ language and interface it with Hadoop through a tool called Hadoop Pipes (Pipes). Pipes is another method that allows other languages to be used with Hadoop, which is the C++ interface to Hadoop MapReduce. Pipes uses sockets as the channel over which the tasktracker communicates with the process running the C++ map or reduce function. Unlike Streaming, which uses standard input and output to communicate with the map and reduce code, Hadoop Pipes uses sockets as the channel over which the tastracker communicates with the process running the C++ map or reduce function.

The Hadoop+GPU system used considers the usage of the CPU and the GPU together rather than simply using one of the processors. Even though the usage of this method is ideal, scheduling MapReduce tasks onto CPU cores and GPU devices for efficient execution is a non-trivial task. One method to overcome the scheduling difficulty is to create a hybrid scheduling technique for GPU-based computer clusters, which minimize the execution time of a submitted job using dynamic profiles of Map tasks running on CPU cores and GPU devices. This hybrid map task scheduling was implemented by Shirahata et al.[48]. Their scheduling algorithm is open source and is used to test if any speedup is obtained when compared to a GPU and CPU implementation of the same program. It is important to note that attempting to implement a scheduling algorithm that can handle the CPU and GPU from scratch is a non-trivial job for a single person, therefore creating a more effective scheduling algorithm for the analysis of HSI data will be set for future work. Additionally the

programs will be tested on a single machine with an NVIDIA GPU card. Therefore, the communication between the Map and Reduce phase can use the multiple cores that are present on the CPU. This particular scheduling algorithm uses the Hadoop 0.20.2 API which is an older version of Hadoop. A lot of work is required to incorporate GPU's within the Hadoop system which is why this particular research will use Shirahata et al. implementation of Hadoop and GPU. Additionally, in order to read in data into the HDFS, a text file is created that divides the hyperspectral image along the spectral level, and each band is written to one line of the text file as a value pair. After the text file is written, it is then placed into the HDFS and the dimensionality reduction was applied. After the Hadoop Pipes job successfully ran, then the results from the HDFS are placed on the local disk. Then one more program is executed in order to translate the comma delimited intensity values into an image. Next, a description of the linear and non-linear feature extraction algorithms will be reviewed.

3.5 Dimensional Reduction

In this section, PCA is first discussed along with the Lanczo's iterative algorithm. Then LLE is be discussed along with and overview of Arnoldi's iterative algorithm. Additionally, a second enhancement is presented for LLE after explaining Arnoldi's iterative algorithm.

3.5.1 PCA

PCA is a technique used to emphasize variation and extract strong patterns in a dataset that are linearly correlated. Consider a simple 3 channel image where each channel pertains to the RGB components of the image. Each channel can be treated as a unit by expressing each group of three corresponding pixels as a vector. Let x_1, x_2 and x_3 be the values of a pixel in each of the three RGB component images expressed as a 3-D column vector, x where:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3.3}$$

This vector can further be expanded to an image with N channels. For this research, we use a hyperspectral image that can contain N channels or bands. Therefore the pixel vector for a hyperspectral image would be represented as:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{3.4}$$

Consider a hyperspectral image containing X rows and Y columns. Since we are dealing with a hyperspectral image, let the number of bands be represented as N. The first step in applying PCA is to extract all the pixel vectors from a hyperspectral image. One pixel vector would contain intensity values for all the bands. In Figure

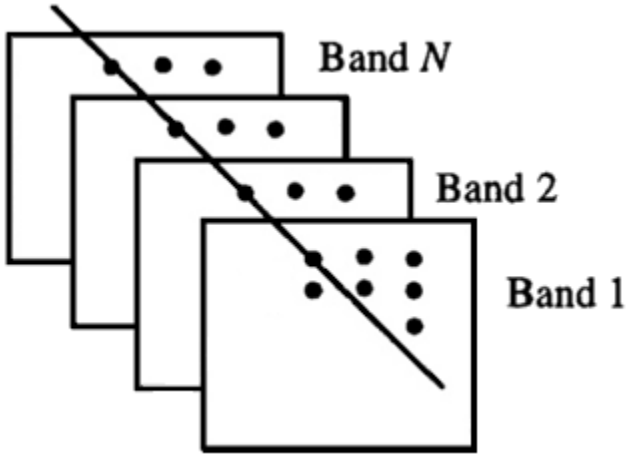


Figure 14.: A pixel vector of size N.

14 let x_i be a pixel vector at the i-th location on a hyperspectral image. That means we will deal with a total of XY pixel vectors; let $K = XY$. Therefore, when working with PCA, we are handling K pixel vectors where each pixel vector contains N elements.

After the extraction of the pixel vectors, we begin the first step of PCA, to calculate the mean vector as:

$$m_x = \frac{1}{K} \sum_{K=1}^{K} x_k \tag{3.5}$$

The mean vector is represented as m_x and K is the total number of pixel vectors. After computing the mean vector, we obtain the covariance matrix represented as C_x. Equation 3.6 depicts the setup of the covariance matrix where x_k^T depicts the transpose of the pixel vector and m_k^T depicts the transpose of the mean vector.

$$C_x = \frac{1}{K}\sum_{K=1}^{K} x_k x_k^T - m_x m_x^T \tag{3.6}$$

Next, it is important to note that C_x is symmetric and real, so finding a set of orthonormal eigenvectors is always possible[23]. Now, let e_i be a set of eigenvectors and let λ_i correspond to eigenvalues. We extract the eigenvectors and their corresponding eigenvalues from the C_x matrix and we create A which is a matrix whose rows are formed from the eigenvectors of C_x so that the first row of A is the eigenvector corresponding to the largest eigenvalue. Now we will use A as a transformation matrix in order to project all the pixel vectors and obtain new values, represented as y. We can accomplish this by applying the following:

$$\mathbf{y} = A(x - m_x) \tag{3.7}$$

The application of equation 3.7 is referred to as the *Hotelling transform* which has some interesting properties such as that the covariance matrix of each y is given in terms of A and C_x with the following expression:

$$C_y = AC_xA^T \tag{3.8}$$

After applying equation 3.7 we have successfully extracted the new principal component bands from a hyperspectral image.

3.5.1.1 Lanczos Iteration

Lanczos algorithm [33, 46] is an iterative algorithm that is an adaptation of power methods to find the eigenvalues and eigenvectors, with the highest magnitude, of an n-th order linear system with a limited number of operations, m, where m is much smaller than n. Lanczos algorithm can be expressed mathematically in the following manner. First suppose we want to extract the eigenvalues from matrix A. So the algorithm begins with an initial vector v_1 of norm unity and then set $\beta_1 \equiv 0$ and $v_0 \equiv 0$. Then the following iteration begins:

$$\text{for } j = 1, 2, \ldots, m \text{ do}$$

$$w_j := Av_j - \beta_j v_j - 1 \tag{3.9}$$

$$\alpha_j := (w_j, v_j) \tag{3.10}$$

$$w_j := w_j - \alpha_j v_j \tag{3.11}$$

$$\beta_{j+1} := \|w_j\|_2 \tag{3.12}$$

$$v_{j+1} := \frac{w_j}{\beta_{j+1}} \tag{3.13}$$

After the iteration ends, then we calculate $w_m = Av_m$ and include it in $\alpha_m = w_m \cdot v_m$. After the above process ends, the algorithms finalizes by creating a tridiagonal matrix T_{mm} depicted as:

$$\mathbf{T}_{mm} = \begin{pmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_m & \alpha_m \end{pmatrix} \tag{3.14}$$

Then one can solve its eigenvalues and corresponding eigenvectors using the QR algorithm. The QR decomposition of a matrix involves decomposing T_{mm} into a product $H_n = QR$ of an orthogonal matrix Q and an upper triangular matrix R.

3.5.1.2 PCA and Lanczos Iteration

In order to enhance PCA, we used Lanczos iterative algorithm when it was time to calculate the principal components, i.e. the eigenvalues and eigenvectors. After we retrieve the eigenvectors that pertain to the eigenvalues of highest magnitude, then we create the transformation matrix A stated in equation 3.7. With the original PCA program, the transformation would have a size $N \times N$ where $N =$ number of bands in image. In our case, this would create a transformation matrix of size 224×224. Although, it is not necessary to create such as matrix since we only need to retain the eigenvalues that contain most of the variance. Therefore we can create an $n \times N$ matrix where $n \ll N$. We would retain n eigenvalue/eigenvector pairs according to the values that retain some percent of the variance which can be done with setting a percent threshold, such as keeping the eigenvalues that only retain

90% of the total variance. In order to compute the n eigenvalues/eigenvectors we then use Lanczos iterative algorithm in order to solely keep a few principal components. With this approach, there is a significant saving in storage and transmission[22], especially when a hyperspectral scene has more than 224 bands.

We will now proceed to the explanation of LLE, similar to the explanation of PCA and its enhancements.

3.5.2 LLE

The general LLE[27, 24] consists of three main steps: search for the nearest neighbors, calculate reconstruction weights and then determine low-dimensional embedding. We can mathematically describe the three steps with the following expressions.

We first find the K-Nearest Neighbors (K-NN):

$$d = \sqrt{(\vec{x_i} - \vec{x_j})^T(\vec{x_i} - \vec{x_j})} \tag{3.15}$$

where d is the Euclidian distance between data point $\vec{x_i}$ and $\vec{x_j}$ in column vectors for $i, j = 1, 2...n$ and n is the number of data points. For this research, n is equivalent to the total number of pixel vectors as shown in Figure 14.

Then, according to the second step, we calculate the reconstruction weights by minimizing:

$$\left\| \vec{x_i} - \sum_{j=1}^{k} w_{ij}\vec{x_j} \right\|^2 \tag{3.16}$$

subject to:

$$\sum_{j=1}^{k} w_{ij} = 1 \tag{3.17}$$

where k is the number of nearest neighbors used for reconstructing each data point. It might not be obvious, but this is a constrained least squares problem that has the following closed-form solution:

$$\vec{w_i} = S_i^{-1}\vec{v} \tag{3.18}$$

where $\vec{v} = [1, 1, \ldots, 1]^T$ and $S_i = (X_i - N_i)^T(X_i - N_i)$. Then X_i is a $d \times k$ matrix whose columns are duplicates of $\vec{x}$ and N_i is also a $d \times k$ matrix whose columns are the k nearest neighbors of $\vec{x}$. Lastly,d is the dimensionality of the original space and the size of $\vec{v}$ is equivalent to the k value previously stated.

The last step is to determine the low-dimensional embedding which is implemented by minimizing:

$$\sum_{i=1}^{n} \left\| \vec{y_j} - \sum_{j=1}^{n} w_{ij}\vec{y_j} \right\|^2 \tag{3.19}$$

where $\vec{y_i}$ is the coordinate of the data point $\vec{x_i}$ in the low-dimensional embedding to be determined with the dimensionality of r. In order to facilitate the low-dimensional embedding, it has been proven [45] $\vec{y_j}$ is equivalent to calculating the eigenvectors corresponding to the r smallest nonzero eigenvalues of $(I - W)^T(I - W)$[24] where I is a $n \times n$ identity matrix and W is also a $n \times n$ matrix, each of whose rows are composed of a data points reconstruction weights. Therefore, by extracting the eigenvector with the smallest eigenvalues, we have successfully mini-

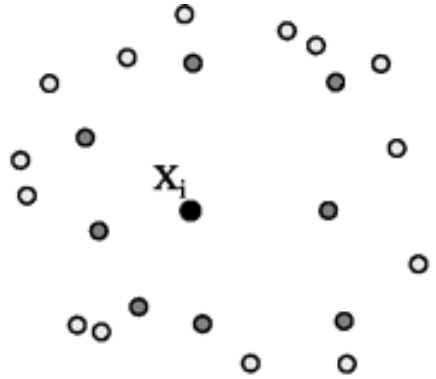


Figure 15.: Obtaining K-NN's for X_i.

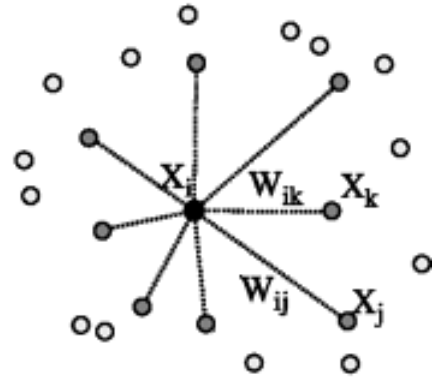


Figure 16.: Obtaining reconstruction coefficients with linear weights.

mized equation 3.19. Additionally, the eigenvector with the smallest eigenvalue (i.e. the eigenvector with an eigenvalue of 0) corresponds to the mean of equation 3.19 and can be discarded to enforce $\sum_{i=1}^{n} y_j = 0$ [14].

A visual representation of this algorithm begins with Figure 15 where we first select the K-NN's for point X_i. Then in Figure 16 we see the reconstruction of point X_i with the linear weights of the K-NN's. Then we complete this algorithm by computing the embedding coordinates for point X_i and we obtain the new point Y_i as described in Figure 17.

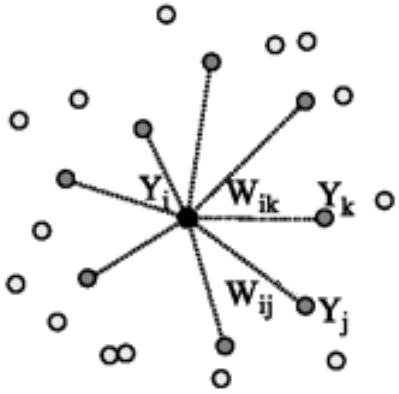


Figure 17.: Mapping coordinates to lower dimension.

In order for the original LLE algorithm to work correctly, matrix S_i produced in the second step, equation 3.18, has to be regulated only when the number of nearest neighbors is more than the input dimensionality (when $k > d$). This can be done by multiplying the diagonal of S_i by a very small number. Furthermore, the between-band correlation for hyperspectral data is significant and matrix S_i often becomes singular when $k < d$ [24]. Therefore, in order to solve for the reconstruction weights, regulation will always be required which is achieved by adding a small value to all the diagonal elements of S_i.

Additionally, it is important to note that LLE is much more intensive in computation and memory consumption compared to PCA. In the worst case scenerio, the computing complexities of the three LLE steps are of order $O(dn^2)$, $O(dnk^3)$ and $O(rn^2)$, respectively where d is the input data dimensionality, k is the number of nearest neighbors, n is the number of data vectors, and r is the output dimensionality. In order to add to complexity, the size of the reconstruction weight matrix W is nxn. Now, lets assume that we have a 640×512 hyperspectral image, then

$n = 327680$. The matrix of such size goes beyond the memory capacity of average PC's even though W is sparse.

3.5.2.1 Arnoldi Iteration

The Arnoldi iteration[46, 31, 1] is an eigenvalue algorithm that finds the eigenvalues of any general square matrix A. Note that in order to find the eigenvalues/vectors of A we can form the characteristic polynomial and then find the roots of the polynomial as the eigenvalues/vectors of A, but this method is not suitable for large sparse matrices. Therefore, to overcome that we run the eigendecomposition starting like the *Power Iteration* method which starts with Ab where b is an arbitrary vector. Then we continue with Ab^2, and so we can write this as: $[Ab, A^2b, A^3b, A^nb]$ where A^nb will converge to the largest or smallest eigenvector of the matrix A. Although, much potentially useful computaiton is wasted by using only yhe final result $A^{n-1}b$ so instead we create the *Krylov matrix* K such that $K_n = [b, Ab, A^2b, A^3b, A^n - 1b]$. This method is unstable, therefore in order to stabilize this method we ensure that each vector is orthogonal to the previous vector, which is referred to as the *Arnoldi iteration*. We ensure the that each vector is orthogonal to the previous vectors by applying the *Gram-Shmidt* orthogonalization procedure. Therefore we start the Arnoldi iteration with an arbitrary vector q_1 with a norm of 1 in order to have an orthonormal basis:

$$||q_1|| = 1 \tag{3.20}$$

Then we repeat for q_k through q_n such that $q_k = Aq_{k-1}$. In order to orthogonalize the vector to all the vectors before k, then we state the following:

$$\text{for } j = 1, 2, \ldots, k-1$$

$$h_{j,k-1} = q_j^T q_k \tag{3.21}$$

$$q_k = q_k - (q_j^T)(h_{j,k-1})q_j \tag{3.22}$$

Where equation 3.21 is the scalar that defines the length of the projection onto the vector j. Then we can normalize q_k, so we end the loop and proceed by setting $h_{k,k-1} = ||q_k||$ and scale q_k to be normal with: $q_k = \frac{q_k}{h_{k,k-1}}$ which finalizes the Arnoldi iteration. After the previous procedure is complete, we produce matrix H_n such that:

$$\mathbf{H}_n = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \ldots & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \ldots & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \ldots & h_{3,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & h_{n,n-1} & h_{n,n} \end{pmatrix} \tag{3.23}$$

Where H_n is the upper *Hessenberg* matrix formed by the numbers $h_{j,k}$ computed by the Arnoldi algorithm. An upper *Hessenberg* matrix is a special kind of square matrix that has zero entries below the first sub-diagonal. In comparison, a lower *Hessenberg* matrix has zero entries above the first super-diagonal. Since the Arnoldi iteration projects matrix A onto the *Krylov* subspace, the projection represented by H_n, we can calculate the eigenvalues and eigenvectors efficiently by applying the QR algorithm.

3.5.2.2 LLE and Arnoldi Iteration

For the LLE, Arnoldis iterative algorithm is useful because, like PCA, we want to only keep a few eigenvalue/eigenvector pairs. Furthermore, it is more useful to use Arnoldis method for LLE because we are handling a large sparse matrix towards the end. In this case, we also only require the eigenvectors that pertain to the smallest eigenvalues rather than obtaining all n eigenvalues/vectors where $n =$ pixel vectors. In this particular case, the savings in storage and transmission is much greater compared to PCA.

The second enhancement done to the original LLE program pertains to the calculation of embedding from eigenvectors of cost matrix. Rather than creating a final sparse matrix with a loop that requires a complexity of $O(n^2)$ and direct access to the coefficients of a sparse matrix, we instead obtain the product of $M = (I - W)^T(I - W)$, where I is the identity matrix and W is a sparse matrix holding all reconstruction weights. This approach improves the time because we take advantage of data reuse by maximizing the utilization of fast local storage. For example, asssuming that we have matrix C such that $C = AB$, then we can decompose both A and B into block matrices. Then we can recursively apply matrix-matrix multiplication on the block matrices which allows for better locality of reference both in space and time of the data used in the product. Therefore this procedure takes advantage of the cache on the system. Additionally, if the system running the program has more than one level of cache, then the blocking can be applied a second time to improve with data access time. This improvement was achieved with the

Basic Linear Algebra Subprograms (BLAS). After we calculate M, we then apply Arnoldis algorithm in order to extract the eigenvalues/eigenvectors with the smallest magnitude. Additionally there will be a test comparison of the initial PCA/LLE program vs. the improved PCA/LLE program will show the difference in runtime speed.

Next we will go over the last part of this section pertaining to the classification of the hyperspectral images.

3.6 Classification

The two classifiers that are used in this research are an unsupervised approach, K-Means, and a supervised method, extraction and classification of homogeneous objects (ECHO). These two classifiers were applied to the hyperspectral scenes on two distinct instances. In the first instance, we apply the classifiers after we reduce the number of dimensions with either PCA or LLE. In the second instance, we apply the classifier to the raw HSI data in order to compare results to the dimensionally reduced images in regards to classification accuracy and time to run each system. The following will be a general description of each algorithm followed by the actual implementation of each classifier.

3.6.1 K-Means

The standard method to approach K-Means uses an iterative refinement technique with two basic steps: the "assignment" step and "update" step.

Let $X = (x_1, x_2, \ldots, x_n)$, where $\forall x \in X$ is a d-dimensional real vector. The

K-Means algorithm then aims to partition n observations into k, where $k \leq n$, sets $S = S_1, S_2, \ldots, S_k$ in order to minimize the within-cluster sum of squares. Given an initial set of k means $m_1, m_2, \ldots, m_k$, the first step assigns each observation to the cluster whose mean yields the least within-cluster sum of squares. Then, we calculate the new means to be the centroids of the observations in the new clusters. The iterative algorithm stops when the assignments no longer change. The minimization objective of the within-cluster sum of squares is described as:

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} (\|x - \mu_i\|^2) \tag{3.24}$$

where μ_i is the mean point in S_i.

The implementation of the K-Means algorithm is done with the Open Foris toolkit and GDAL. Open Foris provides tools that use GDAL in order to apply K-Means to geo-spatial raster data. The process to implement this classifier begins by first generating a grid of points over the hyperspectral image. Then we extract the values from the input image for the pixels that lay on the grid and then write them to a file. Afterwards we run the K-Means algorithm based on the extracted points that were written to a file and then we choose the total number of clusters desired. The final classified image will be a single band 32-bit raster image, so the image will not be visible to a human user. Therefore, the image will have to be converted to an 8-bit image, where the intensity values range between $0 - 255$, thus making it viewable to a human user.

3.6.2 ECHO

The ECHO algorithm [26] consists of two main steps. The first step is to apply a conjunctive object finding algorithm to determine the samples in the scene to be classified. This algorithm begins with a fine partition of the scene and simplifies it by progressively merging adjacent cells that are found to be similar according to certain statistical criteria. The second step is then to classify the entire image using a maximum likelihood (ML) sample classification algorithm. In this manner ECHO benefits from the spatial correlation of picture elements belonging to the same object.

The basic approach of the first step consists of two distinct levels of tests. In the first level of testing, for step one, the pixels are divided by a rectangular grid into small groups. Each group becomes a unit referred to as a cell, provided that it satisfies a relatively mild criterion of homogeneity. The groups that do not fit into the mild criterion of homogeneity are assumed to overlap a boundary and their individual pixels are classified by a no-memory method. A no-memory method means that each pixel is classified individually on the basis of its spectral measurements alone. During the first test, it is a good practice to keep a low rejection rate to reflect the relatively high *a priori* probability of a group being homogeneous. The goal of the first level of testing is to detect as many pixels as possible that lie along boundaries without requiring that the ones detected form closed contours or even be connected.

In the second level of testing, an individual cell is compared to an adjacent field which is a group of one or more connected cells that have previously been merged. If the two samples appear statistically similar then they are merged. Otherwise the cell

is compared to another adjacent field or becomes a new field itself. By successively appending adjacent cells, each field expands until it reaches its natural boundaries, where the rejection rate increases and halting further expansion.

We can express the first step of extraction and classification of homogeneous objects (ECHO) with the following mathematical expressions. First we know that a scene consist of objects whose boundaries form a partition of the scene. Each object in the partition belongs to K classes. Let W_i denote the event that an object belongs to class i. Each pixel in an object is a q-dimensional random variable where q denotes the number of spectral measurements per pixel. It is assumed that the q-variate, marginal, probability density function of a pixel, X, depends only on the class of the object containing X. This is due to the homogeneity of the types of objects typically encountered in remote sensing applications. Now, $p(x|w_i), x \in R^q$ denotes this class-conditional density function for the i-th class. We can also assume that the classes can be define such that $p(x|W_i)$ is approximately multi-variate normal (MVN), which means that:

$$p(x|W_i) = (|2\pi C_i| \exp((x - M_i)^t C_1^{-1}(x - M_i)))^{\frac{-1}{2}} \tag{3.25}$$

for some q-dimensional positive-definite, covariance matrix C_i and some mean vector $M_i \in R^q$. Parametric estimates of these density functions are obtained by estimating M_i and C_i from sets of training data supplied for each class. Furthermore, if $X = (X_1, X_2, \ldots, X_n) \in R^{nq}$ represents a set of pixels in some object, then this set constitutes a "sample" from a population characterized bu one of the

class-conditioned probability density functions. A sample classifier is a strategy for deciding which one, based on n observations. This then leads to the next step in the ECHO algorithm. In the second step of ECHO, the maximum likelihood (ML) classifier begins with the calculation of a probability function from the inputs for classes established from the training sites. Each pixel is then judged as to the class to which it most probably belongs. This classifier assigns X to class i with the following expression:

$$\ln p(X|W_i) = \max j \ln p(X|W_j) \tag{3.26}$$

The general overview of the ECHO algorithm has been reviewed. In this research, the ECHO algorithm is used through an open source tool called Multispec [3]. The first step to classify and image is to first create the statistics defined in the first step of the ECHO algorithm. This is achieved by selecting a rectangular region around the classes that are to be classified. Once all the classes are properly selected, and assuming the number of pixels per class is more than the number of bands in the image, then the training can be initialized. If the number of pixels selected for training are not enough, then an error will be outputted and more pixels will be required for the training to be effectively completed. This error goes hand-in-hand with Hughes phenomenon where more training samples are required as the number of bands increases. Assuming all the classes have been carefully selected and trained, then the classification using ML is applied and the final output would be a fully classified image consisting of all the classes that were previously selected.

CHAPTER 4

EVALUATION AND RESULTS

In this chapter, the results are discussed along with the evaluation metrics. There will also be discussion for sets of tables in order to explain all the results. The testing will be separated by each hyperspectral scene, so the classification results for the vegetation scene will be first then the oil spill scene. The classification results refers to the confusion matrices created to test classification accuracy, omission error, commission error. After the confusion matrices have been discussed, then the overall classification and timing results will be discussed for each image in their respective order. All of the classified images for the vegetation scene are displayed in appendix A and the classified images for the oil spill scene are in appendix B.

For each hyperspectral scene and its respective confusion matrices is explained along with the overall accuracy and timing results. The time results shown here were obtained by taking the average of 10 iterations. Lastly the results between the initial PCA/LLE programs are compared to their enhanced counterparts. If the title contains "no DR", it means that "no Dimensional Reduction" was applied and the classification was applied to the raw HSI image.

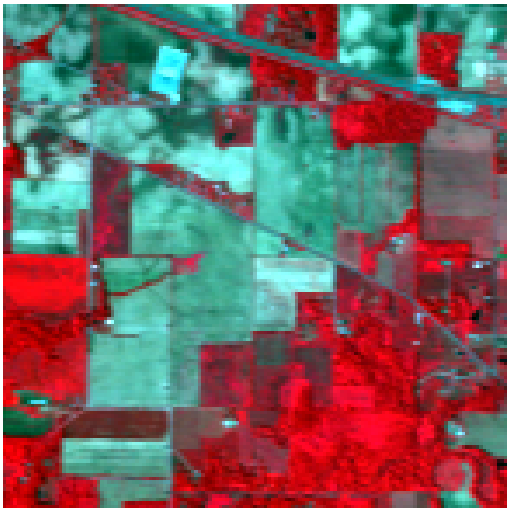
Before continuing, it is important to discuss the limitations due to *Hughe's Phenomenon*. As stated in the beginning of this writeup, some limitations that comes with classifying hyperspectral images is the inability to classify classes that are initially small. This occurs because the number of training samples for a class increases as the number of bands increases. Therefore, there may not be enough

training samples for small class when the number of bands is high, which is a direct result of the *curse of dimensionality*. As an example, refer to the hyperspectral image named Indian Pines in Figure 18a and its groundtruth 18b. This hyperspectral scene contains 16 total classes with a color coded legend shown in Figure 18c. This hyperspectral image contain 220 bands. If this image is classified with the ECHO classifier, then it would be not be possible because the training samples required for each class has to be at least one value more than 220 pixels. We would run into this problem when attempting to classify classes such as the *Grass/pasture-mowed* and *Oats*. Therefore, it is not possible to use this image for testing purposes because applying the classification of the raw hyperspectral image with the ECHO classifier is not possible.

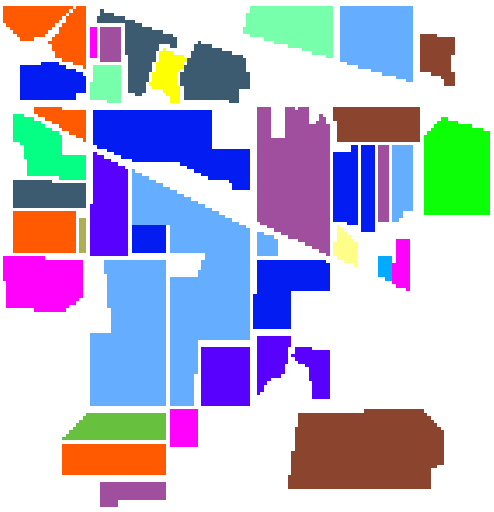
4.1 Evaluation Results

This section reviews the results that were obtained from dimensionality reduction and classification of the images. Testing was applied on two hyperspectral images. The first image, Salinas-A, is taken from the vegetation field and is a subset of the Salinas' California hyperspectral scene taken with AVIRIS. It is comprised of 83 rows, 86 columns and 224 bands with a pixel size of 20 meters. A color image of the Salinas-A scene is shown in Figure 19. The Salinas-A scene was tested with PCA and LLE.

Additionally, the number of bands kept for PCA were selected based on a scree plot and on an investigative study. First, a scree plot displays the eigenvalues associated with a component or factor in descending order versus the number of the



(a) Indian Pines RGB Image.



(b) Indian Pines Groundtruth.

(c) Indian Pines Class Legend.

Figure 18.: Indian Pines Image and its Groundtruth

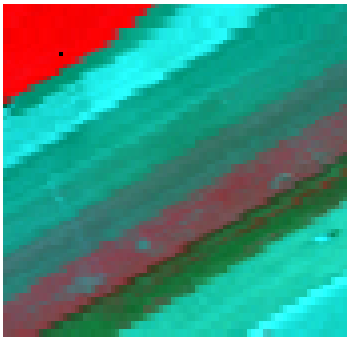


Figure 19.: RGB Representation of Salinas-A Scene.

component or factor. It helps to visually assess which components or factors explain most of the variability in the data. The scree plot for the vegetation scene is displayed in Figure 20. We see that most of the variance is kept within the first few principal componets (x-axis). Although, as stated in this research article [56], vegetation images most of the variance is kept within the first 20 components. Therefore, just to be pedantic, the first 20 components are kept when PCA is applied to the hyperspectral vegetation scene. When applying LLE to the vegetation scene, we keep a total of 10 dimensions because anything less than that returns a singular matrix and the output image is contains plenty of noise depicted as dots and lines. The value of K, for the nearest neighbors, was set to 84. In order to define the value of K, a method called *bootstrapping*[15] is used where the choice of $K = \sqrt{\text{total samples}}$. In this case, $K = \sqrt{83 \times 86}$.

The second image is a scene that was taken, again with the AVIRIS, during the Deepwater Horizon Gulf of Mexico oil spill near the Barataria Bay. For testing purposes, this particular image is split into two different images, a "large" image and a "small" image. The large image was initially produced in order to apply PCA. The

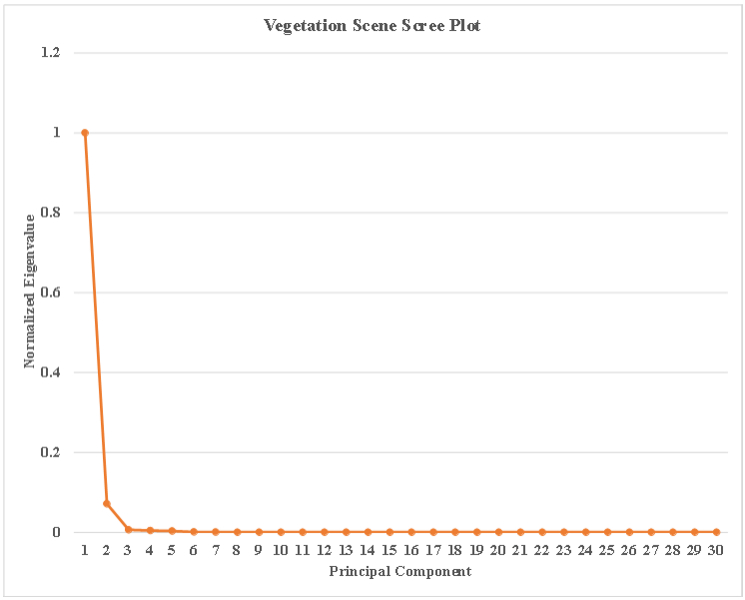
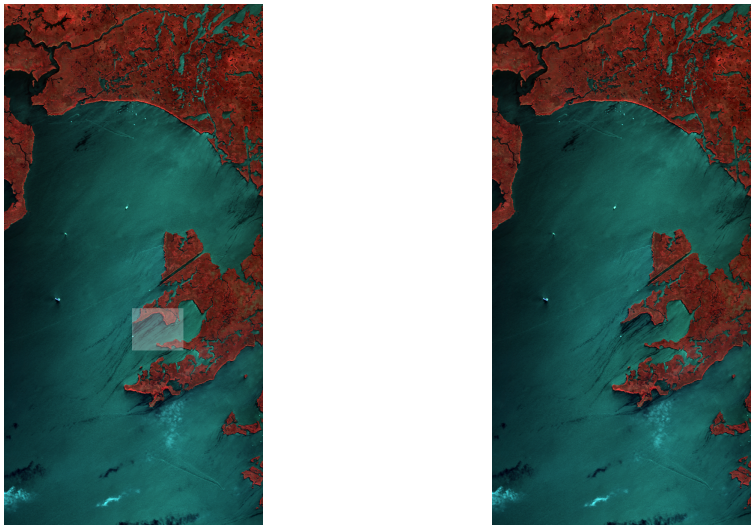


Figure 20.: Scree Plot for Vegetation Scene.

large image is comprised of 1400 rows, 849 columns, 224 bands with a pixel size of 20 meters. A standard color image of the large oil scene is shown in Figure 21b. The smaller image of the oil spill scene was extracted from the large oil spill scene mainly to apply LLE since large images tend to demand more memory with the application of LLE. The small oil spill scene consists of 100 rows, 100 columns, 224 bands and a pixel size of 20 meters. PCA and LLE were applied to the small oil spill scene and Figure 21a shows the highlighted area that pertains to the small oil spill image.

Futhermore, the number of components kept for PCA, in both both small and large image, is 5. This value was obtained after observing the scree plot of the oil spill scene shown in Figure 22. In regards to the LLE algorithm, a total of 7 dimensions were retained since anything less than that returns a singular matrix, as stated with the vegetation scene. Also, the K value used was 100 since $K = \sqrt{100 \times 100} = 100$.



(a) Small Oil Spill Scene. (b) Large Oil Spill Scene.

Figure 21.: Small and Large Oil Spill Image.

All the tests were done on a machine, in the Pixel Island lab supervised by Dr. King. The machine contained an NVIDIA GeForce GTX 780 GPU with an Intel Core i7-4930K CPU.

The last metrics that will be discussed are the classification metrics along with their general form of calculation. Agreement/accuracy represents the probability that the classifier has labeled an image pixel into the ground truth class (the probability of a reference pixel being correctly classified). This can be expressed in the following manner:

$$\text{Agreement/accuracy} = \frac{\text{True Positive}}{\text{Column Total}} \tag{4.1}$$

Overall accuracy refers to the total classification accuracy expressed as:

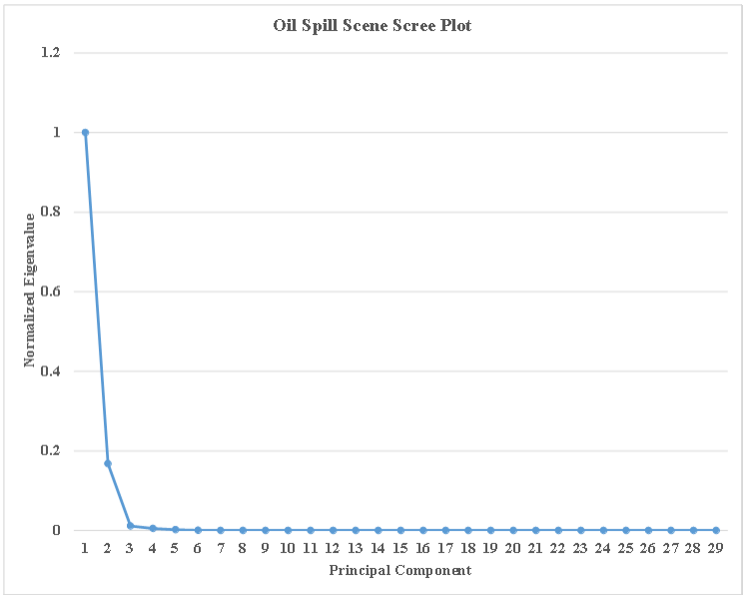


Figure 22.: Scree Plot for Oil Scene.

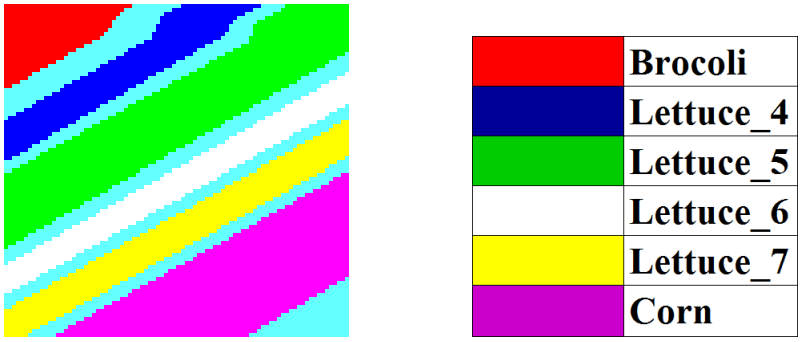
$$\text{Overall accuracy} = \frac{\text{Sum of all True Positives}}{\text{Sum of All Columns Values}} \tag{4.2}$$

Omission error represent pixels that belong to the truth class but fails to be classified into the proper class.

$$\text{Omission error} = \frac{\text{Sum of all False Positives in Class Column}}{\text{Sum of Column Values}} \tag{4.3}$$

Commission error represents pixels that belong to another class but are labeled as belonging that the class.

$$\text{Commission error} = \frac{\text{Sum of all False Positives in Class Row}}{\text{Sum of Row Values}} \tag{4.4}$$



(a) Vegetation Groundtruth. (b) Vegetation Legend.

Figure 23.: Vegetation Groundtruth and respective legend.

4.2 Data Set

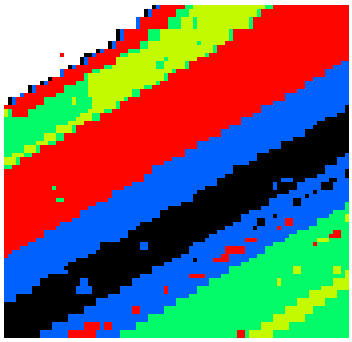
4.2.1 Vegetation Results

The Salinas-A scene consisted of 6 distinct classes shown in Figure 23a and the classes are stated in Figure 23b.

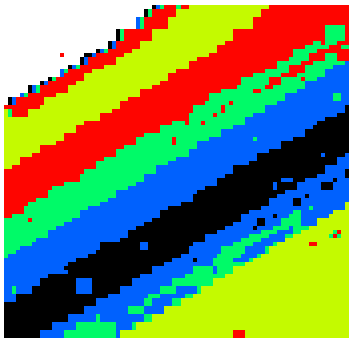
In order to view sample classified images after applying PCA and K-means, refer to Figure 24.

4.2.1.1 Confusion Matrices

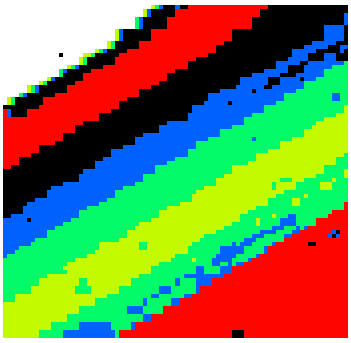
In table I the overall accuracy for each subtable seems to be fairly low, with a maximum accuracy of 49.40% which was given by the Hadoop+GPU and GPU test case. For the first three tables the most difficult classes to detect were Lettuce-5 and the Corn classes. For the CPU test case, the class for Corn was also one of the worst classes to detect and class Lettuce-4, Lettuce-5, Lettuce-7. The third table could have rendered low accuracy due to the application of the full 224 bands in order to



(a) CPU version: PCA+ K-Means.



(b) GPU version: PCA+ K-Means.



(c) K-means on raw image.

Figure 24.: Test 1 Vegetation Classification.

obtain a fully classified data set. Overall, the detection of the Corn class was the most troublesome. One possibility as to why the CPU case had worst results could be due to the K-Means classifier, as we will see in other data tables.

In table II we see that there seems to be a more stable overall accuracy level since the highest accuracy was 73.89% when no dimensionality reductin was applied. The lowest was given by the Hadoop+GPU and GPU test case with 66.56%. For the third test case, it is possible that it gave better accuracy mainly because it required more training samples. Since this table shows results dealt with the supervised ECHO classifier, the higher number of training samples allowed the third table to give better results. Additionally, ECHO seems to be a factor when returning more stable overall results, compared to table I.

In table III we observe that the CPU test case gave the worst overall accuracy with a 40.28% and the Hadoop+GPU and GPU test cases return the highest with a 49.93%. The results given in this data set seem to have a similar pattern to the results given in I, where the CPU test case gave the worst results and the Hadoop+GPU and GPU test case gave the best results. Even though LLE was used in this case, it is very possible that there is a general problem with the application of the K-means classifier with the vegetation dataset, since results seem rather different for the oil spill dataset and in talbe IV.

In table IV we observe that the case when no dimensionality reduction was applied, we achieved the highest classification accuracy by a small margin compared to the other systems. Additionally, it is apparent that the most difficult class to detect was Lettuce-4. Although the overall accuracy was more stable with ECHO

and much more accurate with LLE compared to table I and table III.

After observing the first four tables, we see that the last test case, table IV with LLE and ECHO, returned the best results compared to the first three tables. All the test cases in table IV gave a 70% accuracy or better. Additionally, it seems like the K-means classifier is not very reliable since using it returned the lowest possible overall accuracy. This could be due to the fact that the original Salinas-A image was not corrected. Therefore, there is a possibility that were was a lot of noise involved even before the dimensional reduction was applied.

Table I.: Vegetation Test 1

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	390	0	0	0	0	0
	Lettuce_4	0	584	0	0	0	763
	Lettuce_5	1	32	905	0	0	10
	Lettuce_6	0	0	0	667	38	335
	Lettuce_7	0	0	0	1	761	16
	Corn	0	0	620	6	0	219
Accuracy		99.74%	94.81%	59.34%	98.96%	95.24%	16.31%
Omission Error		0.26%	5.19%	40.66%	1.04%	4.76%	83.69%
Comission Error		22.62%	65.42%	27.37%	58.13%	29.73%	78.59%
Overall Accuracy	49.40%						

(a) Hadoop+GPU and PCA+K-Means

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	390	0	0	0	0	0
	Lettuce_4	0	584	0	0	0	763
	Lettuce_5	1	32	905	0	0	10
	Lettuce_6	0	0	0	667	38	335
	Lettuce_7	0	0	0	1	761	16
	Corn	0	0	620	6	0	219
Accuracy		99.74%	94.81%	59.34%	98.96%	95.24%	16.31%
Omission Error		0.26%	5.19%	40.66%	1.04%	4.76%	83.69%
Comission Error		22.62%	65.42%	27.37%	58.13%	29.73%	78.59%
Overall Accuracy	49.40%						

(b) GPU and PCA+K-Means

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	390	0	0	0	0	0
	Lettuce_4	0	585	0	0	0	763
	Lettuce_5	1	31	888	0	0	10
	Lettuce_6	0	0	0	668	33	385
	Lettuce_7	0	0	0	1	766	13
	Corn	0	0	637	5	0	172
Accuracy		99.74%	94.97%	58.23%	99.11%	95.87%	12.81%
Omission Error		0.26%	5.03%	41.77%	0.89%	4.13%	87.19%
Comission Error		22.62%	65.41%	27.63%	59.42%	29.07%	82.63%
Overall Accuracy	48.60%						

(c) no DR and K-Means

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	390	0	0	0	0	0
	Lettuce_4	0	361	0	0	0	45
	Lettuce_5	1	27	1522	0	0	70
	Lettuce_6	0	0	0	1	768	17
	Lettuce_7	0	0	0	673	31	470
	Corn	0	228	3	0	0	741
Accuracy		99.74%	58.60%	99.80%	0.15%	3.88%	55.17%
Omission Error		0.26%	41.40%	0.20%	99.85%	96.12%	44.83%
Comission Error		22.62%	36.44%	26.08%	99.91%	98.23%	36.07%
Overall Accuracy	42.67%						

(d) CPU and PCA+Kmeans

Table II.: Vegetation Test 2

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	391	0	0	0	0	0
	Lettuce_4	0	562	0	0	0	20
	Lettuce_5	0	54	1503	0	0	18
	Lettuce_6	0	0	22	672	12	383
	Lettuce_7	0	0	0	2	787	86
	Corn	0	0	0	0	0	836
Accuracy		100.00%	91.23%	98.56%	99.70%	98.50%	62.25%
Omission Error		0.00%	8.77%	1.44%	0.30%	1.50%	37.75%
Comission Error		20.37%	26.15%	30.83%	56.28%	41.14%	0.36%
Overall Accuracy	66.56%						

(a) Hadoop+GPU and PCA+ECHO

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	391	0	0	0	0	0
	Lettuce_4	0	562	0	0	0	20
	Lettuce_5	0	54	1503	0	0	18
	Lettuce_6	0	0	22	672	12	383
	Lettuce_7	0	0	0	2	787	86
	Corn	0	0	0	0	0	836
Accuracy		100.00%	91.23%	98.56%	99.70%	98.50%	62.25%
Omission Error		0.00%	8.77%	1.44%	0.30%	1.50%	37.75%
Comission Error		20.37%	26.15%	30.83%	56.28%	41.14%	0.36%
Overall Accuracy	66.56%						

(b) GPU and PCA+ECHO

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	391	0	0	0	0	0
	Lettuce_4	0	564	0	0	0	5
	Lettuce_5	0	52	1525	0	1	12
	Lettuce_6	0	0	0	674	3	0
	Lettuce_7	0	0	0	0	794	0
	Corn	0	0	0	0	1	1326
Accuracy		100.00%	91.56%	100.00%	100.00%	99.37%	98.73%
Omission Error		0.00%	8.44%	0.00%	0.00%	0.63%	1.27%
Comission Error		20.85%	20.34%	31.18%	36.17%	28.34%	14.78%
Overall Accuracy	73.89%						

(c) no DR and ECHO

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	390	0	0	0	0	0
	Lettuce_4	0	550	0	0	0	8
	Lettuce_5	0	20	1081	0	0	0
	Lettuce_6	0	0	0	659	0	0
	Lettuce_7	0	0	0	5	786	0
	Corn	1	46	444	10	13	1335
Accuracy		99.74%	89.29%	70.89%	97.77%	98.37%	99.40%
Omission Error		0.26%	10.71%	29.11%	2.23%	1.63%	0.60%
Comission Error		20.41%	21.09%	16.59%	29.82%	29.70%	48.61%
Overall Accuracy	67.26%						

(d) CPU and PCA+ECHO

Table III.: Vegetation Test 3

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	53	0	0	0	0	716
	Lettuce_4	337	56	0	0	0	25
	Lettuce_5	1	560	1525	7	0	65
	Lettuce_6	0	0	0	617	1	0
	Lettuce_7	0	0	0	50	784	8
	Corn	0	0	0	0	14	529
Accuracy		13.55%	9.09%	100.00%	91.54%	98.12%	39.39%
Omission Error		86.45%	90.91%	0.00%	8.46%	1.88%	60.61%
Comission Error		93.13%	90.86%	46.94%	26.28%	36.98%	33.79%
Overall Accuracy	49.93%						

(a) Hadoop+GPU and LLE+K-Means

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	53	0	0	0	0	716
	Lettuce_4	337	56	0	0	0	25
	Lettuce_5	1	560	1525	7	0	65
	Lettuce_6	0	0	0	617	1	0
	Lettuce_7	0	0	0	50	784	8
	Corn	0	0	0	0	14	529
Accuracy		13.55%	9.09%	100.00%	91.54%	98.12%	39.39%
Omission Error		86.45%	90.91%	0.00%	8.46%	1.88%	60.61%
Comission Error		93.13%	90.86%	46.94%	26.28%	36.98%	33.79%
Overall Accuracy	49.93%						

(b) GPU and LLE+K-Means

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	390	0	0	0	0	0
	Lettuce_4	0	585	0	0	0	763
	Lettuce_5	1	31	888	0	0	10
	Lettuce_6	0	0	0	668	33	385
	Lettuce_7	0	0	0	1	766	13
	Corn	0	0	637	5	0	172
Accuracy		99.74%	94.97%	58.23%	99.11%	95.87%	12.81%
Omission Error		0.26%	5.03%	41.77%	0.89%	4.13%	87.19%
Comission Error		22.62%	65.41%	27.63%	59.42%	29.07%	82.63%
Overall Accuracy	48.60%						

(c) no DR and K-Means

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	0	0	764	7	0	4
	Lettuce_4	0	588	2	0	0	22
	Lettuce_5	1	24	716	593	0	8
	Lettuce_6	0	4	0	0	0	19
	Lettuce_7	390	0	43	74	799	518
	Corn	0	0	0	0	0	772
Accuracy		0.00%	95.45%	46.95%	0.00%	100.00%	57.48%
Omission Error		100.00%	4.55%	53.05%	100.00%	0.00%	42.52%
Comission Error		100.00%	36.50%	59.89%	100.00%	69.64%	0.52%
Overall Accuracy	40.28%						

(d) CPU and LLE+Kmeans

Table IV.: Vegetation Test 4

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	389	0	0	0	0	0
	Lettuce_4	0	385	0	0	0	25
	Lettuce_5	0	29	1518	0	0	3
	Lettuce_6	0	0	6	674	25	0
	Lettuce_7	0	**0**	0	0	760	0
	Corn	2	202	1	0	14	1315
Accuracy		99.49%	62.50%	99.54%	100.00%	95.12%	97.92%
Omission Error		0.51%	37.50%	0.46%	0.00%	4.88%	2.08%
Comission Error		19.13%	28.17%	21.55%	39.82%	25.56%	35.70%
Overall Accuracy	70.62%						

(a) Hadoop+GPU and LLE+ECHO

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	389	0	0	0	0	0
	Lettuce_4	0	385	0	0	0	25
	Lettuce_5	0	29	1518	0	0	3
	Lettuce_6	0	0	6	674	25	0
	Lettuce_7	0	**0**	0	0	760	0
	Corn	2	202	1	0	14	1315
Accuracy		99.49%	62.50%	99.54%	100.00%	95.12%	97.92%
Omission Error		0.51%	37.50%	0.46%	0.00%	4.88%	2.08%
Comission Error		19.13%	28.17%	21.55%	39.82%	25.56%	35.70%
Overall Accuracy	70.62%						

(b) GPU and LLE+ECHO

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	391	0	0	0	0	0
	Lettuce_4	0	564	0	0	0	5
	Lettuce_5	0	52	1525	0	1	12
	Lettuce_6	0	0	0	674	3	0
	Lettuce_7	0	0	0	0	794	0
	Corn	0	0	0	0	1	1326
Accuracy		100.00%	91.56%	100.00%	100.00%	99.37%	98.73%
Omission Error		0.00%	8.44%	0.00%	0.00%	0.63%	1.27%
Comission Error		20.85%	20.34%	31.18%	36.17%	28.34%	14.78%
Overall Accuracy	73.89%						

(c) no DR and ECHO

		Groundtruth					
	Classes	**Brocoli**	**Lettuce_4**	**Lettuce_5**	**Lettuce_6**	**Lettuce_7**	**Corn**
Predicted	**Brocoli**	389	0	0	0	0	0
	Lettuce_4	0	472	0	0	0	28
	Lettuce_5	0	27	1521	0	0	4
	Lettuce_6	0	0	0	672	4	0
	Lettuce_7	0	0	0	2	784	0
	Corn	2	117	4	0	11	1311
Accuracy		99.49%	76.62%	99.74%	99.70%	98.12%	97.62%
Omission Error		0.51%	23.38%	0.26%	0.30%	1.88%	2.38%
Comission Error		19.13%	28.27%	22.95%	32.33%	27.74%	32.67%
Overall Accuracy	72.14%						

(d) CPU and LLE+ECHO

4.2.1.2 Overall Accuracy and Timing Results

The following tables incorporate the time it took to: read the hyperspectral image, run the dimensional reduction (feature extraction) algorithm, write the new projected dimensions and the time it took to complete the image classification. For table V, we see the same overall classification accuracy as shown in the confusion matrices above. In this case, it is important to look at the total time that it took to run each test case. For each test, the Hadoop+GPU system took the longest time out of all the other test cases. This can be due to the higher amount of communication that is required between Hadoop, the CPU and the GPU. The test case that used the GPU only involves communication between the CPU and GPU, and the CPU test case does not communicate with any other devices or systems. Furthermore, as previously mentioned in chapter 3, it is apparent that LLE takes the longest time, compared to PCA, due to the general algorithm. Additionally, even though the K-means seems to be much faster than ECHO, it does not produce the most accurate results. There seems to be some issues with the K-means classifier with the detection of vegetation areas as stated in the previous discussion.

Overall it seems like the Hadoop+GPU system does not provide any advantage to the analysis of hyperspectral vegetation scenes. It is apparent that the Hadoop+GPU system can only provide high fault tolerance by replicating the data onto different nodes. Besides that, it seems to add too much overhead for this research.

Table V.: Vegetation Overall Classification Accuracy/Time Results

	Hadoop+GPU	**GPU**		**CPU**
	PCA	**PCA**	**none**	**PCA**
	Kmeans	**Kmeans**	**Kmeans**	**Kmeans**
Read/Write/Dim. Reduction Time	34	3.067	0	3.134
Classifier time	3.666	3.666	29.893	5.025
Total time (sec)	37.666	6.733	29.893	8.159
Agreement/Accuracy	49.3976	49.3976	48.599	42.673

(a) PCA+K-Means Test

	Hadoop+GPU	**GPU**		**CPU**
	PCA	**PCA**	**none**	**PCA**
	ECHO	**ECHO**	**ECHO**	**ECHO**
Read/Write/Dim. Reduction Time	34	3.055	0	3.122
Classifier time	8	8	24	18
Total time (sec)	42	11.055	24	21.122
Agreement/Accuracy	66.5593	66.5593	73.8862	67.2597

(b) PCA+ECHO Test

	Hadoop+GPU	**GPU**		**CPU**
	LLE	**LLE**	**none**	**LLE**
	Kmeans	**Kmeans**	**Kmeans**	**Kmeans**
Read/Write/Dim. Reduction Time	419	239.1	0	269
Classifier time	1.694	1.694	29.893	1.678
Total time (sec)	420.694	240.794	29.893	270.678
Agreement/Accuracy	49.93	49.93	48.599	40.2774

(c) LLE+K-Means Test

	Hadoop+GPU	**GPU**		**CPU**
	LLE	**LLE**	**none**	**LLE**
	ECHO	**ECHO**	**ECHO**	**ECHO**
Read/Write/Dim. Reduction Time	419	240	0	271
Classifier time	6	6	24	12
Total time (sec)	425	246	24	283
Agreement/Accuracy	70.622	70.622	73.8862	72.135

(d) LLE+ECHO Test

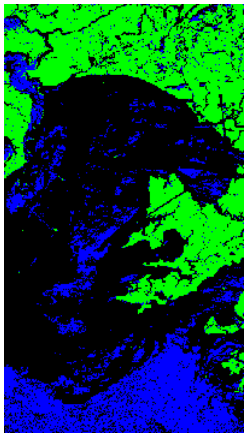
4.2.2 Oil Results

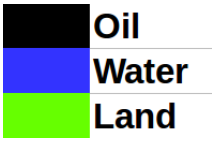
The oil spill scene consisted of 3 distinct classes shown in Figure 25a, where the blue represents water, the green represents land and the black refers to the oil. Figure 25b also displays the classes in a color coded manner. Additionally, keep in mind that PCA was applied to the large and small oil spill image.

4.2.2.1 Confusion Matrices

In table VI we see the results after applying PCA and K-Means on the large image of the oil spill scene. The overall accuracy seems to be low with the highest accuracy being 68% given bu the CPU test case. The detection of oil was fairly good where all the test cases returned an accuracy level greater than 76.60%. The lowest accuracy pertains to the water class with a highest overall accuracy of 5.87% given by the Hadoop+GPU and GPU system. Overall there seems to be no additional surprises when it comes to this dataset since most all of the test cases seem to return similar results for the accuracy, omission and comission error.

In table VII we see the results after applying PCA and K-Means on the large image of the oil spill scene. For this particular scene, we can see that the overall accuracy was not equal to or greater than the results in table VI. This can be due to the supervised nature of the ECHO algorithm. Since ECHO requires training samples, picking the best area to represent a class in an oil spill seems much more difficult. Even though it seems like the detection of oil is difficult with a supervised classifier, it should be pointed out that the ECHO classifier did return a much higher





(a) (b)

Figure 25.: a) Oil Spill Groundtruth. b) Oil Spill Legend.

accuracy percentage for the land class compared to VI. Nonetheless, at this moment, K-Means seems to be doing somewhat better than the ECHO classifier for the large image.

In table VIII we see the results after applying PCA and K-Means on the small image of the oil spill scene. For this image, there was approximately 20 pixels related to water which did not supply enough samples for accuracy checking so the water class has been removed for this table and the for the following tables that use the small oil spill image. Additionally, the results seem rather stable and in the same range for the different test cases because in the end, the K-Means classifier was only able to detect the land and oil much more easily than oil, land and water.

In table IX we see that again we get an overall accuracy within the same range. Again, the results seem rather stable and in the same range for the different test cases because the ECHO classifier was only able to detect the land and oil much more easily than oil, land and water.

For table X and table XI the results seem to resonate within the same range again. In the end, it seems like application of LLE and ECHO were similar or a little worse than the more basic PCA and K-Means. In some instances the K-Means classifer returned better results than ECHO. These results actually seem to be quite the opposite from the vegetation scene. It seems that for the vegetation scene we obtained the highest accuracy with LLE and ECHO while with the oil spill scene, we achieved good results with simply applying PCA and K-Means.

4.2.2.2 Overall Accuracy and Timing Results

Overall it seems like the Hadoop+GPU system does not provide any advantage to the analysis of hyperspectral oil spill scenes as shown in table XII and XIII. Furthermore, in table XIII we also see that the GPU does provide speedup versus the CPU version of LLE. Although it is not a huge difference, the improvement of the GPU implementation of LLE can be improved within the future work. In the end it seems as if LLE and ECHO did not provide much more accuracy compared to PCA and K-Means.

It seems like an unsupervised method of classification seems more appropriate for hyperspectral scenes that pertain to oil spill detection while the ECHO classifier would be much more helpful within the vegetatin scene. Additionally, PCA seems to be a better option for the oil spill images since LLE does not provide results with higher accuracy. Also, the LLE and ECHO fusion seems to work better for vegetation scenes.

Table VI.: Oil Test 1

		Groundtruth		
	Class	land	oil	water
Predicted	land	94557	88486	3034
	oil	87657	551584	80057
	water	7251	44051	5183
Accuracy		49.91%	80.63%	5.87%
Omission Error		50.09%	19.37%	94.13%
Comission Error		49.18%	23.32%	90.82%
Overall Accuracy	67.72%			

(a) Hadoop+GPU and PCA+K-Means

		Groundtruth		
	Class	land	oil	water
Predicted	land	94557	88486	3034
	oil	87657	551584	80057
	water	7251	44051	5183
Accuracy		49.91%	80.63%	5.87%
Omission Error		50.09%	19.37%	94.13%
Comission Error		49.18%	23.32%	90.82%
Overall Accuracy	67.72%			

(b) GPU and PCA+K-Means

		Groundtruth		
	Class	land	oil	water
Predicted	land	118684	108685	1627
	oil	63398	524041	81589
	water	7383	51395	5058
Accuracy		62.64%	76.60%	5.73%
Omission Error		37.36%	23.40%	94.27%
Comission Error		48.17%	21.67%	92.08%
Overall Accuracy	67.35%			

(c) no DR and K-Means

		Groundtruth		
	Class	land	oil	water
Predicted	land	116912	107401	1586
	oil	66990	532904	82444
	water	5563	43816	4244
Accuracy		61.71%	77.90%	4.81%
Omission Error		38.29%	22.10%	95.19%
Comission Error		48.25%	21.90%	92.09%
Overall Accuracy	68.00%			

(d) CPU and PCA+Kmeans

Table VII.: Oil Test 2

		Groundtruth		
	Class	land	oil	water
Predicted	land	151296	142033	2399
	oil	29444	449744	78409
	water	9686	96490	7935
Accuracy		79.45%	65.34%	8.94%
Omission Error		20.55%	34.66%	91.06%
Comission Error		48.84%	19.34%	93.05%
Overall Accuracy	62.95%			

(a) Hadoop+GPU and PCA+ECHO

		Groundtruth		
	Class	land	oil	water
Predicted	land	151296	142033	2399
	oil	29444	449744	78409
	water	9686	96490	7935
Accuracy		79.45%	65.34%	8.94%
Omission Error		20.55%	34.66%	91.06%
Comission Error		48.84%	19.34%	93.05%
Overall Accuracy	62.95%			

(b) GPU and PCA+ECHO

		Groundtruth		
	Class	land	oil	water
Predicted	land	148932	139299	2606
	oil	24660	344048	71184
	water	16834	204920	14953
Accuracy		78.21%	49.99%	16.85%
Omission Error		21.79%	50.01%	83.15%
Comission Error		48.79%	21.79%	93.68%
Overall Accuracy	52.50%			

(c) no DR and PCA+ECHO

		Groundtruth		
	Class	land	oil	water
Predicted	land	133042	124311	1907
	oil	39865	386139	71741
	water	17519	177817	15095
Accuracy		69.87%	56.10%	17.01%
Omission Error		30.13%	43.90%	82.99%
Comission Error		48.68%	22.42%	92.83%
Overall Accuracy	55.23%			

(d) CPU and PCA+ECHO

Table VIII.: Oil Test 3

		Groundtruth	
	Class	land	oil
Predicted	land	964	668
	oil	122	7569
Accuracy		88.77%	91.89%
Omission Error		11.23%	8.11%
Comission Error		40.93%	1.59%
Overall Accuracy	91.53%		

(a) Hadoop+GPU and PCA+K-Means

		Groundtruth	
	Class	land	oil
Predicted	land	964	668
	oil	122	7569
Accuracy		88.77%	91.89%
Omission Error		11.23%	8.11%
Comission Error		40.93%	1.59%
Overall Accuracy	91.53%		

(b) GPU and PCA+K-Means

		Groundtruth	
	Class	land	oil
Predicted	land	1025	676
	oil	57	7602
Accuracy		94.73%	91.83%
Omission Error		5.27%	8.17%
Comission Error		39.74%	0.74%
Overall Accuracy	92.17%		

(c) no DR and K-Means

		Groundtruth	
	Class	land	oil
Predicted	land	947	673
	oil	141	7524
Accuracy		87.04%	91.79%
Omission Error		12.96%	8.21%
Comission Error		41.54%	1.84%
Overall Accuracy	91.23%		

(d) CPU and PCA+Kmeans

Table IX.: Oil Test 4

		Groundtruth	
	Class	land	oil
Predicted	land	1051	791
	oil	82	8049
Accuracy		92.76%	91.05%
Omission Error		7.24%	8.95%
Comission Error		42.94%	1.01%
Overall Accuracy	91.25%		

(a) Hadoop+GPU and PCA+ECHO

		Groundtruth	
	Class	land	oil
Predicted	land	1051	791
	oil	82	8049
Accuracy		92.76%	91.05%
Omission Error		7.24%	8.95%
Comission Error		42.94%	1.01%
Overall Accuracy	91.25%		

(b) GPU and PCA+ECHO

		Groundtruth	
	Class	land	oil
Predicted	land	1073	843
	oil	62	8002
Accuracy		94.54%	90.47%
Omission Error		5.46%	9.53%
Comission Error		44.00%	0.77%
Overall Accuracy	90.93%		

(c) no DR and ECHO

		Groundtruth	
	Class	land	oil
Predicted	land	1021	676
	oil	114	8169
Accuracy		89.96%	92.36%
Omission Error		10.04%	7.64%
Comission Error		39.84%	1.38%
Overall Accuracy	92.08%		

(d) CPU and PCA+ECHO

Table X.: Oil Test 5

		Groundtruth	
	Class	land	oil
Predicted	land	781	587
	oil	288	7525
Accuracy		73.06%	92.76%
Omission Error		26.94%	7.24%
Comission Error		42.91%	3.69%
Overall Accuracy	90.47%		

(a) Hadoop+GPU and LLE+K-Means

		Groundtruth	
	Class	land	oil
Predicted	land	781	587
	oil	288	7525
Accuracy		73.06%	92.76%
Omission Error		26.94%	7.24%
Comission Error		42.91%	3.69%
Overall Accuracy	90.47%		

(b) GPU and LLE+K-Means

		Groundtruth	
	Class	land	oil
Predicted	land	1025	676
	oil	57	7602
Accuracy		94.73%	91.83%
Omission Error		5.27%	8.17%
Comission Error		39.74%	0.74%
Overall Accuracy	92.17%		

(c) no DR and K-Means

		Groundtruth	
	Class	land	oil
Predicted	land	624	480
	oil	466	7544
Accuracy		57.25%	94.02%
Omission Error		42.75%	5.98%
Comission Error		43.48%	5.82%
Overall Accuracy	89.62%		

(d) CPU and LLE+Kmeans

Table XI.: Oil Test 6

		Groundtruth	
	Class	land	oil
Predicted	land	1068	888
	oil	67	7957
Accuracy		94.10%	89.96%
Omission Error		5.90%	10.04%
Comission Error		45.40%	0.83%
Overall Accuracy	90.43%		

(a) Hadoop+GPU and LLE+ECHO

		Groundtruth	
	Class	land	oil
Predicted	land	1068	888
	oil	67	7957
Accuracy		94.10%	89.96%
Omission Error		5.90%	10.04%
Comission Error		45.40%	0.83%
Overall Accuracy	90.43%		

(b) GPU and LLE+ECHO

		Groundtruth	
	Class	land	oil
Predicted	land	1073	843
	oil	62	8002
Accuracy		94.54%	90.47%
Omission Error		5.46%	9.53%
Comission Error		44.00%	0.77%
Overall Accuracy	90.93%		

(c) no DR and ECHO

		Groundtruth	
	Class	land	oil
Predicted	land	1088	981
	oil	47	7864
Accuracy		95.86%	88.91%
Omission Error		4.14%	11.09%
Comission Error		47.41%	0.59%
Overall Accuracy	89.70%		

(d) CPU and LLE+ECHO

Table XII.: Oil Overall Classification Accuracy/Time Results: Large Image

	Hadoop+GPU	**GPU**		**CPU**
	PCA	**PCA**	**none**	**PCA**
	Kmeans	**Kmeans**	**Kmeans**	**Kmeans**
Read/Write/Dim. Reduction Time	157	92.1638	0	95.113691
Classifier time	6.334	6.334	112.597	5.847
Total time (sec)	163.334	98.4978	112.597	100.960691
Agreement/Accuracy	67.7151	67.7151	67.3469	67.9995

(a) PCA+K-Means Test

	Hadoop+GPU	**GPU**		**CPU**
	PCA	**PCA**	**none**	**PCA**
	ECHO	**ECHO**	**ECHO**	**ECHO**
Read/Write/Dim. Reduction Time	157	93.1123	0	96.221784
Classifier time	18	18	360	7
Total time (sec)	178	111.1123	360	103.221784
Agreement/Accuracy	62.9473	62.9473	52.503	55.226

(b) PCA+ECHO Test

Table XIII.: Oil Overall Classification Accuracy/Time Results: Small Image

	Hadoop+GPU	**GPU**		**CPU**
	PCA	**PCA**	**none**	**PCA**
	Kmeans	**Kmeans**	**Kmeans**	**Kmeans**
Read/Write/Dim. Reduction Time	38	1.052	0	1.331
Classifier time	3.859	3.859	76.817	4.063
Total time (sec)	41.859	4.911	76.817	5.394
Agreement/Accuracy	91.53	91.53	92.17	91.23

(a) PCA+K-Means Test

	Hadoop+GPU	**GPU**		**CPU**
	PCA	**PCA**	**none**	**PCA**
	ECHO	**ECHO**	**ECHO**	**ECHO**
Read/Write/Dim. Reduction Time	38	1.1032	0	1.1241
Classifier time	15	15	31	8
Total time (sec)	53	15	31	8
Agreement/Accuracy	91.25	91.25	90.93	92.08

(b) PCA+ECHO Test

	Hadoop+GPU	**GPU**		**CPU**
	LLE	**LLE**	**none**	**LLE**
	Kmeans	**Kmeans**	**Kmeans**	**Kmeans**
Read/Write/Dim. Reduction Time	2465	365.1	0	383.5
Classifier time	1.84	1.84	76.817	1.821
Total time (sec)	2466.84	366.94	76.817	385.321
Agreement/Accuracy	90.47	90.47	92.17	89.62

(c) LLE+K-Means Test

	Hadoop+GPU	**GPU**		**CPU**
	LLE	**LLE**	**none**	**LLE**
	ECHO	**ECHO**	**ECHO**	**ECHO**
Read/Write/Dim. Reduction Time	2465	364.2	0	385.1
Classifier time	9	9	31	9
Total time (sec)	2474	373.2	31	394.1
Agreement/Accuracy	90.43	90.43	90.93	89.7

(d) LLE+ECHO Test

4.2.3 Program Runtime Comparison

The following tests were run between the initial CPU version of the PCA and LLE algorithms. From the previous tables we see that the GPU provided the most speedup particularly for the LLE algorithm, therefore we can assume that the GPU versions of PCA and LLE will run faster than following CPU tests. We simply apply these results to compare the initial programs that were written for PCA/LLE and the improved versions that made the programs more efficient. The first graph seen in Figure 26 was set to retain one dimension at the end. These variables were set only to show a proof of concept. The variable that was adjusted is the number of nearest neighbors (KNN), and they were set from 1 to 100. We can see that as the number of KNN's increased, the time it took to run the initial version of the program takes a much longer time compared to the improved program. It seems that as the number of KNN increases, then the time it takes to run the initial program increases in an exponential growth. In comparison, the improved program seems to be increasing in a somewhat linear manner.

In the figure 27, we compare the PCA time where dimensions kept ranges from 1 to 100 bands. Again, we can see that the improved program time is still under the initial program time, even though the difference in time is not as great as the two programs from the LLE algorithm.

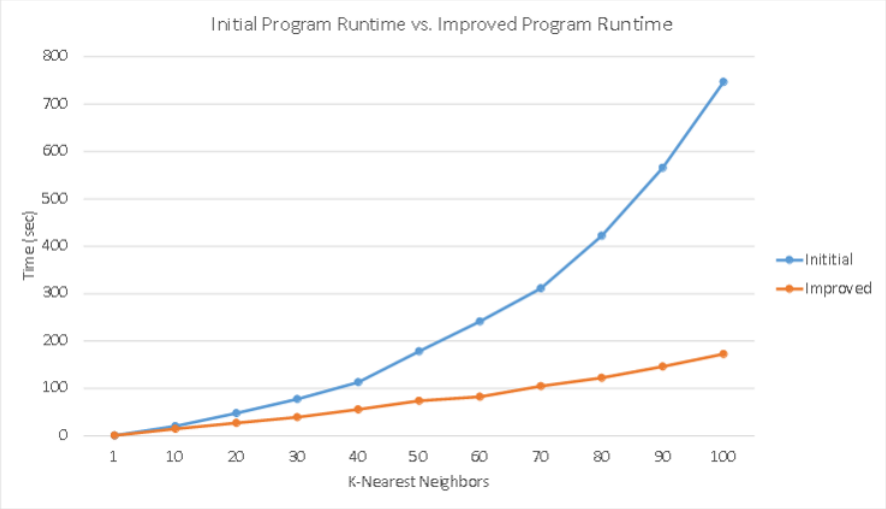


Figure 26.: LLE Program Time Comparison.

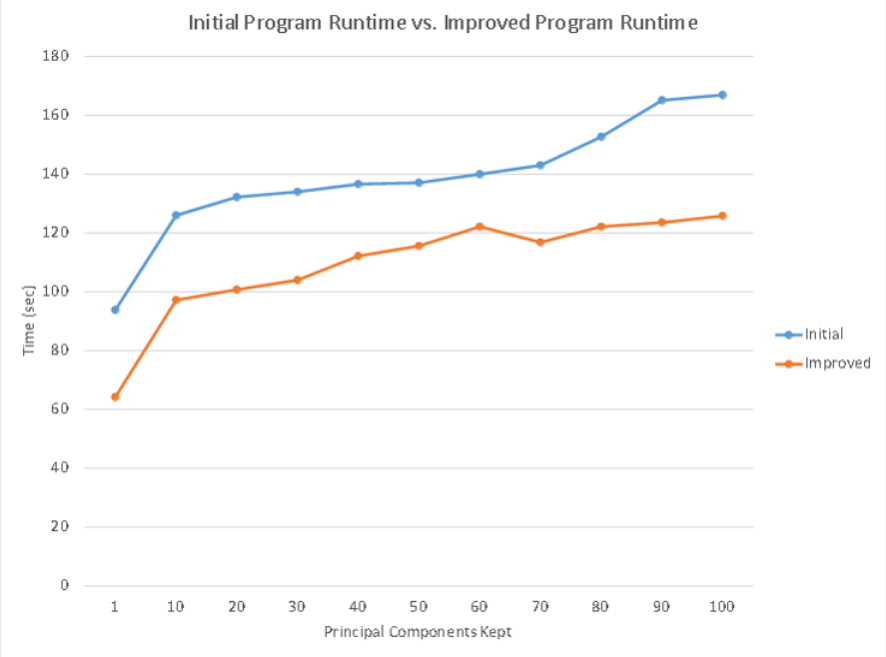


Figure 27.: PCA Program Time Comparison.

CHAPTER 5

CONCLUSION

5.1 Summary

To summarize, this research dealt with the creation of linear and non-linear feature extraction algorithms to run on a Hadoop and GPU system. The images that were being dimensionally reduced were hyperspectral images. One of the tested images dealt with a vegetation scene and the second image dealt with an oil spill scene. After successfully reducing the dimensions of the original hyperspectral image, then we applied two classifiers: K-means and ECHO.

Then this research compared classification results and timing results between the Hadoop+GPU system and tested it against the following test cases: the CPU and GPU test case, a CPU test case and a test case where no dimensional reduction was applied.

5.2 Contribution

To reiterate, my contribution consists of two distinct parts. The first part is the creation of two programs, an implementation of PCA and LLE, and their ability to perform faster than their non-optimized approach with the use of Lanczos and Arnoldis iterative algorithms to obtain eigenvectors of the smallest and highest magnitude. The second part is a thorough comparison of PCA,LLE,K-Means and ECHO involving the time it took to run the feature extraction classification accuracy by im-

plementing programs that run on the three different test cases: Hadoop and GPU, GPU and CPU (one more test case used, that does not require dimensional reduction, was the classification of the raw hyperspectral image).

5.3 Successful Parts

The successful parts for this research include the creation of two enhanced programs: an implementation of PCA and LLE. Both program were able to perform faster than their initial counterparts. PCA was able to run faster due to Lanczos iterative algorithm for eigenvalue/vector extraction. LLE was improved by using Arnoldis iterative algorithm and by removing a portion of the initial program that ran in $O(n^2)$.

5.4 Unsuccessful Parts

In the end, the Hadoop+GPU system did not prove to be faster than its GPU or CPU counterparts. There seems to be too much communication overhead between the mapping, the reduction and the communication between Hadoop and the GPU. Additionally, even though the Hadoop+GPU system provided a speedup that would have been faster than its standard Hadoop counterpart, it still was not enough to provide an accelerated program that reduced the dimensions from a hyperspectral image.

CHAPTER 6

FUTURE WORK

The next logical step for this work is to make it open source and allow developers to contribute to this open source project. Contributions to this work can include other dimensionality reduction methods such as statistical methods or other manifold learning methods. Essentially, this base system can serve as an open source toolbox for hyperpsectral image analysis. Also, it would also be ideal to extend the Hadoop+GPU implementation to run on a multi-node cluster in order to compare the time difference to the single node machine.

Additionally, this system can also be extended to other parallel programming paradigms such as the implementation of a Message Passing Interface (MPI) program or OpenMP. The extensions of this research are many, but the applications must be available to the remote sensing community as an open source project.

REFERENCES

[1] *4. The Implicitly Restarted Arnoldi Method.* ch. 4, pp. 43–66.

[2] Open Source Geospatial Foundation. Geospatial Data Abstraction Library, 2015. http://www.gdal.org/.

[3] Purdue Research Foundation. A Freeware Multispectral Image Data Analysis System, 2015. https://engineering.purdue.edu/ biehl/MultiSpec.

[4] Alam, M., and Sidike, P. Trends in oil spill detection via hyperspectral imaging. In *Electrical Computer Engineering (ICECE), 2012 7th International Conference on* (Dec 2012), pp. 858–862.

[5] Basaran, C., and Kang, K.-D. Grex: An efficient mapreduce framework for graphics processing units. *Journal of Parallel and Distributed Computing 73*, 4 (2013), 522 – 533.

[6] Belkin, M., and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput. 15*, 6 (June 2003), 1373–1396.

[7] Bernabe, S., López, S., Plaza, A., and Sarmiento, R. GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis. *IEEE Geosci. Remote Sensing Lett. 10*, 2 (2013), 221–225.

[8] Boggs, T. Spectral Python, 2014. http://www.spectralpython.net/index.html.

[9] Boiman, O. In defense of nearest-neighbor based image classification.

[10] Cawley, G. C., and Talbot, N. L. C. Preventing over-fitting during model selection via bayesian regularisation of the hyper-parameters. *J. Mach. Learn. Res. 8* (Dec. 2007), 841–861.

[11] Chi, J., and Crawford, M. Selection of landmark points on nonlinear manifolds for spectral unmixing using local homogeneity. *Geoscience and Remote Sensing Letters, IEEE 10*, 4 (July 2013), 711–715.

[12] Clark, R. N., Swayze, G. A., Leifer, I., Livo, K. E., Kokaly, R., Hoefen, T., Lundeen, S., Eastwood, M., Green, R. O., Pearson, N., Sarture, C., McCubbin, I., Roberts, D., Bradley, E., Steele, D., Ryan, T., Dominguez, R., and AVIRIS Team. A method for quantitative mapping of thick oil spills using imaging spectroscopy: U.s. geological survey open-file report 2010âĂŞ1167. 51 p.

[13] Continuum Analytics. Numbapro, 2015. http://docs.continuum.io/numbapro/.

[14] de Ridder, D., and Duin, R. P. Locally linear embedding for classification, 2002.

[15] Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[16] Elteir, M., Lin, H., chun Feng, W., and Scogland, T. Streammr:

An optimized mapreduce framework for amd gpus. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on* (Dec 2011), pp. 364–371.

[17] Fang, W., He, B., Luo, Q., and Govindaraju, N. Mars: Accelerating mapreduce with graphics processors. *Parallel and Distributed Systems, IEEE Transactions on 22*, 4 (April 2011), 608–620.

[18] Farivar, R., Verma, A., Chan, E., and Campbell, R. Mithra: Multiple data independent tasks on a heterogeneous resource architecture. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on* (Aug 2009), pp. 1–10.

[19] Fauvel, M., Chanussot, J., and Benediktsson, J. A. Kernel principal component analysis for the classification of hyperspectral remote sensing data over urban areas. *EURASIP J. Adv. Signal Process 2009* (Jan. 2009), 11:1–11:14.

[20] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. *SIGOPS Oper. Syst. Rev. 37*, 5 (Oct. 2003), 29–43.

[21] GIS Geography. Image classification techniques in remote sensing, 2015.

[22] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[23] Gover, M. Comparative review of applied linear algebra, 3rd ed., by b. noble

and j.w. daniel and linear algebra and its applications, 3rd ed., by g. strang. *Linear Algebra and its Applications 118*, 0 (1989), 159 – 161.

[24] Han, T., and Goodenough, D. Nonlinear feature extraction of hyperspectral data based on locally linear embedding (lle). In *Geoscience and Remote Sensing Symposium, 2005. IGARSS '05. Proceedings. 2005 IEEE International* (July 2005), vol. 2, pp. 1237–1240.

[25] Heylen, R., Burazerovic, D., and Scheunders, P. Non-linear spectral unmixing by geodesic simplex volume maximization. *Selected Topics in Signal Processing, IEEE Journal of 5*, 3 (June 2011), 534–542.

[26] Kettig, R., and Landgrebe, D. Classification of multispectral image data by extraction and classification of homogeneous objects. *Geoscience Electronics, IEEE Transactions on 14*, 1 (Jan 1976), 19–26.

[27] Kim, W. *Manifold Learning for Robust Classification of Hyperspectral Data.* PhD thesis, Purdue University, West Lafayette, Indiana, 2011.

[28] KlÃűckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., and Fasih, A. Pycuda and pyopencl: A scripting-based approach to {GPU} run-time code generation. *Parallel Computing 38*, 3 (2012), 157 – 174.

[29] Kokaly, R. F., Couvillion, B. R., Holloway, J. M., Roberts, D. A., Ustin, S. L., Peterson, S. H., Khanna, S., and Piazza, S. C. Spectroscopic remote sensing of the distribution and persistence of oil from the deep-

water horizon spill in barataria bay marshes. *Remote Sensing of Environment 129*, 0 (2013), 210 – 230.

[30] Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y. D., and Moon, B. Parallel data processing with mapreduce: A survey. *SIGMOD Rec. 40*, 4 (Jan. 2012), 11–20.

[31] Lehoucq, R. B., and Sorensen, D. C. Deflation techniques for an implicitly restarted arnoldi iteration. *SIAM J. Matrix Anal. Appl. 17*, 4 (Oct. 1996), 789–821.

[32] Lunga, D., Prasad, S., Crawford, M., and Ersoy, O. Manifold-learning-based feature extraction for classification of hyperspectral data: A review of advances in manifold learning. *Signal Processing Magazine, IEEE 31*, 1 (Jan 2014), 55–66.

[33] Matam, K., and Kothapalli, K. Gpu accelerated lanczos algorithm with applications. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on* (March 2011), pp. 71–76.

[34] Meigs, A., Otten, L., and Cherezova, T. Ultraspectral imaging: A new contribution to global virtual presence. *Aerospace and Electronic Systems Magazine, IEEE 23*, 10 (Oct 2008), 11–17.

[35] Melgani, F., and Bruzzone, L. Classification of hyperspectral remote sensing images with support vector machines. *Geoscience and Remote Sensing, IEEE Transactions on 42*, 8 (Aug 2004), 1778–1790.

[36] NASA-Jet Propulsion Laboratory. Aviris concept, 2007.

[37] Nokia Corporation. Disco Documentation, 2014. http://discoproject.org/.

[38] Paz, A., and Plaza, A. Cluster versus gpu implementation of an orthogonal target detection algorithm for remotely sensed hyperspectral images. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on* (Sept 2010), pp. 227–234.

[39] Plaza, A. J. Parallel techniques for information extraction from hyperspectral imagery using heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing 68*, 1 (2008), 93 – 111. Parallel Techniques for Information Extraction.

[40] Plaza, J., PÃĽrez, R., Plaza, A., MartÃŋnez, P., and Valencia, D. Mapping oil spills on sea water using spectral mixture analysis of hyperspectral image data. vol. 5995, pp. 599509–599509–8.

[41] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. Evaluating mapreduce for multi-core and multiprocessor systems. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on* (Feb 2007), pp. 13–24.

[42] Richards, J. A., and Jia, X. *Remote Sensing Digital Image Analysis: An Introduction*, 3rd ed. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[43] Robila, S., and Busardo, G. Hyperspectral data processing in a high performance computing environment: A parallel best band selection algorithm. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on* (May 2011), pp. 1424–1431.

[44] Roweis, S. T., and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE 290* (2000), 2323–2326.

[45] Roweis, S. T., and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE 290* (2000), 2323–2326.

[46] Saad, Y. *Numerical methods for large eigenvalue problems*. SIAM, 2011.

[47] Schölkopf, B., Smola, A., and Müller, K.-R. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput. 10*, 5 (July 1998), 1299–1319.

[48] Shirahata, K., Sato, H., and Matsuoka, S. Hybrid map task scheduling for gpu-based heterogeneous clusters. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on* (Nov 2010), pp. 733–740.

[49] Silva, V. D., and Tenenbaum, J. B. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 15* (2003), MIT Press, pp. 705–712.

[50] Stuart, J. A., and Owens, J. D. Multi-gpu mapreduce on gpu clusters. In

Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (Washington, DC, USA, 2011), IPDPS '11, IEEE Computer Society, pp. 1068–1079.

[51] Sun, W., Liu, C., Shi, B., and Li, W. Improved l-isomap for classification of hyperspectral imagery via vector quantization. In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2012 4th Workshop on* (June 2012), pp. 1–4.

[52] Sykas, D., Karathanassi, V., Andreou, C., and Kolokoussis, P. Oil spill thickness estimation using unmixing methods. In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2011 3rd Workshop on* (June 2011), pp. 1–4.

[53] Tenenbaum, J. B., de Silva, V., and Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319.

[54] The Apache Software Foundation. Hadoop, 2015.

[55] Thenkabail, P. S. Optimal hyperspectral narrowbands for discriminating agricultural crops. *Remote Sensing Reviews 20*, 4 (2001), 257–291.

[56] Thenkabail, P. S., Enclona, E. A., Ashton, M. S., and Meer, B. V. D. Accuracy assessments of hyperspectral waveband performance for vegetation analysis applications. *Remote Sensing of Environment 91*, 3âĂŞ4 (2004), 354 – 376.

[57] Vantaram, S. R., and Saber, E. Survey of contemporary trends in color image segmentation. *Journal of Electronic Imaging 21*, 4 (2012), 040901–1–040901–28.

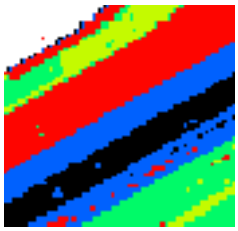
[58] White, T. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[59] Zabalza, J., Ren, J., Yang, M., Zhang, Y., Wang, J., Marshall, S., and Han, J. Novel folded-pca for improved feature extraction and data reduction with hyperspectral imaging and sar in remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing 93*, 0 (2014), 112 – 122.

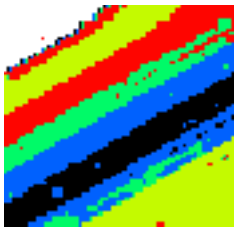
[60] Zhang, Z., and Zha, H. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. *SIAM Journal of Scientific Computing 26* (2002), 313–338.

APPENDIX A

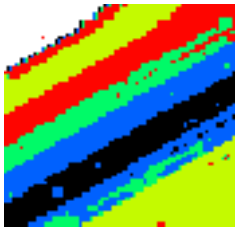
CLASSIFIED IMAGES FOR VEGETATION SCENE



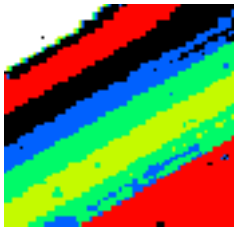
(a) CPU version: PCA+ K-Means.



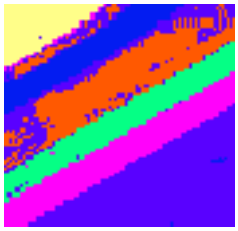
(b) GPU version: PCA+ K-Means.



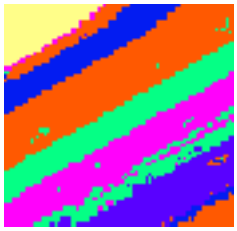
(c) Hadoop version: PCA+ K-Means.



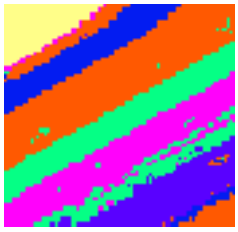
(d) K-means on raw image.



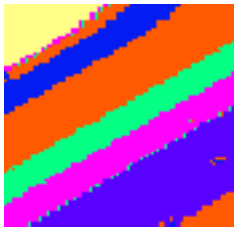
(e) CPU version: PCA+ ECHO.



(f) GPU version: PCA+ ECHO.

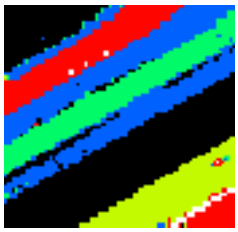


(g) Hadoop version: PCA+ ECHO.

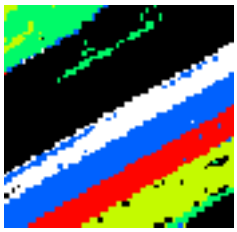


(h) ECHO on raw image.

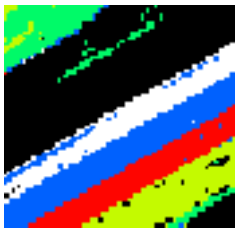
Figure 28.: Test 1 and 2: Vegetation Classification.



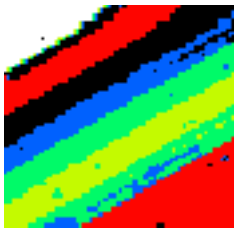
(a) CPU version: LLE+ K-Means.



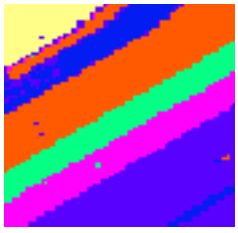
(b) GPU version: LLE+ K-Means.



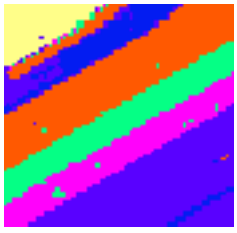
(c) Hadoop version: LLE+ K-Means.



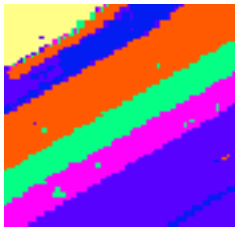
(d) K-means on raw image.



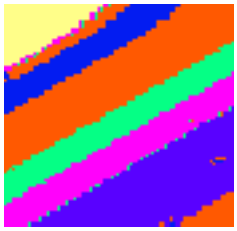
(e) CPU version: LLE+ ECHO.



(f) GPU version: LLE+ ECHO.



(g) Hadoop version: LLE+ ECHO.

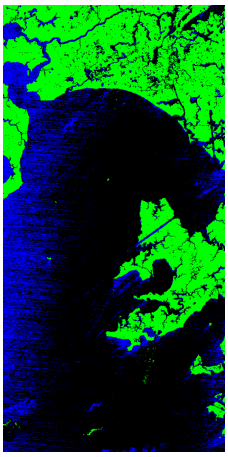


(h) ECHO on raw image.

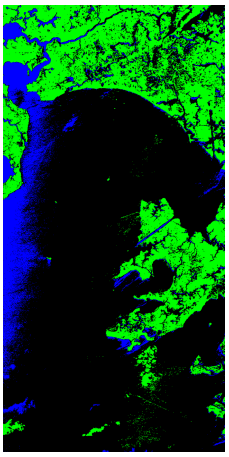
Figure 29.: Test 3 and 4: Vegetation Classification.

APPENDIX B

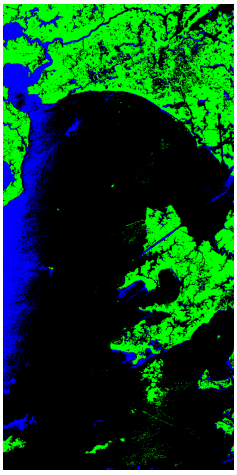
CLASSIFIED IMAGES FOR OIL SPILL SCENE



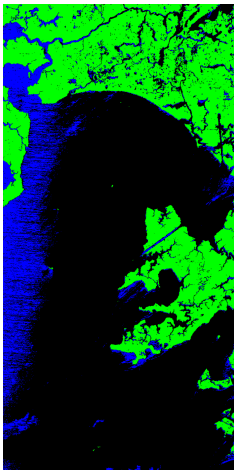
(a) CPU version: PCA+ K-Means.



(b) GPU version: PCA+ K-Means.

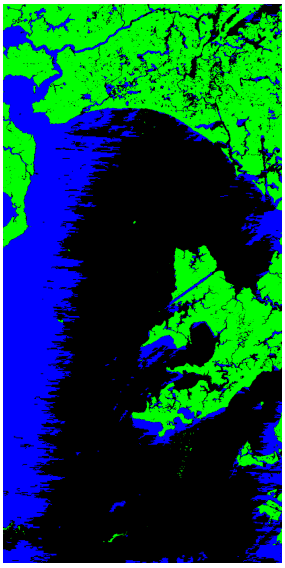


(c) Hadoop version: PCA+ K-Means.

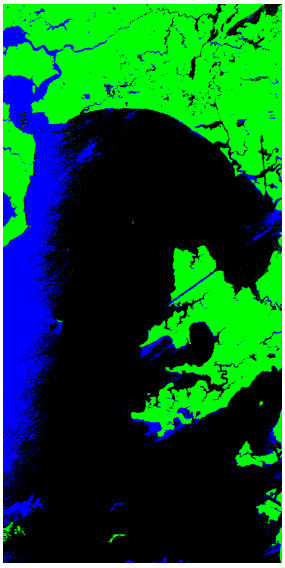


(d) K-means on raw image.

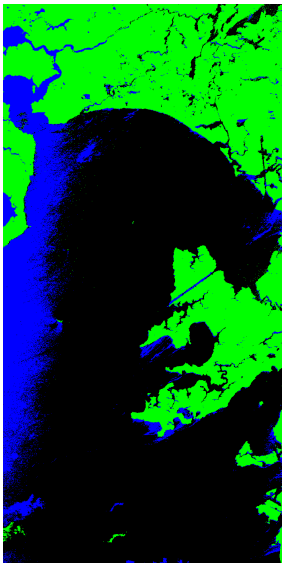
Figure 30.: Test 1: Oil Classification.



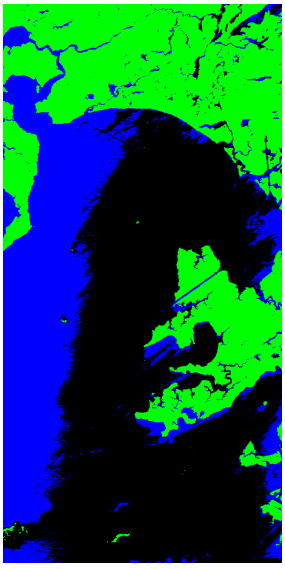
(a) CPU version: PCA+ ECHO.



(b) GPU version: PCA+ ECHO.

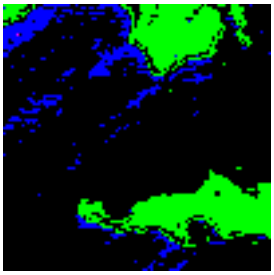


(c) Hadoop version: PCA+ ECHO.

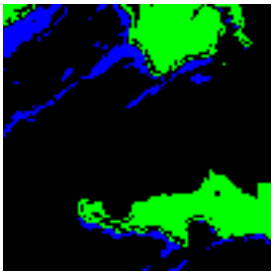


(d) ECHO on raw image.

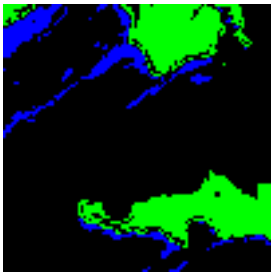
Figure 31.: Test 2: Oil Classification.



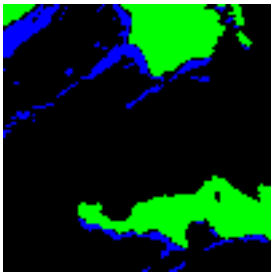
(a) CPU version: PCA+ K-Means.



(b) GPU version: PCA+ K-Means.



(c) Hadoop version: PCA+ K-Means.



(d) K-Means on raw image.

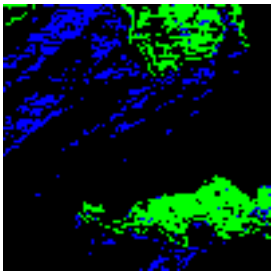
(e) CPU version: PCA + ECHO.

(f) GPU version: PCA + ECHO.

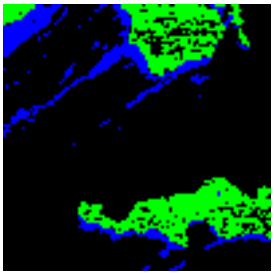
(g) Hadoop version: PCA + ECHO.

(h) ECHO on raw image.

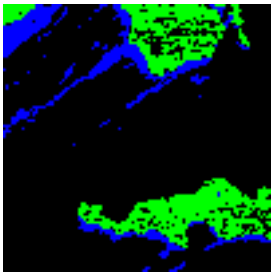
Figure 32.: Test 3 and 4: Oil Classification.



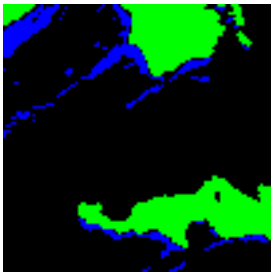
(a) CPU version: LLE+ K-Means.



(b) GPU version: LLE+ K-Means.



(c) Hadoop version: LLE+ K-Means.



(d) K-Means on raw image.

(e) CPU version: LLE + ECHO.

(f) GPU version: LLE + ECHO.

(g) Hadoop version: LLE + ECHO.

(h) ECHO on raw image.

Figure 33.: Test 5 and 6: Oil Classification.