REDUCING WIDE-AREA SATELLITE DATA TO CONCISE SETS FOR MORE EFFICIENT TRAINING AND TESTING OF LAND-COVER CLASSIFIERS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Tommy Y. Chang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Avinash Kak, Chair

School of Electrical and Computer Engineering

Dr. Charles Bouman

School of Electrical and Computer Engineering

Dr. Alexander Quinn

School of Electrical and Computer Engineering

Dr. Tanmay Prakash

School of Electrical and Computer Engineering

Approved by:

Dr. Pedro Irazoqui

Head of the School of Electrical and Computer Engineering

To my grandmother

ACKNOWLEDGMENTS

I would like to thank various people who have helped me tremendously throughout my research and Ph.D program. First of all, I am grateful to Prof. Avinash Kak for giving me the opportunity to conduct this research. The guidance and feedback I have received from him are invaluable. I would also like to give special thanks to Noha Elfiky, German Holguin, Bharath Comandur, Tanmay Prakash, Henry Medeiros, and Johnny Park for the fruitful discussions and collaborations at various stages of my research and to Prof. Alexander Quinn for his very helpful guidelines on research and presentation.

I must also thank my labmates for their friendship and their help with my preparation for the dreadful qualification exam. They include, but are not limited to, Alex Gheith, Nader Alawadi, Dave Kim, Shivani Rao, and Josh Zapf.

Last but not least, I am indebted and thankful to my wife, Jung Suh, for her understanding and patience and to my parents, Shing-Hsiung Chang and Yueh-Ai Chang, for their endless encouragement and support.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Chang, Tommy Y. Ph.D., Purdue University, May 2019. Reducing Wide-Area Satellite Data to Concise Sets for More Efficient Training and Testing of Land-Cover Classifiers. Major Professor: Avinash C. Kak.

Obtaining an accurate estimate of a land-cover classifier's performance over a wide geographic area is a challenging problem due to the need to generate the ground truth that covers the entire area that may be thousands of square kilometers in size. The current best approach constructs a testing dataset by drawing samples randomly from the entire area — with a human supplying the true label for each such sample — with the hope that the selections thus made statistically capture all of the data diversity in the area. A major shortcoming of this approach is that it is difficult for a human to ensure that the information provided by the next data element chosen by the random sampler is non-redundant with respect to the data already collected. In order to reduce the annotation burden, it makes sense to remove any redundancies from the entire dataset before presenting its samples to a human for annotation. This dissertation presents a framework that uses a combination of clustering and compression to create a concise-set representation of the land-cover data for a large geographic area. Whereas clustering is achieved by applying Locality Sensitive Hashing (LSH) to the data elements, compression is achieved through choosing a single data element to represent a given cluster. This framework reduces the annotation burden on the human and makes it more likely that the human would persevere during the annotation stage. We validate our framework experimentally by comparing it with the traditional random sampling approach using WorldView2 satellite imagery.

1. INTRODUCTION

Constructing training and testing datasets for land-cover classifiers that are effective over large geographic areas — areas that may be as large as tens of thousands of square kilometers — places a large burden on the human annotators for supplying the ground truth. The most commonly used approach for creating the datasets in such cases consists of drawing samples randomly in a uniform manner from the entire geographic region. More sophisticated approaches use a random sampler based on the Metropolis-Hastings algorithm [1].

What is significant is that even with the best random samplers, the datasets that are generated tend to be highly redundant. This is for the basic reason that, as each new sample is shown to a human annotator for its true label, it is virtually impossible for the human to remember all of the previously seen samples in order to determine whether the new sample is merely redundant vis-a-vis all the samples collected previously, or whether it really adds additional diversity to the data already collected. Note that this problem is exacerbated by the fact that the human-computer interaction involved in creating a full dataset may last a long time.

So if it is impossible to avoid redundancy in the datasets, the reader may ask if that is really such a bad thing. Most datasets that are out there for the training and testing of machine learning algorithms carry no guarantee of being non-redundant. Obviously, what is most important for a dataset is whether or not it captures all of the diversity in the data as it exists in the real world. As long as this diversity constraint is satisfied, the only price to pay for any redundancy in the data is that it may take longer to train and test a classifier — but the classifier performance would not be impacted by the redundancies.

Unfortunately, the consequences of redundancies in the datasets collected from large geographic regions tend to be not so benign. In light of the challenges created

by the prolonged human-computer interaction, it is difficult to guarantee that a given redundant dataset would adequately capture the diversity associated with the different classes. And when it is practically impossible to assume that a dataset adequately captures all of the diversity associated with the different classes, any redundancies in the data may result in erroneous bounds on the performance of the classifiers when such data is used for testing them. We illustrate this effect with the following simple example: Let's say we are creating a test dataset for a binary classifier and that 80% of the data points collected happen to fall in a small neighborhood of the same point in the feature space. With such a dataset, regardless of the actual performance of the classifier on a "true" dataset, the computed classifier performance would be controlled by the two numbers, 20% and 80%. The classifier would have no choice but to give the same class label to the 80% of the data. If the class label chosen was correct, the computed performance of the classifier could exceed 80% depending on how the classifier performs on the rest of the data. On the other hand, if the label given to the 80% was incorrect, the computed performance of the classifier could be less than 20%, again depending on how the classifier performs on the rest of the data. What this says is that it is particularly important to eliminate redundancies from the datasets created from large geographic areas for the purpose of designing and testing land-cover classifiers.

Generating a concise representation from potentially hundreds of satellite images covering a large geographical region is made extremely challenging by two reasons: The first is, of course, the sheer volume of the data. The second equally important reason has to do with the fact that a pixel cannot be shown in isolation to a human annotator for eliciting its class label for creating the ground truth. It is now well known that for reliable annotation, humans require both the pixel itself and its immediate surround — which we refer to as its background context. Therefore, any automated algorithm for creating a concise representation for human interaction, must compare the pixels both on the basis of the spectral signatures at the pixels themselves and

on the basis of whatever it takes to represent the background contexts for the pixels.[1] When you include the background data for each pixel, you end up having to deal with voluminous amounts of high-dimensional data.

The sheer size of the dataset and its high-dimensionality mean that we are dealing with what is loosely referred to as a big-data problem. Such problems do not allow for exhaustive pairwise comparison of the data elements for the purpose of automated clustering. And, since the data dimensionality can still be high even after applying dimensional reduction, such problems also do not lend themselves to the use of techniques such KD-trees, SR-trees, and cover trees [2–6] because their time or space complexity degrades exponentially with data dimensionality.

In addition to the issues created by the size of the data and its dimensionality, we must also cope with the fact that comparing pixels on the basis of their spectral signatures and on the basis of their background context are two semantically different actions. That is, it would make no sense to lump both the background and the foreground for each pixel into a single vector representation for creating a concise representation.

Yet another source of complexity arises from the fact that similarity constraints for clustering data are generally not transitive. To explain this point, a data sample A can be similar to another data sample B because the magnitude of the difference between their attribute vectors is below some threshold. And, the data sample B may be similar to another data sample C for the same reason. Yet, A may not be similar to C. That is, if we were to directly measure the magnitude of the difference between the attribute vectors for A and C, it may exceed the threshold being used for establishing similarity. Many clustering algorithms get around this problem by assuming that the

[1] A reader might ask: Why is it not sufficient for concise representations to be created from just the pixels themselves, without the need to also factor in the background context for each pixel? Even for a computer algorithm, pixels considered in isolation can result in their being considered similar when in fact they are highly dissimilar. For example, the multispectral signature for a pixel from a concrete road would be very similar to the signature from any number of other structures on the ground — building rooftops, water towers, bridges, etc. Creating a concise representation from just the pixels, without also including a portion of the background for each pixel, would only create a frustrating experience for the human annotators.

number of clusters, k, into which the data must be partitioned is known a priori. Subsequently, a clustering algorithm must find the optimum partitioning of the data so that, say, the average distance of every data point from the center of the cluster to which the data element is assigned is minimized. Deterministic variants of this approach lead to the k-means and other such algorithms. And the probabilistic variants of the same basic idea result in expectation-minimization sorts of algorithms. One can loosen the need for the a-priori knowledge k by testing for different k until some overall quality metric is satisfied. Unfortunately, big-data problems do not lend themselves to such experimentation. In the absence of such logic, a blind application of similarity checking, no matter how it is actually enforced, is highly likely to result in all of the image data elements extracted from all the satellite images to form a single similarity neighborhood.

1.1 Primary Contributions

The main contribution of this dissertation is a solution to all of the issues we have outlined above for reducing a large volume of satellite data to relatively small number of similarity neighborhoods in the underlying feature space and representing each such neighborhood by an exemplar data element that the human is asked to annotate. Subsequently, all of the data elements within any given similarity neighborhood acquire the annotation of its exemplar.

With a data abstraction we refer to as an image patch, we represent each pixel by its foreground spectral signature and a high-dimensional vector that captures its background context. Subsequently, we first reduce the data dimensionality of the background context and then use Locality Sensitive Hashing to compare the pixels on the basis of just the background characterizations. That is followed by refining the clusters obtained with foreground comparisons based on the spectral signatures at the pixels themselves. However, before we carry out the foreground comparisons, we get around the difficulties created by the non-transitivity of the background similarity

constraint by associating with each patch a *similarity neighborhood*, which is the set all other patches that directly satisfy the similarity condition with respect to the former patch. These similarity neighborhoods are converted into what we call a *similarity graph*. A concise set is derived from the similarity graph after the enforcement of the foreground similarity constraint.

1.2 Organization of the Dissertation

In the rest of this dissertation, Chapter 2 will review the current state of the art in multispectral imaging. We will also review the pre-processing steps used to undistort and enhance the satellite images. In Chapter 3, we will present PMISIR — a dynamic data structure that provides rapid visualization of overlapping satellite images. A unique feature of PIMSIR is its support for understanding data variability at the scale of a large geographical area. In Chapter 4, we will investigate the intrinsic dimensionality of satellite image data of an ROI (Region of Interest). Starting with Chapter 5, we will first define the notation of a concise-set representation of the satellite image data and subsequently describe the various stages for its construction. Toward the end of that chapter, we will illustrate how these stages are put together to create a complete processing pipeline. We will validate the effectiveness of our concise-set representation in Chapter 6 by comparing its performance with the traditional random sampling approach. Chapter 7 will introduce cloud computing and present a solution to the task of processing massive datasets that can only be handled by a cluster of computers working together. Finally, Chapter 8 will conclude and present possible future work.

2. MULTISPECTRAL SATELLITE IMAGERY AND DATA ABSTRACTIONS

In this chapter, we first review multispectral systems. Then, we introduce the different data abstractions with regard to the aggregation of the pixels in the satellite images. We will refer back to these data abstractions extensively throughout the rest of the dissertation.

2.1 Multispectral Imagery – A Brief Review

In general, multispectral systems have two specifications – spectral resolution and spatial resolution. Spectral resolution is characterized by the number of bands the frequency range in each band. Spatial resolution, on the other hand, is usually specified by the pixel size on the earth surface. For example, a modern multispectral system may have 8 spectral bands and 0.5 meter per pixel.

There are two types of imagery data: panchromatic and multispectral. These two data are usually recorded simultaneously[1] but they differ in spectral and spatial resolutions. For example, Landsat-1 [8], the earliest multispectral satellite system first introduced in 1972, has four spectral bands covering a spectral range of 500 to 1100 nanometers. Unlike the later modern satellite systems, Landsat-1 does not have a panchromatic sensor. Instead, it has three independent television-like cameras collectively referred to as the Return Beam Vidicon (RBV) sensor. Both the RBV sensor and the multispectral sensor onboard Landsat-1 have the same spatial resolutions of 80 meters. Table 2.1 summarizes the Landsat-1 system.

[1]WorldView2 satellite has a small time delay between its panchromatic and multispectral data [7].

Table 2.1.
The Landsat1 Multispectrum System [8].

Sensor	Spectral resolution [nm]	Spatial Resolution [m]
RBV (Bands 1-3)	480 to 570, 580 to 680, and 700 to 830	80
Green (band 4)	500 to 600	80
Red (band 5)	600 to 700	80
Near IR (band 6)	700 to 800	80
Near IR (band 7)	800 to 1100	80

2.1.1 Modern Multispectral Systems

Modern multispectral sensors are increasingly airborne-based and superior in spatial resolution [9]. The WorldView-3 satellite [10], launched in mid-2014 has a 16-band multispectral sensor and a panchromatic sensor. The spatial resolutions for the multispectral and panchromatic data are 1.24 meters and 0.31 meters, respectively.

For modern airborne-based systems, the UltraCam family of sensors [11] are equipped with a four-band multispectral (Red, Green, Blue, and Nadir IR) sensor and a panchromatic sensor. Unlike satellite-based systems, airborne-based systems can easily achieve spatial resolution less than 0.10 meters. This is the case because the spatial resolution is determined by the flying height, typically specified in terms of altitude needed to achieve 0.10 meters ground resolution. Table 2.2 summarizes the two types of multispectral sensors.

2.1.2 Satellite Scenes

In the satellite terminology, a "satellite scene" is an area on earth captured by the satellite. Depending on the sensor's spatial resolution and its field of view, a satellite scene may require a large amount of digital storage space. For example, it is common to have a satellite scene covering 900 km^2 (30km by 30km) of area and

Table 2.2.
Current state-of-the-art multi-spectrum systems

System	Type	Year	Spectral Bands	Spatial Resolution [m]
WorldView-3	Satellite	2014	16 (400 nm - 2365 nm)	MUL=1.24 PAN=0.31
UltraCam Hawk	Airborne	2014	4 (R,G,B,IR)	MUL≤0.10 PAN≤0.10

Note: Pan = Panchromatic, Mul = Multispectral

taking gigabytes of storage space. Once saved into a storage device, the satellite scene may simply be referred to as a "satellite image" or simply an "image". The size of a satellite image varies. It can contain as many as 120k × 32k of pixels and, depending on the number of bands, take more than 16 gigabytes of computer memory once uncompressed and loaded from the digital storage. Because of its large size, satellite images are typically split into smaller tiles so they can be processed by consumer computers that may not have a lot of memory.

2.1.3 Satellite Data Representation and File Format

Different sensors often have different modes of operation and data representations. For example, WorldView2 data are encoded as either 8-bit or 11-bit values [12]. On the other hand, UltraCam data are 12-bit values [11]. To complicate the matter further, each multispectral system has its own data interpretation and raw data need to be corrected and compensated for various sources of distortion and contamination. In the extreme case, individual sensor element has its own specific calibration and correction. Without these corrections, raw data simply can not be used directly.

After the raw data are pre-processed, they are commonly stored in the Tagged Image File Format (TIFF). TIFF supports more data bands and more data types comparing to other common image file formats such as PPM, PNG, BMP, JPG, etc. All data in a TIFF file must be the same data type (i.e., either 8-bit, 16-bit, 32-bit

or 64-bit) — mixed data types is not allowed. For example, one can not have 8-bit data for band1 and 16-bit data for band2 in a TIFF file.

Because of its generic and versatile format, TIFF is used by just about all Geographic Information System (GIS). Ironically, TIFF does not specify how data should be interpreted or visualized. As a result, GIS software often provide many visualization options for displaying a TIFF image file. For example, when displaying a TIFF file as a color image, the GIS user may need to specify which bands to use for the Red, Green, and Blue channels. The user may also need to adjust the scale so that the resulting color image "look nice" (i.e., not washed out or too dark, etc.). In general, without knowing what and how the data are stored, it can be very difficult and frustrating to display the content of a TIFF file in a meaningful way.

2.1.4 High Spatial Resolution Data

For detecting objects in overhead imagery, it is important to know the image's spatial resolution. Depending on the object of interest, it may be the case that certain objects, such as cross-walk markings and other small objects, simply can not be recognize at low resolution.

One way to get around the low resolution issue is to enhance the multispectral image by applying Panchromatic Sharpening (aka pansharpening). Pansharpening is a process of combining the high-resolution panchromatic image with the corresponding low-resolution multispectral image to create a high-resolution multispectral image. The resulting pansharpened multispectral image has the same spatial resolution as the panchromatic image.

High-resolution images are freely available from map service platforms such as Google Maps and Microsoft Bing Maps. To describe their products and services, map service platforms use a set of predefined spatial resolutions. In particular, the Google Maps API services[2], released in 2005, provides overhead imagery at several predefined

[2] `https://developers.google.com/maps/`

spatial resolutions. Table 2.3 shows a subset of the 24 predefined resolutions. With these definitions, Google makes it easy for software and web applications to query and access overhead imagery at different levels of spatial resolution.

Although Google restricted its free map data to visualization purposes only, it can still be very useful to GIS researchers. For example, a GIS researcher conducting research in road marking detection may use Google Maps services to first visually inspect the road markings at different resolutions and then make an informed decision on the spatial resolution needed for the research.

Table 2.3.
A subset of zoom levels defined by the Google Maps API.

Level	16	17	18	19	20	21	22
Pixel size	2.4m	1.2m	60cm	30cm	15cm	7.5cm	3.7cm

2.1.5 Raw Data Pre-processing

In the satellite terminology, raw data are refereed to as Digital Numbers (DN). These numbers may need to be further processed to in order to be used by off-the-shelf image processing software. In the next five subsections, we review Radiance-to-Reflectance conversion, Panchromatic Sharpening, Orthorectification, Inverse Orthorectification, and Gamma Correction.

Radiance to Reflectance Conversion

Raw data need to be converted into units of energy proportional to the amount of light reflected from the earth surface. This step is especially important when creating a mosaic from multiple tiles, or when analyzing tiles from different sensors. In general, this pre-processing step is complex and takes into account the Earth-Sun configuration as well as the atmosphere effect such as light absorption and scattering

by gas molecules. Instead of modeling the atmospheric effects, a simplified pre-processing step ignores the atmospheric effect and the output is commonly referred to as "top-of-atmosphere reflectance". [12]

Although different satellite systems have different pre-processing steps, the general procedure is as follows: [12]

1. Convert raw data (aka. Digital Numbers (DN)) to Top-of-Atmosphere Radiance (ToAR) made up of three major sources of radiation: unscattered surface-reflected radiation, downwelling surface-reflected skylight, and upwelling path radiance.

2. Convert ToAR from step 1 to either Top-of-Canopy Reflectances (ToCR) or Top-of-Atmosphere Reflectances, which ignores the atmospheric effect.

In general, the conversion equations vary among different sensors and may take different sets of parameters. For the WorldView2 satellite system, the input parameters used are listed below:

- K_{band}: absolute radiometric calibration factor
- $\Delta\lambda_{band}$: effective bandwidth for a given band
- θ_s : solar zenith angle
- Esunλ_{band}: band-averaged solar spectral irradiance at the average Earth-Sun distance.
- UT : universal time the satellite scene was taken
- d_{ES}: Earth-Sun distance at UT

Except for the Esunλ_{band} parameter, which is given in [12], all other parameters can be found or derived from the meta data that come with every WorldView2 satellite image. Top-of-Atmosphere reflectance value is normalized at Earth-Sun distance d_{ES} = 1 AU (Astronomical Units) and at solar zenith angle $\theta_s = 0$ degree. Because the

output values are normalized (i.e., output values are within $[0 \ldots 1]$), they are suitable for fusing, comparing, or combining with other ToA reflectance values derived from other satellite sensors. ToA reflectance is also very important for many multispectral analysis techniques such as Normalized Difference Vegetation Index (NVDI) and Band Ratios [13].

Panchromatic Sharpening

As mentioned previously, pansharpening is a technique that enhances the low-resolution multispectral data with the co-registered high-resolution panchromatic data. There are many pansharpening algorithms. The simplistic method averages the panchromatic value with the multispectral value. In more detail, the multispectral image is first up-sampled to the panchromatic resolution. Let $r_{out}, g_{out}, b_{out}$ be the output red, green, and blue values at a given pixel location and let $p_{in}, r_{in}, g_{in}, b_{in}$ be the input panchromatic and multispectral values. The simplistic method calculates $r_{out}, g_{out}, b_{out}$ by:

$$\begin{aligned} r_{out} &= \frac{r_{in} + p_{in}}{2} \\ g_{out} &= \frac{g_{in} + p_{in}}{2} \\ b_{out} &= \frac{b_{in} + p_{in}}{2} \end{aligned} \tag{2.1}$$

Other more sophisticated algorithms include Weighted Bovey transformation, Intensity-Hue-Saturation transformation, Gram-Schmidt pansharpening, and Principle Component Analysis pansharpening [14]. Perhaps the next simplest pansharpening method is the Bovey transformation:

$$\begin{aligned} p_{est} &= \frac{(r_{in} + g_{in} + b_{in})}{3} \\ r &= \frac{p_{in}}{p_{est}} \\ r_{out} &= r_{in}\ r \\ g_{out} &= g_{in}\ r \\ b_{out} &= b_{in}\ r \end{aligned} \tag{2.2}$$

Fig. 2.1 shows an example of Bovey transformation.

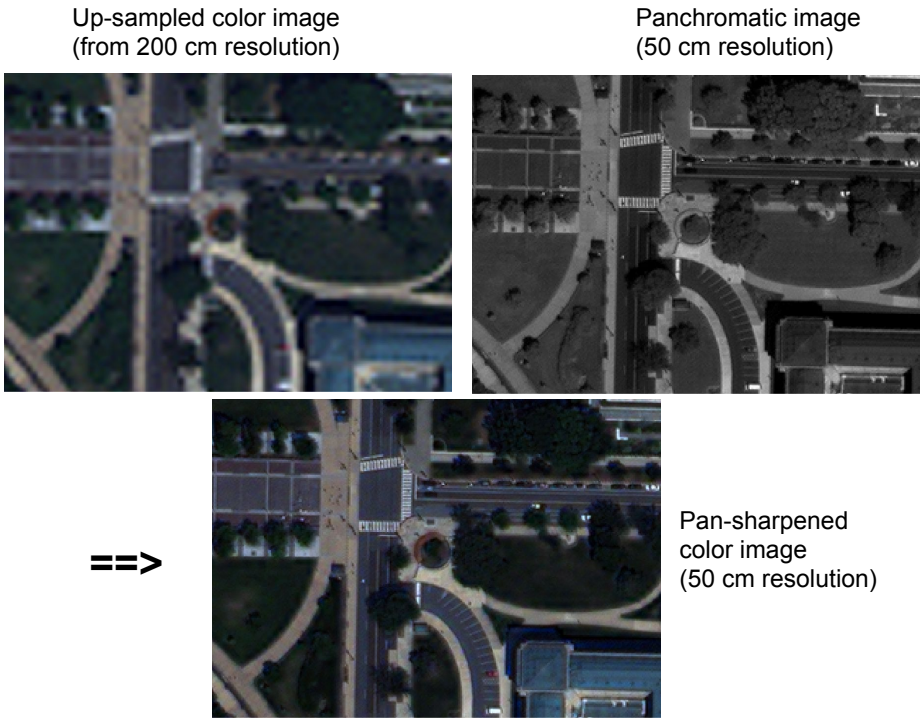


Fig. 2.1. An example of panchromatic sharpening using the Bovey transformation method (Eq. 2.2).

Orthorectification

Orthorectification is the process of transforming raw or ToA corrected satellite image into an orthorectified image such that the pixel locations on the orthorectified image are correctly mapped to a regularly spaced grid of latitude-longitude (lat-long) coordinates. In practice, this process is done indirectly via the RPC (Rational Polynomial Coefficients) equations, as outlined in Fig. 2.2.

The RPC equations [15] transform a point in the world (λ, ϕ, Z) into the corresponding pixel location (S, L) in the image. Here (λ, ϕ) are the longitude and latitude coordinates. Z is the orthometric height in meters and (S, L) stands for

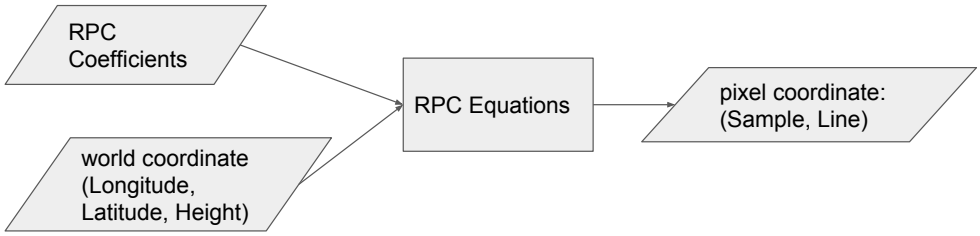


Fig. 2.2. The RPC Camera Model. The RPC equations maps a 3D point in the satellite scene to a 2D pixel in the satellite image.

(Sample, Line) in the satellite terminology. A full RPC camera model consists of 80 coefficients and 10 normalizing parameters. They are described next.

First, the normalized world coordinate (x, y, z) is defined in Eq. 2.3, using six normalizing parameters: $x_o, x_s, y_o, y_s, z_o,$ and z_s.

$$\begin{aligned} x &= \frac{\lambda - x_o}{x_s} \\ y &= \frac{\phi - y_o}{y_s} \\ z &= \frac{Z - z_o}{z_s} \end{aligned} \tag{2.3}$$

Then, the normalized pixel coordinate (u, v) is defined as the ratio of two cubic polynomial functions. Note, each cubic function has 20 coefficients:

$$u = \frac{\sum_{i=1}^{20} C_i f_i^3(x, y, z)}{\sum_{i=1}^{20} D_i f_i^3(x, y, z)}, \qquad v = \frac{\sum_{i=1}^{20} E_i f_i^3(x, y, z)}{\sum_{i=1}^{20} F_i f_i^3(x, y, z)} \tag{2.4}$$

The 80 coefficients are:

$$\begin{aligned}
&\sum_{i=1}^{20} C_i f_i^3(x,y,z) \triangleq C_1 + C_2 \cdot x + C_3 \cdot y + C_4 \cdot z \\
&\quad + C_5 \cdot x^2 + C_6 \cdot xy + C_7 \cdot xz + C_8 \cdot y^2 + C_9 \cdot yz + C_{10} \cdot z^2 \\
&\quad + C_{11} \cdot x^3 + C_{12} \cdot x^2 y + C_{13} \cdot x^2 z \\
&\quad + C_{14} \cdot xy^2 + C_{15} \cdot xyz + C_{16} \cdot xz^2 \\
&\quad + C_{17} \cdot y^3 + C_{18} \cdot y^2 z + C_{19} \cdot yz^2 \\
&\quad + C_{20} \cdot z^3 \\
&\sum_{i=1}^{20} D_i f_i^3(x,y,z) \triangleq D_1 + \ldots + D_{20} \cdot z^3 \\
&\sum_{i=1}^{20} E_i f_i^3(x,y,z) \triangleq E_1 + \ldots + E_{20} \cdot z^3 \\
&\sum_{i=1}^{20} F_i f_i^3(x,y,z) \triangleq F_1 + \ldots + F_{20} \cdot z^3
\end{aligned} \tag{2.5}$$

Finally, four additional normalizing parameters (S_S, S_o, L_S, L_o) are used to convert the normalized pixel coordinate (u, v) back to the (S, L) image pixel coordinate:

$$\begin{aligned}
S &= uS_S + S_o \\
L &= vS_L + L_o
\end{aligned} \tag{2.6}$$

In summary, a full RPC model has 80 coefficients and 10 normalizing parameters. The 10 normalizing parameters are $x_o, y_o, z_o, x_s, y_s, z_s, S_S, S_o, L_S,$ and L_o.

As mentioned previously, height data is needed for accurate orthorectification. When no height data is available, geoid[3] can be used instead. However, using geoid for height may result in location error as large as 1 km. Fortunately, since June 29, 2009, accurate height data have been made freely available from ASTER (Advanced Spaceborne Thermal Emission and Reflection Radiometer). The ASTER GDEM (Global Digital Elevation Model) dataset has 30-meter ground resolution and covers 99% of the globe[4] .

[3]Geoid is a hypotheses surface, closely related to the average sea level.

[4]ASTER GDEM can be downloaded from `https://search.earthdata.nasa.gov/search` or `https://earthexplorer.usgs.gov/`.

We now presents the orthorectification algorithm:

1. Create an empty orthorectified image by constructing a uniform grid of lat-long cells using the locations at the four corners of the satellite image.

2. At each lat-long cell, (ϕ, λ), first look up its orthometric height (height above EGM96 geoid) from GDEM. Then, calculate the GPS ellipsoidal height, Z, by adding the geoid height[5]:

$$\begin{aligned} Z &= \text{GPS ellipsoidal height} \\ &= \text{orthometric height} + \text{geoid height} \end{aligned}$$

3. Project each (ϕ, λ) into the satellite image at pixel (S, L) using and the ellipsoidal height, Z, calculated from the previous step and the RPC equations from Eq. 2.3 to Eq. 2.6.

$$(S, L) = \text{RPC}(\lambda, \phi, Z) \tag{2.7}$$

4. For each pixel (S, L) in the satellite image, apply bilinear interpolation using the four nearest neighbors and assign this interpolated value to the corresponding grid cell, (ϕ, λ).

One way to generate a regularly-spaced lat-long cells from the satellite image is to first project the corner pixels into lat-long coordinates, using geoid for height (i.e., Orthometric height = 0). Then, use the accurate GDEM height data to iteratively refine these four corner pixels to their exact lat-long coordinates. After obtaining the corner pixels in their correct lat-long coordinates, a regularly-spaced grid of lat-long cells can be constructed.

Inverse Orthorectification

For object geo-localization applications, it is desirable to have the object's location expressed in geographic coordinate in terms of longitude and latitude values. The

[5]The geoid height ranges from (-100m to 100m) and can be obtained from the Internet.

traditional approach is to orthorectify the entire satellite image first, and then extract all object instances from it. This way, the object locations in the orthorectified image are already in lat-long coordinates. However, there are two main disadvantages in this traditional approach.

1. Orthorectifying the entire satellite image is computationally expensive, even though it only needs to be done once.

2. Orthorectified image may have a slightly coarser spatial resolution and contains aliasing artifacts.

As mentioned is Section 2.1.4, high resolution images are needed for detecting small objects. It could be the case that the slight lost of spatial resolution together with the aliasing artifacts in the orthorectified image could render an objects detector ineffective. Therefore, instead of running an object detector on the orthorectified image, which takes time to generate, the alternative would be to run the detector directly on the original raw or ToA corrected image. Subsequently, only the extracted locations (in image coordinates) are converted to lat-long via inverse orthorectification. Overall, this process takes far less time than doing a full-blown orthorectification of the entire raw image to start with. [16]

Gamma Correction

As we mentioned previously, raw data are first converted to reflectance values through the process of ToA correction. Subsequently, the ToA corrected image is then further processed by Pansharpening, Orthorectification, etc.

When it comes to the time for the image to be displayed on the display screen, each pixel in the image needs to be transformed into the *standard RGB* (aka sRGB) color space. The process of transforming reflectance to sRGB color is done by applying the *gamma correction*:

Let (r, g, b) be the ToA corrected values for the red, green, and blue channels of a pixel. Then, we can approximate the corresponding sRGB color, (R,G,B), by:

$$\begin{aligned} R &= \lfloor 255\ r^{\frac{1}{2.2}} \rfloor \\ G &= \lfloor 255\ g^{\frac{1}{2.2}} \rfloor \\ B &= \lfloor 255\ b^{\frac{1}{2.2}} \rfloor \end{aligned} \tag{2.8}$$

2.2 Data Abstractions

In this section, we define four data abstractions that we will be using through out the text.

satellite image: A typical large high-resolution satellite image contains about 30000× 30000 pixels, covering 225 sq. km at 0.5 meter per pixel.

tile: A single satellite image is chopped into tiles each covering about 1 km × 1 km area, or 2098 × 2098 array of pixels. Tiles overlap 50 pixels into the borders. See Fig. 2.3 for an illustration.

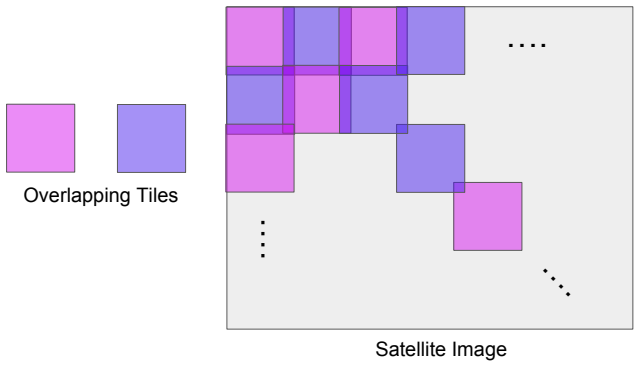


Fig. 2.3. Tiles overlap 50 pixels into the borders.

patch: A patch is a sliding window of 101 × 101 pixels (50 m × 50 m area) inside the tile. The number of patches inside a tile depends on the offset between each adjacent patch. For example, with an offset of 4 pixels, we get a patch every 2

meter. The total number of patches is thus 250000 per tile. See Fig. 2.4 for an illustration.

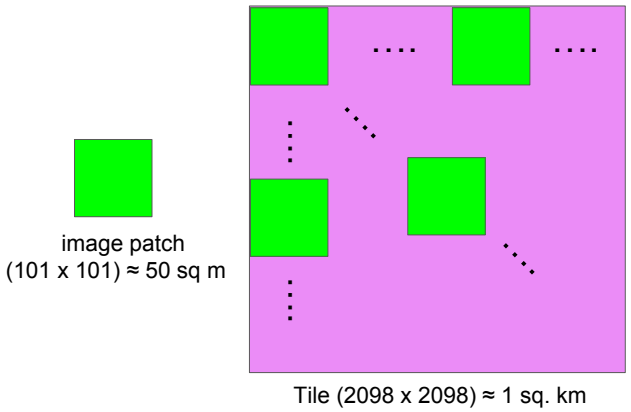


Fig. 2.4. A tile can have 250000 patches when sliding the patch over the tile at one patch per every 4 pixels.

pixel: A pixel represents a 0.5 m $\times$ 0.5 m area on the surface of the earth.

3. PIMSIR FOR UNDERSTANDING MULTIPLE SATELLITE IMAGES OVER LARGE ROI

The material in this chapter is adopted from our recent work [1]. The aim of that work is to address the problem of land-cover classification from a larger geographical perspective. Obviously, before we can design a classifier at the level of an ROI (Region of Interest), we must first come to grips with the data variability over the ROI. Understanding data variability at the scale of a large ROI presents its own challenges and can be thought of as a "Big Data" problem. The challenges are created by the typical fast-response and dynamic-storage needs of any human-interactive computer system that must work with very large variable-sized datasets.[1] We have addressed these challenges by developing a special software tool (named PIMSIR for "Purdue Integrated MultiSpectral Image Representation" tool) that is custom designed to achieve the following:

- Rapid visualization of all of the data in an ROI
- Rapid visualization of the overlaps between the satellite images. Understanding the overlaps is important because any probabilistic modeling of the data at any given geographic *point* is predicated on how much data is available at that point through overlapping satellite views.[2]
- Rapid visualization of the variability of the spectral signatures both spatially and across the views.

What makes PIMSIR versatile is that the data structure it creates for an ROI is dynamic — in the sense that an ROI is allowed to be covered by an arbitrary number

[1]By very large, we mean datasets that are hundreds of gigabytes in size.

[2]In general, probabilistic modeling is with respect to spatial distribution of the observed data. However, in order to address view-to-view data variability issues, one can also talk about probabilistic modeling with respect to the viewing dimension.

of satellite images, with arbitrary degrees of overlap between them. As to the size of the ROI that can be accommodated in this representation, that depends on the number of images. For example, in a ROI that covers about 10,000 km^2 and consists of 189 overlapping images [3], its corresponding PIMSIR structure takes 208 GBytes of disk storage. PIMSIR construction can be made memory-efficient by pre-loading only the relevant images at each cell. Should the available total memory be insufficient, we can always resort to using the POSIX function `mmap` which maps content of a file directly into the computer's virtual memory address space.

In what follows, we will describe how we create the PIMSIR structure for a given ROI. Subsequently, we will describe the information that is stored in PIMSIR for each geographical point in the ROI.

3.1 PIMSIR Data Structure and Construction

Given a raw WorldView2 Multispectral Image (MSI), the first necessary preprocessing step is to apply the Top-of-Atmosphere (ToA) reflectance correction [12] to the images in order to normalize out the view-angle (with respect to the sun angle) variability. Then, the corrected image goes through the orthorectification process that maps the pixels to geo coordinates.

After all of the corrections mentioned above have been applied to the data, creating the PIMSIR structure involves the following steps:

1. Construct a bounding box for an ROI.

2. Rasterize the bounding box with a matrix of sampling points, taking into account implicitly the spatial resolution desired. The results we show in this section are for the case when each cell in the bounding box represents a $2m$ x $2m$ area.

[3] Most of the images in this dataset contain 4 band data: Red, Blue, Green, and Near Infrared(NIR). We plan to incorporate satellite images with arbitrary number of bands in the next version of PIMSIR.

3. Scan the bounding box and, for each cell, calculate its geo lat-long coordinates.

4. Fetch the list of images for the lat-long coordinate at a sampling point. Project the lat-long coordinate into each image and:

   - Record the four pixels that are nearest to the projected point in the image. (The point projected into a image will, in general, not correspond exactly to any of the pixel locations in the image.)
   - Apply bilinear interpolation to the spectral signatures at the four nearest neighbors and return this answer for the image in question.

5. Pool together the spectral signatures collected from all the images that see the geo-point and store them in a compact data structure whose address is held by the bounding-box point in question.

Each cell location in the PIMSIR data contains an array of overlaps, the number of overlaps being equal to the number of images that can see that location. For each overlap, we allocate B bytes for the spectral signatures extracted from each image. In our current implementation, we use $B = 17$ bytes per overlap. Four of these bytes are reserved for the spectral value in each of the four primary bands, and one byte reserved for the pointer to a file that contains the meta data for that satellite image.[4] Fig. 3.1 summarizes the PIMSIR cell data structure. Because each PIMSIR cell can contain different number of overlaps, the size of the cell is not fixed. This dynamic nature of cell size leads us to create an address-look-up-table that maps the cell's grid location to the actual data. This mapping information is stored in the header segment of the file that carries the .pimsir suffix. Fig. 3.2 summarized the overall PIMSIR file data structure.

[4]This obviously limits us to a maximum of 256 overlapping satellite images for any geo-point. We have yet to see a case where that condition would be violated. Nonetheless, in order to "future-proof" PIMSIR, we plan to use two bytes for the pointer to the metafile in the next version of PIMSIR.

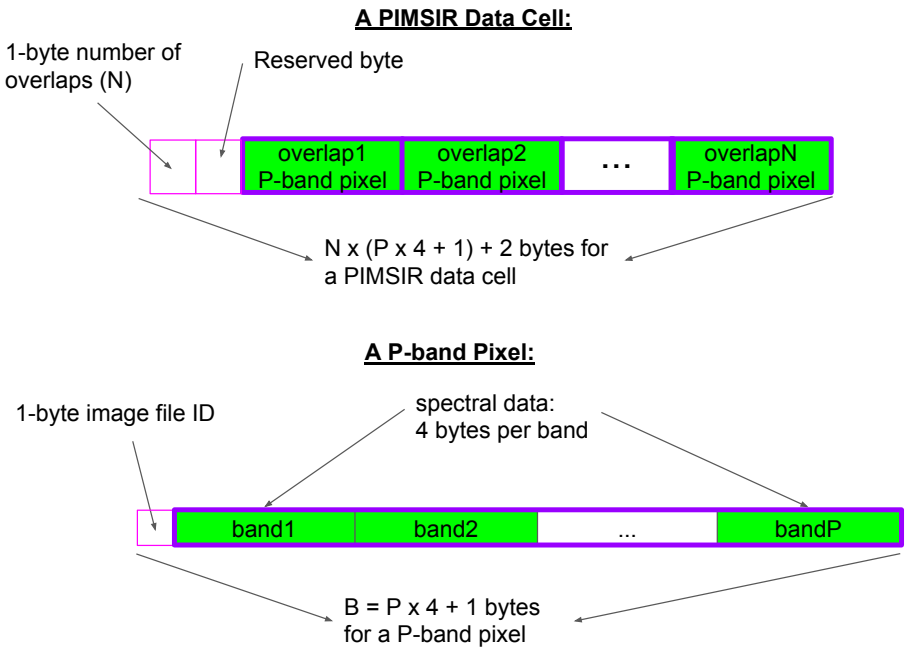


Fig. 3.1. The PIMSIR data cell and its data structure. Each PIMSIR cell can store variable number of overlaps, N. Each overlap stores P=4 bands of spectral data thus taking $B = P \times 4 + 1 = 17$ bytes.

3.2 Viewing Image Overlaps with PIMSIR

Fig. 3.3 shows a Chile ROI of size 10,000 km^2 that is covered by a total of 189 overlapping satellite images in the WorldView2 dataset. The blocky artifacts in Fig. 3.3 are caused by several factors: (1) The image-to-image variability that can be attributed to the different look angles for the sensors aboard the satellites, sun angle variations, and the time of the year when the data was recorded; (2) Small errors in the application of the Top-of-Atmosphere correction that is applied to the pixels; and (3) small errors in image rectification process.

When multiple images contribute to the same sampling point in the ROI-based bounding-box, one of the images is chosen arbitrarily for the purpose of the display as shown in Fig. 3.4. Note that this arbitrary selection is merely for the display and has no bearing whatsoever on the data variability calculations at the point.

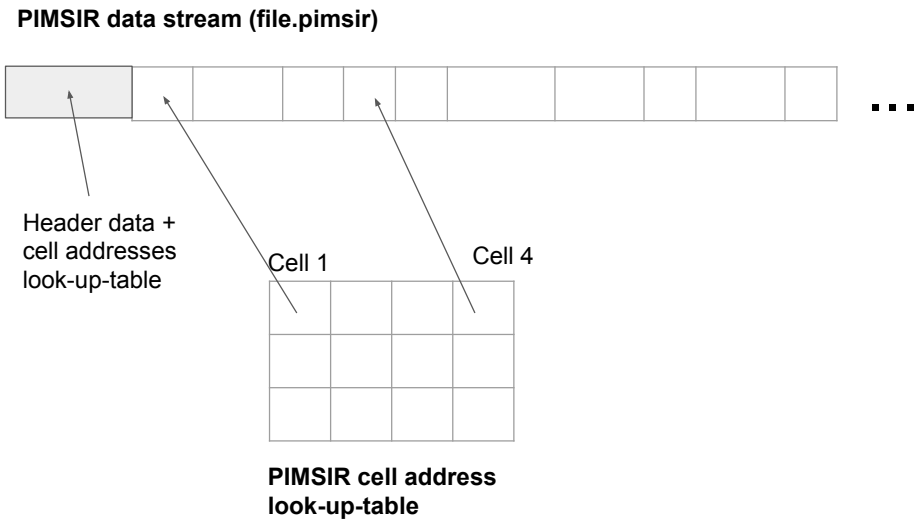


Fig. 3.2. The PIMSIR file data structure. The header of the PMISIR file contains various meta data as well as an address look-up-table that allows constant-time retrieval of data from any cell. Note that data in PIMSIR cells have different sizes in general.

When we examine the overlaps in all of the images for the Chile ROI, we go from one extreme where there is only a single image at a given geo-point to the other extreme when we have 49 images looking at the same geo-point. Fig. 3.5 shows the portions of the Chile ROI where we have data from 49 images. At each cell of the ROI-based bounding box, we arbitrarily selected the RGB from one of the 49 images for constructing the display in Fig. 3.5. Note again that the goal of this display is merely to indicate where we have a total of 49 views looking at the same geo-point.

3.3 Viewing Data Variability Heat Maps with PIMSIR

We show the with-in image data variability through what we refer to as variability heat maps. Fig. 3.6 shows an example of such a heat map. The values depicted in Fig. 3.6 correspond to maximum spectral range, r_{max}, at each point. If we let S_i be

Fig. 3.3. The entire Chile ROI (Region of Interest) covering about 10,000 square kilometers.

the set of 49 spectral measurements for a band i at a given point, then this parameter is calculated using the formulas:

$$r_i = \max(S_i) - \min(S_i) \tag{3.1}$$

$$r_{max} = \max_i(\{r_i\}) \tag{3.2}$$

The value $\max(S_i)$ is the maximum of 49 such band values at a given point and $\min(S_i)$ is the minimum of the same set of spectral signatures. Therefore, the value of r_i is the largest variation within the i^{th} band across all 49 images. And the largest of these variations among the 4 spectral bands is the maximum range, denoted r_{max}. Note that these calculations are carried out after the spectral values are normalized to lie between 0 and 1.0. The data normalization for each image is a part of what is accomplished by the Top-of-Atmosphere correction mentioned earlier. The red regions in Fig. 3.6 correspond to the heat map values above 0.15. On account of the

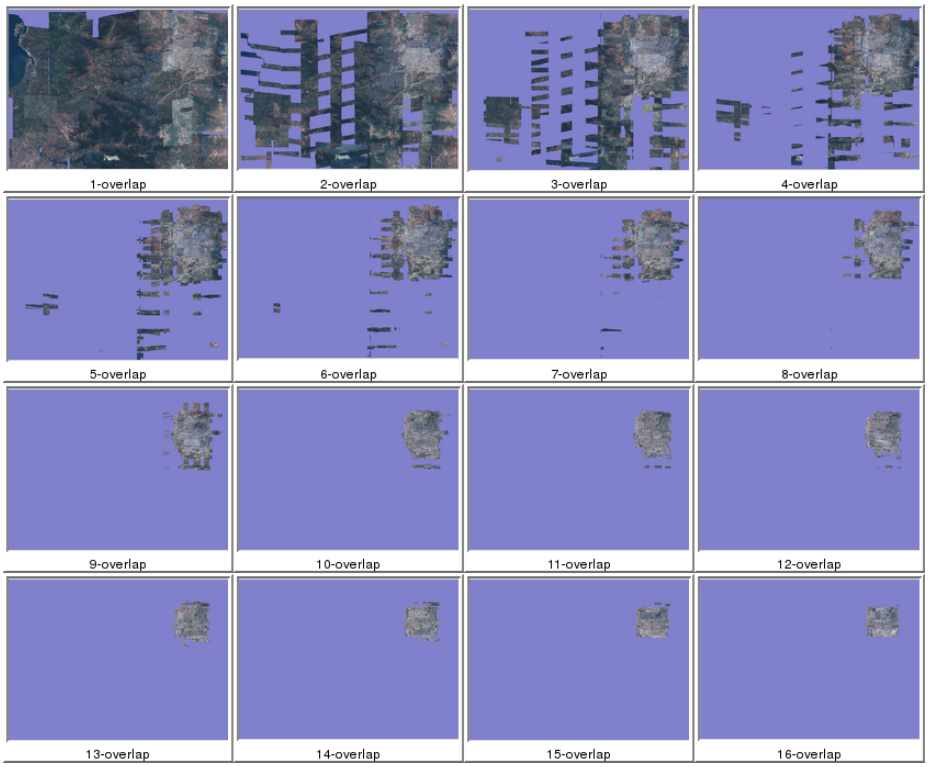


Fig. 3.4. A collage of the first 16 (out of 49) overlaps: 1-overlap, 2-overlap, . . . , 16-overlap.

data normalization, this means that we have more than 15% within-band variation at the points that are shown as red in Fig. 3.6. The points that are shown as green have their within-band variability under 15%. Rapid visualization of the variability allows us to systematically try many different cutoff thresholds and, in this case, 0.15 was chosen as it produced the most visually informative heatmap in the sense that we can see the fine road structure present in the scene.

Comparing the images in Fig. 3.5 and Fig. 3.6, one can infer that the orthorectification errors within the 49 images are sufficiently small and may be ignored in most cases — *this is an important by-product of this study since one is always concerned about the quality of orthorectification.* If this error had been large, the green strands in Fig. 3.6 would not correspond to the roads in Fig. 3.5. Also note that the spec-

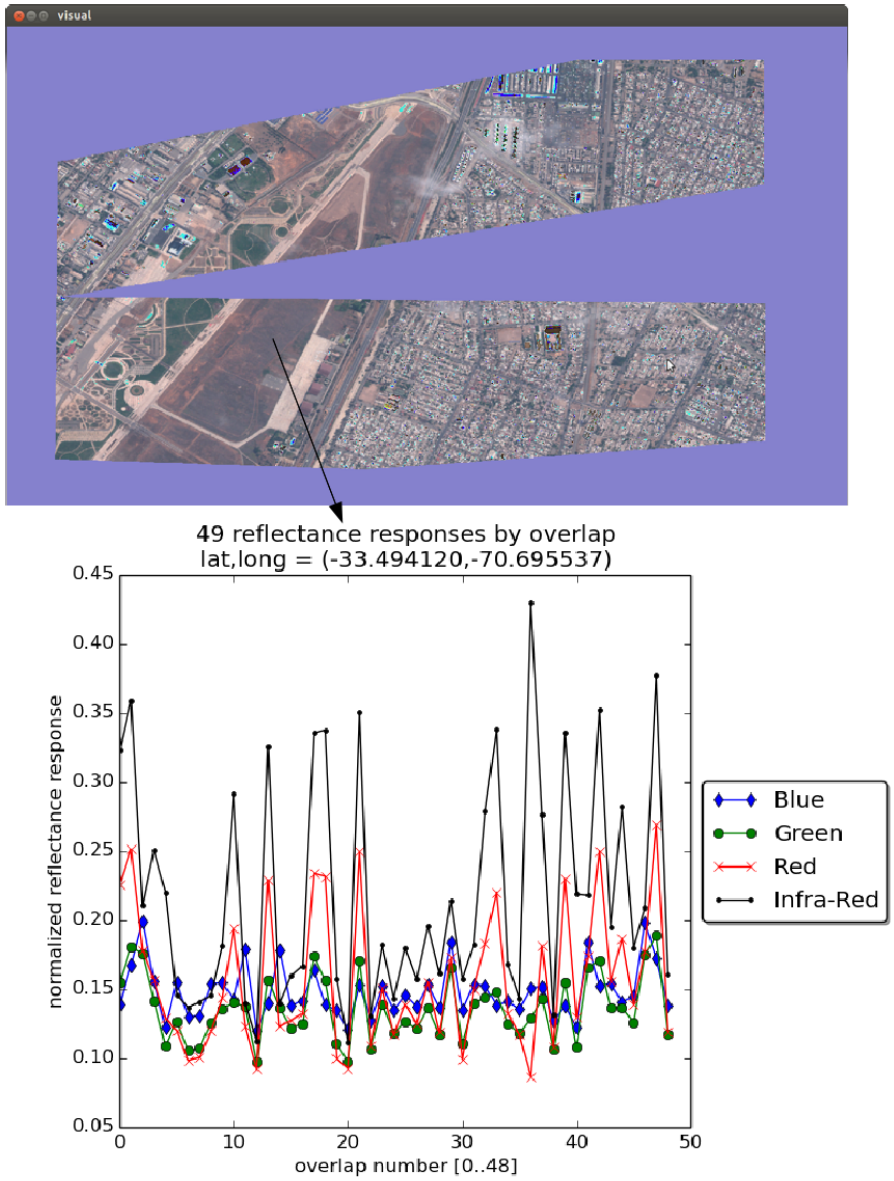


Fig. 3.5. The 49-overlap display shows the region with 49 overlapping MSI images. What is shown in the lower plot are all the spectral signatures for a particular geo-location within the region.

tral signatures are significantly "invariable" across the 49 images for the road pixels. In other words, for this particular sub-region, the points of the earth's surface that

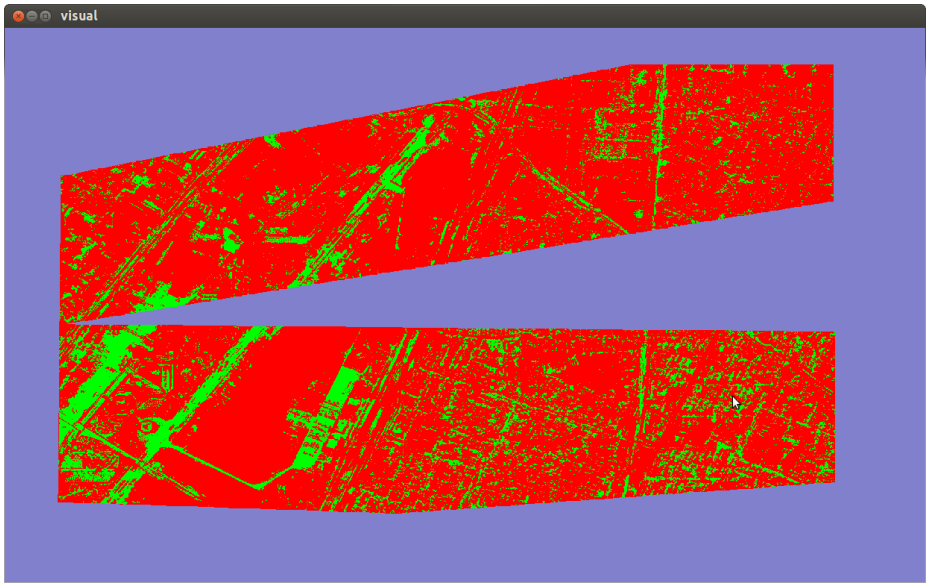


Fig. 3.6. Variability heat map: Green = maximum range $r_{max} < 0.15$.

correspond to the roads produce the same spectral signatures regardless of when the satellite image was taken and the look angle for the image.

Fig. 3.7 shows an example of data variation in a vegetation covered area as caused by seasonal change and Fig. 3.8 shows variation due to view-angle change. For the latter case especially, even when the actual land-cover remains the same, the occlusions created by tall structures cause the spectral signatures collected for a geo-point to vary from one image to another (ie. the parallax effect not fully corrected by orthorectification). Yet another source of data variation is caused by a piece of land being under development during the course of the time when the images were recorded. In this case, the same area can completely change from one land type to another. Fig. 3.9 shows such an example.

While all the previous examples illustrated data variability caused by different sources, it's also good to know that there can exist regions where the data stays constant, more or less. Fig. 3.10 shows the same road area in three different MSI images (selected randomly from the 49 images for the sub-region shown earlier in

Fig. 3.7. Examples of variation due to seasonal change. The same geo-location may need to be labeled as light vegetation and soil at different times of the year.

Fig. 3.8. Example of data variation due to view-angle change. In this case, the small areas in the middle of the two image patches shown should correspond to the same geo-location. But due to different view angles, the area in the right image is occluded by part of the tall building that can be seen in the left image. This results in different spectral signatures for the same geo-point.

Fig. 3.5). In Fig. 3.11 we plot the spectral signatures with respect to the view angles and with respect to time at a particular geo-location located at the center of the three image patches in Fig. 3.10.

The left plot of Fig. 3.11 shows the variability with respect to the look angle and the right shows the variability with respect to the month in which the data was recorded. As the reader can see, there is much less variation in the data for the point chosen. This result is consistent with the overall data variability heat map shown in Fig. 3.6.

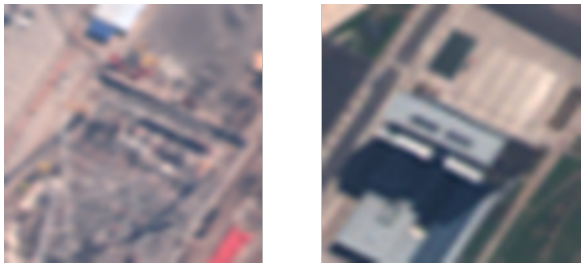


Fig. 3.9. Example of land-cover change caused by urban development. The area in the middle of the image patches shown changes from road/parking lot (left image) to building (right image).

Fig. 3.10. The three image patches cropped from three different MSI images centered at the same geo-location over a road surface.

3.3.1 Fast Variability Heat Map Computation using Integral Image

In order to support rapid visualization of the data variability heat map over any region in the entire ROI, we need to have a fast way to down-sample the data region so that the region can be displayed on the computer monitor, which has limited number of pixels. In other words, we need to compute or retrieve the average r_{max} value over any region quickly. To support this need, we developed a technique based on integral image [17]. With integral image, we are able to quickly sum up all pixel values inside any rectangular region in the image. In fact, it takes only two subtractions and one addition operations to compute the sum of all pixels inside a rectangular region, regardless of the size of the region. See Fig. 3.12 for an illustration.

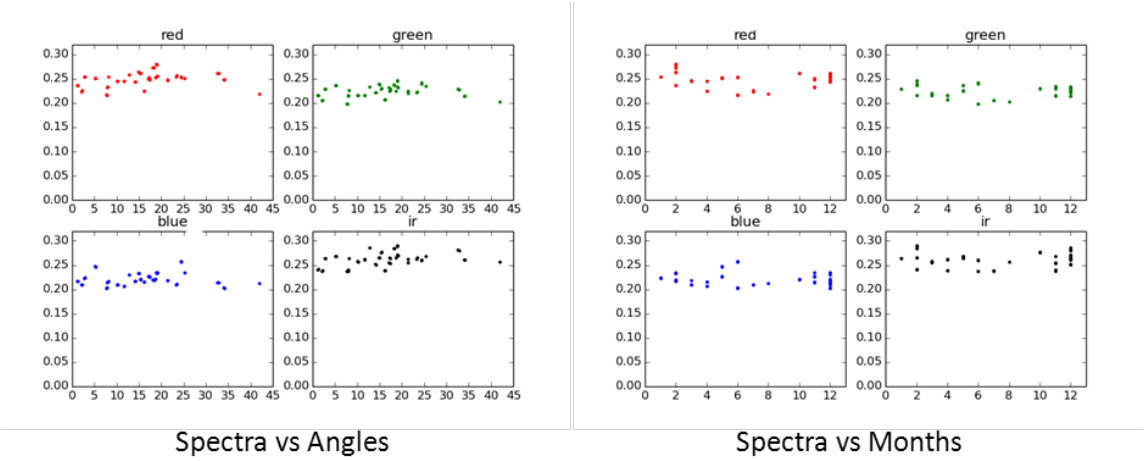


Fig. 3.11. Road spectral response versus Off-Nadir Angle and Month.

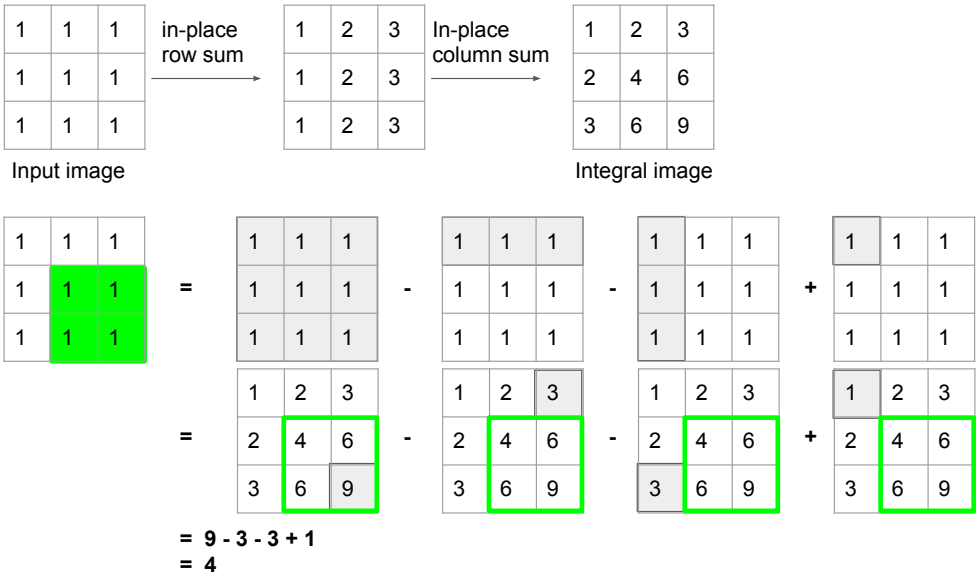


Fig. 3.12. Integral image can be computed in-place by first computing the row sums and followed by the column sums. In this example, the integral image is used to quickly compute the sum of the green rectangular region. Regardless of the size of the region, only three arithmetic operations (two subtractions and one addition) and 4 memory locations (the shaded elements in the integral image) are needed.

Our technique first constructs the integral image for r_{max} over the entire ROI once and saves it into a file. Subsequently, the integral image gets loaded into the memory together with the PIMSIR data. Next, when the PIMSIR visualization software wants to display the variability heat map of a region in the ROI, a downsampled heat map for that region can be quickly created with the help of integral image. Specifically, the procedure to create a downsampled heat map is as follows:

1. Get the size of the rectangular region in the ROI to be displayed. Let's name this region R and let its size be $n \times n$ cells.

2. Get the size of the rectangular window on the monitor that the region R is to be project to. Let this rectangular window be named W and let its size be $m \times m$ pixels.

3. Compute the block size, $b = \lfloor \frac{n}{m} \rfloor$.

4. For each $b \times b$ cells in R, compute the average r_{max} value over that rectangular region using the pre-computed r_{max} integral image.

5. Assign each average r_{max} value to the corresponding pixel in W.

6. Color-code W (e.g., green for values < 0.15) and display it on the screen.

4. INTRINSIC DIMENSIONALITY OF IMAGE PATCHES

This chapter investigates how intrinsic dimensionality depends on various attributes associated with the dataset. Specifically, given a set of L*a*b* color histograms extracted from a dataset consisting of image patches, the attributes we want to investigate are: dataset size, image patch size, and the number of bins in the L*a*b* color histogram.

In general, dimensionality reduction is used when we want to represent a feature by a fewer number of dimensions so that the data become easier to manipulate and less noisy to classify [18–21]. As to how much reduction can be achieved, it depends on the dataset's intrinsic dimensionality, which is typically defined as the minimum number of dimensions needed to describes the dataset completely and without loss of any information[1] [22].

However, when a dataset consists of low-dimensional data corrupted with high-dimensional noise, the intrinsic dimensionality of the dataset is often estimated using Principal Component Analysis (PCA) as "the number of [principal] components that hold the majority of the information" [23].

PCA, which uses linear projection, is not the only popular method for estimating the intrinsic dimensionality of a dataset. Other methods include manifold projection [24] and correlation dimension [22, 25, 26]. Manifold projection methods are especially appropriate when data lie on a low-dimensional nonlinear manifold. Correlation dimension methods, on the other hand, are more applicable for time-series data.

[1]Consider a trivial example of constructing a dataset by rotating a set of 2D points in 3D space. In this case, the resulting dataset consists of 3-dimensional data but its intrinsic dimensionality is 2, not 3.

4.1 Transforming the Multispectral Data Prior to Dimensionality Reduction

Multispectral data transformation is often done for the purpose of extracting meaningful features in order to create an effective land-cover classifier [1, 13]. Depending on the targeting land types and the classification algorithm used, different transformation or feature extraction may be appropriate. Examples of commonly used multispectral features include the National Difference Vegetation Index (NDVI) and Band Difference Ratios (BDR) [13, 27, 28].

Some features, such as color histogram and Haar-like features are high dimensional data. For these high-dimensional features, the number of dimensions can easily go above 10,000. In such cases, it makes sense to reduce the data dimensionality for the purpose of achieving more efficient data handling as well as potentially better classification performance.

In the next section, we give an overview of various color spaces that are commonly used for describing and representing colors. Having a good understanding of color representation is important because after all, our goal is to model the satellite image data in such a way that the concept of similarity is captured in agreement with human's interpretation.

4.1.1 The Advantage of L*a*b* Color Representation for Clustering

In this section, we first review the CIELAB color space, also known as the L*a*b* color space. We give reasons for why L*a*b* representation is appropriate for color clustering, especially when taking human perception into account. Then, we describe how we represent an image patch by its L*a*b* color histogram.

A Brief Review of the sRGB Color Space

Perhaps the most commonly used color system today is the Standard Default Color system (sRGB) created by Hewlett-Packard and Microsoft in 1996. Subsequently in 1999, sRGB was standardized by the International Electrotechnical Commission as `IEC 61966-2-1:1999` and used primary for displaying color on the computer displays.

sRGB is based on the trichromatic theory of human color vision. The theory states that any perceived color can be represented by a mixture of three primary colors. For sRGB, the three primary colors are conveniently represented by three 8-bit integers as their intensities. Fig. 4.1 shows a chromaticity diagram containing the sRGB gamut, which is the range of colors in the sRGB color space. In this chromaticity diagram, the blue horseshoe shape region encompasses all perceptible colors (specified by their x,y chromaticity coordinates) in the human color vision. Note that the sRGB gamut occupies only a portion of the horseshape region. Therefore, sRGB can not represent all colors perceivable to human. Nevertheless, sRGB is widely used because of its practicality in hardware implementation.

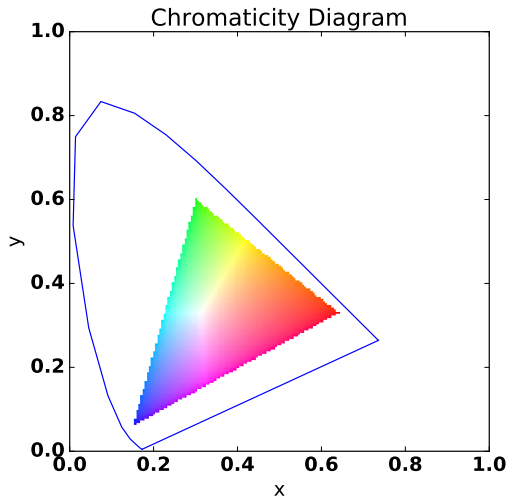


Fig. 4.1. The chromaticity diagram is often used to show and characterize the gamut of a color space. Here the sRGB gamut is shown. The blue line encompasses the region of all possible colors in the human color vision while the triangular-shaped gamut shows the range of color inside the sRGB color space. Note, the sRGB gamut shown here is displayed in full brightness (i.e., at each chromaticity coordinate, max(R,G,B) = 255).

Although the sRGB system is simple and convenient to process by a computer, the colors organized in this space is not suitable for meaningful clustering. This is the case because sRGB space is not *perceptually uniform*. In other words, if we were to visually cluster similar[2] colors together by drawing ellipsoids in the sRGB color space, we will see a large variation in ellipsoid size. In an ideal perceptually uniform color space, these ellipsoids would all be spheres having the same size instead.

A Brief Review of the HSV Color Space

A common alternative to sRGB is the HSV (Hue, Saturation, Value)[3] color space. HSV and its variants were created in the late 1970's by computer graphics researchers

[2]The color similarity metric can be established using the Just Notifiable Difference (JND) measurement of human vision.

[3]Two other variants of HSV are HSL (Hue, Saturation, Lightness) and HSI (Hue, Saturation, Intensity).

working with colors. At the time, these researchers needed a color space that can be used to generate and modify colors in a more intuitive way. Their general approach was to tilt the RGB color cube on its origin and map the colors onto a 2D polar-coordinate system. The vertical axis is then a measure of brightness while the 2D polar coordinate specifies the chromaticity of the color. In this way, similar colors have similar Hue values, neglecting the brightness and richness effect. Fig. 4.2 demonstrates the intuitiveness of the HSV space. As the user changes each of the H, S, and V values, the manipulated color changes in a predictable way. In general, H controls the "base color", S modifies the color's richness, and V changes the brightness.

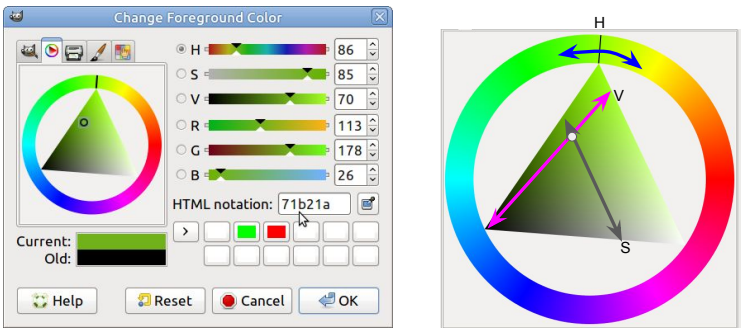


Fig. 4.2. HSV color space is commonly used in graphics software such as the GIMP (GNU Image Manipulation Program) software. A simple graphical interface allows the user to intuitively pick the desired color. On the color wheel shown on the right figure, the user can controls the overall color by changing the Hue (marked by H and blue), Saturation (marked by S and black), and Value (marked by V and pink).

A Brief Review of the L*a*b* Perceptually Uniform Color Space

Although HSV and its variants are more intuitive to use than sRGB, they are not perceptually uniform neither. Work on creating perceptually uniform color spaces actually started around the same time HSV color space was being developed. In fact, two perceptually uniform color spaces have existed since 1976. They were named L*u*v (aka CIELUV) and L*a*b* (aka CIELAB) by the International Commission

on Illumination (CIE) [29]. In both color spaces, L* is a measure of *lightness* that is linearly proportional to human's perception of brightness. The u* and v* components in the CIELUV space are derived from the *uniform chromaticity scale diagram* (aka the UCS diagram), which is perceptually uniform version of the original CIE1931 chromaticity diagram. Similarly, the a* and b* components in CIELAB are also perceptually uniform as well but they correspond to the red-green and the yellow-blue opponent colors in human vision. In other words, the value of a* indicates the degree of red-ness or green-ness, depending on the sign of the value. For example, a large positive value of a* indicates a large presence of red-ness. On the other hand, a large negative value indicates a large presence of green-ness. Similar logic goes for the b* component, but for the yellow-ness and blue-ness mix instead.

As mentioned previously, CIELUV is a direct extension to the UCS diagram that transforms the original CIE1931 (x,y) chromaticity diagram into a more perceptually uniform (u',v') chromaticity diagram. Fig. 4.3 shows a side-by-side visual comparison of the original CIE1931 (x,y) chromaticity diagram and the UCS (u',v') diagram. Fig. 4.4 shows the CIELUV color space at three different values of L*, which is a measure of lightness. In all three subplots of Fig. 4.4, only valid[4] sRGB colors are shown. Note that the range of (u', v') values depends on the value L*. For details on CIELUV definition and equations, please see Appendix A.

[4] R,G,B $\in [0 \dots 255]$

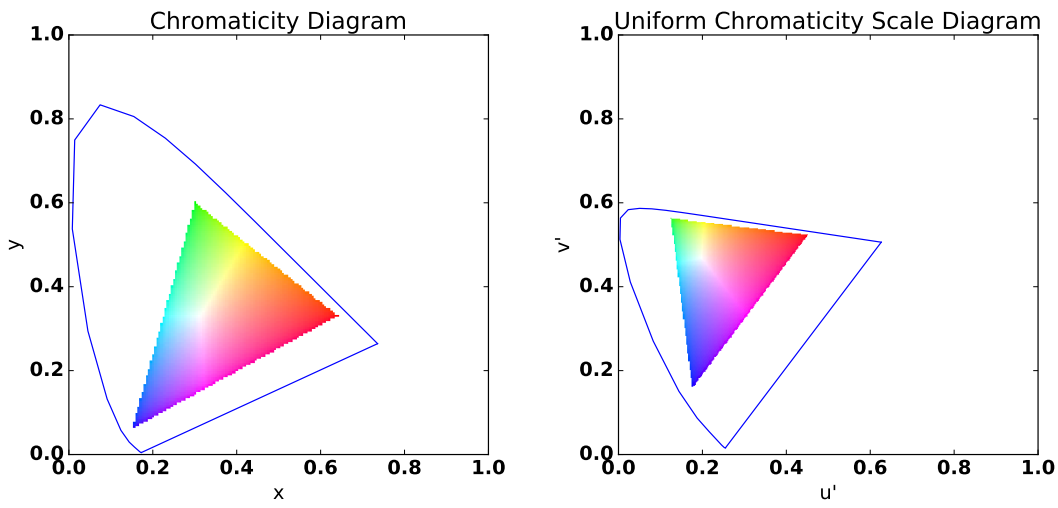


Fig. 4.3. Visual comparison of the two chromaticity diagrams. The sRGB gamut is shown in both plots. The blue line encompasses the region of all possible colors in the human color vision. The left plot shows the CIE1931 chromaticity diagram while the right plot shows the CIE1976 uniform chromaticity scale diagram (aka the UCS diagram). The UCS diagram is perceptually uniform in the sense that the Euclidean distance between points on the UCS diagram is more consistent with human's perception of color similarity. Note, the sRGB gamut shown here is displayed in full brightness (i.e., at each chromaticity coordinate, max(R,G,B) = 255).

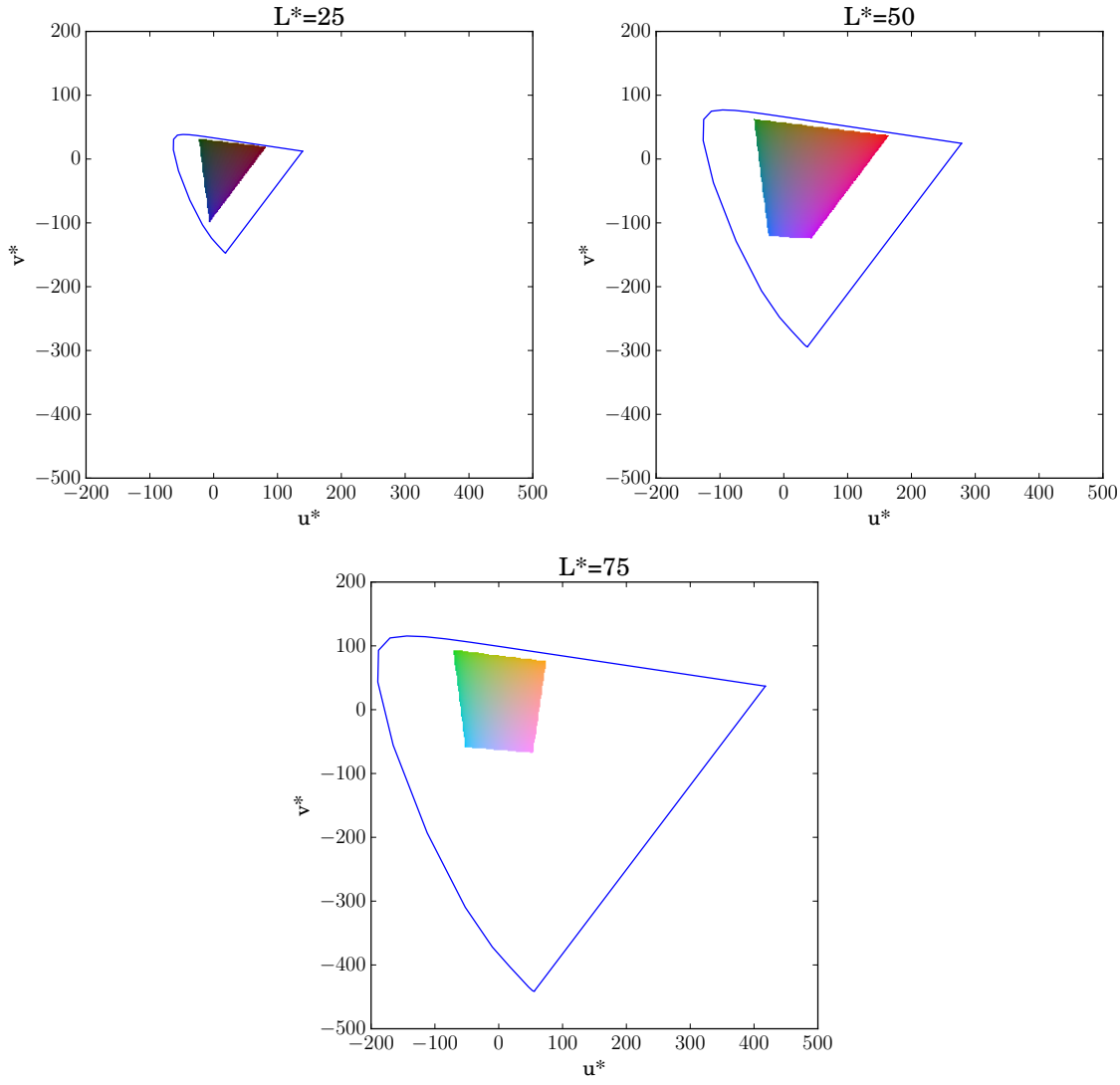


Fig. 4.4. The three subplots show the sRGB gamut in the CIELUV color space. Each subplot shows a different luminance value for L*. The blue line encompasses the region of all possible colors in the human color vision.

Similar to the CIELUV color space, the CIELAB color space is also perceptually uniform. CIELAB has two major advantages over CIELUV. First, the a* and b* values in the CIELAB color space do not depend on L*. This fact can be seen by comparing Fig 4.4 and Fig. 4.5. As for the second advantage, both a* and b* axes in CIELAB are physically meaningful in that they are aligned to two pairs of opponent

colors that exist in the human color vision. To demonstrate the opponent colors, Fig. 4.6 shows a* and b* contour lines overlaid on top of the sRGB gamut. As can be seen in the figure, a* varies from red-ness to green-ness and b* varies from blue-ness to yellow-ness.

With regard to their applications, CIELUV is commonly used for characterizing displays while CIELAB is used more for matching colored surfaces and paints. [30] For details on CIELAB definition and equations, please see Appendix A.

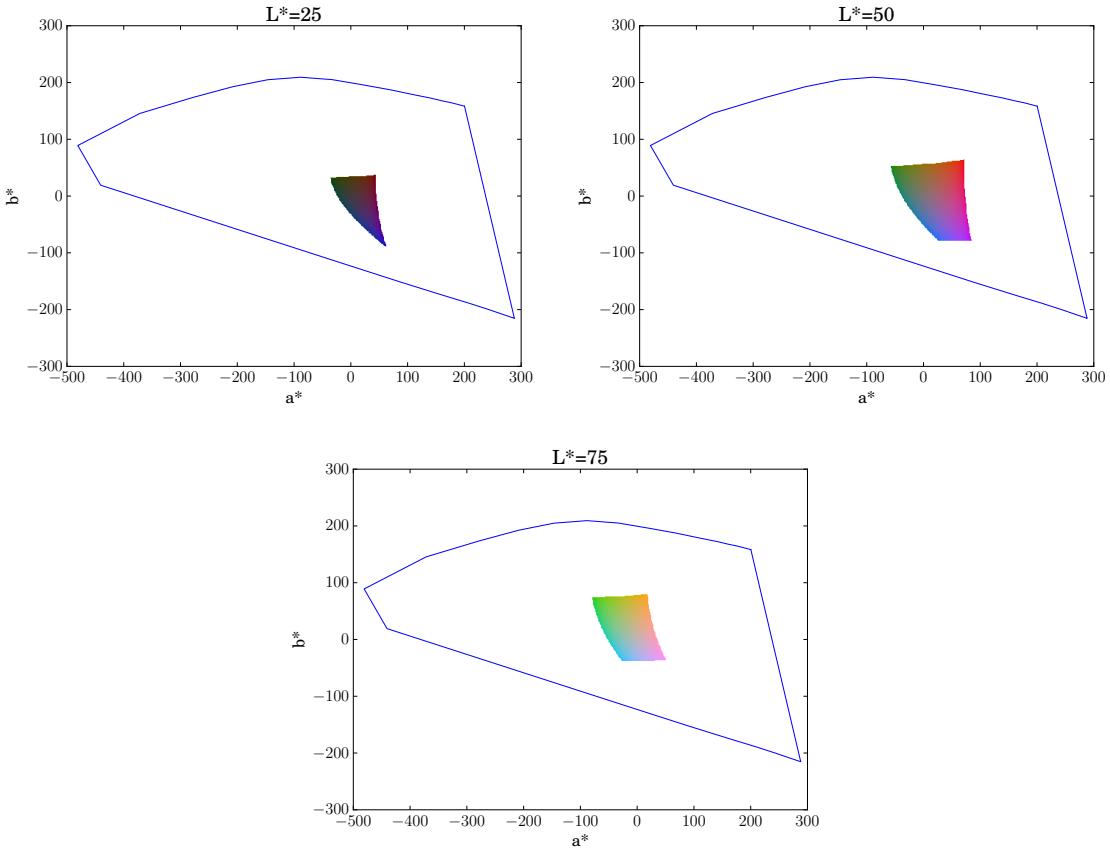
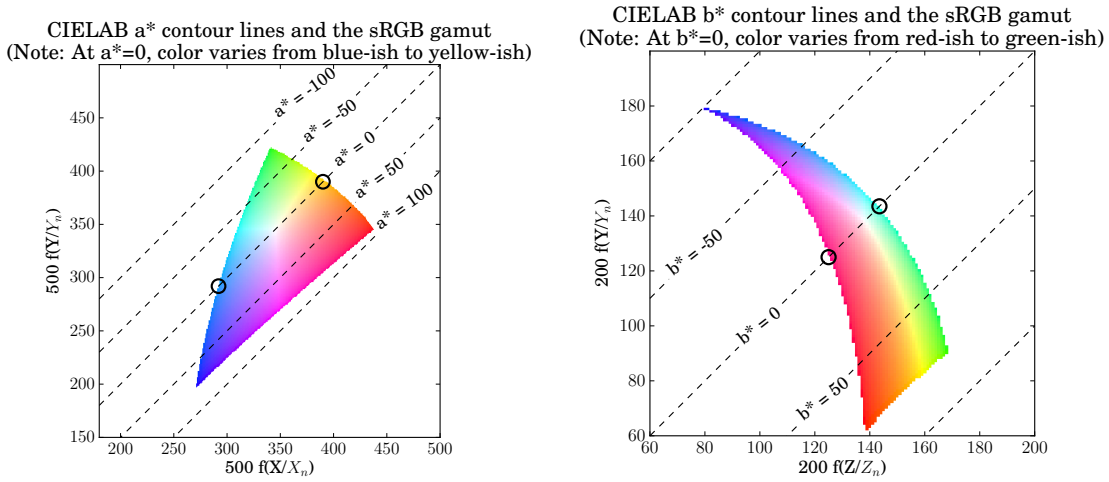


Fig. 4.5. The subplots show the sRGB gamut in the CIELAB color space. Each subplot shows a different luminance value for L*. The blue line encompasses the region of all possible colors in the human color vision.



(a) sRGB gamut and a* contour lines

(b) sRGB gamut and b* contour lines

Fig. 4.6. CIELAB is an opponent color space. From the sRGB gamut, we can see the a* contour lines vary from green to lack of green. Similarly, the b* contour lines vary from blue to lack of blue. At b*=0, only a* varies and we see the color goes from red to green. Similarly, at a*=0, only *b varies and the color goes from blue to yellow. The locations of these colors are marked by black circles.

4.1.2 Creating Color Histogram Using the L*a*b* Color Space

In order to create a 3D color histogram, we must first partition the 3D color space into voxels. Each voxel then represents a bin in the histogram. Specifically, we first quantize the CIELAB color space into b^3 bins, where b is the number of bins in each of the L*, a*, and b* axes. As b increases, the number of bins increases and the colors in the neighboring bins will be more similar to each other. As to the choice of b, we noted from the study in [31] that b does not need to be more than 66[5] in order to adequately represent different colors by different bins. In other words, as b increases and approaches 66, it becomes harder for human to distinguish colors from adjacent bins.

[5]In [31], the voxel size for the CIELAB color space was determined to be $1.5 \times 3.0 \times 3.0$.

Since our input data are in the sRGB color space, we can further reduce the number of bins in the CIELAB color histogram by removing the invalid bins – those bins that will always be empty due to the fact that the sRGB color space is a subset of the CIELAB color space (See Fig. 4.5). As it turns out, the range of all sRGB colors in terms of CIELAB coordinates are: L*$\in [0 \ldots 100]$, a*$\in [-86.183 \ldots 98.233]$, and b*$\in [-107.857 \ldots 94.478]$. Table 4.1 shows the histogram sizes for both sRGB and CIELAB color spaces. From the table, we see that $b = 32$ gives us a 9024-dimensional L*a*b* color histogram.

Table 4.1.
Color histogram sizes for sRGB and CIELAB color spaces.

b bins per axis	sRGB histogram size	L*a*b* histogram size
4	64	45
8	512	245
12	1728	652
16	4096	1388
20	8000	2490
24	13824	4080
28	21952	6228
32	32768	9024
64	262144	64508

4.2 A Brief Survey of Previous Work on Dimensionality Reduction on Satellite Data

Most of the literature on dimensionality reduction for satellite data deal with extracting salient or meaningful feature for classification applications [32–34]. Dimensionality reduction also found applications in visualization of satellite data [35].

For the rest of this section, we give three typical and representative examples of how PCA is applied to satellite data.

First, the work of [32] proposes an algorithm using PCA and K-Means to automatically detect changes between two co-registered panchromatic satellite images. The approach is said to be applicable to multispectral images as well. The algorithm first computes the absolute differences between each pair of pixels between the two co-registered images. Then, PCA is applied on this "difference image". More specifically, PCA is applied on 4x4 non-overlapping blocks taken from the difference image. That is, each 4x4 block is represent by a 16-dimensional feature vector and each of these 16 dimensions corresponds to the absolute difference at the particular pixel location within the 4x4 block. Of the 16 dimensions, only the top 3 components from PCA is retained. The use of PCA in this case is to perform 2-class clustering in a lower dimensional space. The two classes are given the labels: "Change" and "No Change". As for the number of components to keep, the work states that no significant difference is observed when using more than 3 components.

For another typical multispectral data classification application, the work of [33] proposes using PCA to remove haze. The algorithm uses PCA to identify and remove the "cloudiness band", which is found to be related to the smallest principal component. More specifically, PCA is applied on pixels from a continuous region where the haze is located. Each pixel in that region is represented by a 6-dimensional feature vector made up of multispectral channels 1-5 and 7. PCA is then used to remove the haze artifact, which is identified to reside on the data dimension that is least varying.

Finally, the work of [35] introduces a visualization method called Decorrelation Stretch that became popular for enhancing multispectral satellite images. Because data in the adjacent bands are highly correlated in multispectral satellite imagery, the idea of Decorrelation Stretch is to use PCA to decorrelate (aka "whiten") the spectral channels. This has the effect of highlighting data variation and thus making the hidden details easier to see. The algorithm first collects a representative sample of 1000 pixels from a multispectral image in which each pixel is represented by 3

multispectral channels. PCA is then applied on these 1000 3-dimensional feature vectors. After PCA, all components are kept but normalized by their individual variances. The result is then subsequently displayed in sRGB or some other color spaces.

4.3 Factors Affecting the Intrinsic Dimensionality of Image Patches

In this section, we describe five PCA experiments in which we examine some factors affecting the intrinsic dimensionality of image patches. Please refer to section 2.2 in Chapter 2 for the definition of satellite image, tile, and image patch.

4.3.1 Dataset Size

In this experiment, we examine the effect of dataset size on the intrinsic dimensionality. By dataset size, we mean the number of samples used to carry out PCA. First, we note that when the number of samples in the dataset is less than the number of dimensions in the original feature space, the number of principal components from PCA is at most the size of the dataset. For example, if the dataset has 100 samples, then, PCA can produce at most 100 principal components. Next, we expect the number of principal components to increase as the dataset gets bigger. However, this increase will plateau toward the intrinsic data dimensionality. With these insights in mind, our experimental setup is as follows:

1. Randomly select 10 satellite images from the Australia ROI (Region of Interest) [1].

2. Randomly select P patchesfrom the 10 satellite images. The total number of patches is thus $10 \times P$.

3. Compute the 9024-dimensional L*a*b* color histogram for each patch. (see Section 4.1.2). This gives us a dataset consisting of $10 \times P$ feature vectors.

4. Apply PCA on the dataset and retain 99.9 percent of the total energy[6]. Record the number of principal components.

5. Repeat from step 2 with $P \in \{2, 3, \ldots, 10\}$, resulting in nine datasets with sizes $= \{20, 30, \ldots, 100\}$.

6. Run 10 trials starting from step 1, each trial processes data from a new random set of 10 satellite images.

7. Over the 10 trials, create nine boxplots, one per datasets defined in step 5.

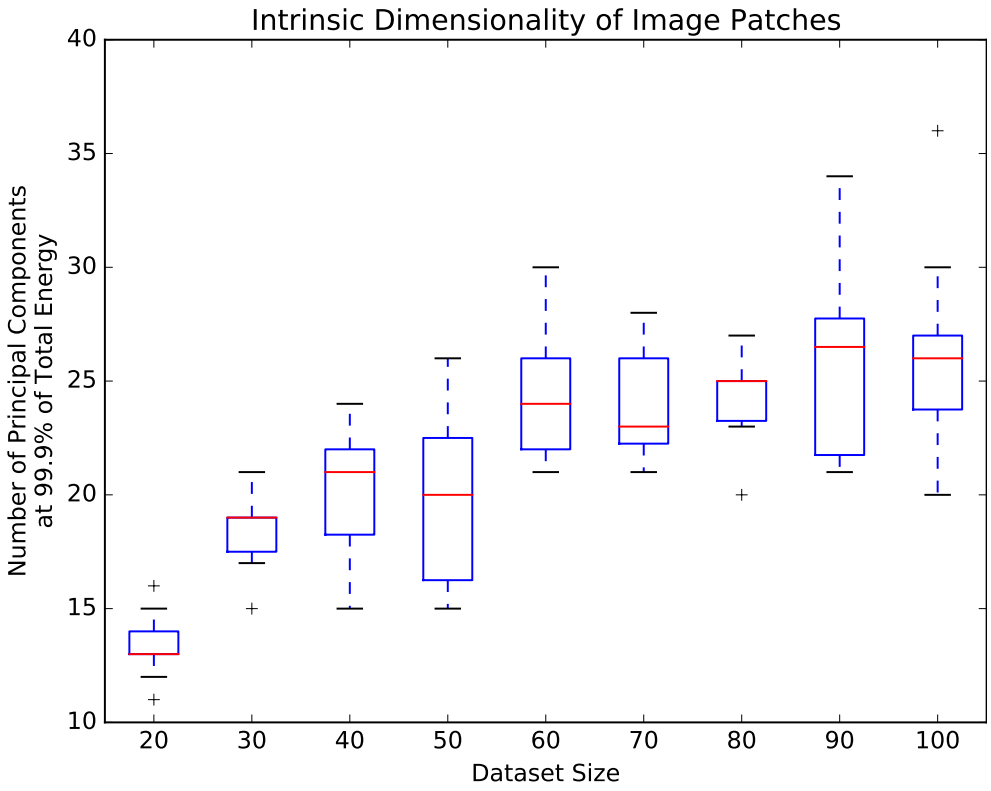


Fig. 4.7. Boxplots over 10 trials, each trial randomly selects 10 satellite images and estimates the intrinsic dimensionality by the number of principal components needed to retain 99.9% of the total energy.

[6]We compute the total energy by taking the sum of all eigenvalues.

We observe from Fig. 4.7 that for dataset sizes larger than 60, the median number of principal components plateaus toward 25, while the maximum number reaches 36 at size = 100. From these observations, we conservatively estimate the intrinsic dimensionality of image patches in the Australia ROI to be more than 36 and less than 100. Although this estimate is specific to the Australia ROI, the dramatic reduction of dimensionality from 9024 to less than 100 is typical to other ROI as well.

4.3.2 Data Diversity: Single-Satellite vs Multi-Satellite

In this experiment, we examine the effect on data variability when drawing image patches randomly from a single satellite image and when drawing from multiple satellite images. Because different locations on the earth surface tend to look differently, we expect a random set of image patches drawn from multiple satellite images be more diverse. In other words, if we were to apply PCA on the datasets, there should be a noticeable difference between the two datasets in terms of the number of principal components. More specifically, a random set of image patches sampled from the same satellite image is likely to have fewer number of principal components than a set of image patches from multiple satellite images covering a wider area.

Our experimental setup is as follows:

1. Randomly select 10 satellite images out of the 127 images in the Australia ROI.
2. Create two groups of tiles:
   - Single-satellite Group: Randomly select 10 tiles from a particular satellite image, S, among the 10 satellite images selected in step 1.
   - Multi-satellite Group: Randomly select 10 tiles, one from each of the 10 satellite images selected in step 1.
3. From each tile group defined in step 2, randomly select 100 patches from each tile. This give each group a total of 1000[7] image patches.

[7]As we saw from Section 4.3.1, the intrinsic dimensionality of image patches in the Australia ROI is estimated to be no more than 100. This piece of information is useful for the purpose of setting

4. Compute the 9024-Dimensional color histogram for each image patch in the two groups defined in step 2. Each group can now be represented by a dataset of 1000 color histograms (i.e., feature vectors).

5. Apply PCA to the two datasets defined in step 4, and for each PCA, retain 90% of the total energy.

6. Record the difference in number of principal components between the two datasets defined in step 4. A positive difference means that the multi-satellite dataset requires more dimensions.

7. Repeat from step 3, each time select a different satellite image, S, for the Single-satellite group. Since there are 10 satellites, this step will repeat 10 times.

8. Repeat from step 1 three times, each trail will draw data from a random set of areas in the Australia ROI. We refer to these 3 sets of areas as Region1, Region2, and Region3.

9. Create three boxplots, one per each region described in step 8 and each boxplot summarizes over the 10 trials defined in step 7.

---

the size of dataset. As a general and conservative rule of thumb, a dataset should be about 5-10 times the number of data dimensions. Therefore, we set our dataset size for this experiment to $100 \times 10 = 1000$.

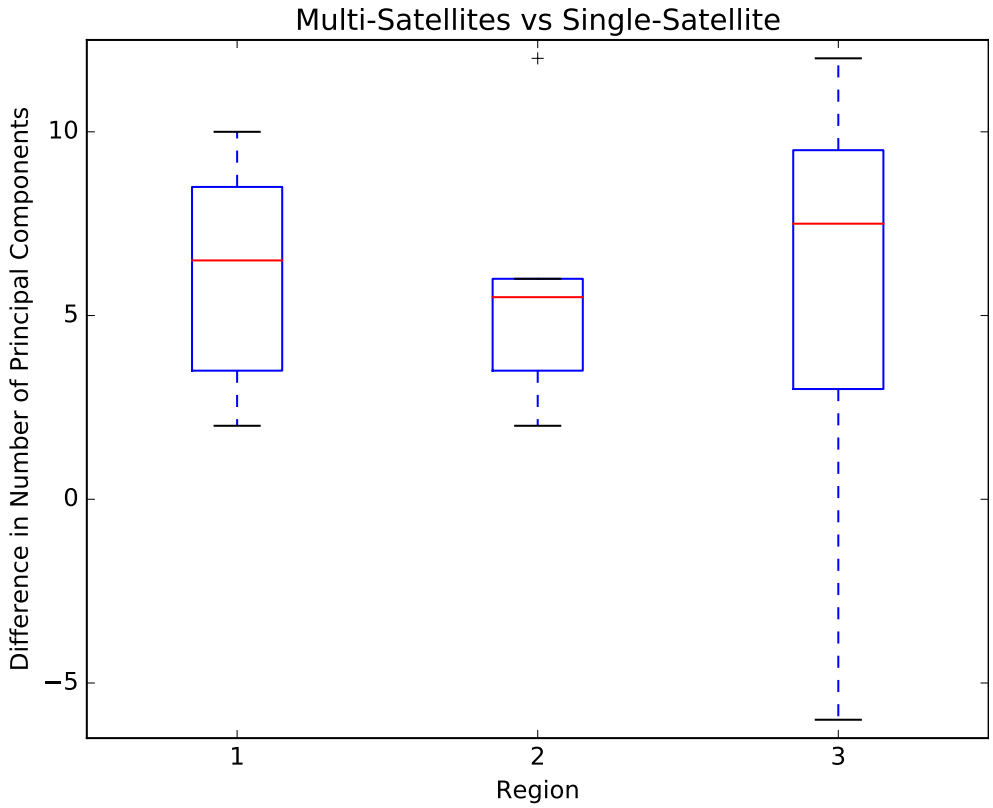


Fig. 4.8. Boxplots of three randomly-selected regions. Each boxplot summarizes, over 10 trials, the differences in the number of principal components between the two groups of datasets: Multi-satellite and Single-satellite.

From Fig. 4.8, we see that with the exception of Region 3, all 10 random collections of image patches drawn from multiple satellites have more principal components compared to collections drawn from a single satellite image. Our observation is based on the difference in the number of principal components at 90% of total energy. As for Region 3, the multi-satellite dataset has more principal components than the single-satellite dataset in 8 out of the 10 trials.

4.3.3 Fraction of the Total Energy

In this experiment, we look at two questions about PCA and dimensional reduction in terms of the total energy kept. The questions are:

1. How does the number of principal components increase as a function of energy retained?

2. How much error, in terms of Cosine distance[8], is present in the feature vector after reconstruction?

Our experimental setup is as follows:

1. Randomly select 10 satellite images out of the 127 images in the Australia ROI. For each of the 10 satellite images, randomly select 5 tiles. This gives us a diverse dataset consisting of 50 random tiles and over multiple satellite images from the Australia ROI.

2. Randomly select 10 image patches per tile. Represent each patch by its 9024-dimensional L*a*b* color histogram. This produces a dataset of size $50 \times 10 = 500$ color histograms[9].

3. Apply PCA on the dataset defined in step 2 and retain $E\%$ of the total energy. Record the number of principal components.

4. Project the dataset from step 2 onto the lower-dimensional space, which is represented by the principal components from step 3.

5. Reconstruct data in the projected lower-dimensional space back to the original high-dimensional feature space.

6. Determine the reconstruction error by:

[8]By Cosine distance, we mean the angle between two vectors.
[9]The histograms are normalized.

   (a) Compute the angular distances between the original data and their reconstructed counterparts. This is the residual error.

   (b) Compute the average residual error over all data.

7. To determine the relationship between the number of principal components and the percentage of total energy retained, set E to:

   - $\{0.95, 0.96, 0.97, 0.98, 0.99\}$ (coarse resolution)
   - $\{0.991, 0.993, 0.995, 0.997, 0.999\}$ (finer resolution for higher percentages)

8. Repeat step 7 10 times. Each trial draws a random set of image patches from the particular 50 tiles selected in step 1.

9. For each value of E, create boxplots to answers the questions stated at the beginning of the section. Note that each boxplot summarizes over 10 random trials created in step 8.

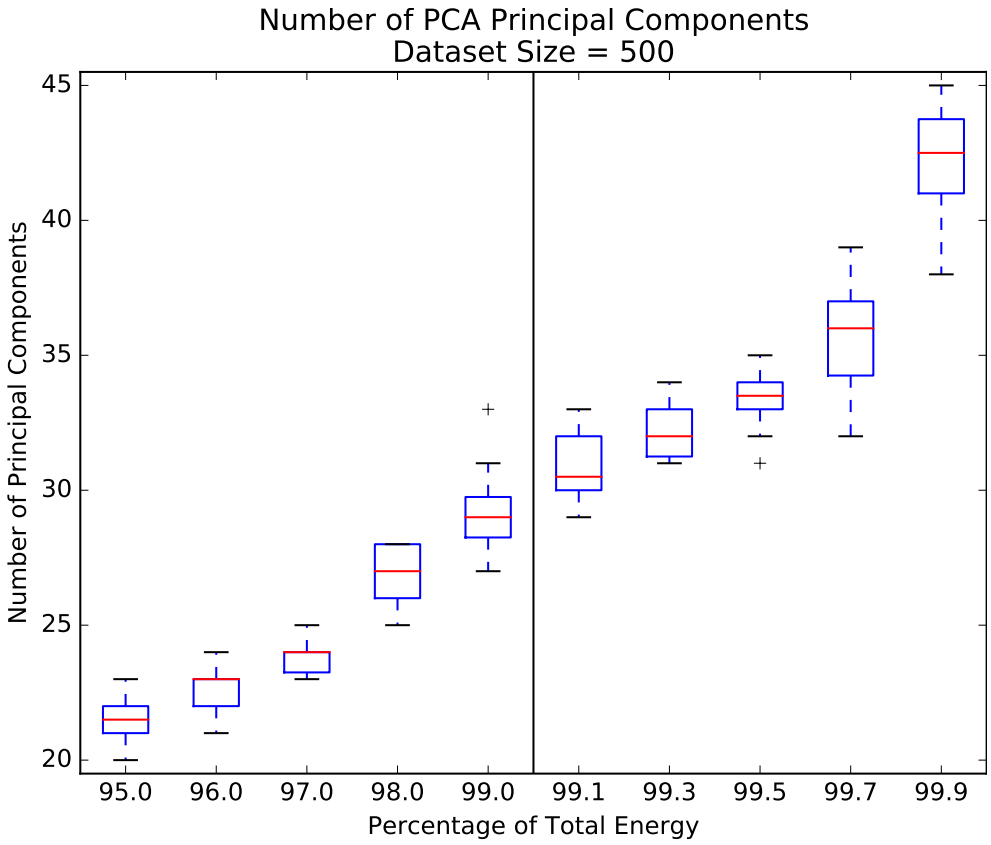


Fig. 4.9. Each boxplot summarizes, over 10 trials, the number of principal components when retaining a particular percentage of the total energy.

Fig. 4.9 shows that with 99.9% of the total energy, the number of dimensions in the projected space is less than 45.

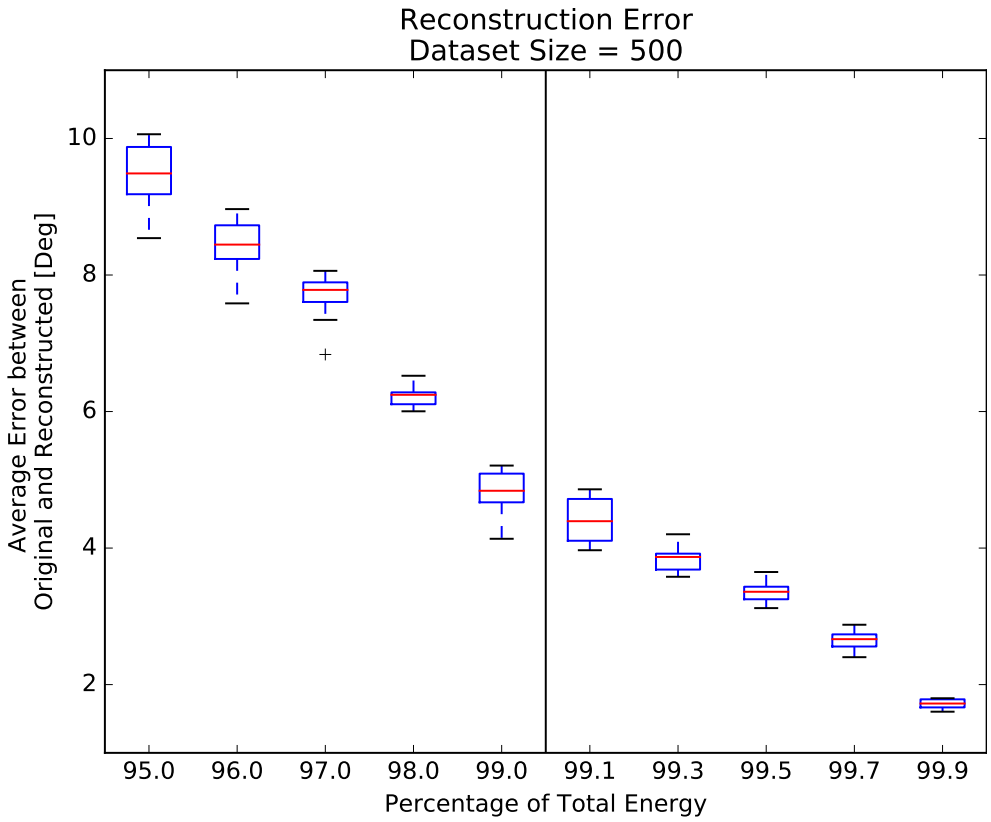


Fig. 4.10. Each boxplot summarizes, over 10 trials, the average error between the original feature vectors and their corresponding reconstructed feature vectors.

With regard to the reconstruction error, Fig. 4.10 shows average error between original feature vectors and their reconstruction from the low-dimensional projected space. As shown in the figure, the average error is less than 2 degrees at 99.9% of the total energy.

4.3.4 Image Patch Size

In this experiment, we examine how the size of the image patches affects the intrinsic dimensionality of the dataset. As one may suspect, the content of the satellite scene places a big role on the intrinsic dimensionality of the dataset. Consider a dataset consisting of image patches taken from satellite images of nothing but dessert

or ocean. Because all pixels are essentially the same color, changing the size of the image patch will not change the normalized color histogram and we will see no change in intrinsic dimensionality as we vary the image patch size. On the other hand, if the dataset consists of image patches that have their dominant colors getting more dominant as the patch size gets bigger, then, the intrinsic dimensionality actually goes down as the size of the image patch gets larger. This is the case because as the image patch gets bigger, the normalized histograms get less "noisy" due to the peak at the mode of the normalized histogram getting higher — suppressing the histogram bin value everywhere else (i.e., the histogram has larger entropy). Finally, we can also imagine the case where the histogram gets nosier (i.e., having lower entropy) as the image patch size gets larger. In this scenario, the intrinsic dimensionality goes up as the image patch size goes up.

Experimental Setup:

1. Randomly select 10 satellite images out of the 127 images in the Australia ROI. For each of the 10 satellite images, randomly select 5 tiles. This gives us a diverse dataset consisting of 50 random tiles and over multiple satellites in the Australia ROI.

2. For each tile selected in step 1, randomly select 10 patch locations. This gives us a collection of $50 \times 10 = 500$ image patch locations.

3. For each patch location in step 2, crop out a patch of size P by P pixels. This gives us a dataset of 500 image patches.

4. Apply PCA on the dataset from step 3, and keep 99.9% of the total energy. Record the number of principal components.

5. Repeat from step 3, with image patch width $P \in \{21, 41, 61, 81, 101, 201\}$.

6. Repeat from step 2 for 9 more times, each trial draws a random set of patches from the tiles selected in step 1.

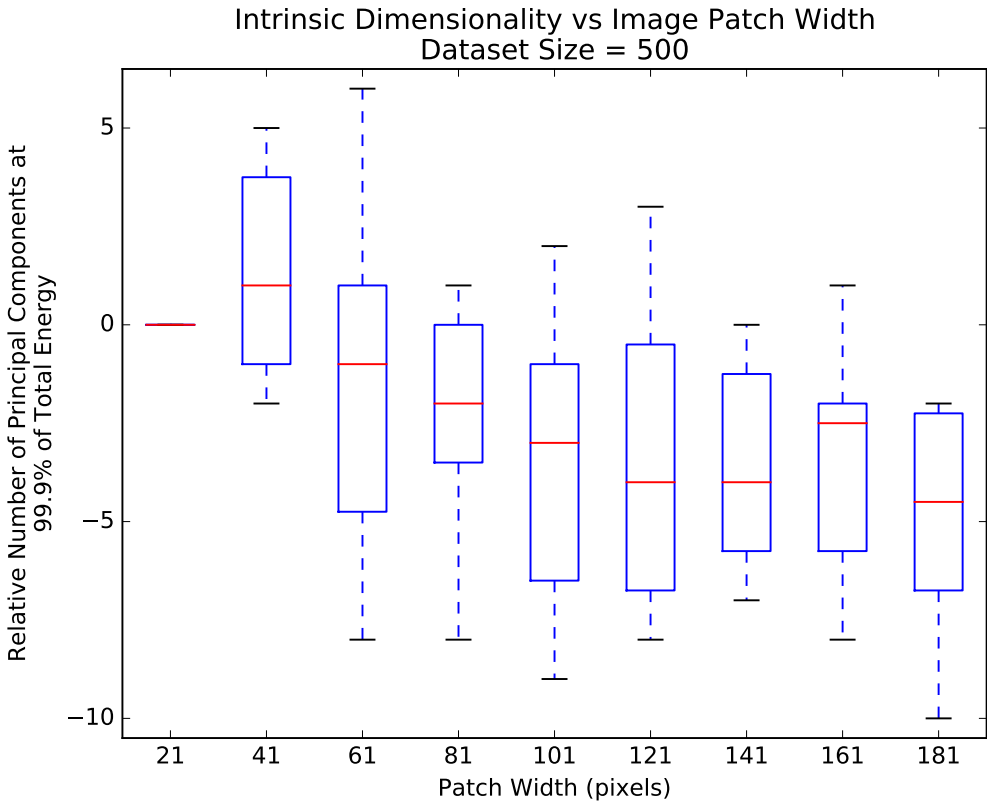


Fig. 4.11. For each particular image patch width, the boxplot summarizes, over 10 trials, the relative number of principal components obtained after PCA. The relative number is calculated by subtracting the number of components obtained for patch width = 21.

From Fig. 4.11, we see that as image patch gets larger, the intrinsic dimensionality of the datasets tend to decrease and then plateau. This suggests there isn't much data variation in this particular ROI.

4.3.5 Histogram Quantization

In this experiment, we study the effect of changing the number of dimensions in the feature space. Since we are using histogram as our feature, we can increase the number of dimensions by simply reducing the histogram bin size. Intuitively, as the feature dimensionality goes up, the number of total distinct feature vectors also goes

up. Therefore, a *random* dataset in high-dimensional space should have variation in many more dimensions and therefore requiring more principal components.Our experimental setup is as follows:

1. Randomly select 10 satellite images out of the 127 images in the Australia ROI. For each of the 10 satellite images, randomly select 5 tiles. This gives us a diverse dataset consisting of 50 random tiles and over multiple satellites in the Australia ROI.

2. For each tile selected in step 1, randomly select 10 image patches. This gives us a diverse dataset of size $50 \times 10 = 500$ image patches.

3. Compute L*a*b* color histogram for each image patch in the dataset and quantize the histogram into Q bins per axis. The dataset now consists of L*a*b* color histograms.

4. Apply PCA on the dataset from step 3 and keep 99.9% of the total energy. Record the number of principal components.

5. Repeat from step 2, with histogram quantization set to $Q \in \{4, 8, 16, 32, 64\}$

6. Repeat step 5 10 times, each trial draws a random dataset from the tiles selected in step 1.

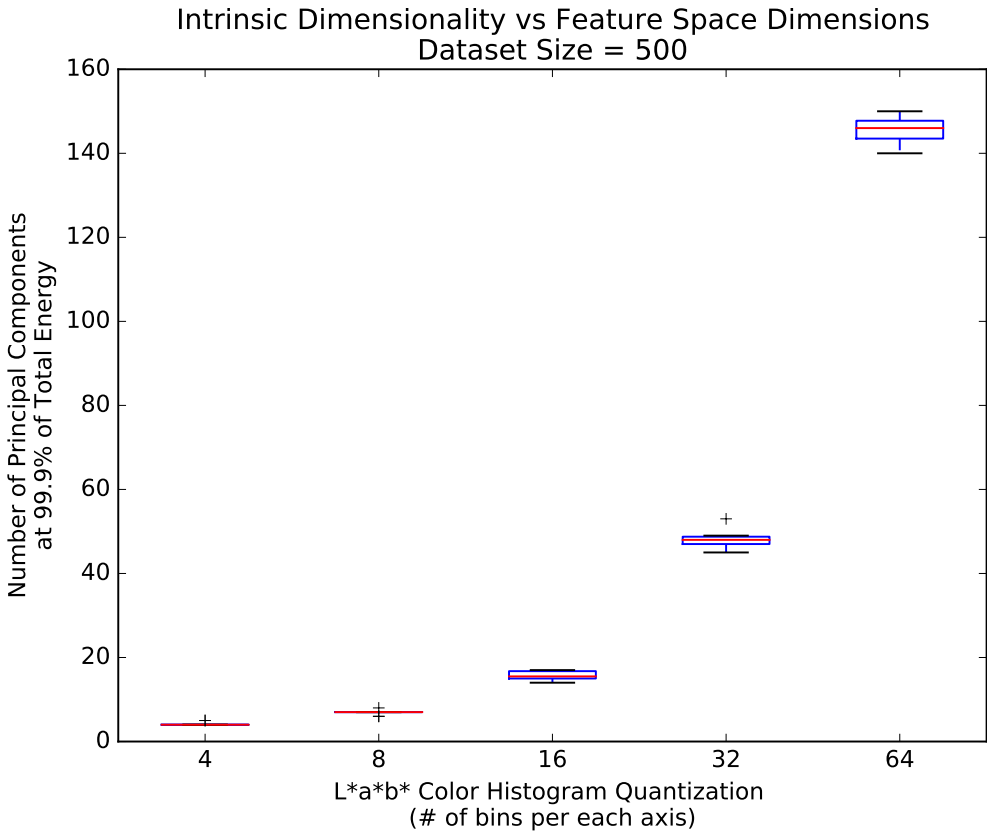


Fig. 4.12. Each boxplot summarizes, over 10 trials, the number of principal components after applying PCA on the quantized histograms.

From Fig. 4.12, we see that as the number of bins in the color histogram increases, the number of principal components also increases.

4.4 Histogram Quantization and Cosine Distance

In this section, we investigate how Cosine distance between two histograms behave as the number of histogram bins changes. To answer this question, we first make the following observations:

- The larger the histogram bin size, the fewer the total number of bins and thus fewer dimensions in the feature space.

- The angle between two histograms can become larger, smaller, or remains the same as we change the bin size.

Next, let us look at the effect of quantization on numerical values. A simple and common approach to quantization is shown in the equation below:

$$q = \left\lfloor \frac{x}{\text{binSize}} \right\rfloor \tag{4.1}$$

where x is normalized to $[0 \ldots 1)$ and $1/\text{binSize}$ is the number of quantization levels. The overall quantization effect is that x either remains at the same position, or it moves to the left on the number line. Similarly, in 2D, a point can move down, left, diagonally, or remains at the same position. See Fig. 4.13 for an illustration.

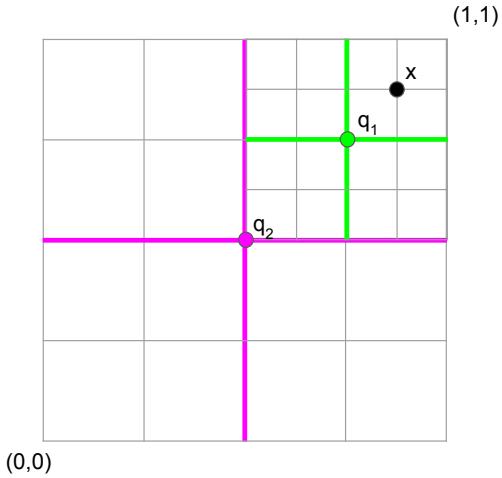


Fig. 4.13. The effect of quantization using equation (4.1) on 2D data. In this illustration, the original point x gets moved further down and left after each quantization step (i.e., binSize $= \frac{1}{4}$ for point q_1 and $\frac{1}{2}$ for q_2.). If the original point were at q_2, then, the two quantization steps will have no effect (i.e., dose not introduce any quantization error).

Now, let's take a look at the process of creating a histogram from a collection of 2D points. As illustrated in Fig 4.14, each cell in the 2D grid becomes a bin, forming a 3D bar graph. As for the effect of quantization on the histogram, we observe the following:

- Smaller adjacent bins get combined into a larger bin.
- Bins get fatter and taller after quantization.
- Quantization reduces the total number of bins in the histogram.

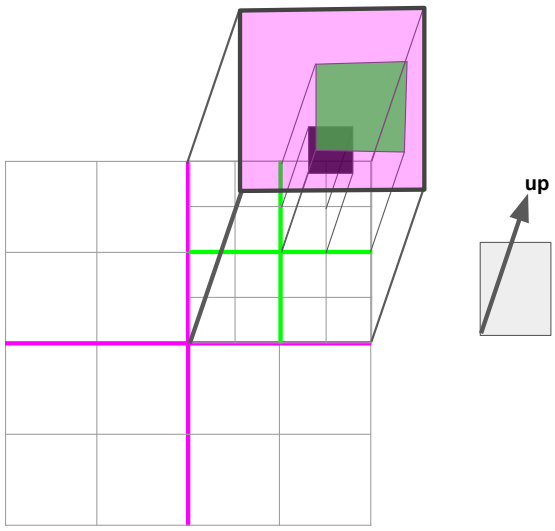


Fig. 4.14. A histogram of 2D points looks like a 3D bar graph. The height of the cell corresponds to the number of points in the cell. In this illustration, there are three levels of quantizations — the black, green, and pink bins. Their bin sizes are $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{1}{2}$, respectively.

Furthermore,

- Converting a multi-dimensional histogram into a 1D histogram has no effect on the computation of Cosine distance. This property is a direct consequence of the dot product definition – ordering of addition does not matter.
- After quantization, the angle between two histograms can behave in three ways:
  - Case1: angle stays about the same.

    Obviously, when two histograms are identical, the angle between them is always 0 regardless of the quantization level. Another way for the angle

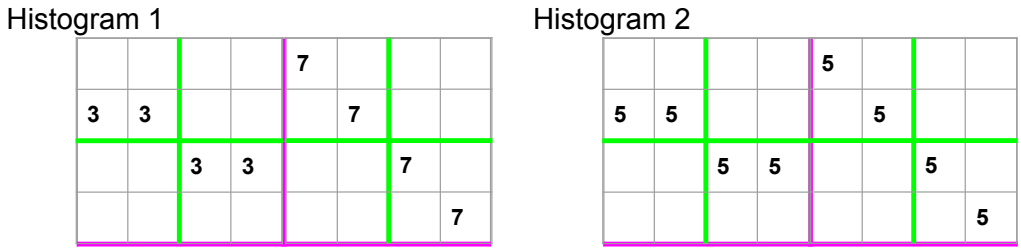
to remain the same is when the non-zero counts inside the merged bin all have the same count value. See Fig 4.15 for an illustration.

– Case2: angle becomes bigger.

  After quantization, it could happen that a merged bin contains just one non-zero bin with large count while the rest of the merged bins contain many non-zero bins with small counts. When this happens, the isolated dominant bin can remains dominant after quantization. If two histograms formed in this way have the same dominant bin location but differ significantly in their count values, then the angle between them will be larger after quantization. Fig 4.16 for an illustration.

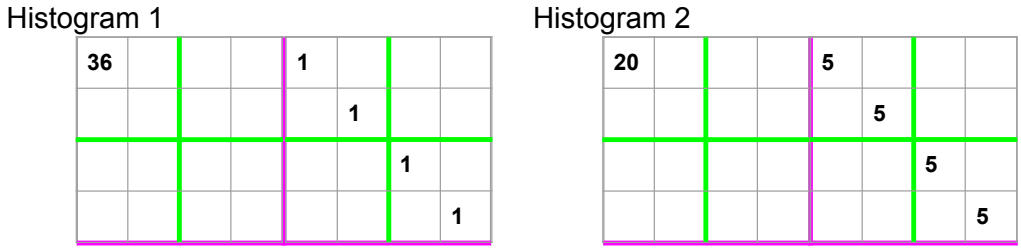
– Case3: angle becomes smaller.

  The angle between two histograms could become smaller after quantization when the two histograms are uniformly distributed. That is, the non-zero bins all have similar counts and locations. See Fig 4.17 for an illustration.



- **High-res**: AngDist ([3,3,3,3,7,7,7,7], [5,5,5,5,5,5,5,5]) = 21.8
- **Mid-res**: AngDist ([6,6,14,14], [10,10,10,10]) = 21.8
- **Low-res**: AngDist ([12,28], [20,20]) = 21.8

Fig. 4.15. Angle between two histograms can remain the same after quantization. Consider two histograms having the same non-zero bin locations. Quantization has no effect on the angle between the two histograms as long as all non-zero smaller bins have the same count inside the bigger merged bin. For example, in histogram 1, {3,3} gets merged into {6}, {7,7} gets merged into {14}, {6,6} gets merged into {12}, {14,14} gets merged into {28}, etc. The angular distance between the two histograms remains the same at 21.8 degrees.



- **High-res**: AngDist([36,1,1,1,1], [20,5,5,5,5]) = 23.3
- **Mid-res**: AngDist([36,2,2], [20,10,10]) = 30.7
- **Low-res**: AngDist([36,4], [20,20]) = 38.6

Fig. 4.16. Angle between two histograms can get bigger after quantization. This can happen if the dominant bin remains throughout the quantization as illustrated in this particular example. The angular distance increases from 23.3 to 38.6.

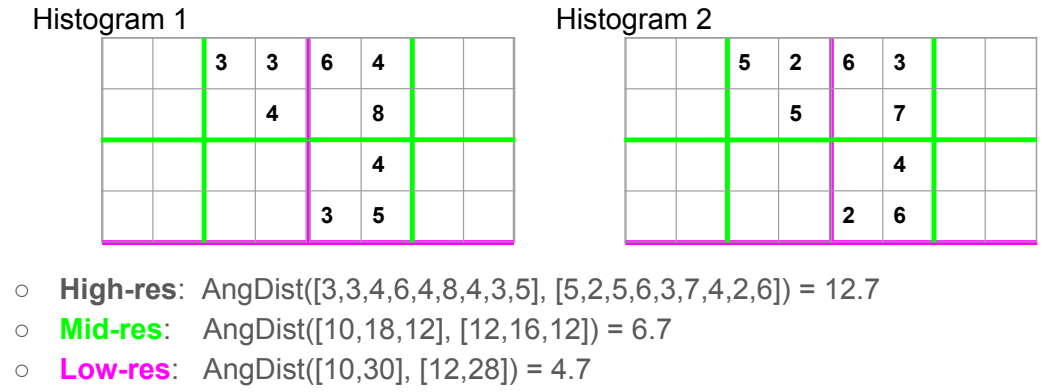


Fig. 4.17. Angle between two histograms can get smaller after quantization. This can happen when histograms are uniformly distributed. That is, the non-zero bins all have similar counts and locations. As non-zero bins get merged, the resulting histograms become more and more similar. In this particular example, the angular distance decreases from 12.7 to 4.7.

In general, error in measurement can be due low sensor resolution in the measuring instrument. This leads to quantization error. For Cosine distance, quantization error can go in either direction, resulting in the angle between two histograms to either increase, decrease, or remains the same after quantization.

4.4.1 A Closer Look at Histogram Quantization

Mathematically, the angle between two vectors is a function of Cosine similarity, which is computed by taking the ratio of dot product and product of magnitudes:

$$\cos(\theta) = \frac{\vec{v}_1 \cdot \vec{v}_2}{\|\vec{v}_1\|\|\vec{v}_2\|} \tag{4.2}$$

Observe that the angle, θ, between the two vectors, $\vec{v}_1$ and $\vec{v}_2$, is large when the Cosine similarity is small and vice versa. This means, the numerator and the denominator in equation (4.2) can grow at different rates after quantization. If the numerator grows faster than the denominator after quantization, then, the Cosine

similarity will get larger, and we will see the angle between the two vectors get smaller after quantization.

Therefore, to predict the effect of histogram quantization, one has to be able to predict the growth rates of the dot product and the growth rate of the magnitudes. These growth rates depend on the the distribution of the histogram and a detailed analysis of this dependency seems difficult. Nevertheless, in the next section, we will demonstrate two specific cases using simulated datasets.

4.4.2 Using Simulated Datasets to Demonstrate the Effects of Histogram Quantization on Cosine Distance

To demonstrate the effect of histogram magnetization, our procedure is as follows:

1. Create a simulated dataset consisting of 100 image patches, each of 101 x 101 pixels.
2. Using a quantization level = Q bins per axis, compute the L*a*b* color histogram for each image patch. The dataset now consists of 100 L*a*b* color histograms.
3. Compute all pair-wise angular distance among the histograms in the dataset. Record the average value.
4. Repeat from step 2 with a different quantization $Q \in [4, 8, 16, 32, 64]$.
5. Repeat from step 1 10 times, each trial draws a new dataset.

With regard to how we simulate the datasets, we first define two populations below:

- Uniform Population - Each unit in this population is a color in the CIELAB color space. In other words, the population consists of units uniformly distributed over the entire CIELAB color space.
- Dominant Population - Unlike the Uniform Population, there are two clusters of colors in this population. One is a small cluster modeled by a Gaussian

distribution with a small variance. The other cluster is bigger and further away.

Next, we create two datasets from these two populations:

- Uniform Dataset - Each datum is a 101x101 L*a*b* image with pixels drawn randomly from the Uniform Population. We create the dataset this way to model the example given in Fig. 4.17.

- Dominant Dataset - Unlike the Uniform Dataset, this dataset consists of L*a*b* image patches with varying portions of pixels drawn from both clusters in the Dominant Population. Specifically, half of the dataset consists of image patches each having 90% of pixels drawn from the smaller cluster and the rest 10% drawn from the the bigger cluster. The other half of the dataset is constructed in the same way except the role of clusters are reversed – i.e., 10% from the smaller cluster and 90% from the bigger cluster. We create the dataset this way to model the example given in Fig. 4.16.

Finally, for each of the two dataset types, we create five boxplots, one per each quantization level. The boxplots summarize the average angle between pairwise histograms over 10 trials. See Fig 4.18 and Fig 4.19 for the boxplots.

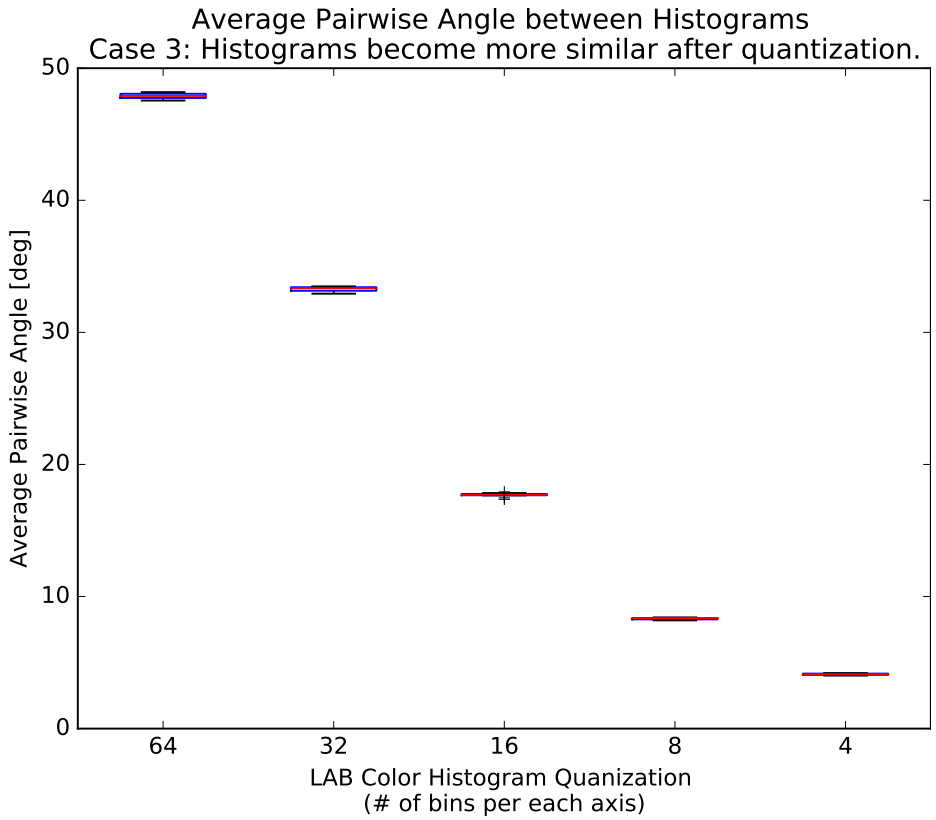


Fig. 4.18. Boxplots of five quantization levels. Each boxplot summarizes, over 10 simulated datasets, the average angle between $\binom{100}{2}$ pairwise histograms. The Uniform Dataset exhibits the phenomenon that indicates histograms becoming more similar after quantization. At 64-bins per axis, the average angle between pairwise histograms is about 50 degrees. On the other hand, at 4-bins per axis, the average pairwise angle goes down to about 5 degrees.

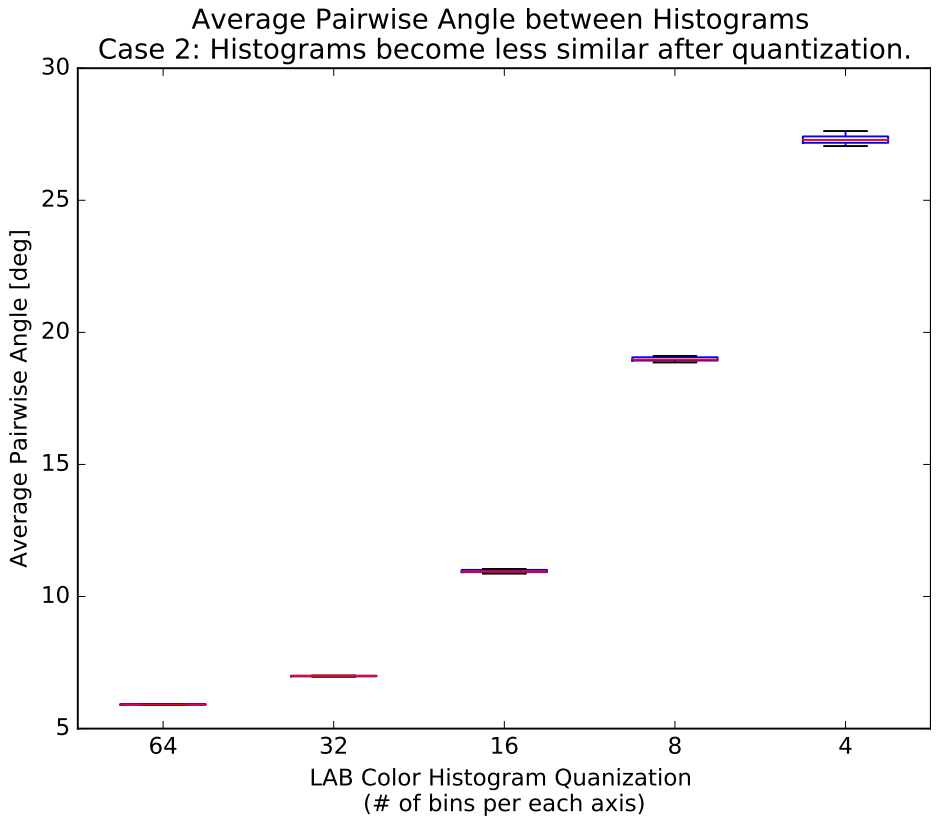


Fig. 4.19. Boxplots of five quantization levels. Each boxplot summarizes, over 10 simulated datasets, the average angle between $\binom{100}{2}$ pairwise histograms. The Dominant Dataset exhibits the phenomenon that indicates histograms becoming less similar after quantization. At 64-bins per axis, we see that the average angle between pairwise histograms is about 6 degrees. On the other hand, at 4-bins per axis, the average pairwise angle goes up to about 27 degrees.

4.5 Calculating Cosine Distance from PCA Representation

Since the PCA representation is zero-mean based, using the Cosine distance metric directly on the reduced-dimensionality vectors would not be appropriate because we may want to apply a similarity threshold that is tuned on one dataset to the PCA representation of another dataset that may have a very different data distribution, and thus different mean.

The baseline approach to computing Cosine distance on PCA representation is to first reconstruct the data from its low-dimensional representation back to the original high-dimensional space and then compute the Cosine distance based on the reconstructed data. This approach is computational expensive. Fortunately, as we will show next, we can get the same result by first shifting the PCA representation by the projected mean and then augment the shifted result with an additional "bias dimension".

Let $\{\vec{x}_1, \ldots, \vec{x}_n\}$ be the original dataset in the high dimensional space and let $\boldsymbol{X}$ be the column matrix representation of the data, i.e., $\boldsymbol{X} = [\vec{x}_1 \ldots \vec{x}_n]$. Similarity, let $\{\vec{v}_1, \ldots, \vec{v}_n\}$ be the augmented low-dimensional vectors. The procedure for computing $\vec{v}_i$ is as follows:

1. Compute the mean vector $\vec{m}$, and the individual deviations $\vec{z}_i$ as shown in Eq. 4.3 and 4.4.

$$\vec{m} = \frac{1}{n} \sum_{i=1}^{n} \vec{x}_i \tag{4.3}$$

$$\vec{z}_i = \vec{x}_i - \vec{m} \tag{4.4}$$

2. Obtain the deviation matrix $\boldsymbol{Z}$ using Eq. 4.5.

$$\boldsymbol{Z} = [\vec{z}_1 \ldots \vec{z}_p] \tag{4.5}$$

3. Obtain the eigenvectors and eigenvalues of the covariance matrix , $\boldsymbol{Z}\boldsymbol{Z}^T$, and let $\boldsymbol{W_k}$ be the column matrix consists of the k most-significant eigenvectors.

$$\boldsymbol{W_k} = \text{k most-significant eigenvectors of } \boldsymbol{Z}\boldsymbol{Z}^T \tag{4.6}$$

4. Compute $\vec{y}$ and $\vec{o}$, the PCA representation and the projection of the mean by Eq. 4.7 and Eq. 4.8, respectively.

$$\vec{y}_i = \boldsymbol{W_k}^T \vec{z}_i \tag{4.7}$$

$$\vec{o} = \boldsymbol{W_k}^T \vec{m} \tag{4.8}$$

5. Shift the PCA representation by the projected mean, as shown in Eq. 4.10.

$$\vec{y'}_i = \vec{y}_i + \vec{o} \tag{4.9}$$

$$= \vec{y}_i + \boldsymbol{W_k}^T \vec{m} \tag{4.10}$$

6. Augment the shifted low-dimensional vector, $\vec{y'}_i$, by a bias dimension. The final augmented low-dimensional representation, $\vec{v}_i$, is then given in Eq. 4.13.

$$\cancel{\boldsymbol{W_k}} = \text{rest of eigenvectors not in } \boldsymbol{W_k} \tag{4.11}$$

$$\vec{v}_i = \begin{bmatrix} \vec{y'}_i \\ \|\cancel{\boldsymbol{W_k}}^T \vec{m}\| \end{bmatrix} \tag{4.12}$$

$$= \begin{bmatrix} \vec{y}_i + \boldsymbol{W_k}^T \vec{m} \\ \|\cancel{\boldsymbol{W_k}}^T \vec{m}\| \end{bmatrix} \tag{4.13}$$

To prove that Cosine similarity between data in this augmented lower-dimensional space is the same as the Cosine similarity between the corresponding reconstructed data in the original high-dimensional space, we just need to verified that the dot products are equal. First, let $\vec{r}_i$ be reconstructed from PCA representation, $\vec{y}_i$, using Eq. 4.14, then, we verify the result $\vec{r}_i^T \vec{r}_j = \vec{v}_i^T \vec{v}_j$ in equations Eq. 4.15 to Eq. 4.26:

$$\vec{r}_i = \boldsymbol{W_k} \vec{y}_i + \vec{m} \tag{4.14}$$

$$\vec{r}_i^T \vec{r}_j = (\boldsymbol{W_k} \vec{y}_i + \vec{m})^T (\boldsymbol{W_k} \vec{y}_j + \vec{m}) \tag{4.15}$$

$$= (\vec{y}_i^T \boldsymbol{W_k}^T + \vec{m}^T)(\boldsymbol{W_k} \vec{y}_j + \vec{m}) \tag{4.16}$$

$$= \vec{y}_i^T \boldsymbol{W_k}^T \boldsymbol{W_k} \vec{y}_j + \vec{y}_i^T \boldsymbol{W_k}^T \vec{m} + \vec{m}^T \boldsymbol{W_k} \vec{y}_j + \vec{m}^T \vec{m} \tag{4.17}$$

$$= \vec{y}_i^T \vec{y}_j + \vec{y}_i^T \boldsymbol{W_k}^T \vec{m} + \vec{m}^T \boldsymbol{W_k} \vec{y}_j + \|\vec{m}\|^2 \tag{4.18}$$

$$\vec{v}_i^T \vec{v}_j = \left[(\vec{y}_i + \boldsymbol{W_k}^T \vec{m})^T \ \|\cancel{\boldsymbol{W_k}^T \vec{m}}\| \right] \begin{bmatrix} \vec{y}_j + \boldsymbol{W_k}^T \vec{m} \\ \|\cancel{\boldsymbol{W_k}^T \vec{m}}\| \end{bmatrix} \tag{4.19}$$

$$= (\vec{y}_i + \boldsymbol{W_k}^T \vec{m})^T (\vec{y}_j + \boldsymbol{W_k}^T \vec{m}) + \|\cancel{\boldsymbol{W_k}^T \vec{m}}\|^2 \tag{4.20}$$

$$= (\vec{y}_i^T + \vec{m}^T \boldsymbol{W_k})(\vec{y}_j + \boldsymbol{W_k}^T \vec{m}) + \|\cancel{\boldsymbol{W_k}^T \vec{m}}\|^2 \tag{4.21}$$

$$= \vec{y}_i^T \vec{y}_j + \vec{y}_i^T \boldsymbol{W_k}^T \vec{m} + \vec{m}^T \boldsymbol{W_k} \vec{y}_j + \vec{m}^T \boldsymbol{W_k} \boldsymbol{W_k}^T \vec{m} \tag{4.22}$$

$$= \vec{y}_i^T \vec{y}_j + \vec{y}_i^T \boldsymbol{W_k}^T \vec{m} + \vec{m}^T \boldsymbol{W_k} \vec{y}_j + \vec{m}^T (I - \cancel{\boldsymbol{W_k} \boldsymbol{W_k}^T}) \vec{m} + \|\cancel{\boldsymbol{W_k}^T \vec{m}}\|^2 \tag{4.23}$$

$$= \vec{y}_i^T \vec{y}_j + \vec{y}_i^T \boldsymbol{W_k}^T \vec{m} + \vec{m}^T \boldsymbol{W_k} \vec{y}_j + \vec{m}^T \vec{m} - \vec{m}^T \cancel{\boldsymbol{W_k} \boldsymbol{W_k}^T} \vec{m} + \|\cancel{\boldsymbol{W_k}^T \vec{m}}\|^2 \tag{4.24}$$

$$= \vec{y}_i^T \vec{y}_j + \vec{y}_i^T \boldsymbol{W_k}^T \vec{m} + \vec{m}^T \boldsymbol{W_k} \vec{y}_j + \|\vec{m}\|^2 - \|\cancel{\boldsymbol{W_k}^T \vec{m}}\|^2 + \|\cancel{\boldsymbol{W_k}^T \vec{m}}\|^2 \tag{4.25}$$

$$= \vec{y}_i^T \vec{y}_j + \vec{y}_i^T \boldsymbol{W_k}^T \vec{m} + \vec{m}^T \boldsymbol{W_k} \vec{y}_j + \|\vec{m}\|^2 \tag{4.26}$$

4.6 Dimensional Reduction Using FastMap

There exist many dimensionality reduction strategies for multidimensional data, however most are not appropriate for the big data scenarios involving tens of millions of patches extracted from satellite images. Consider, for example, what is perhaps the most commonly used method for dimensionality reduction, PCA, which carries out an eigendecomposition of the covariance matrix of the data. In our case, the data would consist of 9024-element vectors for the L*a*b* color histograms, whose covariance matrix would be of size 9024×9024, a matrix with close to 100 million elements. Now if we only had a small number of patches from which to generate the reduced-dimensionality representation, we could take advantage of the fact the rank of the covariance matrix would not exceed the number of patches available and translate that fact into a highly efficient algorithm for the eigendecomposition of the covariance matrix [36]. However, that is not case with the work described in this research — with the number of patches running into millions, we have no hard constraint on the rank of the covariance matrix. We run into similar problems if we try to use the other methods, such as Incremental PCA [37] and Sparse PCA [38]. In light of these

difficulties associated with the more traditional approaches, we have chosen to use the FastMap [39] method for our work.

FastMap works by preserving, up to certain level of precision depending on the stopping criterion used, all pairwise distances among the entire dataset while reducing the data dimensionality as much as possible. The algorithm has time complexity of $O(d \times n)$ where d is the number of dimensions after reduction and n the size of the dataset.

The FastMap algorithm involves just three basic steps: First, it takes the entire dataset and quickly finds the two furthest points away from each other. Then, taking the line joining these two points as the first axis in the reduced dimensionality representation of the data, the algorithm then projects all the data points onto this line to calculate the first coordinate value of the points in the reduced-dimensionality representation. Finally, the algorithm projects all the data points into a hyperplane perpendicular to the axis just constructed. These three steps are repeated with the data projected into the hyperplane until the stopping criterion is met. Fig. 4.20 summarizes these three basic steps and Appendix B gives details on how the projections are computed.

As for choosing the stopping criterion, one could obviously stop when the desired number of dimensions is reached. In general, though, that is not likely to be a useful criterion since one would not know the desired dimensionality in advance. A more useful criterion consists of stopping the iterations when the low-dimensional subspace retains a certain specified fraction of the total variation in the data.

To elaborate, note that each axis is formed by a line segment like the (P_a, P_b) segment shown in Fig. 4.20. The length of this segment determines the range of data variation on that axis. The data variation decreases in each iteration. Therefore, we can stop when the segment length is less than some fraction of the sum of the segment lengths encountered so far. More formally, let L_i be the length of the line segment at iteration i, we stop when the following inequality is satisfied:

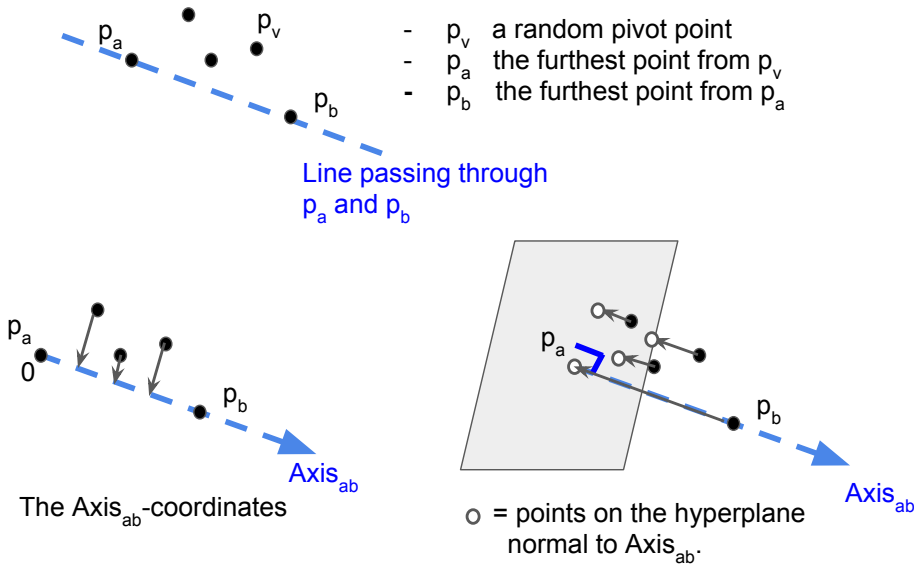


Fig. 4.20. FastMap first finds p_a and p_b, two furthermost points away for each other. This step takes $2 \times n$ comparisons, where n is the number of points. It then maps all points onto the line segment formed by this pair of points for estimating the first coordinate of all the points in the reduced-dimensionality representation. Finally, it maps all points into a hyperplane that is normal to the line passing through p_a and p_b. These three steps are repeated with the points in the hyperplane. Each such repetition adds one more dimension to the low-dimensional representation of the entire dataset.

$$\frac{L_d}{\sum_{i=1}^{d} L_i} < T \tag{4.27}$$

As for the value of T, we found that the value of 0.001 (which is tantamount to retaining 99.9% of the total variation) limits the pair-wise Cosine distance error to less than one degree. Note that we obtained this result using our data, which are normalized histograms.

4.6.1 Calculating Cosine Distance from Fastmap Representation

The Fastmap representation can not be used to calculate Cosine directly because Fastmap works to preserve pairwise distances in the Euclidean space. Fortunately, we can shift the resulting low-dimensional representation in such a way to allow recovering of the Cosine distances in the original high-dimensional space. The procedure is as follows:

1. Insert the null vector, $\vec{0}$, into the dataset that consists of normalized histograms.

2. Using Fastmap, obtain the low-dimensional representation of the entire dataset. Let this dataset be $\boldsymbol{F} = \{\vec{f_0}, \vec{f_1} \dots \vec{f_n}\}$, where $\vec{f_0}$ is the projection of the null vector.

3. Shift the low-dimensional representation by $\vec{f_0}$ and remove it from the dataset. Specifically, let $\boldsymbol{V} = \{\vec{v}_1 \dots \vec{v}_n\}$ be the final low-dimensional representation, construct $\boldsymbol{V}$ using Eq. 4.28:

$$\boldsymbol{V} = \{\vec{f_i} + \vec{f_0}\}, \forall i \in [1 \dots n] \tag{4.28}$$

5. CONCISE-SET REPRESENTATION

Obtaining an accurate estimate of a land-cover classifier's performance over a wide geographic area is a challenging problem due to the need to generate the ground truth that covers the entire area that may be thousands of square kilometers in size.

The current best approach that addresses this problem constructs a testing dataset by drawing samples randomly from the entire area — with a human supplying the true label for each such sample — with the hope that the selections thus made statistically capture all of the data diversity in the area. A major shortcoming of this approach is that the datasets thus generated tend not to be concise since, in a human-computer interactive session that may last a long time, it is difficult for a human to ensure that the information provided by the next data element chosen by the random sampler for human annotation is non-redundant with respect to the data already collected — even when the new data element belongs to a different geographic location.

In order to reduce this annotation burden[1] on the human, it makes sense to remove any redundancies from the entire dataset before presenting its samples to a human for annotation. Towards that end, this chapter presents a framework that uses a combination of clustering and compression for creating a concise-set representation of the land-cover data for a large geographic area. Whereas clustering is achieved by applying Locality Sensitive Hashing (LSH) to the data elements, compression is achieved through choosing a single data element to represent a given cluster. Both these steps are applied to the raw data — that is, to the data prior to its annotation by a human. This considerably reduces the annotation burden on the human and

[1]By annotation burden, we mean the effort involved not only in assigning the ground-truth labels to the data samples, but also in looking for informative and non-redundant data samples. This effort is not to be confused with annotation efficiency, which may involve using human-computer software tools to collect and annotate large sets of data samples quickly.

makes it more likely that the human would persevere during the annotation stage until — hopefully — all of the data diversity has been adequately captured.

In the next chapter, we validate our framework experimentally by comparing it with the traditional random sampling approach using WorldView2 satellite imagery.

5.1 Related Work

In addition to the straightforward Simple Random Sampling (SRS) approach, researchers have also proposed "Stratified Sampling" and "Systematic Sampling" for creating representing datasets from large populations. The systematic sampling approach first sorts the population into a list and then sub-samples from this sorted list. On the other hand, the stratified sampling approach divides the population into homogeneous subgroups and then samples within each subgroup using either SRS or systematic sampling [40]. The authors of [41] have proposed a complex 3-level stratified sampling approach and make use of various prior knowledge about the dataset. In their work, the first level of stratified sampling organizes the mapped areas by their meta-data such as the mapping method, remote sensing source, resolution, acquisition date, etc. The second level uses prior knowledge about the content of the mapped area (e.g., road, building, green areas, etc). And the third level uses finer features present in the mapped areas such as the location of the individual object/inspection unit. On the other hand, [42] proposes a simpler two-stage cluster sampling approach based on the classification map. The approach we present here can be considered to be a combination of the basic notions in Stratified Sampling and Systematic Sampling.

With regard to land-cover classification, early classifiers that could be employed over large geographic areas use only low and medium spatial resolution imagery ranging from 8km to 15m per pixel [43–45]. However, during the last ten years, spatial resolution in satellite images has improved rapidly. It is now common to find satellite data at a very high resolution (VHR) of 0.5m per pixel or better. Fortunately, during the same time period, computer processors, memory, and storage all have become

faster and cheaper. Today, it is not uncommon any longer for research labs to work on land-cover classifiers involving large datasets [46] and VHR satellite images [47].

5.2 Proposed Approach

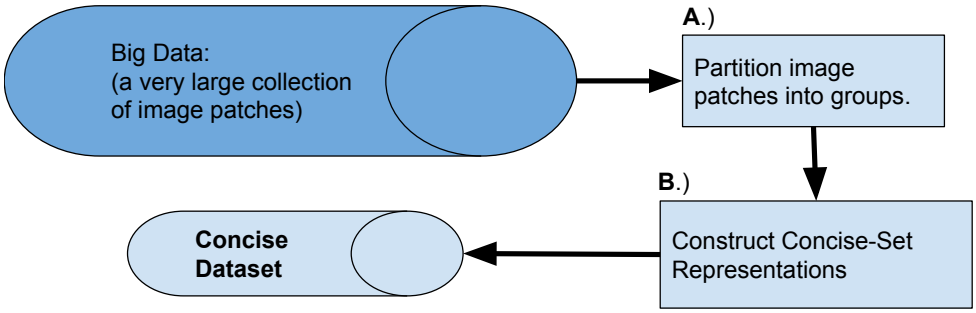


Fig. 5.1. The overall system block diagram.

In Fig. 5.1, we show a block diagram for the overall framework. Block A of the figure, "Partition image patches into groups", is optional but needed if you wish to create disjoint multiple evaluation datasets for testing a classifier and do so through a single human-computer annotation-elicitation session.[2] The algorithm itself, which is the main focus of this chapter, is in Block B, "Construct Concise-Set Representations", of the figure. As we will demonstrate, this algorithm uses a hybrid sampling approach to select a concise (meaning, non-redundant) dataset of the entire data.

5.2.1 Representation of the Population: Content, Unit, and Size

The first issue to resolve when eliciting ground-truth annotations from a human is the content of each "unit" of the data that is shown to the human. Even though the end-goal of annotation elicitation is to collect the class labels for a collection of pixels,

[2]As to why one would want to create disjoint evaluation datasets for a classifier is discussed in Section 5.3.

a human observer is often not able to make a judgment about the class label of an individual pixel if it is shown as a single piece of data without any neighboring pixels. Experience has taught us that it is best to show a pixel along with its immediate surround, which we will refer to as a *patch*, in order for a human to figure out what the ground-truth label should be at the center of the patch. The human would be asked to label only the central pixel in a patch , with all the surrounding pixels merely providing a geographic context for the pixel at the center.

5.2.2 Measuring the Similarity Between Satellite Image Patches

To determine whether two image patches are similar, we first represent each patch with two different feature vectors, one for the color histogram that represents the background pixels and the other for the spectral values at the foreground pixel. For a patch to be considered similar to another patch, the similarity criterion must be satisfied for both the background and the foreground .

To compute the color histogram for the background pixels, we first transform the RGB color space to the perceptually uniform CIELAB color space. Then, we quantize that space into b^3 bins, where b is the number of bins along each of the L*, a*, and b* axes. Note that some of these b^3 bins will always be empty since the RGB color space is a subset of the CIELAB color space [48] (See Section 4.1.1 in Chapter 4 for an illustration of the CIELAB color space). Representing this histogram with a b^3-dimensional vector, we subsequently reduce its dimensionality by, first, retaining only the valid L*a*b* histogram bins, and, then, by applying FastMap as described in Sections 6.1.2 and 5.2.4, respectively. We will use d to denote the retained dimensionality for the color representation of the background pixels.

To measure the similarity between any two background histograms, we use the angular distance metric defined by Eq. 5.1:

$$\text{dist}_{\text{Angle}}(\vec{v}_1, \vec{v}_2) = \frac{180}{\pi} \cos^{-1}\left(\frac{\vec{v}_1^T \vec{v}_2}{|\vec{v}_1|\ |\vec{v}_2|}\right) \tag{5.1}$$

where $\vec{v}_1$ and $\vec{v}_2$ are the d-dimensional vector representations of the two histograms, respectively. Generally, we normalize the histogram vectors so that they are of unit magnitude, which does away with the denominator in the above formula.

As for characterizing the foreground (i.e., the center pixel of the image patch), we use its spectral signature consisting of the spectral responses measured at the corresponding surface location on earth. For example, in a 4-band satellite image, each pixel has 4 values, one per spectral band, and the spectral signature is a vector consisting of these four numbers. To measure the similarity between any two spectral signatures, we use the "L1 distance metric". Note that both the Cosine distance metric for the histogram vectors and the L1 metric for the foreground spectral vectors are fast to compute – an important consideration in big-data processing.

5.2.3 Similarity Search

Since a patch is represented by two *semantically different* characterizations — a d-dimensional vector for the background color distribution and a 4-dimensional vector for the foreground spectral values — that raises the question of how to actually form the similarity groups, especially because we want to enforce the similarity constraint on the two characterizations conjunctively. Note that, in our big-data context, we do not have the luxury of comparing every pair of patches to decide whether or not they belong to the same similarity group.[3] We have three options:

Option 1: Concatenate the d-dimensional color-histogram vector for the background with the 4-dimensional spectral-property vector for the foreground to form a single vector representation for a patch and then apply one of several distance metrics to the vectors for comparing the similarity of the patches.

[3]If it were possible to compare *every* pair of the patches, the two similarity conditions could be enforced simultaneously in each pair-wise comparison.

Option 2: First cluster the patches with respect to just the foreground pixels and then subject each of the clusters thus obtained to further sub-clustering on the basis of the similarity of the background color-histogram vectors.

Option 3: By reversing the two steps in the previous option; that is, by first clustering the patches with respect to the background color-histogram vectors and then further sub-clustering those clusters on the basis of the similarity of the foreground spectral vectors.

Despite its appearance to the contrary, the first option listed above is not appropriate since it cannot guarantee conjunctive enforcement of the two separate and distinct similarity constraints. And the second and the third options are logically equivalent.

What is interesting is that while the second and the third options are logically equivalent, they entail different degrees of computational effort to arrive at the same final conclusion. The main reason for that has to do with how the vectors that represent the color histograms for the background are distributed vis-a-vis the distribution of the spectral vectors that represent the foreground.

The distribution of the histogram vectors is such that with the linear-time LSH algorithm applied to such vectors, it is possible to implement a fast approximated solution for creating sufficiently small clusters so that a subsequent pairwise comparison of the samples within each histogram-similarity based cluster for enforcing the similarity of the spectral vectors results in an overall computationally efficient structure for the conjunctive enforcement of the two different similarity constraints. The opposite approach would consist of first applying a Euclidean-distance based LSH to cluster the entire data on the basis of the similarity of the spectral vectors and then subjecting the data points withing each resulting cluster to the histogram based similarity constraint. Unfortunately, the clusters generated by the second approach tend to be much loo large, making the overall computation relatively inefficient. For

the reasons explained above, we chose Option 3 for the conjunctive enforcement of the two different similar constraints.

Permeating all three options listed above, including obviously our chosen Option 3, are the consequences of the non-transitivity of the similarity constraints that we mentioned earlier in the Introduction. As stated there, if we were to apply any of the three options to the entire data set, we are highly likely to end up with a single concise set, which is not a very useful thing to happen. To get around this difficulty, we introduce the notion of *similarity graph* in Section 5.2.5. A similarity graph is generated by applying a pairwise spectral comparison criterion to all the patches considered similar by the LSH algorithm (on the basis of the background similarity through the histograms associated with the backgrounds). These pairwise comparisons yield what we call similarity neighborhoods. Every patch in a given similarity neighborhood is *directly* within the similarity distance of the patch for which the similarity neighborhood was constructed. The collection of all such similarity neighborhoods constitutes the similarity graph. Note that it is likely that there would patches that would be shared by different nodes.

In the rest of this section, we first describe how we reduce the data dimensionality of the color-histogram vectors before clustering them using LSH. Subsequently, we bring in the spectral data vectors for the foreground pixels to further refine the clusters.

5.2.4 Reducing the Dimensionality of the Histogram Representation for the Background Pixels in a Patch

As explained in Section 5.2.2, the background pixels in a patch are represented through a three dimensional histogram in the L*a*b* space. As we will show in Sections 6.1.1 and 6.1.2, the bin structure used for the histogram results in a vector representation of the background that has 9024 elements in it. This is obviously much too large a dimensionality. Fortunately, with dimensionality reeducation, we

can bring it down to less than 100, depending on the data in the ROI (Region of Interest).

There exist many dimensionality reduction strategies for multidimensional data, however most are not appropriate for the big data scenarios involving tens of millions of patches extracted from satellite images. Consider, for example, what is perhaps the most commonly used method for dimensionality reduction, PCA, which carries out an eigendecomposition of the covariance matrix of the data. In our case, the data would consist of 9024-element vectors for the L*a*b* histograms, whose covariance matrix would be of size 9024×9024, a matrix with close to 100 million elements. Now if we only had a small number of patches from which to generate the reduced-dimensionality representation, we could take advantage of the fact the rank of the covariance matrix would not exceed the number of patches available and translate that fact into a highly efficient algorithm for the eigendecomposition of the covariance matrix [36]. However, that is not case with the work described in this research — with the number of patches running into millions, we have no hard constraint on the rank of the covariance matrix. We run into similar problems if we try to use the other methods, such as Incremental PCA [37] and Sparse PCA [38]. In light of these difficulties associated with the more traditional approaches, we have chosen to use the FastMap [39] method for our work. We give an overview of FastMap in Section 4.6 of Chapter 4.

5.2.5 Creating a Similarity Graph for the Image Patches

Keeping in mind that the vector representations for the background color histograms can still reside in a high-dimensional space after dimensionality reduction, even the supposedly efficient algorithms that avoid exhaustive pairwise comparisons, such as those based on nearest neighbor search (NNS) with KD-trees, SR-trees, and cover trees [2–6] are not appropriate for solving our problem of forming similarity neighborhoods from the background color-histogram vectors because their perfor-

mance (either the running time or the memory requirement) degrades exponentially as the data dimensionality increases.

Locality Sensitive Hashing (LSH) [49, 50], on the other hand, has emerged as an attractive alternative to tree-based nearest neighbor search algorithms for high-dimensional data. Just like the tree-based approaches, LSH does not make exhaustive pair-wise comparisons. Additionally, and most importantly, LSH can be implemented to have constant average search time, making it highly desirable for similarity based searching in very large datasets. The only drawback is that LSH is an approximated nearest neighbor (ANN) algorithm and may not always find the exact nearest neighbor. Nevertheless, LSH is suitable for applications when datasets are large and finding the exact nearest neighbor isn't critical. It has been shown that for high dimensional data, LSH significantly outperforms SR-tree, a representative of tree-decomposition-based indexing techniques [51].

To briefly review how LSH works, as its name implies, LSH uses locality sensitive hashing for nearest neighbor search. A hash function is considered to be locality sensitive if it places "nearby" samples in the same bucket with a high probability, and if it places "far apart" samples in different buckets, again with a high probability. Two data samples are considered to be "nearby" if the distance between them is at most d_1 and two data samples are considered "far apart" if the distance between them is at least $d_2 = c \times d_1$, where $c > 1$ is the approximation factor. The quality of such a hash function is measured by two probabilities p_1 and p_2, where the former is the probability of collision for "nearby" samples and the latter the probability of collision for "far apart" samples. For obvious reasons, you'd want p_1 to be as high as possible and p_2 to be as low as possible.

In practice, it is not possible to find a single hash function with the property described above. However, it has been shown that a large number of hash functions working together in an AND-OR structure can possess this property [52]. One starts out with a basic hash function that places nearby samples in the same bucket with a high probability, but that, at the same time, places any two far-away samples in

the same bucket with NOT a sufficiently low probability. Subsequently, one can require that for any two given samples to be considered similar they must be in the same bucket for a set of different hash functions, these multiple hash functions being random variations of the same basic hash function with respect to at least one of its parameters. (We'll use r to denote the number of hash functions in such a set.) This is referred to as enforcing an 'AND' operation over r hash functions to significantly decrease the probability of two far-apart samples being considered similar. Since the 'AND' operation can also somewhat reduce the probability of nearby samples as being considered similar, we take an 'OR' b sets of r hash functions to restore or further enhance that probability. Choosing r and b in order to achieve desired values for p_1 and p_2 becomes a design issue for any implementation of LSH. Fig 5.3 plots the AND-OR construction curve for some values of r and b.

Hyperplane LSH [53], a commonly used implementation of LSH for similarity measure shown in Eq. 5.1, consists of using randomly oriented hyperplanes for hashing. A hyperplane gives us a two-bucket hash table: When a numerical data sample is projected on a hyperplane perpendicular to a hyperplane passing through the origin, the projection is either in the positive half-space corresponding to that hyperplane or the negative half space. See Fig. 5.4 for an illustration. By constructing b sets of such randomly placed hyperplanes, with r hyperplanes in each set, we can achieve the desired discriminations between nearby and far-apart data samples. In Appendix C, we present a simulated study on the performance of Hyperplane LSH.

Applying LSH on the background color-histogram vectors, each patch p is associated with a set of patches that are directly within the angular similarity threshold of p on the basis of just the background color-histogram similarity. For each patch p, the set of all similar patches thus discovered constitutes p's similarity neighborhood.

Subsequently, patches and their neighborhoods are converted into a similarity graph by testing within each neighborhood for patches having similar foreground spectral signatures with respect to the associate patch. This process is illustrated in Fig. 5.5. The output of this exercise is represented by a similarity graph in which a

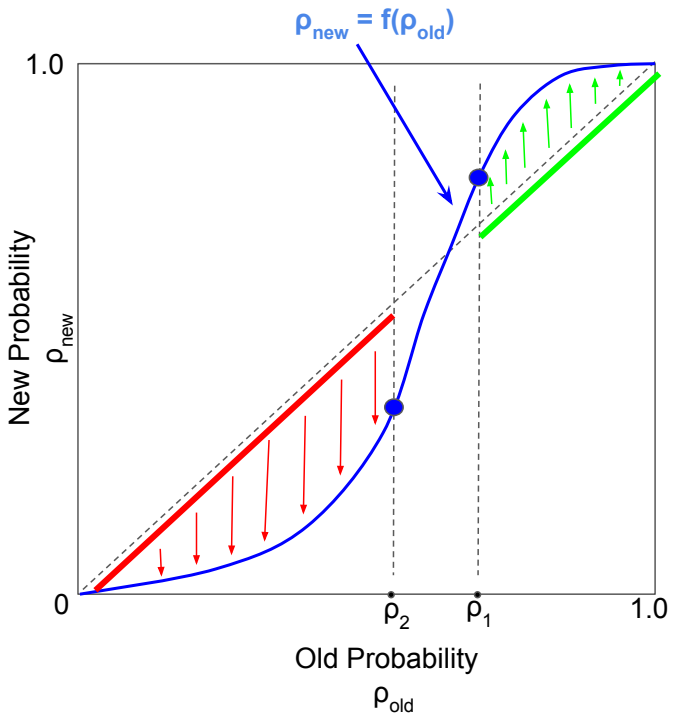


Fig. 5.2. The idea behind AND-OR construction is to change the probability of bucket collision. We want the probability of collision for "nearby" samples to go up above p_1 while the probability of collision for "far apart" samples to go down below p_2. If we cascade many such constructions in series, then, we can achieve very high p_1 and very low p_2 at the cost of more computation.

pair of two vertices, with each vertex corresponding to an image patch, share an edge if they are similar both with respect to the background and the foreground contexts. Although the worst-case time complexity of this algorithm is given by $O(|V|^2)$, where V is the number of patches, the worst case happens only when the entire similarity graph is a clique (for which the number of edges is quadratic on the number of vertices; that is $|E| = O(|V|^2)$).

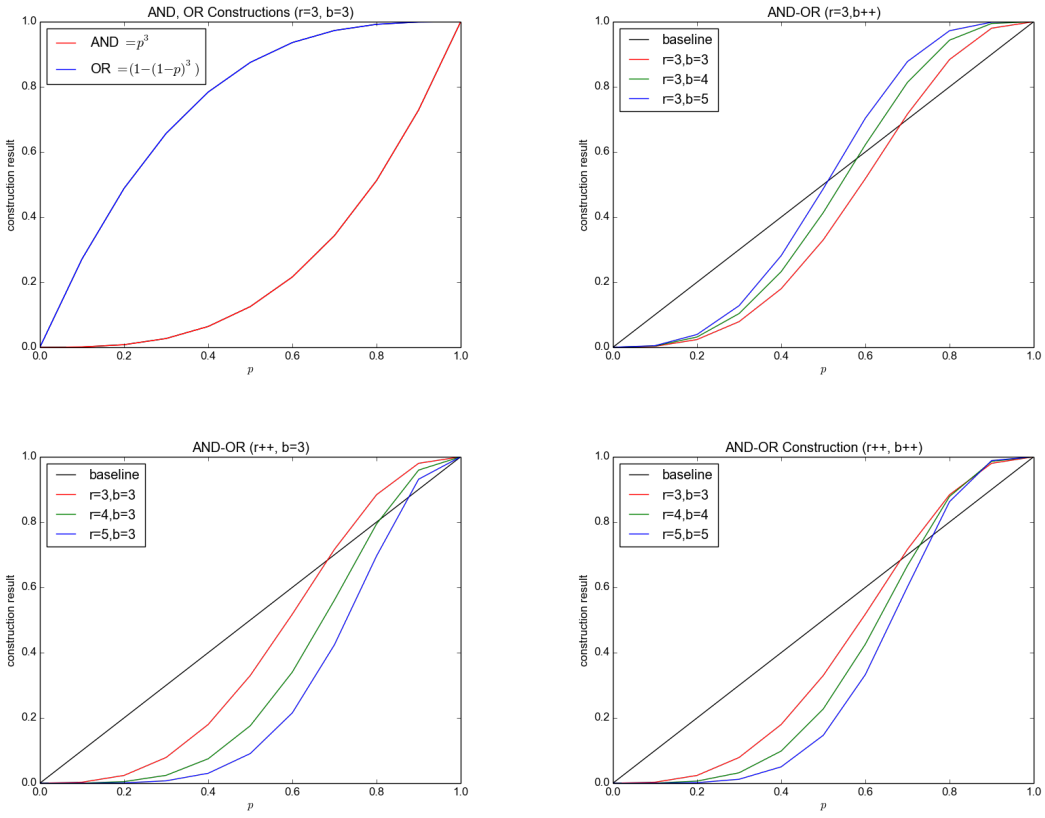


Fig. 5.3. Single-stage AND-OR construction $= 1-(1-p^r)^b$. Combinations of r and b values gives different effects and shifts the fixed point along the diagonal line.

5.2.6 Population Compression

We represent a pixel along with the image patch that provides its surrounding context by a vertex in the similarity graph mentioned previously. When a particular vertex is selected for inclusion in the concise dataset, all other vertices that are similar to it are subsequently marked as redundant in the similarity neighborhood of the selected vertex. The task of data reduction is then to find a minimal set of vertices that maximally cover the overall redundant set of vertices. This optimization problem

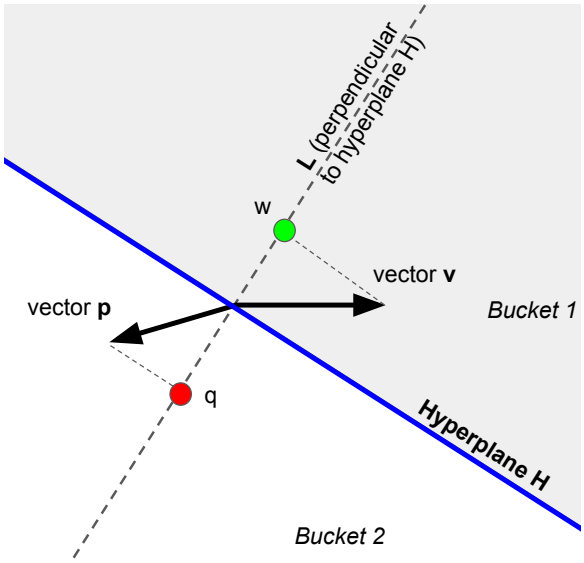


Fig. 5.4. An illustration of Hyperplane LSH. Hyperplane H partitions the space into two buckets. Points q and w are the projections of vectors $\vec{p}$ and $\vec{q}$ onto the perpendicular hyperplane L. In this example, w is in bucket 1 and q is in bucket 2.

can be formulated as the "Set Cover problem"[4], which is a well-known NP-Complete problem [54]. A known approximated solution using greedy algorithm, as described in the subsection that follows, can produce a solution that is guaranteed to be within $O(\log|V|)$ factor of the optimal solution where V is the set of vertices in the graph. In terms of algorithmic complexity, the greedy algorithm runs in $O(|E|)$ time, where E is the set of edges in the graph.

5.2.7 Creating an Initial Concise-set Representation of the Population

In the previous section, we mentioned using a greedy algorithm to find an approximated solution to the Set Cover problem. We now describe the algorithm in detail and show how it returns an concise-set representation of the target population. As

[4]The Set Cover problem is closely related to the Dominating Set problem in graph theory.

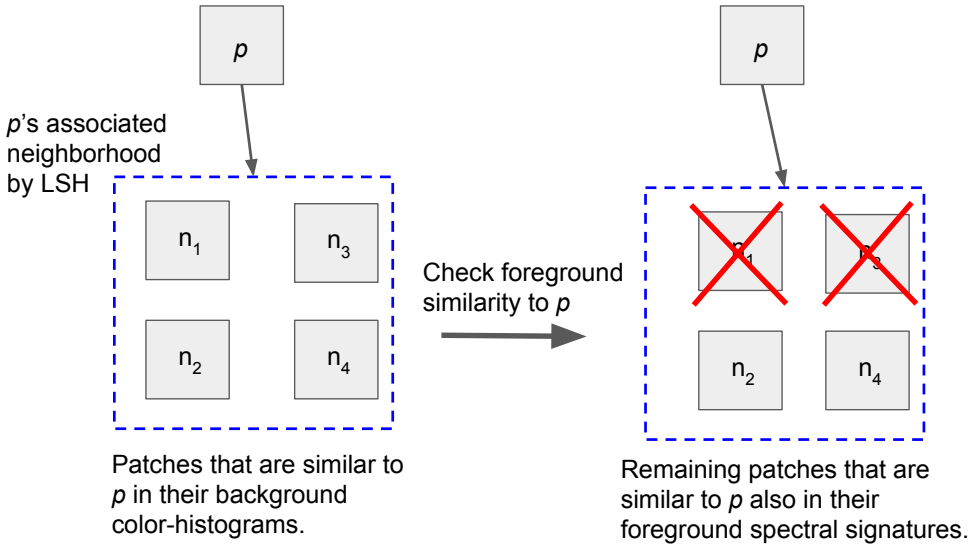


Fig. 5.5. In this example, a patch p is associated with a neighborhood consisting of 4 patches, n_1 to n_4, that are similar in their background color-histograms to p. After checking their foreground spectral signatures with p, only n_2 and n_4 remain in the neighborhood. This process is done for $p \in \{\text{all patches}\}$ and the overall result is a similarity graph in which two vertices (patches) share an edge if they are similar to each other in both foreground and background contexts.

we will see in Section 5.4, this representation can be refined subsequently to yield a final representation.

Algorithm 1 Create an Initial Concise-set Representation

Input: U = Set of all items. (e.g., image patches)

Output: A concise-set representation of U

1: $S \leftarrow \{\{\text{LSH-GetItemsSimilarTo}(e)\}, \forall e \in U\}$
   // S = Set of candidate similar-item sets (clusters).
2: $R \leftarrow [\,]$ // Array of cluster representatives.
3: $W \leftarrow [\,]$ // Array of redundancy weights.
4: $C \leftarrow [\,]$ // Array of similar-item sets (clusters).
5: **while** SomeItemsNotCovered(U, C) == True **do**
6: $(c^*, r^*) \leftarrow \text{GetMaxCluster}(S)$ // Max cluster, c^*, and its repre-
7: $S \leftarrow \text{RemoveAndUpdateClusters}(S, c^*)$ sentative, r^*.
8: $R \leftarrow \text{Append } (R, r^*)$
9: $C \leftarrow \text{Append } (C, c^*)$
10: $W \leftarrow \text{Append } (W, |c^*|)$
11: **end while**
12: **return** (R, C, W)

Algorithm 1 takes as input a set of indices representing image patches IDs in the population. The algorithm outputs a triplet, (R, C, W), as the concise-set representation of the population, where:

- R is the *concise dataset* that is an array of cluster representatives, with one representative for each cluster of vertices in the approximated similarity graph.
- C is an array of clusters. That is, for each i, $C[i]$ is a cluster represented by a set of vertices. Each vertex in $C[i]$ corresponds to an image patch that is similar to the cluster representative, $R[i]$.
- W is an array of cluster sizes. That is, $W[i]$ is the size of cluster $C[i]$ for which $R[i]$ is the corresponding cluster representative.

5.2.8 Annotating the Initial Concise Dataset

After we have created a concise-set representation of the image patch population, we proceed to annotate the center pixels (the foreground pixels) of the image patches retained in the concise dataset. That is, a human annotator looks at image patches associated in the R array, as returned by Algorithm 1, and assign ground-truth labels to their center pixels. The human annotator does not assign labels to the surrounding pixels in the image patch. In keeping with our earlier discussion, an image patch is modeled as containing contextual pixels (i.e., the background) surrounding the center pixel (i.e., the foreground) for which we want the human annotator to supply a class label.

5.2.9 On Extending the Concise-Set Representative Label to the Other Members of the Same Set

Obviously, the most straightforward way to extend the human-supplied annotation label for a cluster representative in the R array is to simply assign the same label to all the other members in same cluster.

However, it is possible to conceive of alternatives to the obvious mentioned above that have ramifications regarding the size of the overall representation created for a large dataset involving hundreds of satellite images. One could, for example, argue that since — *seemingly* — all the other members in a cluster are redundant vis-a-vis the cluster representative for constructing or evaluating a classifier, why not just retain *only* the cluster representatives and discard the rest of the data. The problem with that logic is that such a data reduction could significantly impact the class probabilities associated with different land types in a geographic regions and, consequently, result in erroneous classification performance results (regardless of the choice of the classifier).

To get around this difficulty, and, at the same time, to benefit from the compression made possible by the R array, we could associate the size of each cluster with

each cluster representative in R. This is indeed one of the options made available by our concise-set framework when we generate the final representation for the satellite data. We refer to this as the "The Weighted Representative Method (WRM)" for creating the final representation.

When not using the weighted representative method, the system simply extends the human-supplied annotations for each cluster representative in R to the rest of the rest of the cluster members. In order to make a distinction with WRM, we refer to this method as "The Whole Cluster Method (WCM)".

Fig. 5.6 illustrates an example of estimating the confusion matrix using the "weighted-representative" method, and Fig. 5.7 illustrates an example of estimating the confusion matrix using the "whole-cluster" method. As the reader would expect, the "weighted-representative" method is simple and fast, but it tends to either overestimate or underestimate the classifier's true performance. On the other hand, the whole-cluster method produces a better performance estimate, although at the cost of doing more work. Note that the annotation effort is the same for both methods.

5.2.10 A Quality Coefficient for Choosing the Best Value for the LSH Similarity Threshold

It should be obvious to the reader that the validity of the confusion matrices as produced by the two methods presented in the previous subsection depends significantly on the similarity distance threshold used in the LSH algorithm. A similarity distance threshold that is too large would degrade the quality of the concise dataset with regard to the following two considerations: (1) We will have increased tendency of the data samples from disparate classes to populate the same clusters; and (2) The dataset may end up with fewer land-type classes than there actually are in the satellite data.

At the same time, a similarity distance threshold that is too small would generate too many small clusters, which would increase the human burden associated with

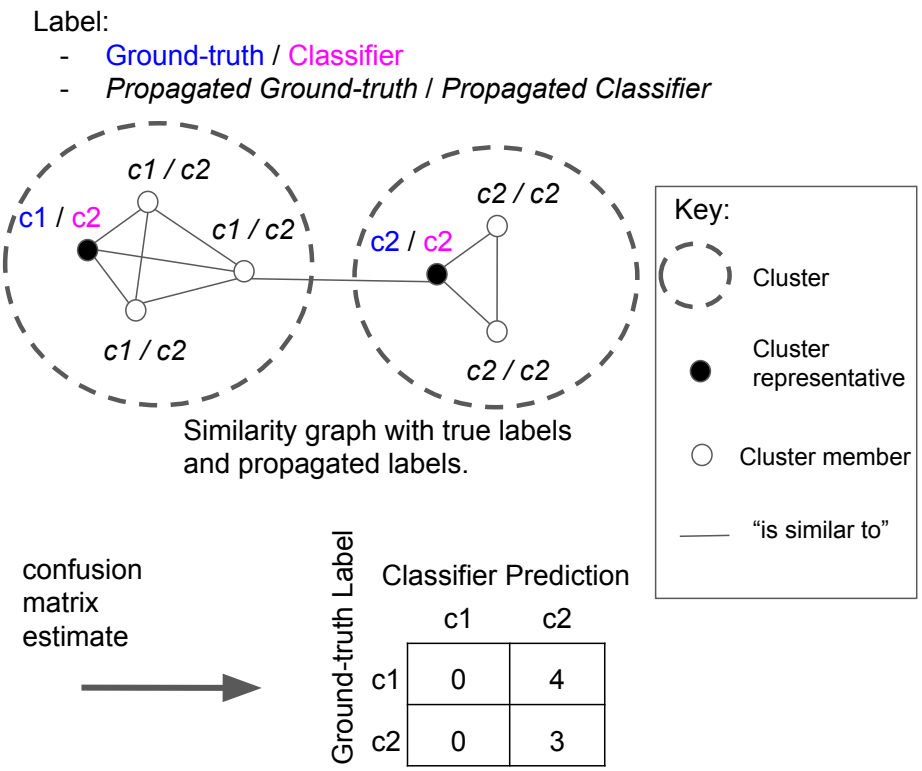


Fig. 5.6. An example of the "weighted-representative" method: The similarity graph shown here has two clusters depicted by the dashed circles. Each cluster has a representative shown as a black dot. The classifier is applied to only the cluster representatives and the classifier generated labels for the representatives propagated to the rest of the cluster. Each vertex is shown with two labels, one for the ground-truth and the other for classifier-generated, and, in each case, they are both propagated from the cluster representative. In this example, there are four vertices labeled "c1/c2" and therefore the corresponding "c1/c2" entry in the estimated confusion matrix is 4. Similarly for the "c2/c2" entry.

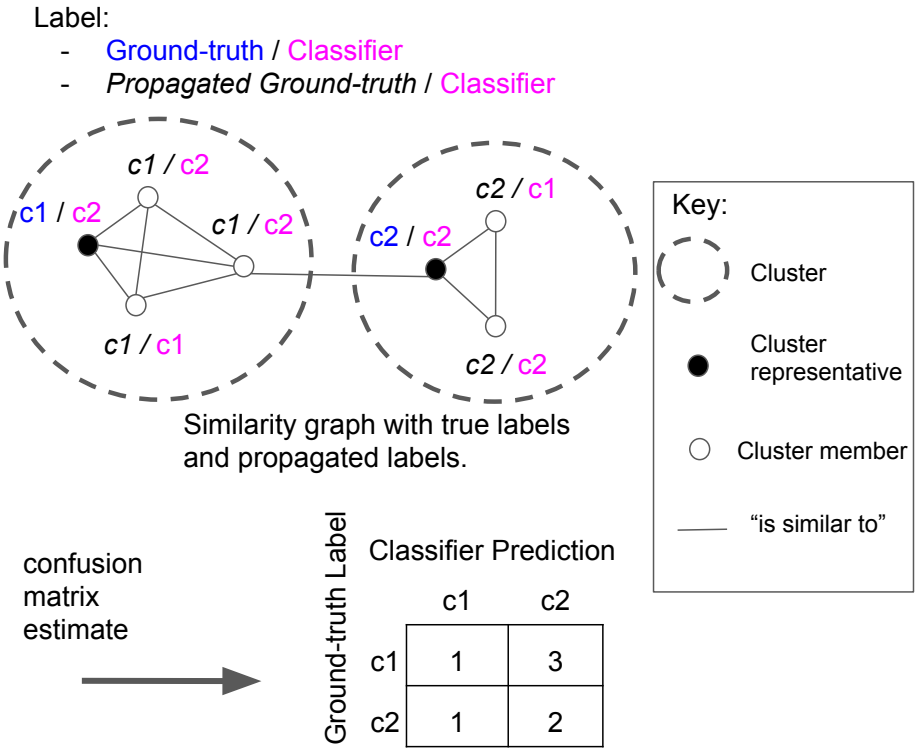


Fig. 5.7. An example of the "whole-cluster method": The depiction here parallels the one shown in Fig. 5.6 except for the fact that the classifier is applied to every member of each cluster. For each vertex, the first label is the ground-truth label as propagated from the cluster representative and the second label is as produced by the classifier. In the example shown, there are three vertices labeled "c1/c2" and therefore the corresponding "c1/c2" entry in the estimated confusion matrix is 3. Similarly for the other entries in the confusion matrix.

supplying the ground-truth label for the representative of each cluster. That leads to the question of whether there is any automatic way to determine a good value to use for the similarity distance threshold. As we discuss below, the answer to the above question is yes.

Our answer presented in this section is based on the following observation: Since the similarity neighborhoods returned by the LSH algorithm consist of the vertices that are hashed into the same bucket, it is possible for a similarity neighborhood to intersect multiple clusters as returned by LSH (see Fig. 5.8). When a vertex lies simultaneously in multiple similarity neighborhoods, it *may* acquire a set of different propagated *class* labels.[5] Such a vertex contributes to inconsistencies in the labeling of the data. We claim that when all the propagated class labels are consistent, we have chosen a good value for the similarity distance threshold. So, if we can find a way to estimate the number of the vertices with inconsistently propagated class labels, we can assess the appropriateness of the value chosen for the similarity distance threshold.

Let T be the total number of vertices and let I be the number of vertices with inconsistent class labels in different clusters, we define *ground-truth consistency* of the concise-set representation as:

$$\text{ground-truth consistency} = 1 - \frac{I}{T} \tag{5.2}$$

Obviously, ground-truth consistency is only meaningful when there are overlapping clusters.

[5]As to the reason for *may*, first note that LSH will form multiple distinct clusters for the same ground-truth class label. LSH forms a cluster on the basis of the approximate similarity of vertices. Subsequently, the human annotator labels one cluster representative and then that label is propagated to all the other members. For an example of there being multiple clusters for the same ground-truth class label, think of the pixels corresponding to the label "road". Since roads, in general, are made from different materials — concrete, asphalt, gravel, or just plain dirt — any automatic clusterer is likely to place the road image pixels in different clusters that may or may not be overlapping. Such different clusters for the same class label are NOT the source of inconsistency we are talking about. For the sort of inconsistencies we are talking about, consider the image pixels that, through propagation from the cluster representatives, simultaneously acquire two different labels such as "roof" and "road". This can easily happen since in many parts of the world we have roofs, especially flat roofs, that are made from the same materials that go into road construction. So human-annotated "road" pixels may get hashed into an LSH bucket that also contains "roof" pixels and vice versa.

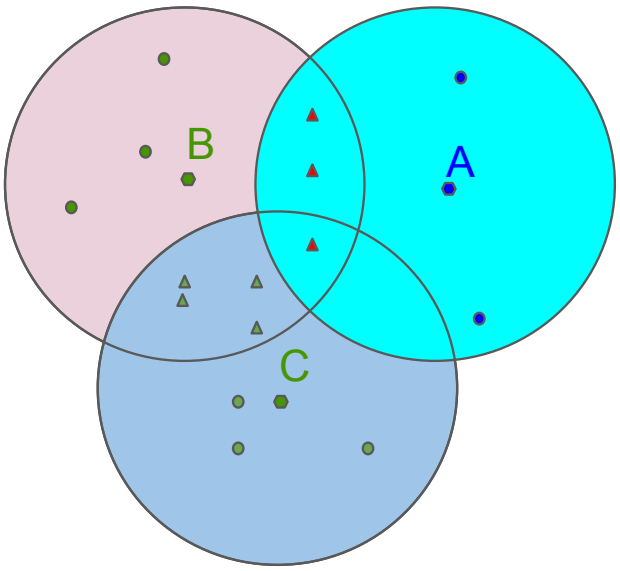


Fig. 5.8. In this example for illustrating the notion of consistency, while we have three overlapping clusters in some feature space, two of the clusters, represented by the cluster representatives B and C, carry the same propagated ground-truth label. On the other hand, the cluster represented by A carries a different propagated label. Note that true ground-truth labels are provided only for the cluster representatives. We have a total of 18 vertices in the three clusters. In the figure, small circular dots represent vertices that belong to only one cluster while small triangles are vertices that simultaneously belong to two or more clusters. We see that 7 of the 18 vertices have two or more cluster memberships. However, on account of the equivalency of the class labels for B and C, only three vertices have different class labels. Therefore, the ground-truth consistency (See Eq. 5.2) is $1 - \frac{3}{18} = 0.833$.

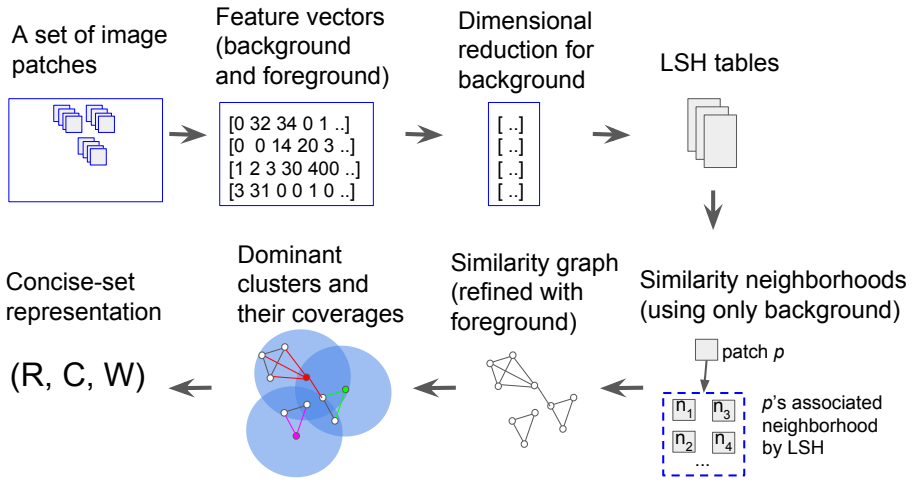


Fig. 5.9. The processing pipeline for creating a concise-set representation.

5.2.11 The Complete Processing Pipeline For Generating a Concise-set Representation

Having described in the preceding subsections the different aspects of our proposed approach, we now summarize the overall framework for creating a concise-set representation for satellite data. Fig. 5.9 give an overview picture of the pipeline.

1. Collect all units in the target population, which in our case would be image pixels along with the associated image patches. (See Section 5.2.1)

2. Optionally, partition the population using a vertex attribute, to be discussed in Section 5.3.

3. Extract both background and foreground features for all image patches. (See Section 5.2.2)

4. Using FastMap, apply dimensional reduction to the background histograms. (See Section 5.2.4)

5. Using LSH, create background similarity neighborhoods for the entire population. (See Section 5.2.5)

6. Create the similarity graph by refining the background similarity neighborhoods with the foreground similarity constraint. (See Fig. 5.5)

7. Find dominant clusters within the refined similarity graph and create the concise-set representation using Algorithm 1.

8. Annotate the cluster representatives with the ground-truth labels (See Section 5.2.8).

9. Calculate the ground-truth consistency using Eq. 5.2.

5.3 Population Partition — Creating Multiple Evaluation Datasets Simultaneously

We now return to the optional Block A of Fig. 5.1 ("Partition image patches into groups"). As is done with stratified sampling, we first partition the main dataset using a criterion that depends on the reason for why you would want to create partitioned evaluation datasets. Subsequently, a concise-set representation is created for each partition.

In the context of satellite imagery that covers large geographic areas, one of the main reasons for creating partitioned datasets is that one may want to investigate the performance of a classifier for subsets of the overall data that possess certain attributes. Another equally valid reason partitioning a satellite image dataset is to distribute the work among a cluster of computers.

In general, depending on how the overall dataset is partitioned, we have two strategies for combining the concise-set representations: *union* and *hierarchical*. The union strategy simply takes the union of the representations. That is, suppose we have G partitions, then:

$$(R_f, C_f, W_f) = (\cup_{i=1}^{G} R_i, \cup_{i=1}^{G} C_i, \cup_{i=1}^{G} W_i). \tag{5.3}$$

The union combining strategy is simple and suitable for the cases when the concise-set representations have little overlap among them in the feature space. On the other

hand, the hierarchical strategy combines only the concise datasets (i.e., the cluster representatives) and creates a completely new concise-set representation from them using Algorithm 1. That is,

$$(R_f, C_f, W_f) = \text{CreateConciseSetRep}(\cup_{i=1}^{G} R_i). \tag{5.4}$$

This hierarchical combining strategy can produce a much smaller concise dataset than the union combining strategy and is particularly suitable for partitions that are somewhat arbitrary and large. A good use case for the hierarchical combining strategy is when dealing with many satellite images covering a wide-area region. In this case, a natural partitioning strategy is to treat each satellite image as a partition. However, as we will see in Chapter 7, there is another more practical partitioning strategy for working with multiple satellite images.

When creating a new concise-set representation using the hierarchical combining strategy, care must be taken to prevent some merged member becoming dissimilar to their new cluster representative. This situation happens because similarity constraints for clustering data are generally not transitive. To get around this issue, we merge two clusters only when more than 95% of the merged members are still similar to their new cluster representative.

5.4 Refining the Concise Dataset

We mentioned earlier in Section 5.2.10, if the similarity threshold is too large, one can run into the following two problems: (1) Each cluster produced by LSH would be impure in the sense of containing samples from disparate land-type classes; and (2) The concise dataset may not represent all of the land-type classes. The ground-truth consistency measure presented earlier takes care of the first problem. This section focuses on how to fine-tune the concise dataset so that all of the land-type classes are represented in it. Note that fine-tuning the concise dataset will also lead to an improved ground-truth consistency.

An important consideration in refining the concise dataset is that all of the previously supplied annotations by the human must not be discarded and that the human should be asked for any new annotations only when absolutely necessary. And, as it turns out, the only condition under which the human would need to be consulted again is when the total number of land-cover classes is less than what is believed to be in the geographical region of interest. In what follows, we present a concise-set refinement procedure with this property.

The main idea in this refinement is to repeatedly shrink the clusters and, at the same time, introduce more cluster representatives into the dataset. This iterative process repeats until the human is satisfied with the size, diversity, and representativeness in the concise dataset.

Since there may be many clusters in the concise-set representation, reducing all clusters simultaneously will create too many uncovered vertices at once and thus overburdening the human. Instead, we pick one cluster at a time. As to which cluster to pick, we define "the most-impure" cluster as the cluster that has the largest distance variance inside the cluster (See Fig. 5.10).

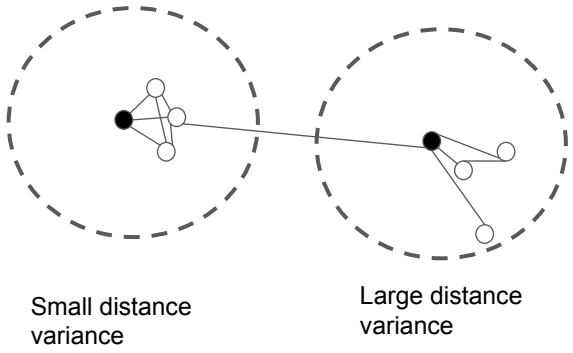


Fig. 5.10. A similarity graph consisting of two clusters. The cluster on the right has a larger distance variance inside the cluster. All distances are calculated relative to the cluster representative. Thus, the right cluster is "more impure" than the left cluster.

Once the most-impure cluster has been found, we proceed to determine a new radius for the cluster. While there are many ways to come up with a new radius, our heuristic is to model the distances as having a bimodal distribution. That is, we assume the distances fall into either one of the two categories: *near* or *far*. We can then calculate the optimal radius by using Otsu's algorithm [55] modified for handling continuous values. Fig. 5.11 shows an example of shrinking a cluster. For detail on our modified Otsu's algorithm, see Appendix D.

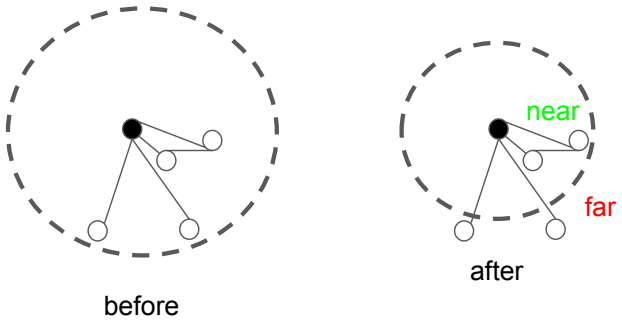


Fig. 5.11. A heuristic for shrinking a cluster. We assume cluster member distances have a bimodal distribution; i.e., member are either "near" or "far" from the cluster representative. The optimal threshold can be calculated using Otsu's Algorithm modified for continuous values. Refer to Algorithm 2 in Appendix D for detail.

Because uncovered vertices do not belong to any cluster, they are not represented in the concise-set representation. Therefor, we run the greedy set-cover algorithm again but only on the uncovered vertices. This process creates additional clusters. Fig. 5.12 shows an example.

In practice, a similarity measure between two items may not be single numeric value, but rather a logical condition on multiple numeric values. In our case, we define two satellite image patches to be similar when 1.) their background color histograms are similar, *and* 2.) their foreground spectral signatures are also similar. To get around this issue, we first get two most-impure clusters, one based on the background color histograms and the other based on the foreground spectral signatures. We then

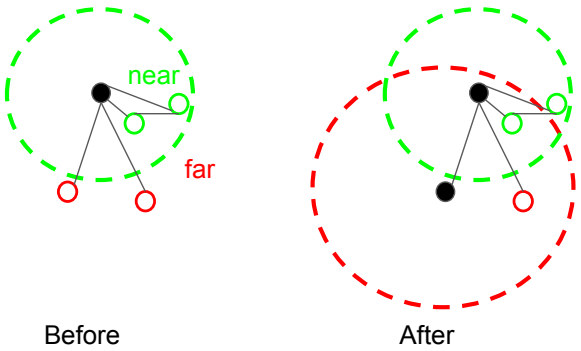


Fig. 5.12. Creating additional clusters from uncovered vertices: In this example, a new cluster with the default radius is created to represent the uncovered vertices. The black dots indicate the cluster representatives and their ground-truth labels are provided by human.

pick the most-impure cluster that has a smaller number of *far* vertices. For example, suppose cluster A and cluster B are the most impure clusters in their corresponding feature spaces but cluster B has fewer *far* vertices than cluster A. In this case, our heuristic will select cluster B for radius reduction. See Fig. 5.13 for an illustration. The rational behind this heuristic is to minimize human labor. A larger number of *far* vertices could introduce more clusters and thus requiring more human annotation since every cluster needs to have its cluster representative annotated.

When a new clusters is created, it can be annotated immediately by the human or saved into a queue for later annotation in batch.

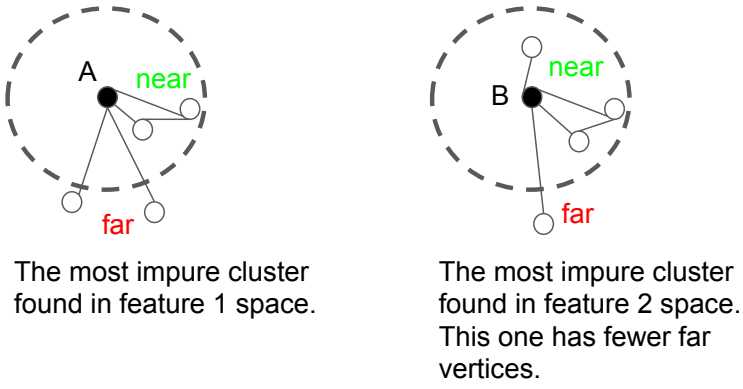


Fig. 5.13. Heuristic for selecting an impure cluster among different feature spaces: Cluster A and B are the most impure clusters in their corresponding feature spaces. Cluster B is chosen over cluster A for radius reduction because it has a smaller number of *far* vertices.

6. CONCISE-SET REPRESENTATION RESULT

Recall that at the heart of our approach for constructing a concise-set representation of the data is the construction of an approximate similarity graph with the help of LSH. As mentioned in Section 5.2.6, each vertex of this similarity graph represents a pixel along with the image patch that provide geographical context for the pixel. When the vertices are tested for similarity, it is done on the basis of both the spectral signatures at the pixels themselves and the attributes of the associated image patches. As mentioned earlier, each image patch is represented by a histogram of the pixel "colors" in the patch.

Therefore, as the reader can imagine, how many bins to use for the patch histograms and what similarity thresholds to use when comparing the histogram-based feature vectors, etc., are some of the critical experimental parameters in our approach. The goal of this section is to discuss how we choose values for these parameters. Our parameter selection strategy is conservative in the sense that we aim to minimize the computation overhead while keeping the final concise dataset small. Using these parameters, we will present our experiments and the results in Section 6.2.

6.1 Setting the Experimental Parameters for WorldView2 Imagery

In this section, we will describe how we set the parameters mentioned above for WorldView2 satellite imagery. This dataset consists of Very High Resolution (VHR) satellite images with ground resolution at 0.5m per pixel.[1] At this level of resolution, it is possible for a human annotator to identify typical narrow one-lane roads and alleys. In order to apply our framework to this data, we must first set the size for

[1] We use 4-band WorldView2 data that is ToA corrected. ToA stands for Top-of-Atmosphere. This correction is a standard procedure for normalizing satellite data taken at different angles and distances relative to the earth's surface being imaged.

Fig. 6.1. A typical region in the Chile ROI.

the image patches. We have experimentally determined that patches of size 101 × 101 pixels give is a good balance between competing requirements. This size is large enough to provide a reliable context for the labeling of the center pixel and small enough that it does not include too much diversity within a single patch.

For the purpose of exploring the best choices to make for the parameters, we manually annotated a dataset from the Chile ROI (Region of Interest) [1]. Fig. 6.1 shows a typical area in this ROI. The visually recognizable different regions in this area that are homogeneous in terms of the class labels are demarcated with a graphical tool. Five major land-types are used in this exercise: building, road, tree, water, and soil. The output of this exercise is a set of pixels along with their image patches, with each pixel along with its associated image patch serving as a vertex in the similarity graph. The important thing to remember is that every vertex created in this manner has a

human-supplied class label. Subsequently, we use this data to investigate the power of our proposed formalism for creating concise-set representations for the different choices for the parameters of the framework.

We will show in Section 6.2 that the parameters set in this manner from the data collected over Chile work effectively in another part of the world that is significantly different — Australia. That fact provided further validation for our framework.

6.1.1 Color Spaces and Histogram Quantization

As mentioned earlier, we refer to the center pixel in an image patch as the foreground and the rest of the pixels in the patch as the background. And, as mentioned in Section 5.2.2, we represent the background of an image patch by its color histogram. In our experiments, we investigated both the RGB and CIELAB color spaces for the histogram. The RGB space results in b^3 bins. Subsequently, the choice of b controls the size of the generated histogram (i.e., $b = 4$ results in $4^3 = 64$ bins and, therefore, a 64-dimensional feature vector). To reduce the number of bins in the histogram, we examined the CIELAB color space and remove the invalid bins – those bins that are not occupied by any color in the RGB space. We showed the obtained histogram sizes for both RGB and CIELAB in Table 4.1. There is also another reason for using CIELAB color space over other color spaces that are not perceptually uniform. In our case, we want the image patches to be grouped together according to human perception of color similarity.

To determine the best value to use for b, we examined a wide range of values between $b = 4$ to $b = 64$. Note that there is no need to examine $b \geq 66$ as it has been shown to be the sufficient for CIELAB color quantization [31]. We refer the readers to section 4.1.2 for the discussion on CIELAB color space.

For each value of b, we constructed a similarity graph as described in Section 5.2.5 using the Chile dataset. Then, we calculated the cardinality of each cluster and found average cardinality over all the clusters.

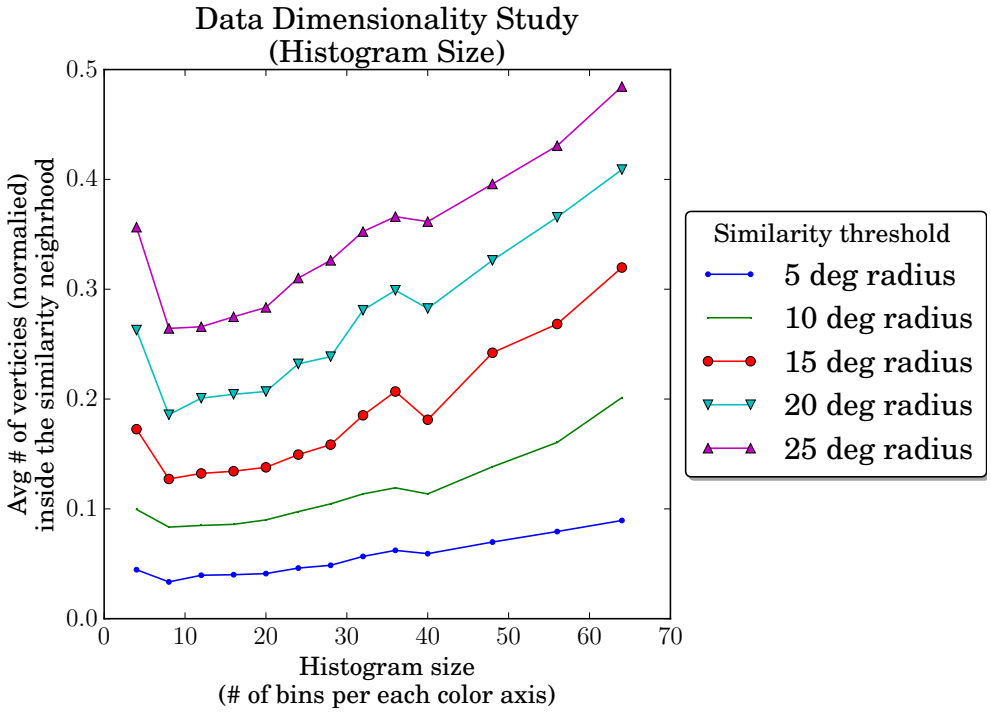


Fig. 6.2. Proportion of the data found inside the similarity neighborhoods (clusters).

Fig. 6.2 illustrates the effect of histogram size on the average cardinality of the clusters for different similarity thresholds. We observe a decrease in the average values between $b = 4$ and $b = 8$ followed by a gradual increase after $b = 8$. It seems as the histogram size increases, more and more data become similar to one another. We note that this phenomenon does not always happen in general and can be attributed to data distribution and quantization effect.

6.1.2 Calculating the Best Value to Use for the Similarity Threshold

In this section, we investigate the best choice to make for the similarity threshold that is needed by the LSH algorithm. Recall that we are talking about similarity between a pair of vertices in the similarity graph, with vertex standing for a pixel along with the associated image patch. In the previous section, we concluded that

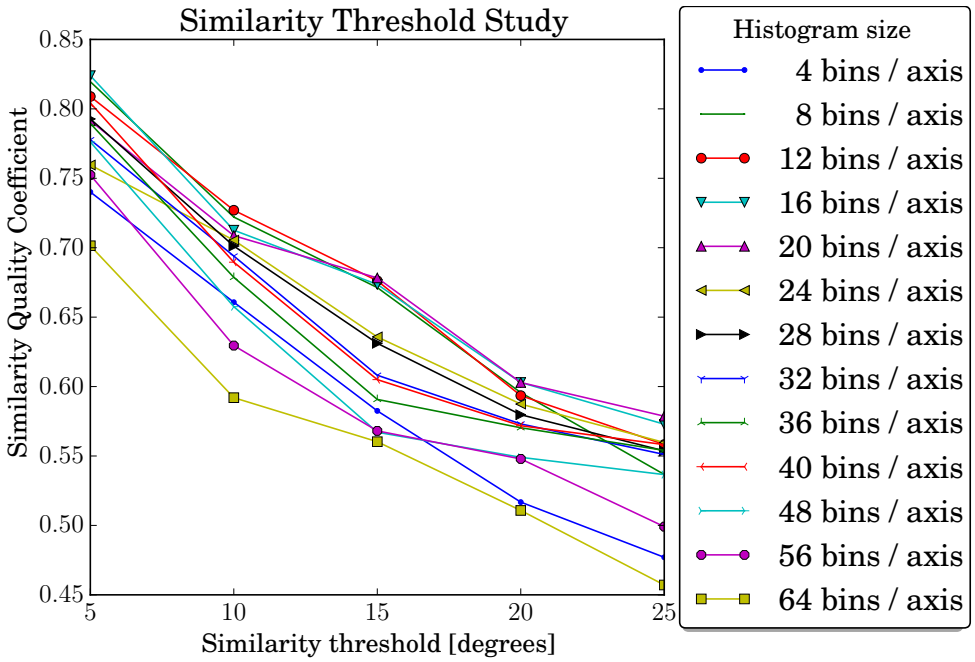


Fig. 6.3. Plots of background Similarity Quality Coefficient (SQC) as a function of similarity threshold.

even with the same similarity threshold, how the data gets clustered changes when the data dimensionality changes.

We use the following logic to evaluate the quality of a similarity threshold: We examine each cluster and count the number of vertices in the cluster whose human-supplied class labels are the same as the human-supplied label for the cluster representative. This count is normalized by the cardinality of the cluster. The average of this ratio over all the clusters is a quality coefficient for a given similarity threshold. We refer to this coefficient as the "Similarity Quality Coefficient (SQC)". In Fig. 6.3, we show the dependence of SQC on different values for the similarity threshold. Note, each plot corresponds to a different histogram size.

We see in Fig. 6.3 that SQC values are greater than 50% for similarity thresholds less than 20 degrees, regardless of the histogram size. Therefore, in our experiment, we conservatively set the similarity threshold at 10 degrees so that at least 60% of the

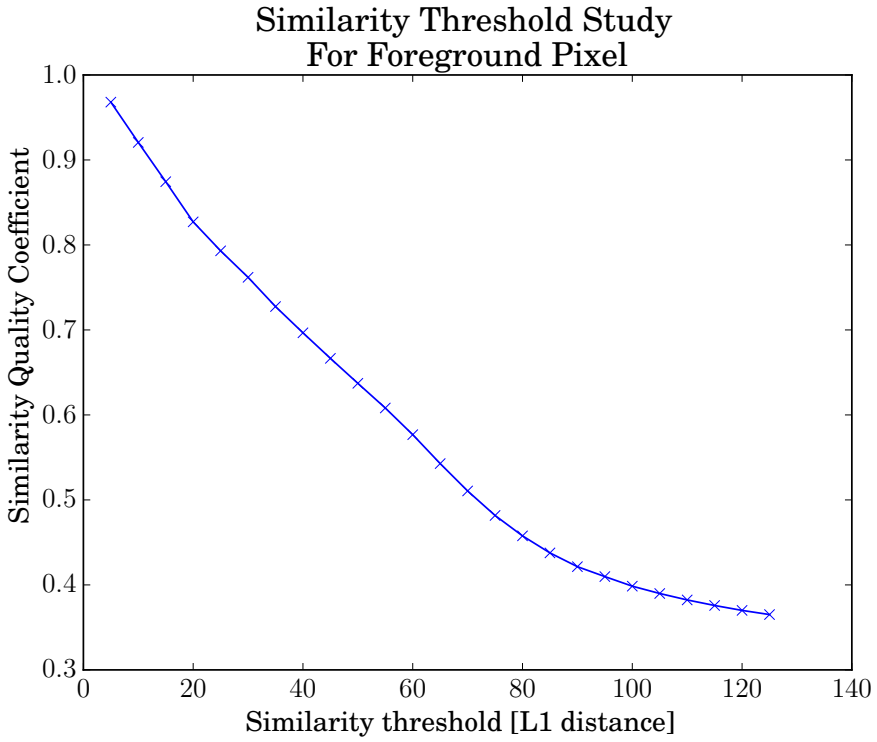


Fig. 6.4. Plot of foreground Similarity Quality Coefficient (SQC) as a function of similarity threshold.

image patches found within a cluster can be expected to have the same ground-truth label as the cluster representative. We could have picked any values smaller than 20, but a tighter similarity threshold increases the hashing time in LSH. We pick 10 over 15 because we would like to have higher SQC values while still keeping the hashing computation low. As for the histogram size, we picked $b = 32$ (9024 dimensions) for our experiments (See Table 4.1). We noted from the study in [31] that b does not need to be more than 66 and thus $b = 32$ seemed like a good trade-off between color fidelity and computational efficiency. We obtained the same result when we repeated the similarity threshold study on the lower-dimensional data described in Section 5.2.4

Fig. 6.4 shows the SQC plot for the image foreground (i.e., the spectral signature of the center pixel). Using this plot, we set the similarity threshold at 50 so that about

60% (i.e., slightly over a majority) of the image patches found within this threshold can be expected to have the same ground-truth label as the cluster representative.

6.2 Validation

A most important aspect of the comparative results we report in this section is that the datasets we used for these results are drawn from a part of the world that is different from what was used in Section 6.1 for parameter estimation. We refer to these datasets as our "validation datasets". *We however use the same parameters that we estimated in Section 6.1 to create a concise-set representation of the validation datasets.*[2] Whereas the dataset used in Section 6.1 came from Chile, the validation datasets are from Australia. Fig. 6.5 shows an example of the area from which the validation datasets were drawn.

6.2.1 Classifier Evaluation Experiments

As we mention later here, the true human-supplied class identity is known for every element in the validation datasets. This allows us to calculate the "true" confusion matrix on a validation dataset for any given classifier. We have chosen a Support Vector Machine (SVM) classifier created using the approach in [1] for this validation study since SVM-based classifiers are arguably the most commonly used classifiers used today for land-cover classification with satellite images.

In this section, we will show that the confusion matrix calculated from the concise-set representation of the validation dataset is significantly closer to the true confusion matrix as compared to the confusion matrix calculated by the traditional random sampling method. In order to convince the reader that the above stated result is not a chance result — that is, a result that is specific to one particular random sampling of the data — we repeat the traditional random sampling method multiple times.

[2]What that implies is that the parameters estimated in Section 6.1 possess some measure of generality. The extent of this generality is yet to be investigated.

Fig. 6.5. A 1km by 1km region in the Australia ROI.

As we did for the parameter estimation dataset in Section 6.1, for the validation datasets used in this section, we manually supply the class label for $1,240,590$ pixels within a 1 km $\times$1 km area in the Australia ROI.[3] Hence, the largest validation dataset we can draw has $1,240,590$ units. Note that each unit in the dataset is represented by a 101×101 image patch centered at the annotated pixel.

The following considerations go into calculating and comparing the confusion matrices produced by the concise-set representation method and the traditional random sampling method:

[3]The manual annotations were supplied with the graphical tools that we mentioned in Section 6.1. As mentioned there, these tools allow us to quickly demarcate large sections of a satellite image that are homogeneous with respect to a class label. This allows the system to quickly assign a class label to large set of pixels, all at the same time.

- To estimate the true confusion matrix from the concise-set representation, we use the "whole-cluster" method described in Section 5.2.9.

- To directly compare a pair of confusion matrices, we first normalize the confusion matrices and then compute the sum-of-squared-difference (SSD) between the two confusion matrices being compared. A small SSD value indicates a close match between the two confusion matrices.

- To normalize a confusion matrix, we divide each matrix entry by the sum of all entries. For example, let $\hat{\text{CM}}$ be the normalized version of the confusion matrix CM, then:

$$\hat{\text{CM}} = \frac{\text{CM}}{\text{sum}(\text{CM})}. \tag{6.1}$$

- To compute the SSD value between two normalize confusion matrices, we take the Frobenius norm of the matrix difference. That is, $\text{SSD} = ||\hat{\text{CM}}_1 - \hat{\text{CM}}_2||_F$ where $||A||_F$ denotes the Frobenius norm of matrix A. For example, let $\mathbf{T}$ and $\mathbf{E}$ be two matrices of the same size, then:

$$\text{SSD}(\mathbf{T}, \mathbf{E}) = \sum_i \sum_j (T_{i,j} - E_{i,j})^2. \tag{6.2}$$

Experiment 1: Concise-set Representation versus Random Sampling

We repeat SVM based classification using the traditional random sampling approach 100 times and, for each trial, compare the classifier performance individually with what is obtained through the concise-set representation approach. The plot in Fig. 6.6 shows the SSD values for the 100 random trials. Here the SSD value quantifies the performance error relative to the true performance obtained from the entire validation dataset. The particular validation dataset used in this experiment has 1000 units and is randomly drawn from the $1,240,590$ annotated data pool mentioned in the previous section. The size of the random sample in each trial is set to 126 to match the size of the concise dataset. Note that we purposely keep the validation

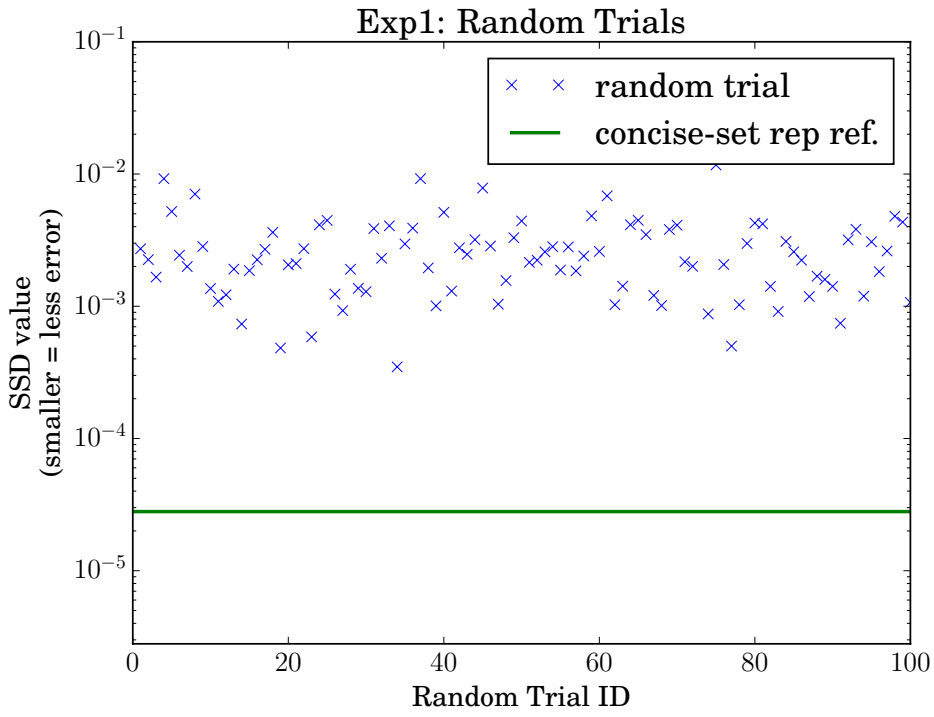


Fig. 6.6. None of the SSD values from the 100 random trials gives a better performance estimate than the SSD value obtained using the concise-set representation approach. Validation dataset size = 1000. Concise dataset size = 126. Random dataset size used for each trial = 126. Ground-truth consistency = 0.997.

dataset small for this experiment so that we can run multiple trials within a short time. We study the effect of larger validation dataset in Experiment 6.2.1.

We observe from the plot that none of the SSD values from the random sampling approach is smaller than the SSD value obtained using the proposed approach. As for the ground-truth consistency estimate, the value in this case is 0.997 (See Section 5.2.10 and Eq. 5.2).

Experiment 2: Concise-set Representation versus the Average of the Results Obtained with Randomly Drawn Samples

In this experiment, we average the confusion matrices obtained by the traditional random sampling approach over many trials and then compare it with the the confusion matrix obtained through the concise-set representation approach. We then repeat the averaging experiment 100 times and plot the min, max, 5^{th}, 50^{th}, and 95^{th} percentiles curves. These percentile curves give us additional insights into the effectiveness of our concise-set representation approach.

As made evident by Fig. 6.7, the median (i.e., 50^{th} percentile) curve in the figure indicates that taking the average of at least 80 random trials is needed to achieve a more accurate confusion matrix than the concise-set representation approach 50% of the time. We can draw similar conclusions regarding the best-case scenarios from the "best" curve. We see that in the best case, averaging of at least 10 trials are needed for the random sampling approach to have any chance of outperforming the concise-set approach.

Experiment 3: Concise-set Representation versus Random Datasets of Different Sizes

The goal of this experiment is to examine the effect of dataset size on the random sampling approach vis-a-vis the results obtained with our concise-set representation. In particular, we wish to investigate how large a randomly drawn dataset must be in order for it to outperform the concise-set representation. We repeat the experiment 100 times for each size and plot the values of the min, max, 5^{th}, 50^{th}, and 95^{th} percentiles as shown in Fig. 6.8. The figure indicates that for the random-sampling approach to have any chance of outperforming the concise-set representation, the size of a randomly drawn dataset should be at least 4.7 times larger than the concise dataset.

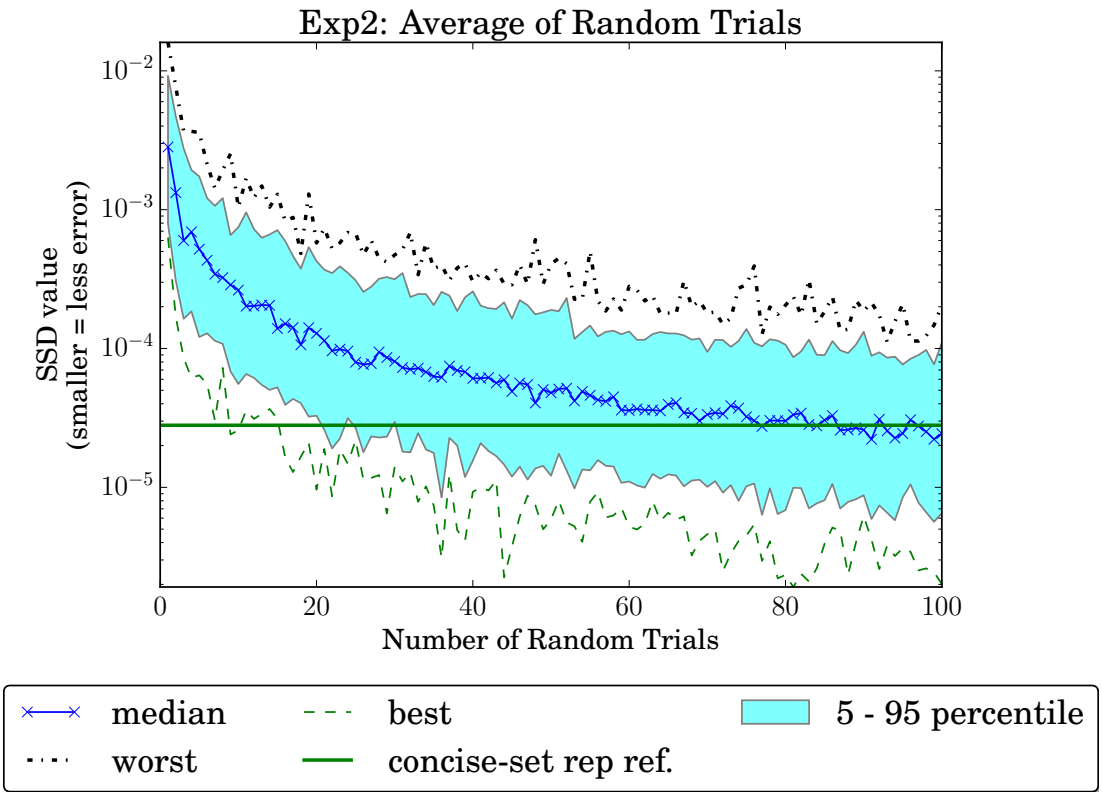


Fig. 6.7. From the "best" performance curve, we see that averaging over at least 10 random trials is needed for the random sampling approach to have any chance of outperforming the concise-set representation. Validation dataset size = 1000. Concise dataset size = 126. Random dataset size used for each trial = 126. Number of repeated experiments per trial = 100. Ground-truth consistency = 0.997.

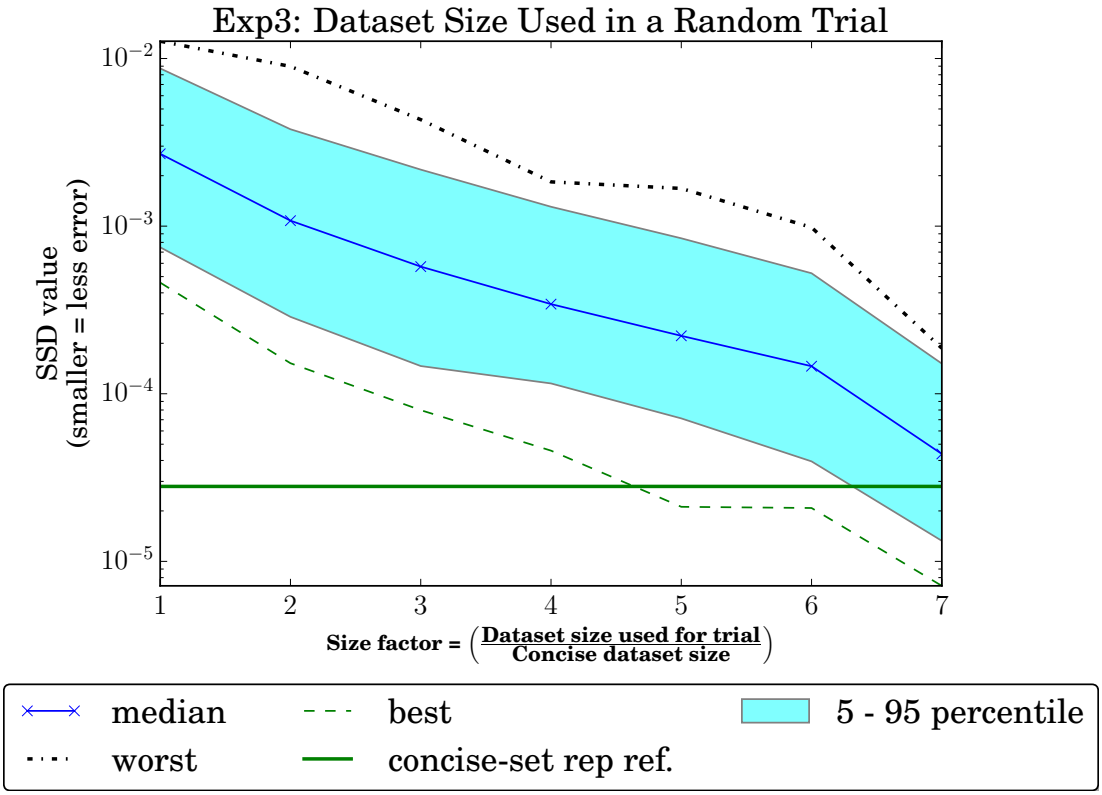


Fig. 6.8. Performance of randomly drawn datasets of different sizes. From the "best" performance curve, we see that a random dataset needs to be at least 4.7 times larger than the concise dataset in order to have any chance of outperforming the concise-set representation approach. Validation dataset size = 1000. Concise dataset size = 126. Number of trials with differently sized random datasets = 100. Ground-truth consistency = 0.997.

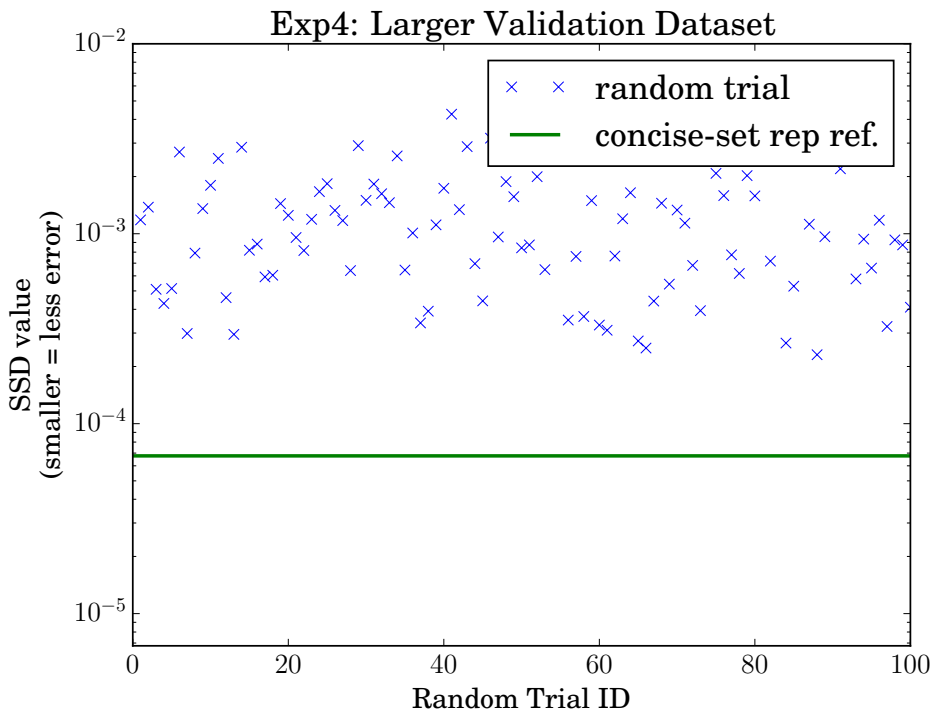


Fig. 6.9. With a larger validation dataset (10000 units instead of 1000), the concise-set representation approach continues to outperform the random sampling approach. Concise dataset size = 379. Random dataset size used for each of the 100 trials = 379. Ground-truth consistency = 0.9717.

Experiment 4: Larger Validation Dataset

In this experiment, we repeat Experiment 6.2.1 but with a larger validation dataset. The result shown in Fig. 6.9 indicates that the concise-set representation approach still outperforms the traditional random sampling approach.

Experiment 5: Evaluations with Multiple Validation Datasets

To show that our approach works on other validation datasets, we repeat the last experiment (Section 6.2.1) on 5 validation datasets, each containing 10000 random samples drawn from the annotated pool described in Section 6.2.

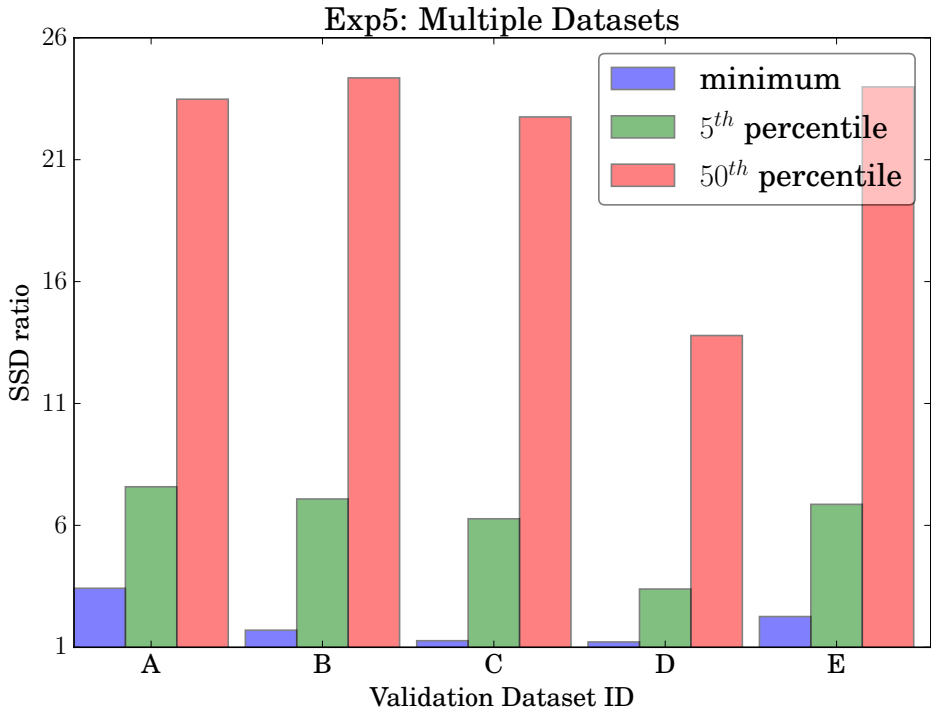


Fig. 6.10. Each group of three bars is a run of Exp4 (See Section 6.2.1) but on a different validation dataset. The SSD ratios are computed by Eq. 6.3. When the minimum ratio is above 1.0, it means that the concise-set representation approach is better than the random-sampling approach in all of the 100 random trials.

Fig. 6.10 shows a bar chart consisting of five different validation datasets. The performance of each dataset is summarized by three SSD ratios calculated from Eq. 6.3 for K $\in [0, 5, 50]$:

$$\text{SSD Ratio} = \frac{\text{K-percentile(100 random trials)}}{\text{SSD from Concise-set rep.}} \tag{6.3}$$

Note that depending on the region, the minimum ratio can be close to 1.0. For example, in a region where not much diversity exists, the traditional random-sampling approach will do just as well as the concise-set representation approach. For another example, if the region is highly diverse, then the concise dataset will be larger, as more clusters are formed. In that case, the best performance by the random-sampling approach might improve due to the larger number of samples used in its random dataset, which has the same size as the concise dataset.

6.2.2 Classifier Training Experiments

In the previous section, we compared two methods of evaluating classifiers. We showed that our concise-set representation method gives more accurate result. In this section, we compare two methods of creating a training set for the purpose of training classifiers. The two methods compared are the traditional random sampling approach and our concise-set representation approach. We will refer to the training set created by the traditional method as the "random training set" and our method as the "concise training set", which is represented by the (R, W) tuple where R is the array of cluster representatives and W is the array of corresponding cluster sizes (See section 5.2.7).

For training a land-cover classifier, we followed the work in [1] and trained a SVM using "band ratio" features. When training with the concise training set, we take the cluster sizes into account by weighting the training example at each cluster representative with its cluster size. As for the metric used in comparing classifiers, we compute the classifier accuracy by taking the sum of the diagonal elements in the normalized confusion matrix.

Experiment A: Learning Rate as a Function of Training Set Size

In this experiment, we are interested in answering the question of how two different training set selection strategies affect the classifier accuracy. To answer this question, we first randomly split a validation dataset into two parts – one part for the training pool, which is where the training sets will be drawn from, and the other part for the test set. We describe the experiment protocols next.

Traditional random sampling training strategy:

1. Initialize T to a small number.
2. Create a training set by drawing a random subset of size T from the training pool.
3. Train a classifier from the training set.
4. Compute the classifier accuracy using the test set.
5. Repeat 100 times from Step 2.
6. Repeat from Step 2 with $T = T + 1$ until T reaches the size of the training pool.

Concise-set representation training strategy:

1. Create an initial concise-set representation of the training pool.
2. Train a classifier from the concise training set.
3. Compute the classifier accuracy using the test set.
4. Refine the concise-set representation by shrinking the most-impure cluster (See section 5.4 in Chapter 5) and let T be the size of the new concise training set.
5. Repeat from Step 2 until T reaches the size of the training pool.

Fig. 6.11 shows the learning curve of each classifier. If we look at the best-case curve for the traditional random sampling approach, we see that it is roughly the same performance as the concise-set representation approach. On the other hand, the median-case curve is much worse and does not catch up to the concise-set representation approach until about four fifths of the entire training pool is used for training the classifier.

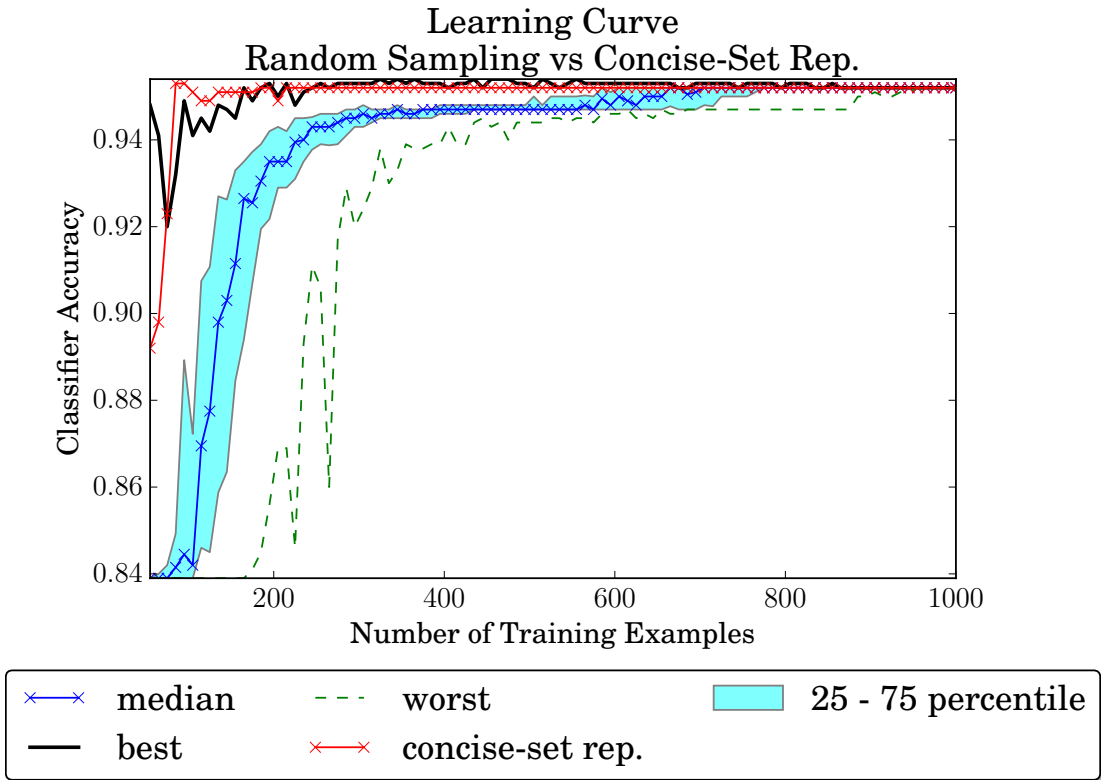


Fig. 6.11. Learning curves show how classifier accuracy improve as a function of training dataset size. Validation dataset size = 2000; Training pool size = 1000; Test set size = 1000.

Experiment B: Statistical Significance Test for Comparing Training Set Selection Strategies

In this experiment, we are interested in answering the question: Is the performance difference between the two training set selection strategies significant? To answer this question, we turn to statistical analysis. Our experimental setup is as follows:

1. Randomly split the validation dataset into two parts – one part for the training pool in which training sets will be drawn from, and the other part for the test set.

2. Create an initial concise-set representation of the training pool. Initialize T to the size of the concise dataset.

3. For the concise-set representation training strategy:

   (a) Further refine the concise-set representation until the size of the concise dataset grows to T (See section 5.4).

   (b) Train a classifier with the resulting concise training set.

4. For the traditional random sampling training strategy:

   (a) Draw a random training set of size T from the training pool.

   (b) Train a classifier using the random training set.

   (c) Repeat with 20 different random training sets. This creates 20 classifiers from which we can compute the median, 75-percentile, and 90-percentile performances later.

5. Compute the classifier accuracy on the test set created in step 1.

6. Repeat from step 3 with $T = 10\%, 20\%, \ldots, 100\%$ of the training pool size.

7. Repeat 100 times from step 1 to create 100 matched pairs to be used in statistical analysis later.

Tables 6.1, 6.2, and 6.3 compare the result obtained from the concise-set representation training strategy with the median, 75-percentile, and 90-percentile results obtained from repetition of the random sampling training approach. As can be seen from Table 6.1, compared to the median result, the concise-set representation approach is better in all 9 non-trivial training sizes (10% to 90% of the training pool)[4]. As for the 75-percentile result shown in Table 6.2, we see that the random sampling strategy starts to catch up and the two training strategies are tie in 5 out of 10 training sizes (30%, 40%, 60%, 70%, and 100%). In the case of the 90-percentile result shown in Table 6.3, the concise-set representation approach is better when the training set is small (at 10% and 20% sizes).

Because of the randomness involved in drawing a random training set, the traditional approach to training the classifier makes sense only if we can perform multiple trials and select the best result among them. But it is often not practical in practice to have multiple trials because for each trial, we also need to annotate every new sample in the new training set. On the other hand, the concise-set representation approach does not need multiple trials and still produces a classifier that is more accurate than the median-performing classifier obtained by the traditional random sampling approach. Therefor, the concise-set representation approach can be used to quickly training a good classifier and with minimal annotation effort.

[4]When 100% of the training pool is used for training, the two training strategies are equivalent and thus always give the same performance.

Table 6.1.
Paired Student's t-test: Comparing with the Median Performance

Training Size (%)	10	20	30	40	50
Mean of Diffs	0.1039	0.0089	0.0020	0.0009	0.0006
P-value	0.0000	0.0000	0.0000	0.0000	0.0000
P-value < 0.050	Yes	Yes	Yes	Yes	Yes
ConciseSet is better	Yes	Yes	Yes	Yes	Yes
Training Size (%)	60	70	80	90	100
Mean of Diffs	0.0004	0.0004	0.0003	0.0001	0.0000
P-value	0.0003	0.0002	0.0006	0.0184	nan
P-value < 0.050	Yes	Yes	Yes	Yes	No
ConciseSet is better	Yes	Yes	Yes	Yes	N/A

Table 6.2.
Paired Student's t-test: Comparing with the 75-percentile Performance

Training Size (%)	10	20	30	40	50
Mean of Diffs	0.0592	0.0030	0.0003	-0.0001	-0.0003
P-value	0.0000	0.0000	0.2324	0.4884	0.0025
P-value < 0.050	Yes	Yes	No	No	Yes
ConciseSet is better	Yes	Yes	N/A	N/A	No
Training Size (%)	60	70	80	90	100
Mean of Diffs	-0.0002	-0.0001	-0.0002	-0.0002	0.0000
P-value	0.1101	0.0957	0.0228	0.0105	nan
P-value < 0.050	No	No	Yes	Yes	No
ConciseSet is better	N/A	N/A	No	No	N/A

Table 6.3.
Paired Student's t-test: Comparing with the 90-percentile Performance

Training Size (%)	10	20	30	40	50
Mean of Diffs	0.0401	0.0013	-0.0004	-0.0005	-0.0005
P-value	0.0000	0.0034	0.0499	0.0031	0.0000
P-value < 0.050	Yes	Yes	Yes	Yes	Yes
ConciseSet is better	Yes	Yes	No	No	No
Training Size (%)	60	70	80	90	100
Mean of Diffs	-0.0004	-0.0003	-0.0003	-0.0003	0.0000
P-value	0.0006	0.0002	0.0001	0.0000	nan
P-value < 0.050	Yes	Yes	Yes	Yes	No
ConciseSet is better	No	No	No	No	N/A

7. SCALING UP THE CONCISE-SET REPRESENTATION TO HANDLE BIG DATA

In this chapter, we address the scalability issues faced when applying the concise-set representation to a potentially very large dataset, aka the Big Data.

7.1 A Brief Review of Big Data Processing

The nature of Big Data changes over time. What used to be considered Big Data 30 years ago are trivial to handle by today's technology. Therefore, the definition of Big Data must be defined in terms of the technology that is timeless. One such definition is given by [56]:

> ... data, perhaps, whose analysis requires massively parallel software running on tens, hundreds, or even thousands of servers.

In other words, if something is Big Data today, then we can expect its processing to require a large cluster of today's commodity computers working together in parallel.

In the next section, we review Map-Reduce – a parallel processing paradigm made popular by Google's *MapReduce programming model* [57].

7.1.1 The Map-Reduce Processing Paradigm

In functional computer programming languages such as Lisp, `map()` is a function that takes as input a list of items and an operator. The output of `map()` is a list consists of the outputs obtained from applying the operator to each item in the input list. For example, `map([1,2,3], square)` returns `[1,4,9]`. In this example, three `square` operations were carried out. Note that the semantic of the `map()` function

naturally allows simultaneous executions. This is the case because the operator can be applied to all items in the input list independently and simultaneously.

The opposite of `map()` is `reduce()`. Like `map()`, `reduce()` also takes as input a list of items and an operator. However, unlike `map()`, the operator in a reduce function takes the entire list as input and return just one item. For example, `reduce([1,2,3], sum)` returns `6`. Here `[1,2,3]` is the input list and `6` is the returned item after applying the operator `sum` to the list.

Perhaps not obvious, the reduce function can also be parallelized. But only when the operator has the associative property. For example `reduce([1,2,3], sum)` could be executed by three reduce function calls: `reduce([reduce([1,2], sum), reduce([3],sum)], sum)`. Of these three invocations of the reduce function, two can run simultaneously. They are: `reduce([1,2], sum)` and `reduce([3],sum)`.

To summarize, both the map and reduce functions are parallelizable. Map function distributes; Reduce function coalesces. These two primitive functions form the basis of the MapReduce programming model, to be reviewed next.

7.1.2 The MapReduce Programming Model

In this section, we briefly review the MapReduce Programming Model [57]. First, we review some relevant terms:

Worker

A processing unit.

Master

The head worker that schedules and assigns tasks to workers.

Task

A description of the work to be performed.

Map Phase

A phase in the MapReduce execution flow when the map tasks are run.

Map Function

The C++[1] function invoked when a worker is performing the map task.

Reduce Phase

A phase in the MapReduce execution flow when the reduce tasks are run.

Reduce Function

The C++ function invoked when a worker is performing the reduce task.

Key-Value Pair

A pair of custom data encapsulations: *key* and *value*.

Intermediate File

A file that stores output key-value pairs; The intermediate files are generated by workers during the map phase.

The MapReduce programming model separates data processing and cluster management. As a result, the programmer only has to focus on data processing. All other aspect of the program, such as cluster management and inter-process communication, are hidden away from the programmer. Therefore, the programmer's only job is to implement the map and reduce functions.

During the map phase, the *master* assigns each available *worker* a *map task*. The worker then fetches the assigned data chunk accordingly and hands it over as a *key-value pair* to the *map function*, which is provided by the programmer. The map function processes the input key-value pair and outputs another key-value pair. This output key-value pair is stored in one of the R *intermediate files*. After the worker finishes processing the assigned data chunk, it reports the intermediate file's storage location to the *master*. The map phase is said to be completed when all of the M map tasks are finished. Fig. 7.1 shows the block diagram of the map phase. The values of M and R are typically chosen by the programmer.

[1]Google's original implementation of MapReduce was written in C++. Later implementations, such as Hadoop (`https://hadoop.apache.org/`), allows other programming languages to be used.

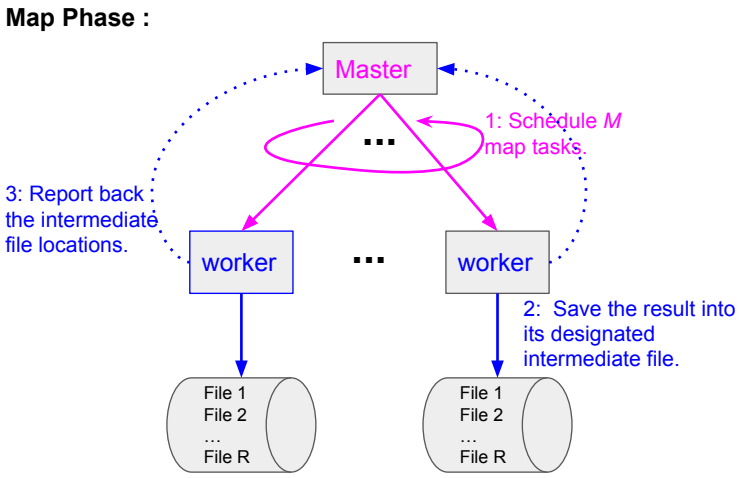


Fig. 7.1. The *Map Phase* of the MapReduce Programming Model.

During the reduce phase, the master assigns a *reduce task* to an available worker. The worker then fetches the assigned set of intermediate files (e.g., all the "File 3"s) and groups the key-value pairs from all file in the set by their keys. Subsequently, the worker passes the grouped key-value pairs to the *reduce function* and stores the output of the function into an output file. Finally, the worker sends the location of the output file back to the master. The reduce phase is said to be completed when all of the R reduce tasks are finished. Fig. 7.2 shows the block diagram of the reduce phase.

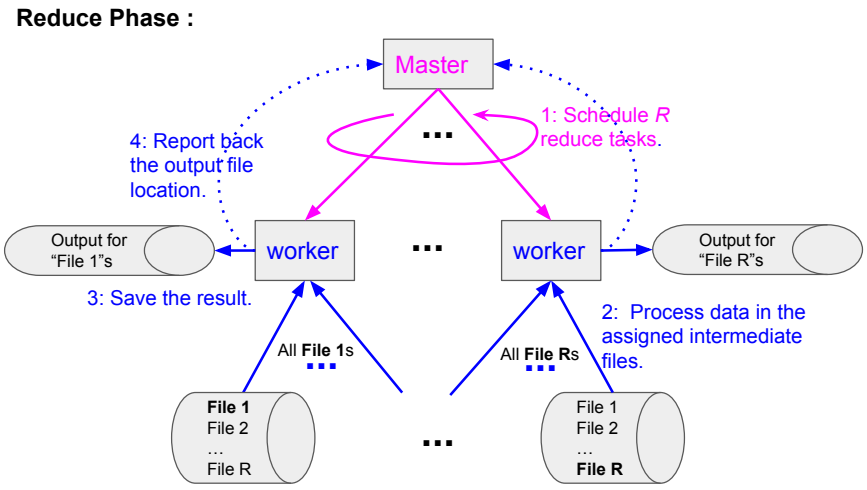


Fig. 7.2. The *Reduce Phase* of the MapReduce Programming Model.

7.1.3 The Cloud Computing Services Model

So far, we have reviewed the Map-Reduce processing paradigm and examined a particular programming model called MapReduce introduced by Google. Next, we will review *Cloud Computing* and see how it fits in the overall picture of big-data processing.

In general, cloud computing is a computing infrastructure that provides configurable and on-demand computing resources. It has three layers of abstraction [58]. They are:

- layer1: Infrastructure as a Service (IaaS)
- layer2: Platform as a Service (PaaS)
- layer3: Software as a Service (SaaS)

Among these layers of abstraction, higher layers are built from the services provided by the lower layers. At the bottom-most layer, we have the IaaS layer that provides services for accessing the bare-metal hardware. Examples of IaaS implementations

include Amazon Elastic Compute Cloud[2] (aka Amazon EC2), OpenStack[3], and Eucalyptus[4].

Common to all IaaS implementations, users can upload their own virtual machine (VM) in the form of an operating system disk image file. Once uploaded, users can then deploy as many instances of their VMs as allowed by the available resource and the usage policy. To instantiating a VM, the user chooses a "resource profile" that specifies the desired hardware configuration in terms of CPU, memory, storage, and network. For example, a low-demanding resource profile may specify 2 CPUs, 4 GB of memory, and 8 GB of disk storage.

The next layer of abstraction above IaaS is PaaS, which provides services on the "platform" level. At this level of abstraction, users can quickly build big-data applications without having to manage the underling hardware infrastructure. A good example of PaaS is the Google App Engine framework[5].

Finally, the top-most layer in the cloud computing abstraction is SaaS. SaaS provides services to users on the "software" level. Examples of SaaS include E-mail service such as Google e-mail[6], on-line video sharing and streaming services such YouTube[7] and Netflex[8], social networking service such as Facebook[9], and massive open online course services (MOOC) such as Coursera[10] and Udacity[11]. These SaaS examples all share a few things in common – they are user-friendly and can simultaneously serve many users and manage massive amount of data.

[2] https://aws.amazon.com/ec2
[3] http://www.openstack.org
[4] https://github.com/eucalyptus/eucalyptus
[5] https://cloud.google.com/appengine/
[6] http://www.gmail.com/
[7] http://www.youtube.com/
[8] http://www.netflix.com/
[9] http://www.facebook.com/
[10] http://www.coursera.com/
[11] http://www.udacity.com/

In summary, cloud computing provides all the essential and supporting services needed to implement the Map-Reduce processing paradigm, which is the core basis of big-data processing.

7.2 Creating the Concise-set Representation using the Map-Reduce Processing Paradigm

To create a concise-set representation for a large set of satellite images, we apply the Map-Reduce processing paradigm and implement our system on a cloud-computing platform. In the next two sections, we will first review the overall system and then we will specify the various map and reduce phases within the system.

7.2.1 System Overview

Fig. 7.3 shows the system overview. The input to the system is a set of high-resolution satellite images that cover a wide-area region and the output is a merged concise dataset.

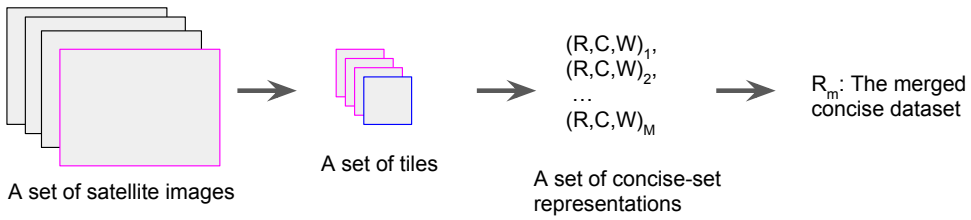


Fig. 7.3. Processing a wide-area region involves three steps. First we partition the satellite images into a set of tiles. Then, we create a concise-set representation for each tile. And finally, we combine all concise-set representations into a merged concise dataset for annotation.

The first step in the processing pipeline is arranging each satellite image into a set of overlapping tiles, as shown in Fig. 7.4. Next, each tile is converted into a concise-set representation by the process summarized in Fig. 7.5 and described in detail in

Chapter 5. And finally, all concise-set representations are combined to form a merged concise dataset using a hierarchical merging strategy. The hierarchical merging step was discussed in Section 5.3 of Chapter5.

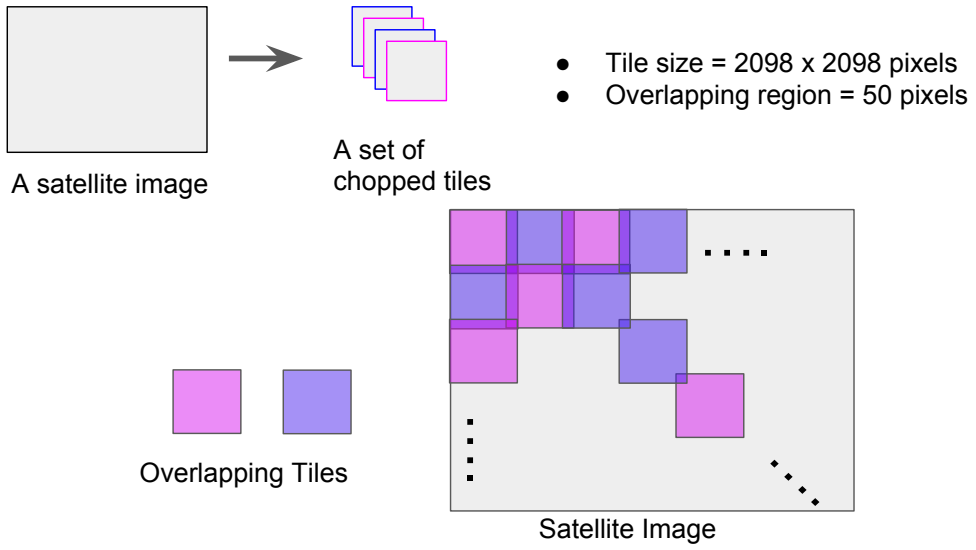


Fig. 7.4. Arranging a satellite image into overlapping tiles.

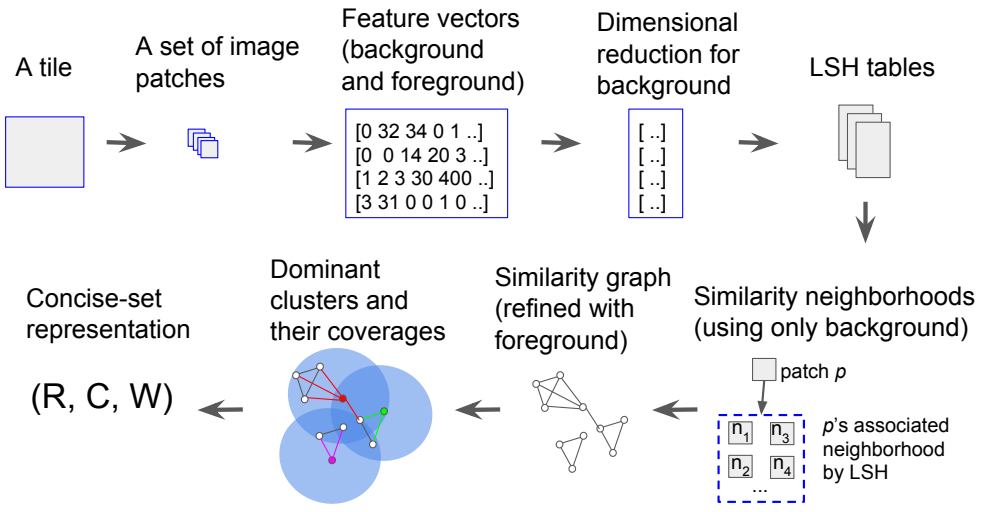


Fig. 7.5. Converting a tile into a concise-set representation. See Chapter 5 for more details.

7.2.2 Map and Reduce Phases

In this section, we present the map and reduce phases that form the basis of our system.

Map Phase 1: Map Satellite Images to Sets of Tiles

- Phase input = A list of satellite images.
- Map function:
  - Input = A satellite image.
  - Output = A list of tiles.

Map Phase 2: Map Tiles to Feature Files

- Phase input = A list of tiles.
- Map function:
  - Input = A tile.
  - Output = A feature file. This file contains features extracted from all image patches.

The process of creating a feature file from a tile is summarized in Fig. 7.6.

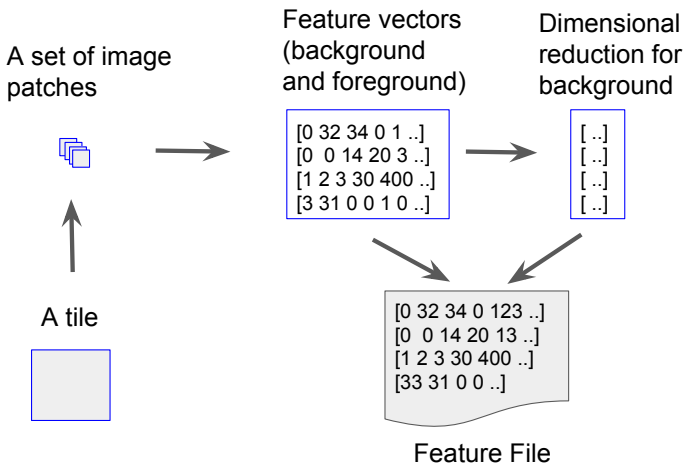


Fig. 7.6. Creating a feature file from a tile.

Map Phase 3: Map Feature Files to Concise-Set Files

- Phase input = A list of feature files.
- Map function:
  - Input = A feature file.
  - Output = A concise-set file.

The process of creating a concise-set file from a feature file is summarized in Fig. 7.7.

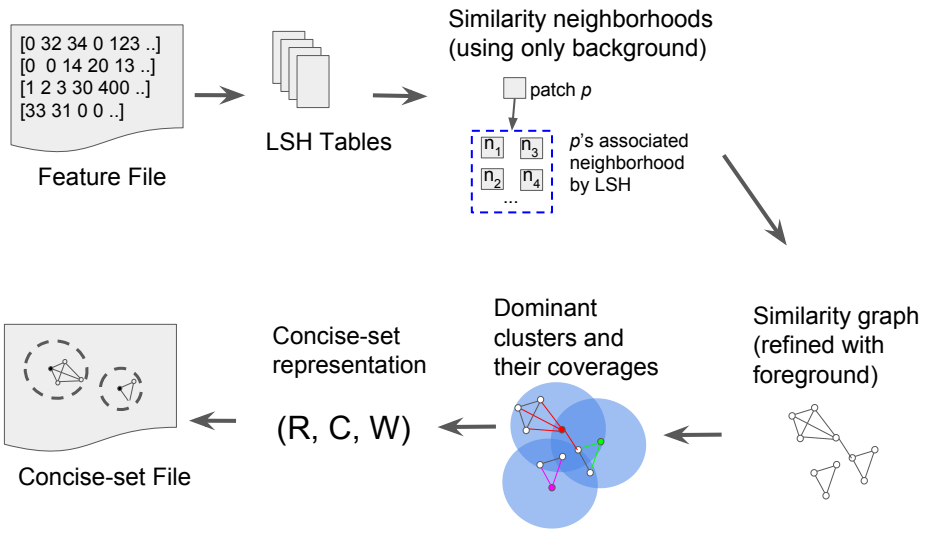


Fig. 7.7. Creating a concise-set file from a feature file.

Reduce Phase: Reduce concise-set Files to a Merged Concise-Set File

- Phase input = A list of concise-set files.
- Reduce function:
  - Input = A list of concise-set files.
  - Output = The merged concise-set file.

Fig. 7.8 summarizes the process of combining multiple concise-set files to create a merged concise-set file.

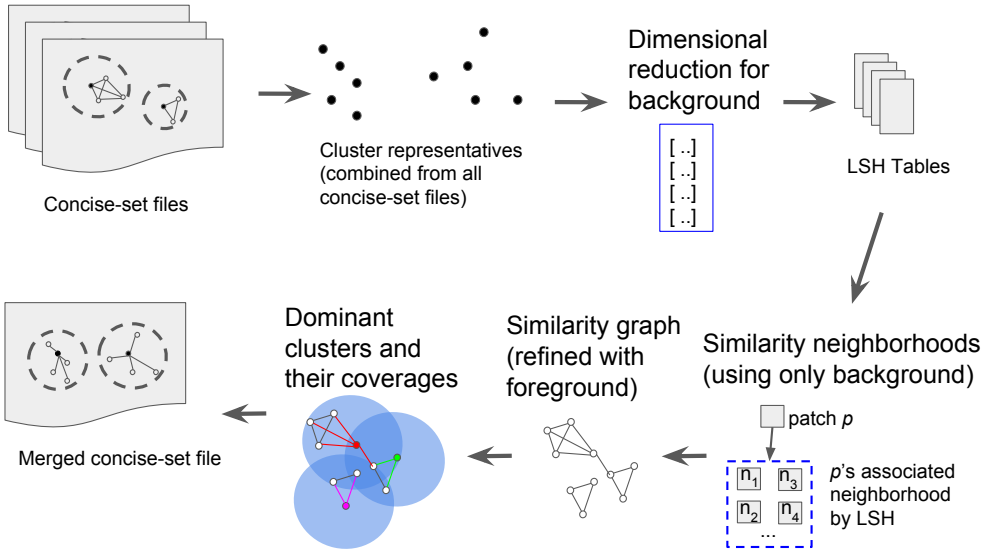


Fig. 7.8. The reduce phase merges concise-set files into one using the hierarchical merging method described in Section 5.3.

7.3 Algorithm Complexity Analysis

To understand how well our system scales to Big Data, we must analyze all algorithms used in the system. In the next three sections, we will analyze the run-time complexity of three major algorithms we used in our system. These three sections will help us understand the current limitation and also provide potential future research directions.

7.3.1 Indexing the Dataset using Hyperplane LSH

In this section, we investigate the question pertaining to the cost of using Hyperplane LSH to index the dataset. Recall that the purpose of indexing the dataset is to support efficient neighborhood querying. Ideally, given an item, we could like to retrieve all of its neighbors in constant time. Two indexing methods that supports this fast retrieval are the brute-force method and LSH. The brute-force approach is not appropriate because the number of comparisons it takes is a quadratic function

of the size of the dataset. On the other hand, the number of comparisons for the LSH approach is linear in dataset size.

As shown in Appendix C, the cost of using LSH to index the dataset depends on the size of the dataset as well as the number of hashes needed to achieve the desired performance guarantee. When the number of hashes is large, due to desiring a high performance guarantee, the cost of indexing the dataset dominates the cost of data dimensional reduction. Specifically, the cost of indexing a dataset with Hyperplane LSH is linear in the number of data dimensions, thus a 100x reduction in data dimension translates to two order of magnitude in speedup.

To show it with complexity analysis, let d and k be the number of data dimensions before and after applying FastMap (See Section 4.6 of Chapter 4). Then, the time complexity of FastMap is $O(d\ k\ n)$, where n is the size of the dataset. On the other hand, the time complexity of indexing the dataset with Hyperplane LSH is $O(h\ d\ n)$, where h is the number of hashes per datum. The total running times for indexing a dataset with and without dimensional reduction is given in Eq. 7.3 and q. 7.1, respectively. The speedup effect due to the reduced data dimensionality is thus $\theta(\frac{d}{k})$. As a result, when we reduce the data dimensionality of the background color-histogram vectors from 9024 down to about 100, we can expect to get about two order of magnitude in speedup.

$$T_{\text{without dimensional reduction}} = O(h\ d\ n) \tag{7.1}$$

$$T_{\text{with dimensional reduction}} = O(d\ k\ n) + O(h\ k\ n) \tag{7.2}$$

$$= O(h\ k\ n) \quad \text{when } h \gg d \tag{7.3}$$

With regard to the the appropriateness of using LSH for large datasets, we note that when the dataset is small, indexing the dataset with LSH actually takes longer time than the brute-force indexing method. This is the case simply because $O(n)$ can grow faster than $O(n^2)$ for values of n below some threshold. For Hyperplane LSH, this threshold is $\theta(h)$. As an example, with the LSH parameters set to: $d_1 = 15$,

$d_2 = 37.5$, desired $p_1 = 0.99$, and desired $p_2 = 0.001$, the value of h is 37278 (see Table C.1). The means, in order to take advantage of Hyperplane LSH, the dataset must not be smaller than h. Otherwise, we might as well use the brute-force indexing method.

7.3.2 Sifting Through Neighbor Candidates in a LSH Bucket

In the previous section, we mentioned that LSH indexing has $O(h\ d\ n)$ time complexity. Here n is the size of the dataset, h is a function of the LSH performance guarantee, and d is the number of data dimensions. We stated that LSH indexing is linear in n because in our case d is typically less than 100 and h is a constant that does not change.

Because LSH is an approximated method, the collection of neighbor candidates inside the LSH bucket[12] of a given input query may include false positives as well as duplicates. In the worst case, when every item is similar to all other items, it will take $O(n^2)$ time to remove false positives and duplicates from every neighborhood. This is the case because there are n neighborhoods, one per datum, and each neighborhood contains $O(n)$ neighbor candidates in the worst case.

7.3.3 Extracting the Dominating Clusters/Neighborhoods

As we mentioned in Section 5.2.6 of Chapter 5, the greedy solution for the Dominating Set problem runs in $O(|E|)$ time where E is the set of edges in the graph. In the worst case, when the graph is dense, $|E| = O(|V|^2)$ the greedy solution takes $O(|E|) = O(|V|^2)$ where V is the set of vertices in the graph. In our application, the number of vertices in the similarity graph is the size of the dataset. Therefore, extracting the dominating clusters will take $O(n^2)$ time to run in the worst case scenario. Here n is the size of the dataset.

[12] A LSH bucket is actually a collection of buckets from multiple hash tables.

7.4 Discussion

We have presented our system design for handling the big-data situation in which the entire dataset is too big to be processed by a single computing unit. Our key approach is employing Map-Reduce processing paradigm to distribute the computation across multiple computing units. With this approach, we mitigate the run-time complexity issues mentioned in the previous section.

In general, when a $O(n^2)$ algorithm can not be avoid, the strategy is to keep n small enough such that the algorithm can still finish within a reasonable amount of time. If we break the entire dataset into small chunks/partitions, then each chunk can be processed quickly even though the complexity of the processing algorithm is $O(n^2)$. Furthermore, if we process all these chucks in parallel across many computing units, then the overall processing time is the time it takes to process a single chunk plus the merging time it takes to execute the subsequent merging step.

The current implementation of our system is written in Python programming language and this implementation can handle chunk size of $n = 250000$. Further improvement in execution speed requires using optimized programming language such as C++ and replacing the quadratic-time algorithms mentioned in the previous section with linear-time equivalents, if possible.

8. CONCLUSION AND FUTURE WORK

It can be mentally exhausting for human annotators to generate the ground truth needed for evaluating land-cover classifiers meant for large geographic regions covered by hundreds of satellite images. Human annotators end up wasting time by not realizing that new annotations may not be adding any additional discriminatory information to those already supplied. The work we have presented in this dissertation seeks to alleviate the annotation burden by reducing redundancies in the data through fast, albeit approximate, clustering by the LSH algorithm. Subsequently, the human is asked to annotate only the cluster representatives, in other words, the concise dataset, with one representative per cluster. Given the fairly wide range of the attributes derived from the spectral signatures that are used in land-cover classification, LSH may represent each land-type by hundreds of clusters. Nonetheless, providing annotations for the cluster representatives takes far less work than for the individual pixels in the satellite images that cover a wide area.

What adds to the power of our approach is our demonstration that the approach is not overly sensitive to the choice of the parameters for the LSH algorithm. In our demonstration, we estimated the good values to use for the parameters from the satellite images that cover Chile and then used them to create a concise-set representation of the data in Australia. We validated our approach through a comparison with traditional random-sampling based methods that are typically used for large datasets. We showed that for the same annotation effort, the concise-set approach to data representation outperforms the traditional random sampling approach.

With regard to potential future research directions, one possibility is to explore other linear-time clustering methods that also work well for high-dimensional data. Even though Hyperplane LSH, which we used in this dissertation, has indexing time linear to the data size and dimension, it also has a high overhead. As we explained in

Chapter 7, LSH only becomes faster than the brute-force approach when the size of the dataset exceeds certain threshold. This threshold is proportional to the product of AND and OR constructions. One way to lower this overhead is to relax the performance guarantee. But this is obviously undesirable. Another way may be to transform the dataset in certain ways so that the data are more spread out spatially, effectively relaxing the similarity threshold needed for clustering. Another future research direction is to reduce overall computation by not removing false positives from every cluster/neighborhood. Using this strategy, we can modify Algorithm 1 in Chapter 5 so that we only remove false positives from a cluster when the cluster has a high chance of being a dominant cluster. We can also achieve significant saving in computation by avoiding creating a dense similarity graph in the first place. For example, if we know a cluster has no chance of becoming dominant, then, we don't need to add its edges into the similarity graph. If we can enforce the similarity graph to always be sparse, then extracting dominant clusters will always run in linear time.

REFERENCES

REFERENCES

[1] T. Chang, B. Comandur, J. Park, and A. C. Kak, "A variance-based bayesian framework for improving land-cover classification through wide-area learning from large geographic regions," *Computer Vision and Image Understanding*, vol. 147, pp. 3–22, 2016.

[2] Y. Sakurai, M. Yoshikawa, S. Uemura, H. Kojima *et al.*, "The a-tree: An index structure for high-dimensional spaces using relative approximation," in *VLDB*, vol. 2000, 2000, pp. 516–526.

[3] D. M. Mount, "ANN Programming Manual," https://www.cs.umd.edu/~mount/ANN/Files/1.1.2/ANNmanual_1.1.pdf, 2010.

[4] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB*, vol. 98, 1998, pp. 194–205.

[5] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 97–104.

[6] M. Izbicki and C. Shelton, "Faster cover trees," in *International Conference on Machine Learning*, 2015, pp. 1162–1170.

[7] B. Salehi, "Urban land cover classification and moving vehicle extraction using very high resolution satellite imagery," Ph.D. dissertation, Department of Geodesy and Geomatics Engineering, University of New Brunswick, Fredericton, New Brunswick, Canada, 2012.

[8] A. M. Mika, "Three decades of landsat instruments," *Photogrammetric Engineering & Remote Sensing*, vol. 63, no. 7, pp. 839–852, 1997.

[9] C. Ünsalan and K. L. Boyer, *Multispectral Satellite Image Understanding: From Land Classification to Building and Road Detection*, ser. Advances in Computer Vision and Pattern Recognition. Springer, 2011.

[10] N. W. Longbotham, F. Pacifici, S. Malitz, W. Baugh, and G. Camps-Valls, "Measuring the spatial and spectral performance of worldview-3," in *Hyperspectral Imaging and Sounding of the Environment*. Optical Society of America, 2015, pp. HW3B–2.

[11] Microsoft, "Ultracam," http://download.microsoft.com/download/4/2/1/4216DF68-F652-4C64-9832-A77A3697DA03/UALL-E-OV-1213-1.0-Letter%20web.pdf, 2014.

[12] T. Updike and C. Comp, "Radiometric use of worldview-2 imagery," Digital Globe, Tech. Rep., 2010.

[13] G. Marchisio, F. Pacifici, and C. Padwick, "On the relative predictive value of the new spectral bands in the worldwiew-2 sensor," in *Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2010, pp. 2723–2726.

[14] G. Sarp, "Spectral and spatial quality analysis of pan-sharpening algorithms: A case study in istanbul," *European Journal of Remote Sensing*, vol. 47, no. 1, pp. 19–28, 2014.

[15] G. Dial and J. Grodecki, "Rpc replacement camera models," in *Proc. ASPRS Annual Conference*, 2005, pp. 1–5.

[16] T. Prakash, B. Comandur, T. Chang, N. Elfiky, and A. Kak, "A generic road-following framework for detecting markings and objects in satellite imagery," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4729–4741, 2015.

[17] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[18] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE transactions on information theory*, vol. 14, no. 1, pp. 55–63, 1968.

[19] J. M. Van Campenhout, "On the Peaking of the Hughes Mean Recognition Accuracy: The Resolution of an Apparent Paradox," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 8, no. 5, pp. 390–395, 1978.

[20] G. V. Trunk, "A problem of dimensionality: A simple example," *IEEE Transactions on pattern analysis and machine intelligence*, no. 3, pp. 306–307, 1979.

[21] M. C. Alonso, J. A. Malpica, and A. M. de Agirre, "Consequences of the hughes phenomenon on some classification techniques," in *Proc. ASPRS annual conference*, 2011, pp. 1–5.

[22] F. Camastra and A. Vinciarelli, "Estimating the intrinsic dimension of data with a fractal-based method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 10, pp. 1404–1407, 2002.

[23] K. Koonsanit, C. Jaruskulchai, and A. Eiumnoh, "Band selection for dimension reduction in hyper spectral image using integrated information gain and principal components analysis technique," *International Journal of Machine Learning and Computing*, vol. 2, no. 3, pp. 248–251, 2012.

[24] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *SCIENCE*, vol. 290, pp. 2323–2326, 2000.

[25] J. Theiler, "Efficient algorithm for estimating the correlation dimension from a set of discrete points," *Physical review A*, vol. 36, no. 9, pp. 4456–4462, 1987.

[26] E. Levina and P. J. Bickel, "Maximum likelihood estimation of intrinsic dimension," in *Advances in neural information processing systems*, 2005, pp. 777–784.

[27] R. B. Myneni, F. G. Hall, P. J. Sellers, and A. L. Marshak, "The interpretation of spectral vegetation indexes," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 2, pp. 481–486, 1995.

[28] A. F. Wolf, "Using worldview-2 vis-nir multispectral imagery to support land mapping and feature extraction using normalized difference index ratios," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVIII*, vol. 8390. International Society for Optics and Photonics, 2012, p. 83900N.

[29] CIE Technical Committee 1-48, "Colorimetry (3rd edition)," International Commission on Illumination, Tech. Rep., 2004.

[30] F. S. Welsh, "More exact & flexible color system now in use," in *Finish Notes - The newsletter of architectural finishes investigation.* Frank S. Welsh company, 1993, vol. 1, no. 2.

[31] A. Pujol and L. Chen, "Color quantization for image processing using self information," in *International Conference on Information, Communications Signal Processing.* IEEE, Dec 2007, pp. 1–5.

[32] T. Celik, "Unsupervised change detection in satellite images using principal component analysis and k-means clustering," *IEEE Geoscience and Remote Sensing Letters*, vol. 6, no. 4, pp. 772–776, 2009.

[33] S. García-López, J. Delgado, J. Cardenal, and J. Caracuel, "Using pca transformation to remove the tenuous cloudiness effect in multispectral satellite sensor images," *International Journal of Remote Sensing*, vol. 26, no. 1, pp. 209–216, 2005.

[34] L. Journaux, I. Foucherot, and P. Gouton, "Multispectral satellite images processing through dimensionality reduction," in *Signal Processing for Image Enhancement and Multimedia Processing.* Springer, 2008, pp. 59–66.

[35] R. Alley, "Algorithm theoretical basis document for decorrelation stretch," Jet Propulsion Lab, Tech. Rep., 1996.

[36] A. C. Kak, "Constructing Optimal Subspaces for Pattern Classification," https://engineering.purdue.edu/kak/Tutorials/OptimalSubspaces.pdf, 2018.

[37] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International journal of computer vision*, vol. 77, no. 1-3, pp. 125–141, 2008.

[38] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *Journal of computational and graphical statistics*, vol. 15, no. 2, pp. 265–286, 2006.

[39] C. Faloutsos and K.-I. Lin, *FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets.* ACM, 1995, vol. 24, no. 2.

[40] A. C. Tamhane and D. D. Dunlop, *Statistics and Data Analysis from Elementary to Intermediate.* Upper Saddle River, NJ, USA: Prentice Hall, 2000.

[41] H. Xie, X. Tong, W. Meng, D. Liang, Z. Wang, and W. Shi, "A multilevel stratified spatial sampling approach for the quality assessment of remote-sensing-derived products," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4699–4713, 2015.

[42] J. Wickham, S. Stehman, J. Smith, T. Wade, and L. Yang, "A priori evaluation of two-stage cluster sampling for accuracy assessment of large-area land-cover maps," *International Journal of Remote Sensing*, vol. 25, no. 6, pp. 1235–1252, 2004.

[43] R. De Fries, M. Hansen, J. Townshend, and R. Sohlberg, "Global land cover classifications at 8 km spatial resolution: the use of training data derived from landsat imagery in decision tree classifiers," *International Journal of Remote Sensing*, vol. 19, no. 16, pp. 3141–3168, 1998.

[44] C. Homer, C. Huang, L. Yang, B. Wylie, and M. Coan, "Development of a 2001 national land-cover database for the united states," *Photogrammetric Engineering & Remote Sensing*, vol. 70, no. 7, pp. 829–840, 2004.

[45] K. Karantzalos, D. Bliziotis, and A. Karmas, "A scalable geospatial web service for near real-time, high-resolution land cover mapping," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4665–4674, 2015.

[46] N. C. Codella, G. Hua, A. Natsev, and J. R. Smith, "Towards large scale land-cover recognition of satellite images," in *International Conference on Information, Communications and Signal Processing (ICICS)*. IEEE, 2011, pp. 1–5.

[47] C. Jacqueminet, S. Kermadi, K. Michel, D. Béal, M. Gagnage, F. Branger, S. Jankowfsky, and I. Braud, "Land cover mapping using aerial and vhr satellite images for distributed hydrological modelling of periurban catchments: Application to the yzeron catchment (lyon, france)," *Journal of Hydrology*, vol. 485, pp. 68–83, 2013.

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

[49] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[50] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors [lecture notes]," *IEEE Signal processing magazine*, vol. 25, no. 2, pp. 128–131, 2008.

[51] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, vol. 99, no. 6, 1999, pp. 518–529.

[52] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011.

[53] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 380–388.

[54] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[55] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[56] A. Jacobs, "The pathologies of big data," *Commun. ACM*, vol. 52, no. 8, pp. 36–44, Aug. 2009.

[57] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[58] P. M. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards & Technology, Tech. Rep., 2011, Special Publication 800-145.

[59] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 950–961.

[60] K. Terasawa and Y. Tanaka, "Spherical LSH for approximate nearest neighbor search on unit hypersphere," in *Workshop on Algorithms and Data Structures*. Springer, 2007, pp. 27–38.

[61] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: self-tuning indexes for similarity search," in *Proceedings of the 14th international conference on World Wide Web*. ACM, 2005, pp. 651–660.

APPENDIX

A. CIELUV AND CIELAB FORMULAS

In Chapter 4, Section 4.1.1, we gave an overview of the CIELUV and CIELAB color spaces [29]. In this appendix, we list the formulas used for describing these two color spaces in terms CIEXYZ color space.

A.0.1 sRGB to CIEXYZ

Given a sRGB color, (R,G,B), the CIEXYZ color can be approximated by:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1895 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} \left(\frac{R}{255}\right)^{2.2} \\ \left(\frac{G}{255}\right)^{2.2} \\ \left(\frac{B}{255}\right)^{2.2} \end{bmatrix} \tag{A.1}$$

A.0.2 CIEXYZ to CIELUV

Given a CIEXYZ color, (X, Y, Z), the CIELUV color is calculated by:

$$\begin{aligned} L* &= 116\ f\left(\frac{Y}{Y_n}\right) - 16 \\ u* &= 13\ L*\ (u' - u'_n) \\ v* &= 116\ L*\ (v' - v'_n) \end{aligned} \tag{A.2}$$

where (u', v') is the *uniform chromaticity scale diagram* defined by:

$$\begin{aligned} u' &= \frac{4\ X}{X + 15\ Y + 3\ Z} \\ v' &= \frac{9\ Y}{X + 15\ Y + 3\ Z} \end{aligned} \tag{A.3}$$

and Y_n, u'_n, v'_n correspond to the white point used. For the D65 white point, the values of (X_n, Y_n, Z_n) are:

$$X_n = 95.047 \tag{A.4}$$

$$Y_n = 100.00 \tag{A.5}$$

$$Z_n = 108.883 \tag{A.6}$$

As for the gamma-correction function, f, it is given by:

$$f(t) = \begin{cases} t^{\frac{1}{3}} & \text{if } t > (\frac{6}{29})^3 \\ \frac{t}{3(\frac{6}{29})^2} + \frac{4}{29} & \text{otherwise} \end{cases} \tag{A.7}$$

A.0.3 CIEXYZ to CIELAB

Given a CIEXYZ color, (X, Y, Z), the CIELAB color is calculated by:

$$\begin{aligned} L* &= 116\ f\left(\frac{Y}{Y_n}\right) - 16 \\ a* &= 500\ \left(f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right) \\ b* &= 200\ \left(f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right) \end{aligned} \tag{A.8}$$

Here X_n, Y_n, Z_n, and f are defined in equations Eq.A.4 to Eq.A.7.

B. FASTMAP PROJECTION

In Chapter 4, Section 4.6, we gave a general overview of the FastMap dimensionality reduction method without going into how this method projects data onto the axis and the perpendicular hyperplane. As it turns out, we only need to know the Pythagorean theorem to see how FastMap projection works.

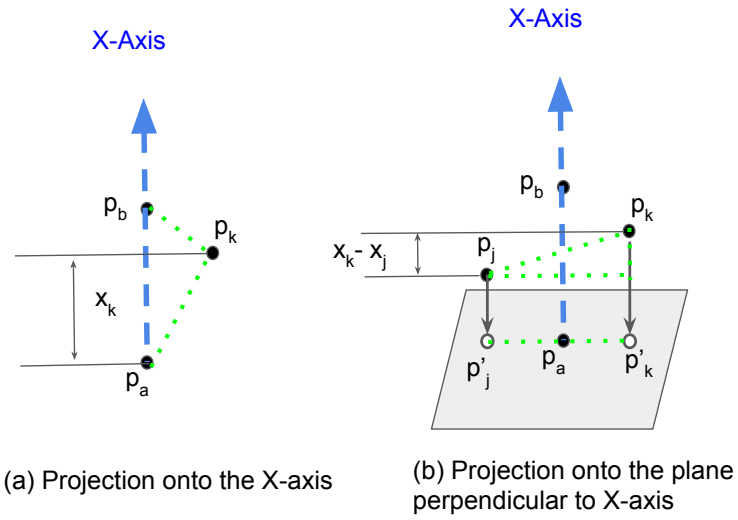


Fig. B.1. (a). The projection of an arbitrary point p_k onto the axis formed by the points p_a and p_b has length x_k relative to p_a. (b). Points p'_j and p'_k are the projections of p_j and p_k onto the hyperplane that is perpendicular to the X-axis.

Referring to Fig. B.1(a), we can compute x_k, the first coordinate value of point p_k, if we know the distance between any pair of points. Let $d_{j,k,0}$ denote the original distance between points p_j and p_k. Then, x_k can be solved from the Pythagorean equation:

$$d^2_{a,k,0} = x^2_k + (d^2_{b,k,0} - (d_{a,b,0} - x_k)^2)^2 \tag{B.1}$$

Solving Eq. B.1 for x_k, we get:

$$x_k = \frac{d_{a,k,0}^2 + d_{a,b,0}^2 - d_{b,k,0}^2}{2\ d_{a,b,0}} \tag{B.2}$$

After we compute the first coordinate value of all the points in the dataset, we can repeat Eq. B.2 for the second coordinate value. The only thing we need to change is the distance function. Let $d_{j,k,1}$ denote the distance between points p'_j and p'_k on the hyperplane that is perpendicular to the first axis (See Fig. B.1(b)). And let y_k be the second coordinate value for the point p_k. Then, the formula for y_k is analogous to Eq. B.2:

$$y_k = \frac{d_{a,k,1}^2 + d_{a,b,1}^2 - d_{b,k,1}^2}{2\ d_{a,b,1}} \tag{B.3}$$

As it turns out, it is easy to formulate $d_{j,k,1}$ in terms of $d_{j,k,0}$. Referring to Fig. B.1(b) and applying the Pythagorean theorem again, we get:

$$d_{j,k,1}^2 = d_{j,k,0}^2 - (x_j - x_k)^2 \tag{B.4}$$

Eq. B.4 has a general recursive form:

$$d_{j,k,d}^2 = d_{j,k,d-1}^2 - (\text{coord}(j,d) - \text{coord}(k,d))^2 \tag{B.5}$$

Here $\text{coord}(j,d)$ is the d^{th} coordinate value of point p_j in the lower-dimensional FastMap representation.

C. TESTING HYPERPLANE LSH WITH A SIMULATED STUDY

In this appendix, we present a simulated study on how hyperplane LSH performs, especially with high-dimensional data. The simulation consists of the following steps:

1. In a D-dimensional vector space, generate 20 randomly selected points on the surface of a unit sphere (which simulates the unit sphere used for the background color-histogram vectors). Associate with each point a circular region on the surface of the surface of the sphere. The radius of this circular region represents the angular similarity threshold used for the histogram vectors. Let the value of this radius be θ. Repeat this process until the 20 selected points are such that their associated circular regions are non-overlapping.[1] See Fig. C.1 for an example of such non-overlapping circular areas on the surface of a sphere in 3D. Each circular area on the surface subtends a hypercone at the center of the sphere as shown in the figure.

2. For each hypercone, generate 1000 random D-dimensional vectors of unit magnitude that are uniformly distributed within d_1 degrees from the axis of the hypercone. Also generate another 1000 random vectors between d_1 and d_2 degrees from the axis of the hypercone. Here d_1 and d_2 are the "nearby" and the "far apart" thresholds mentioned in Section 5.2.5. Note that the region between d_1 and d_2 is the "transition region" and LSH does not make any performance guarantee for that region.

3. Insert all $20 \times 2000 = 20000$ vectors into the LSH tables.

[1] This is not to imply that the histogram vectors for the actual satellite data can be partitioned into disjoint clusters. The disjointedness assumption is being used in the simulation study because the scope of the simulation is limited to investigating the effectiveness of LSH in pulling together the data elements in each non-overlapping circular area on the surface of the sphere.

4. For each query vector corresponding to the axis of the hypercone, retrieve candidate "nearby" vectors. Compute the true-positive and false-positive rates.
5. Repeat with dimension $D \in \{30, 300, 3000\}$.

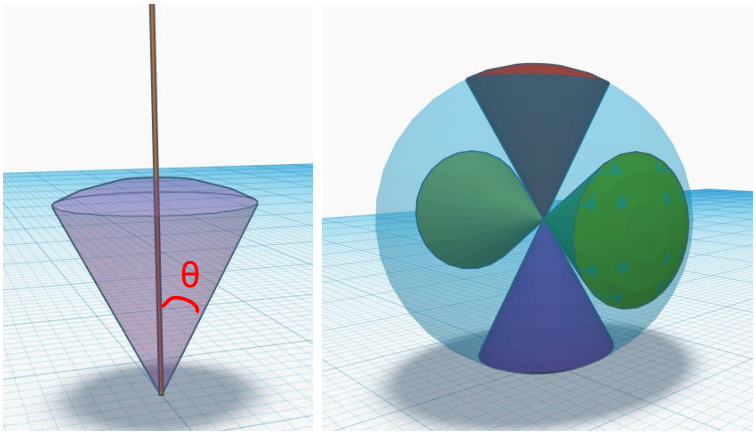


Fig. C.1. The image on the right shows four non-overlapping hypercones in 3D. Here, θ is the angle of each hypercone. All non-overlapping hypercones are positioned at a common origin.

For d_1 set to 15°, Table C.1 shows the values of d_2 and the number of hash function calls as a function of c,[2] the approximating factor mentioned earlier in our brief explanation of how LSH works. The LSH parameters for the desired true positive rate (desired p_1) and the false positive rate (desired p_2) are set to 0.99 and 0.001, respectively. The numbers of AND and OR constructions are calculated by solving r and b, respectively, in the two simultaneous equations:

$$\text{desired } p_1 = 1 - (1 - p_1^r)^b \tag{C.1}$$

$$\text{desired } p_2 = 1 - (1 - p_2^r)^b \tag{C.2}$$

As for the values of p_1 and p_2, they depend on the type of hash function used. For hyperplane LSH, they are given: $p_1 = \frac{180-d_1}{180}$ and $p_2 = \frac{180-d_2}{180}$. See Fig. C.2 for an example.

[2] See Section 5.2.5 for a brief review of LSH as well as the description of the following variables used in the rest of this paragraph: c, d_1, d_2, p_1, p_2, the desired p_1, and the desired p_2.

Table C.1.
Number of hash function calls as a function of approximation factor in hyperplane LSH. ($d_1 = 15.0$, desired $p_1 = 0.99$, desired $p_2 = 0.001$)

Approx. Factor (c)	d_2	# of ANDs	# of ORs	# of hash calls
1.5	22.5	181	31840155	5763068055
2.0	30.0	88	9739	857032
2.5	37.5	57	654	37278
3.0	45.0	41	160	6560
3.5	52.5	32	72	2304

From Table C.1, we see that although smaller approximation factor leads to a tighter transition region, $[d_1, d_2]$, it can result in dramatic increase in the number of hash function calls[3]. For example, with $c = 2.5$, the number of hash calls to index one feature vector is 37278, which is a large number. Nevertheless, improvements such as multi-probe LSH [59] and cross-polytope LSH [60] have been developed to reduce the number of hash calls needed.

In general, regardless of what variant of LSH is used, the number of hash calls needed in order to achieve the desired performance guarantee will only increase as the approximation factor approaches to 1.0. Other variants of LSH, such as LSH Forest [61], can save considerable computation by focusing on obtaining high precision performance while ignoring recall performance. Such strategy works well for applications that want to quickly retrieve just a few nearby neighbors per a given query.

Table C.2 summarizes the result of our simulation study whose experimental protocol is given at the beginning of this section. For the experimental parameters, we let $d_1 = 15.0$, $c = 2.5$ (ie., $d_2 = 37.5$), and $\theta = 37.5$ (ie., $\theta = d_2$). From this table, we observe that LSH's performance in terms of true-positive and false-positive rates

[3]This dramatic increase is mainly due to the large increase in the number of OR constructions (ie., number of hash function calls = product of AND and OR constructions).

Table C.2.
Hyperplane LSH performance as a function of data dimensionality. LSH design parameters: $d_1 = 15°$, $c = 2.5$ ($d_2 = 37.5°$), desired $p_1 = 0.99$, desired $p_2 = 0.001$. Result averaged over 20 hypercones.

Dimensions	30	300	3000
TPR (avg)	1.000	0.999	1.000
TPR (std)	5.357e-04	7.392e-04	3.996e-04
FPR (avg)	0.007	0.007	0.007
FPR (std)	2.561e-04	3.974e-04	2.842e-04
FPR-trans (avg)	0.267	0.267	0.266
FPR-trans (std)	9.992e-03	1.551e-02	1.109e-02
FPR-far (avg)	0.000	0.000	0.000
FPR-far (std)	0.000e+00	0.000e+00	0.000e+00

(TPR and FPR) do not depend on the data dimensionality. In all three cases of data dimensionality, the true-positive and false-positive rates are indeed within the design parameter. Specifically, the TPR values in the second and third rows of the table exceed the desired p_1 value of 0.99 while the FPR-far values in the last two rows of the table stay below the desired p_2 value of 0.001.

Interestingly, we see that all false detection come from the transition region $[d_1, d_2]$ and the average false-detection rate in that region is about 26% for all three dimensionality cases (see rows six and seven of Table C.2). This result agrees with the theoretical result shown in figures Fig. C.2 and Fig. C.3. Specifically, Fig. C.3 shows an average hash collision probability of 0.2648 in the transition region. We also summarize the overall false-positive rate in rows four and five of Table C.2. As for the formulas used for the calculation, they are listed in equations Eq. C.3 to Eq. C.6.

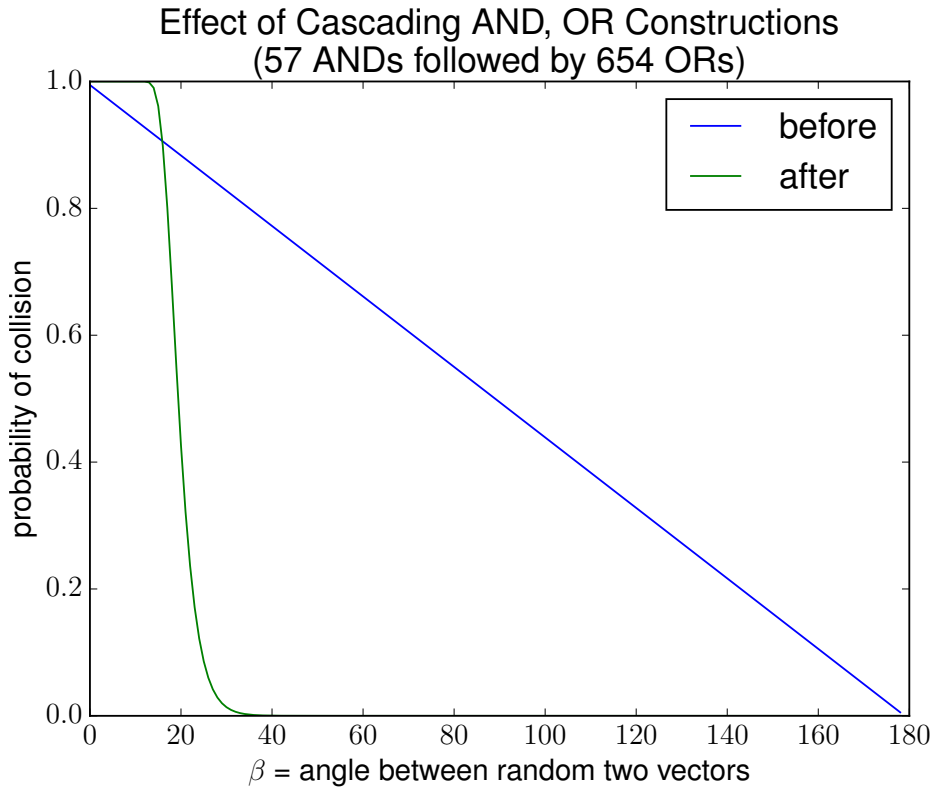


Fig. C.2. Probability of hash collisions for hyperplane LSH before and after AND, OR constructions. The "unamplified" probability is given by the "before" curve $p_{\text{before}}(\beta) = \frac{1-\beta}{180}$. and the "after" probability is described by the equation $p_{\text{after}} = 1 - (1 - p_{\text{before}}^r)^b$, where r is the number of AND constructions and b is the number of OR constructions. The r and b values are calculated by solving two simultaneous equations relating two LSH performance guarantee conditions in Eq. C.1 and Eq. C.2. Here, the LSH design parameters are: $d_1 = 15$, $d_2 = 37.5$, desired $p_1 = 0.99$, desired $p_2 = 0.001$.

$$\text{TPR} = \frac{\text{\# of true positives}}{\text{total \# of positives}} \tag{C.3}$$

$$\text{FPR} = \frac{\text{\# of false positives}}{\text{total \# of negatives}} \tag{C.4}$$

$$\text{FPR-trans} = \frac{\text{\# of false positives within } [d_1, d_2]}{\text{total \# of data within } [d_1, d_2]} \tag{C.5}$$

$$\text{FPR-far} = \frac{\text{\# of false positives} > d_2}{\text{total \# of data} > d_2} \tag{C.6}$$

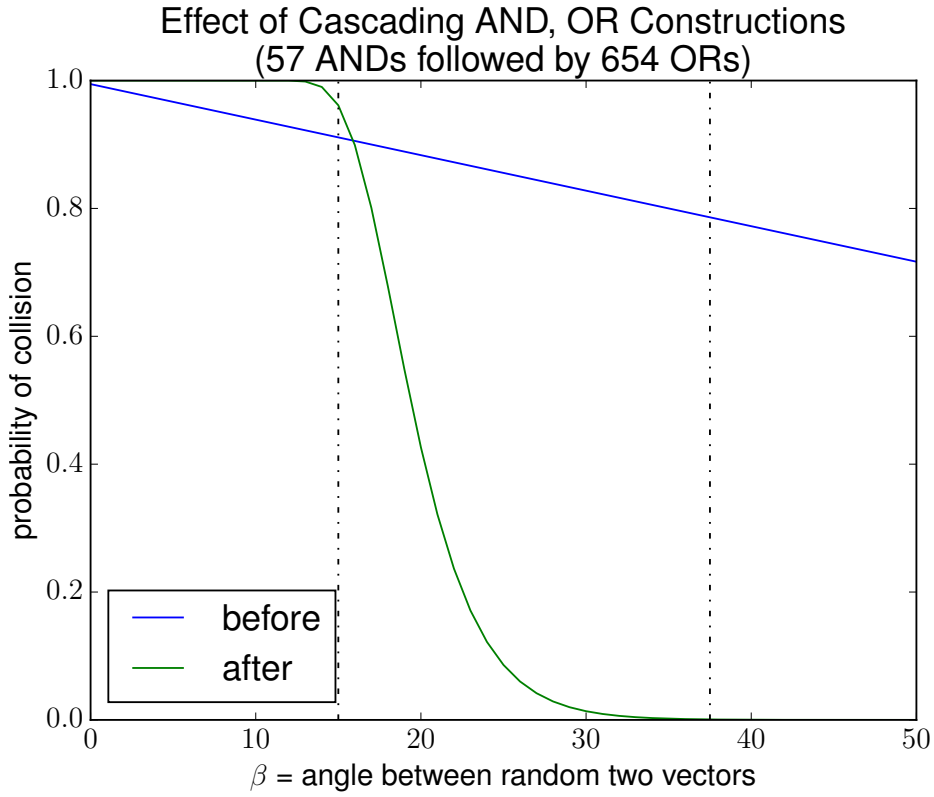


Fig. C.3. Probability of hash collision for hyperplane LSH (see Fig. C.2) zoomed into the transition region: ($\beta \in [d_1, d_2] = [15, 37.5]$). Here, the transition region is marked by the dash vertical lines. In this transition region, the "after" curve averages, in the limit, to 0.2648814...

D. OTSU'S ALGORITHM FOR REAL VALUES

The original Otsu's algorithm [55] takes a collection of discrete values (eg., pixel intensities) and determines the optimal threshold to separate the collection into two classes. The optimality criterion bases on minimizing the "within-class variance", defined as:

$$\sigma_W^2 = w_1 \ \sigma_1^2 + w_2 \ \sigma_2^2 \tag{D.1}$$

Here w_1 and w_2 are the two class probabilities for class 1 and class 2, respectively. And σ_1^2 and σ_2^2 are the two class variances. Otsu shows that to minimize σ_W^2, one can just maximize the "between-class variance", which simplifies to:

$$\sigma_B^2 = w_1 \ w_2 \ (\mu_1 - \mu_2)^2 \tag{D.2}$$

Therefor, to find the optimal threshold, k, Otsu's algorithm searches through all possible values of k (eg., $k \in [0, 1, \ldots, 255]$) and returns the one that maximizes σ_B^2. To calculate σ_B^2, we can use the next four equations to compute w_1, w_2, μ_1 and μ_2 given a particular candidate threshold value k:

$$w_1(k) = \sum_{i=0}^{k} p_i \tag{D.3}$$

$$w_2(k) = 1 - w_1(k) \tag{D.4}$$

$$\mu_1(k) = \sum_{i=0}^{k} i \ \frac{p_i}{w_1(k)} \tag{D.5}$$

$$\mu_2(k) = \sum_{i=k+1}^{L} i \ \frac{p_i}{w_2(k)} \tag{D.6}$$

Here p_i is the probability/frequency of value i occurring in the dataset and L is the largest possible data value (eg., $L = 255$).

In order to adopt Otsu's algorithm for continuous data, we first observe that the optimal threshold is a datum in the dataset. In other words, let the dataset be arranged in a sorted array $\boldsymbol{S} = [x_1, x_2, \ldots, x_n]$, where $x_i \leq x_j, \forall i < j$. Then, we want to find the optimal threshold, x_k, such that class $1 = [x_1, \ldots, x_k]$ and class $2 = [x_{k+1}, \ldots, x_n]$. To do this, we modify Otsu's algorithm as follows:

1. Instead of iterating through all possible threshold values, which can be infinite for continuous data, we just iterate through all data elements in the dataset $\boldsymbol{S}$, starting from the $\boldsymbol{S}[1]$.
2. At each iteration into the $\boldsymbol{S}$ array, we compute w_1, w_2, μ_1 and μ_2 as a function of array index k and use them to compute $\sigma_B^2(k)$: (See Eq. D.2)

$$w_1(k) = \frac{k}{n} \tag{D.7}$$

$$w_2(k) = 1 - w_1(k) \tag{D.8}$$

$$\mu_1(k) = \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{S}[i] \tag{D.9}$$

$$\mu_2(k) = \frac{1}{n-k} \sum_{i=k+1}^{n} \boldsymbol{S}[i] \tag{D.10}$$

3. Return $\boldsymbol{S}[k]$ where $\sigma_B^2(k)$ is the maximum.

Algorithm 2 shows our modified Otsu's algorithm for handling continuous data.

Algorithm 2 Otsu's Algorithm – Modified for Real Values

Input: R = An array of real values.

Output: The optimal threshold to separate R into two groups.

$S \leftarrow$ Sort (R)
$L \leftarrow$ Length (R) // total number of data
$u_0 \leftarrow 0.0; u_1 \leftarrow 0.0;$ // within class means
$s \leftarrow 0.0$ // running sum
$\sigma^2 \leftarrow 0.0$ // max variance
$t \leftarrow 0.0$ // optimal threshold
$s_t \leftarrow$ Sum (R) // global sum
for $k = [0 \ldots L-1]$ **do**
    $n \leftarrow k + 1$ // number of data seen so far
    $s \leftarrow s + S[k]$
    $u_0 \leftarrow \frac{s}{n}$
    $u_1 \leftarrow \frac{(s_t - s)}{(L-n)}$
    $w_0 \leftarrow \frac{n}{L}$ // class1 probability
    $w_1 \leftarrow 1.0 - w_0$ // class2 probability
    $\sigma_b^2 \leftarrow w_0\ w_1\ (u_1 - u_0)^2$
    if $\sigma_b^2 > \sigma^2$ **then**
        $t \leftarrow S[k]$
        $\sigma^2 \leftarrow \sigma_b^2$
    end if
end for
return t

VITA

VITA

Tommy Chang received B.S. in Electrical Engineering and B.S. in Computer Science from University of Maryland in 1997 and 1998, respectively. He joined the Intelligent Systems Division at the National Institute of Standards and Technology in 2000 and received M.S. in Computer Science from Johns Hopkins University in 2008. Since 2011, he has been pursuing Ph.D. in the school of Electrical and Computer Engineering at Purdue University. His research interests include computer vision, cloud computing, and robotics.