

12-17-2010

Nonattribution Properties of JPEG Quantization Tables

Punnya Tuladhar
University of New Orleans

Follow this and additional works at: <https://scholarworks.uno.edu/td>

Recommended Citation

Tuladhar, Punnya, "Nonattribution Properties of JPEG Quantization Tables" (2010). *University of New Orleans Theses and Dissertations*. 1261.
<https://scholarworks.uno.edu/td/1261>

This Thesis is protected by copyright and/or related rights. It has been brought to you by ScholarWorks@UNO with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in University of New Orleans Theses and Dissertations by an authorized administrator of ScholarWorks@UNO. For more information, please contact scholarworks@uno.edu.

Non-attribution Properties of JPEG Quantization Tables

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by

Punya Tuladhar

B.E. Pulchowk Campus, Nepal, 2004

December, 2010

ACKNOWLEDGMENT

I am grateful to Dr. Vassil Roussev, my thesis advisor, for providing me topic of this thesis and guiding and instructing me through every step of this research. Without his help, this thesis wouldn't be possible. Also, I would like to show my gratitude to Dr. Golden Richard and Dr. Shengru Tu for being members of the thesis committee.

CONTENTS

<i>List of Figures</i>	-----	<i>iv</i>
<i>Abstract</i>	-----	<i>v</i>
<i>Chapter 1:</i> <i>Introduction</i>	-----	<i>1</i>
<i>Chapter 2:</i> <i>Design of the Study</i>	-----	<i>3</i>
2.1 <i>Structure of JPEG</i>	-----	<i>3</i>
2.2 <i>Source Data</i>	-----	<i>6</i>
<i>Chapter 3:</i> <i>Analysis</i>	-----	<i>9</i>
<i>References</i>	-----	<i>12</i>
<i>Appendices</i>	-----	<i>13</i>
A: <i>FFDBExtractor.java</i>	-----	<i>13</i>
B: <i>FFC4Extractor.java</i>	-----	<i>14</i>
C: <i>hash_calc.sh</i>	-----	<i>16</i>
D: <i>Format.java</i>	-----	<i>16</i>
E: <i>freq.sh</i>	-----	<i>17</i>
F: <i>ExifExtractor.java</i>	-----	<i>17</i>
G: <i>q-table statistics by make</i>	-----	<i>19</i>
H: <i>q-table statistics by model</i>	-----	<i>20</i>
I: <i>q-table statistics by photoshop</i> <i>version</i>	-----	<i>21</i>
J: <i>h-table statistics by make</i>	-----	<i>21</i>
K: <i>h-table statistics by model</i>	-----	<i>22</i>
L: <i>h-table statistics by photoshop</i> <i>version</i>	-----	<i>23</i>
<i>Vita</i>	-----	<i>24</i>

LIST OF FIGURES

<i>1 JPEG File Organization</i>	-----	<i>4</i>
<i>2 ffdb_hash</i>	-----	<i>7</i>
<i>3 ffdb_hash_formatted</i>	-----	<i>7</i>
<i>4 ffdb_hash_freq</i>	-----	<i>7</i>
<i>5 data_set_final</i>	-----	<i>8</i>
<i>6 ffdb_data_freq</i>	-----	<i>8</i>
<i>7 empirical_cdf</i>	-----	<i>9</i>
<i>8 freq_vs_class</i>	-----	<i>10</i>
<i>9 distribution_unique</i>	-----	<i>10</i>

ABSTRACT

In digital forensics, source camera identification of digital images has drawn attention in recent years. An image does contain information of its camera and/or editing software somewhere in it. But the interest of this research is to find manufacturers (henceforth will be called make and model) of a camera using only the header information, such as quantization table and huffman table, of the JPEG encoding.

Having done a research on around 110, 000 images, we reached to state that *"For all practical purposes, using quantization and huffman tables alone to predict a camera make and model isn't a viable approach"*. We found no correlation between quantization and huffman tables of images and makes of camera. Rather, quantization or huffman table is determined by the quality factors like resolution, RGB values, intensity etc. of an image and standard settings of the camera.

KEYWORDS:

identification of camera, huffman table, quantization table, jpeg exif data, correlation of camera and quantization

CHAPTER 1: INTRODUCTION

Attribution is a critical component of forensic analysis both in the physical and the digital world. One particular attribution problem, source camera identification for digital images, has received quite a bit of attention in recent years. Indeed, a number of research results point rather unambiguously that it is possible to reliably identify the source camera based on artifacts that are a function of the inherent sensor imperfections of each camera.

A related problem is that of source camera *model* identification: given an image, identify the model and/or manufacturer of the source camera. It is increasingly becoming accepted wisdom that a camera *model* fingerprint could be derived using only the header information, such as the quantization and Huffman tables, of the JPEG encoding. The underlying assumption is that camera manufacturers produce unique implementations for each camera model (or at least each line of camera models) and that such uniqueness can be readily observed from the basic parameters of the image. The idea is conceptually similar to an approach routinely used by network security tools to map out network services.

Farid's original study of images from 204 cameras that started this line of work. This was followed up by a large-scale statistical study which concluded that *"while the JPEG quantization table is clearly not unique, it (paired with the image resolution) is reasonably effective at narrowing the source of an image to a single camera make and model or to a*

small set of possible cameras". Some tools, such as JPEGsnoop maintain a collection of "camera fingerprints".

Our own experience in attempting to apply a similar methodology on a real-world sample ran into serious problems. In our work, we present an empirical study of ~110,000 JPEG quantization tables (Q-tables) that contradicts the conclusions by previous work. Our inquiry was- Given a set of JPEG, what is the likelihood that we can correctly identify make/model based on Q-tables?

CHAPTER 2: DESIGN OF THE STUDY

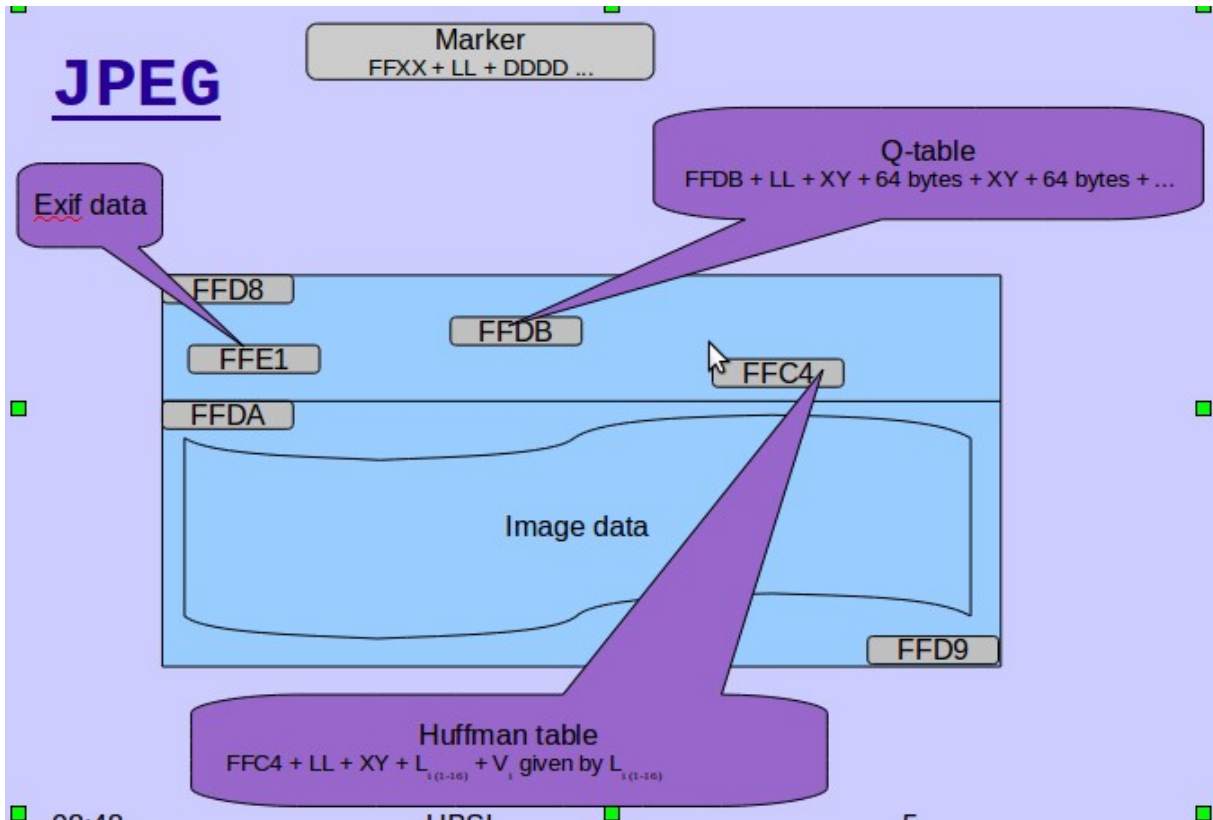
Our research comprised of 4 different phases-

1. Extraction of quantization and huffman tables.
2. Hash Generation
3. Extraction of Exif data.
4. Statistical Analysis

In the first phase, we extracted quantization and huffman tables from 109,774 images downloaded from <http://domex.nps.edu/corp/files/govdocs1/>. The resulted quantization tables were ~204,000 in number whose hash values were generated using md5sum in second phase. Then exif information of images and source camera or software were extracted; which were combined with the hash values and resulted data set was used for statistical analysis.

2.1 Structure of JPEG

Figure below shows the file organization of JPEG format. JPEG [Joint Photographic Experts Group] or jpg refers to a standard organization, a method of file compression, and sometimes file format. The JPEG File Interchange Format [JFIF] is designed to allow files containing JPEG-encoded data streams to be exchanged between otherwise incompatible systems and applications. A JFIF file is basically a JPEG data stream with a few restrictions and identifying marker. Both JPEG and JFIF data are byte streams, always storing 16-bit word values in big endian format. JPEG data in general is stored as stream of blocks, and each block is identified by a marker value.



Fig#1: JPEG File Organization

Every JPEG file starts with binary value 'FFD8', ends with binary value 'FFD9'. There are several binary FFX data in JPEG data, they are called "Marker", and it means the period of JPEG information data. FFD8 means SOI(Start of image), FFD9 means EOI(End of image). Basic format of Marker is below.

FFXX + Length + Data

Details of each marker can be found at [itu-t81.pdf](#) documentation.

If there is a marker like this-

FF C1 00 0C XX XX XX ...

It means this marker(FFC1) has 000C(equal 12)bytes of data including the length [2 bytes] of this length specifier too. It follows only 10 bytes of data after 000C.

In JPEG format, some of markers describe data, then SOS(Start of stream) marker placed. After the SOS marker, JPEG image stream starts and terminates with EOI marker. Real data starts with the marker FFDA.

Exif:

Basically, Exif file format is the same as JPEG file format. Exif inserts some of image/digicam information data and thumbnail image to JPEG in conformity to JPEG specification. Exif data starts with the marker FFE1 and contains information like make, model of camera, resolution, size, dates, software etc.

Quantization Table:

Quantization table is an 8*8 table which controls the quantization quality in a jpeg image. Typically, an image has 2 (or 3) such tables. It is generated by cameras or photo editing softwares for each image which differ depending on both the quality settings of cameras and on the standards which cameras or softwares follow.

In image header, this table starts with the marker FFDB in the format as shown below:

FFDB + LL + 00 + 64 byte data + 01 + 64 byte data + 02 + ...

where the sequence 00, 01, 02 appears like incremented table counters but in fact they are combination of element precision

of data (0 or 1) and destination identifier (0 to 3) at the decoder. LL is total length including length of LL.

Huffman Table:

Huffman coding is used in jpeg compression. Huffman tables generated by cameras and photo editing softwares. This table starts with marker FFC4 in the format as below:

FFC4 + LL + XY + $L_{i(1-16)} + V_i$ given by $L_{i(1-16)}$
LL is total length of all huffman tables, length of length specifier inclusive,
X is 0 or 1 indicating DC table or AC table respectively,
Y is destination identifier (0 to 3) at the decoder,
Each L_i gives number of huffman codes for each of the 16 possible lengths,
 V_i is V_i bytes of codes given by each L_i .
This can be more clear by the formula:
 $LL = 2 + \text{sum of } (17 + m_t), t = 1 \text{ to } n.$
and
 $m_t = \text{sum of } L_i, i = 1-16$

2.2 Source Data

The original data set "ffdb_hash" required to perform statistical analysis was obtained by running hash_calc.sh on quantization tables which were obtained by running FFDBExtractor.java (see appendix). This was then reformatted to "ffdb_hash_formatted" file using Format.java. Using freq.sh, frequency of the hashes from ffdb_hash was computed and saved to "ffdb_hash_freq". Certain number of exif data extracted from all ~110K images, using ExifExtractor.java, were first normalized so that it contained only meaningful words and then saved to "data_set_final".

```
##### ffdb_hash #####
0656ffc785f02947576c3b671c50db9b 000107.00.ff-db
54e464a3c6056f015f7dale86a0086f8 000107.01.ff-db
c30a1cdefc2d7ef4914b2bfdbf33743c 000108.00.ff-db
6307cac23d9b65d1c95cd695a22d1f26 000108.01.ff-db
0656ffc785f02947576c3b671c50db9b 000109.00.ff-db
54e464a3c6056f015f7dale86a0086f8 000109.01.ff-db
0656ffc785f02947576c3b671c50db9b 000110.00.ff-db
54e464a3c6056f015f7dale86a0086f8 000110.01.ff-db
```

Fig#2: ffdb_hash

```
##### ffdb_hash_formatted #####
000107.jpg 00 0656ffc785f02947576c3b671c50db9b
000107.jpg 01 54e464a3c6056f015f7dale86a0086f8
000108.jpg 00 c30a1cdefc2d7ef4914b2bfdbf33743c
000108.jpg 01 6307cac23d9b65d1c95cd695a22d1f26
000109.jpg 00 0656ffc785f02947576c3b671c50db9b
000109.jpg 01 54e464a3c6056f015f7dale86a0086f8
000110.jpg 00 0656ffc785f02947576c3b671c50db9b
000110.jpg 01 54e464a3c6056f015f7dale86a0086f8
```

Fig#3: ffdb_hash_formatted

```
##### ffdb_hash_freq #####
21784 197abd2b5741a96ff5b83121f760ed52
19561 5524f5067b8475f1e7a29fd90c1d98ef
12505 c714773c7a90b0f95a0a714f3e18729e
12331 8a7c44033e511bbf00563b8beea563a6
8717 784d68ba9112308689114a6816c628ce
5588 0e4362ba11351c4808b4e46aeaf94170
5541 3fee0917e01167d5795ed5e4f283e3ad
5539 2417671783bf9c3261707fb94f7b6cc3
```

Fig#4: ffdb_hash_freq

```
##### data_set_final #####  
795043.jpg,0.056,,NIKON,d1x,CS2Mac  
875003.jpg,0.3072,, ,3.0  
983694.jpg,0.4456,,KODAK,z740,CSWin  
800534.jpg,3.24,, ,,  
417505.jpg,1.6875,, ,CSWin  
542256.jpg,0.0702,60%, , ,
```

Fig#5: data_set_final

Then joining 3 data sets – `ffdb_hash_formatted`, `ffdb_hash_freq` and `data_set_final`, a master data set for statistical analysis was created.

```
##### ffdb_data_freq #####  
000ac84aa6fc3ae3c76872e55dc50c1d 164933.jpg 0.1 null null null null 01 388  
000ac84aa6fc3ae3c76872e55dc50c1d 164935.jpg 0.1 null null null null 01 388  
000ac84aa6fc3ae3c76872e55dc50c1d 173364.jpg 0.0 null null null null 01 388  
000ac84aa6fc3ae3c76872e55dc50c1d 195389.jpg 0.3 null null null null 01 388  
000ac84aa6fc3ae3c76872e55dc50c1d 251934.jpg 0.0 null olympus ud600 null 01 388  
000ac84aa6fc3ae3c76872e55dc50c1d 253354.jpg 8.0 null sony dsc-h9 null 01 388  
000ac84aa6fc3ae3c76872e55dc50c1d 255338.jpg 0.2 null panasonic dmc-lz2 null 01 388
```

Fig#6: ffdb_data_freq

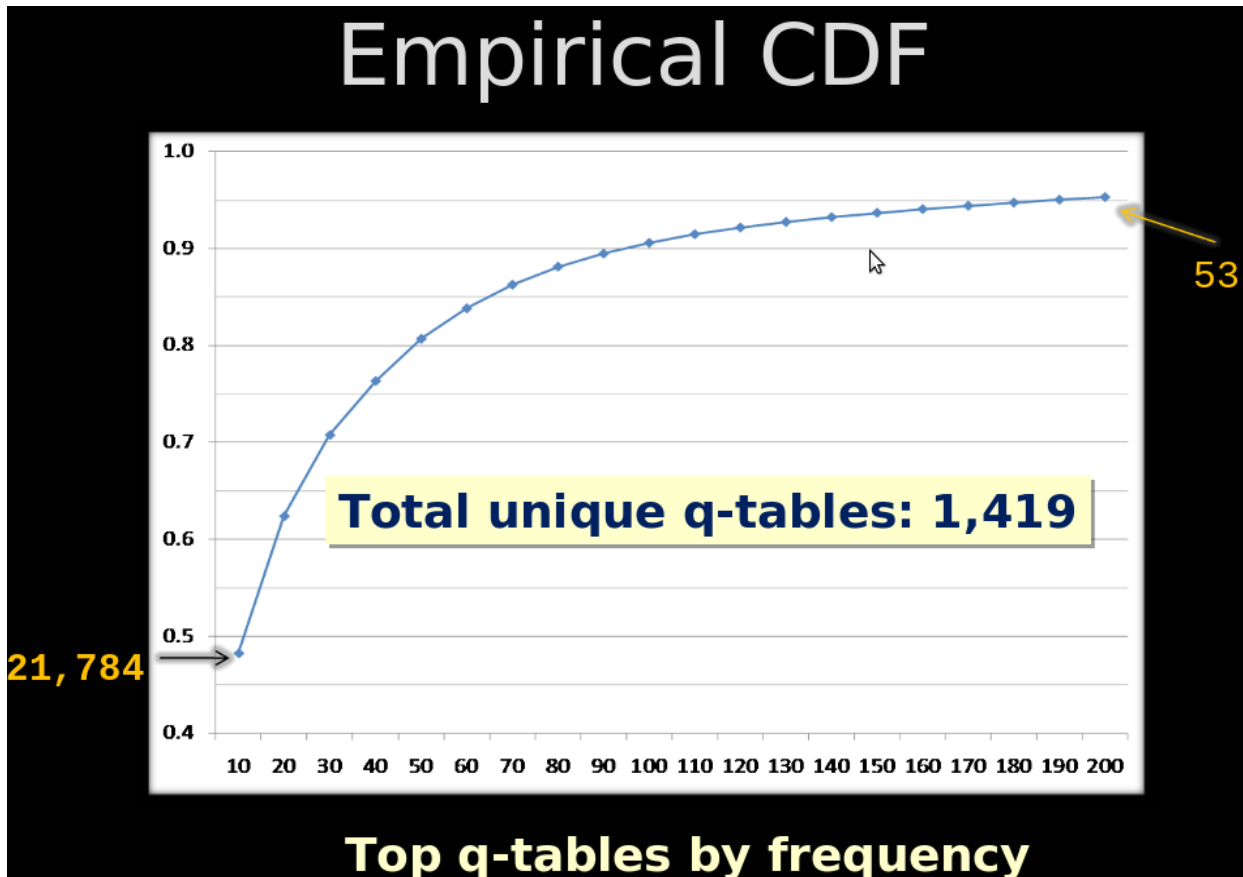
Similarly, `ffc4_hash`, `ffc4_hash_formatted` and `ffc4_hash_freq` were also generated.

CHAPTER 3: ANALYSIS

The file "ffdb_hash_formatted" contains ~204K quantization tables whose distribution looks like -

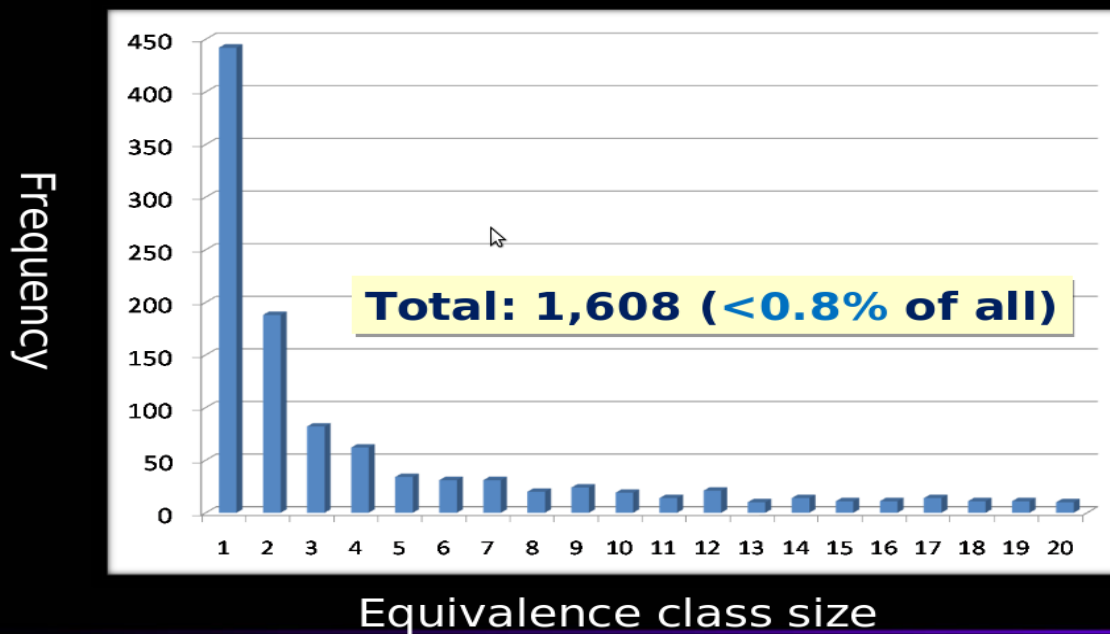
- 9% one table/image
- 89% two tables/image
- 2% three tables/image

Cumulative Distribution Function graph and class size vs frequency graph drawn for obtained results are given below.



Fig#7: empirical_cdf

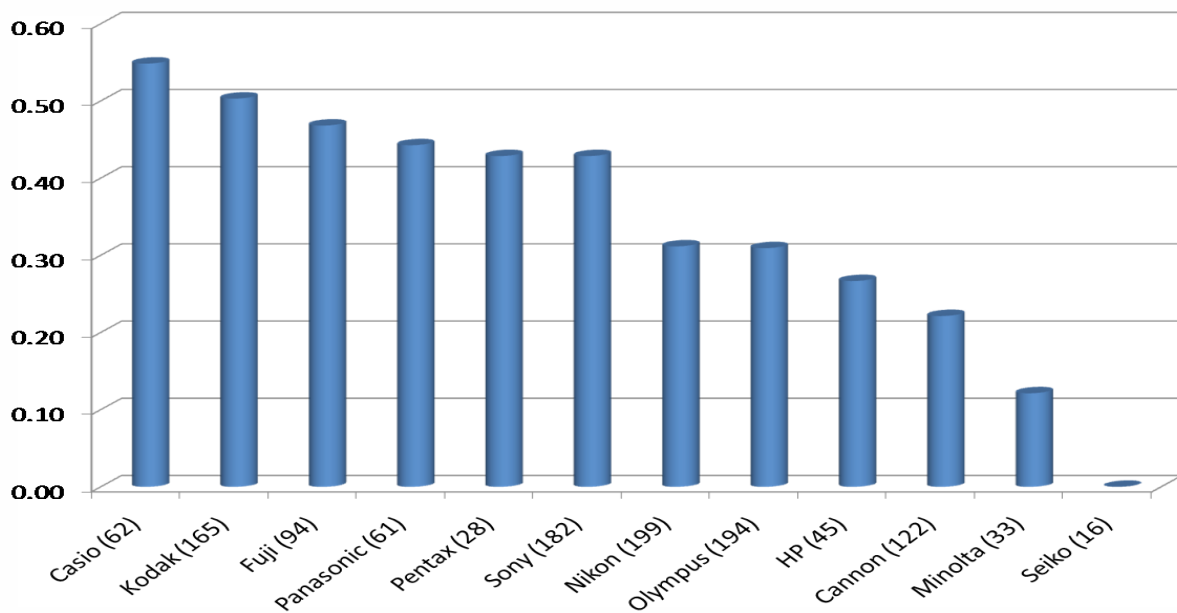
The 'unique' end of the data



17

Fig#8: freq_vs_class

There were 1,419 unique ones whose distribution chart is



Fig#9: distribution_unique

Our preliminary findings are:

- The most frequent table accounted for over 10% of all occurrences.
- The top 10 most frequent tables accounted for almost 50% of all occurrences.
- The top 200 accounted for over 95%.
- The 200th ranked table occurred 53 times (with several different makes).
- There were less than 450 unique and less than 200 duplicate tables.
- There were a total of 1,060 tables(0.5% of total) that occurred 20 times or less (and had any hope as identifying features).

- Considering table uniqueness by make (fraction of tables observed only from the specific make):
 - ◆ 1 make (Seiko) had no make-specific table.
 - ◆ 5 had only 10-30% of their tables as make-specific.
 - ◆ 6 had 40-55%.

For both quantization and huffman tables, statistics based on make, model and photoshop version were calculated (see appendix) with the help of our generated files – ffdb_data_freq and ffc4_data_freq.

Observing these statistics, we can say that high frequency hash values (representing quantization or huffman tables) are associated with different makes, models and software versions.

REFERENCES

- [1] ITU-T81.pdf
- [2] <http://park2.wakwak.com/~tsuruzoh/Computer/Digicams/exif-e.html>
- [3] <http://www.fileformat.info/format/jpeg/egff.htm>
- [4] <http://impulseadventure.com>
- [6] H. Farid, "Digital Image Ballistics from JPEG Quantization", Technical Report, TR2006-583, Dartmouth College, Computer Science.
- [7] H. Farid, "Digital Image Ballistics from JPEG Quantization: A Followup Study", Technical Report, TR2008-638, Dartmouth College, Computer Science.
- [8] Dr. Vassil Roussev, "JPEG Quantization Ballistics Revisited", NeFX 2010.

APPENDICES

A: FFDBExtractor.java

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.RandomAccessFile;
/*
 * version: 3
 * Extracts ff-db (quantization table).
 * QUANT. TABLE FORMAT:
 * ffdb |total-length|00|64bytes-datal01|64bytes-datal02 ...
 */
public class FFDBExtractor {
    private RandomAccessFile jpegSource;

    public static void main(String[] args) throws Exception{
        File folder = new File("/media/corpora/jpg-set");
        File[] listOfFiles = folder.listFiles();

        for (int i = 0; i < listOfFiles.length; i++) {
            String fullName = listOfFiles[i].getName();
            System.out.println(i + ". Extracting ... " + fullName);
            FFDBExtractor parser = new FFDBExtractor("/media/corpora/jpg-set/" + fullName);
            parser.parse(fullName.substring(0, fullName.lastIndexOf(".") + 1)); //include "."
        }
        System.out.println("Finished");
    }

    public FFDBExtractor(String file) {
        try {
            this.jpegSource = new RandomAccessFile(file, "r");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void parse(String fileName) throws IOException{
        this.jpegSource.seek(2);
        int c, j=0, k=0;

        while((c = this.jpegSource.read()) != -1){
            //read marker first
            String first = Integer.toHexString(c);
            String sec = Integer.toHexString(this.jpegSource.read());
            String lastName = "." + first + "-" + sec; //eg .ff-db; . for file extension
            //read 2 bytes of length
            String len1 = Integer.toHexString(this.jpegSource.read());
            String len2 = Integer.toHexString(this.jpegSource.read());
            int length = Integer.parseInt(len1 + len2, 16);
        }
    }
}
```

```

        //if data stream started , end parsing
        if(first.equals("ff") && sec.equals("da")){
            break;
        } else if(first.equals("ff") && sec.equals("db")){
            int times = length/64; //size of quantization table = 64 bytes
            for(int n = 0; n < times; n++){
                FileOutputStream fosdb = new FileOutputStream("ffdb/" + fileName +
String.format("%02d", j++) + lastName);
                //skip 1 byte which is quantization table counter.
                this.jpegSource.read();
                for(int i = 0; i < 64; i++){
                    fosdb.write(this.jpegSource.read());
                }

                fosdb.close();
                fosdb = null;
            }
        } else {
            Long current = this.jpegSource.getFilePointer();
            this.jpegSource.getChannel().position(current + length - 2);
        }
    }

    this.jpegSource.close();
    this.jpegSource = null;
}
}

```

B: FFC4Extractor.java

```

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.RandomAccessFile;
/*
 * version: 3
 * Extracts ff-c4 (huffman table).
 * HUFF TABLE FORMAT:
 * ffc4|total-length|lac-dcl16bytes-length|data-llac-dcl16bytes-length|datal-ll... where
 * 16bytes-length: 11, 12, ... 116
 * data-l: val of 11 + val of 12 + ... + val of 116
 */
public class FFC4Extractor {
    private RandomAccessFile jpegSource;

    public static void main(String[] args) throws Exception{
        File folder = new File("/media/corpora/jpg-set");
        File[] listOfFiles = folder.listFiles();
    }
}

```

```

for (int i = 0; i < listOfFiles.length; i++) {
    String fullName = listOfFiles[i].getName();
    System.out.println(i + ". Extracting ... " + fullName);
    FFC4Extractor parser = new FFC4Extractor("/media/corpora/jpg-set/" + fullName);
        parser.parse(fullName.substring(0, fullName.lastIndexOf(".") + 1)); //include "."
    }

    System.out.println("Finished");
}

public FFC4Extractor(String file) {
    try {
        this.jpegSource = new RandomAccessFile(file, "r");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void parse(String fileName) throws IOException{
    this.jpegSource.seek(2);
    int c, j=0, k=0;

    while((c = this.jpegSource.read()) != -1){
        //read 2 bytes marker
        String first = Integer.toHexString(c);
        String sec = Integer.toHexString(this.jpegSource.read());
        String lastName = "." + first + "-" + sec; //eg .ff-c4; . as file extension

        //read 2 bytes length
        String len1 = Integer.toHexString(this.jpegSource.read());
        String len2 = Integer.toHexString(this.jpegSource.read());
        int length = Integer.parseInt(len1 + len2, 16);

        //if data stream started , end parsing
        if(first.equals("ff") && sec.equals("da")){
            break;
        } else if (first.equals("ff") && sec.equals("c4")){
            long total = 2; //2bytes-total-length which is ffc4total-length
            do{
                FileOutputStream fosc4 = new FileOutputStream("ffc4/" + fileName +
String.format("%02d", k++) + lastName);
                //skip 1 byte which is huffman table counter.
                this.jpegSource.read();

                //read next 16 bytes which give total # of huffman codes
                int subTotal = 0;
                for(int h = 0; h < 16; h++){
                    String len = Integer.toHexString(this.jpegSource.read());
                    subTotal += Integer.parseInt(len, 16);
                }
            }

```

```

        //extract codes
        for(int i = 0; i < subTotal; i++){
            fosc4.write(this.jpegSource.read());
        }

        fosc4.close();
        fosc4 = null;

        total += 17 + subTotal; // 1byte-counter + 16bytes-length
    }while(length > total);
} else {
    Long current = this.jpegSource.getFilePointer();
    this.jpegSource.getChannel().position(current + length - 2);
}
}

this.jpegSource.close();
this.jpegSource = null;
}}

```

C: hash_calc.sh

```

##### hash_calc.sh #####
#!/bin/sh
echo "Generating hash ..."
for f in *.*
do
    md5sum $f >> ffdb_hash
done
echo "Done."

```

D: Format.java

```

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.RandomAccessFile;

/*
 * cmd: java Format ffdb_hash
 * input : hash *.xx.ff-db
 * output: x.jpg xx hash
 *
 */

public class Format {
    RandomAccessFile jpegSource = null;
    FileOutputStream fosdb = null;

    public Format(String name){
        try {
            jpegSource = new RandomAccessFile(name, "r");

```

```

        fosdb = new FileOutputStream(name + "_formatted");
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

public static void main(String[] args){
    Format hc = new Format(args[0]);
    System.out.println("Re-formatting ...");
    hc.reFormat();
    System.out.println("Done.");
}

private void reFormat(){
    try {
        String line = null;

        while((line = jpegSource.readLine()) != null){
            String[] parts = line.split(" ");
            String[] segs = parts[1].split("\\.");

            String hash_freq = segs[0] + ".jpg " + segs[1] + " " + parts[0] + "\n";
            fosdb.write(hash_freq.getBytes());
        }

        jpegSource.close();
        jpegSource = null;
        fosdb.close();
        fosdb = null;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

E: freq.sh

```

##### freq.sh #####
sort | awk '{print $1}' | awk '{c[$1]++;} END{for(i in c){printf "%4d\t%s\n",
c[i], i}}' | sort -rn

```

F: ExifExtractor.java

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

/* command:-

```

```

* exiftool -make -model -xresolution -yresolution -resolutionunit -compressedbitsperpixel
* -bitspersample -compression -datetimeoriginal -orientation -quality 005408.jpg
*/

public class ExifExtractor {
    public static void main(String args[]) {
        String s = null;
        BufferedReader stdInput;
        Process p = null;
        String cmd;

        File folder = new File("/media/corpora/jpg-set");
        File[] listOfFiles = folder.listFiles();
        FileOutputStream fosdb = null;

        try {
            fosdb = new FileOutputStream("exif_data");
        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        }

        for (int i = 0; i < listOfFiles.length; i++) {
            String make = " ", model = " ", xres = " ", yres = " ", resunit = " ", bitspp = " ",
            bitsps = " ", comp = " ", date = " ", orient = " ", qly = " ";

            try {
                String fullName = listOfFiles[i].getName();
                cmd = "ExifTool/exiftool -make -model -xresolution -yresolution -resolutionunit
-compressedbitsperpixel" + " -bitspersample -compression -modifydate -quality /media/corpora/jpg-set/" +
fullName;

                p = Runtime.getRuntime().exec(cmd);
                System.out.println(i + ". Extracting ... " + fullName);
                stdInput = new BufferedReader(new InputStreamReader(p.getInputStream()));

                while ((s = stdInput.readLine()) != null) {
                    String[] parts = s.split(": ");
                    if (parts[0].equals("Make")") {
                        if(parts.length > 1)
                            make = parts[1];
                    } else if (parts[0].equals("Camera Model Name")") {
                        if(parts.length > 1)
                            model = parts[1];
                    } else if (parts[0].equals("X Resolution")") {
                        if(parts.length > 1)
                            xres = parts[1];
                    } else if (parts[0].equals("Y Resolution")") {
                        if(parts.length > 1)
                            yres = parts[1];
                    } else if (parts[0].equals("Resolution Unit")") {
                        if(parts.length > 1)
                            resunit = parts[1];
                    }
                }
            }
        }
    }
}

```



```

        } else if (parts[0].equals("Compressed Bits Per Pixel    ")) {
            if(parts.length > 1)
                bitspp = parts[1];
        } else if (parts[0].equals("Bits Per Sample            ")) {
            if(parts.length > 1)
                bitsps = parts[1];
        } else if (parts[0].equals("Compression              ")) {
            if(parts.length > 1)
                comp = parts[1];
        } else if (parts[0].equals("Modify Date              ")) {
            if(parts.length > 1)
                date = parts[1].split(" ")[0];
        } else if (parts[0].equals("Orientation              ")) {
            if(parts.length > 1)
                orient = parts[1];
        } else if (parts[0].equals("Quality                  ")) {
            if(parts.length > 1)
                qly = parts[1];
        }
    }

    String exifData = fullName + "|" + make + "|" + model + "|" + xres + "|" + yres +
    "|" +
    resunit + "|" + bitspp + "|" + bitsps + "|" + comp + "|" + date + "|" +
    orient + "|" + qly + "\n";

    fosdb.write(exifData.getBytes());

    if(null != p)
        p.destroy();
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(-1);
    }
}

try {
    fosdb.close();
    fosdb = null;
} catch (IOException e) {
    e.printStackTrace();
}

System.exit(0);
}
}

```

G: q-table statistics by make

```

21784 197abd2b5741a96ff5b83121f760ed52 sony
21784 197abd2b5741a96ff5b83121f760ed52 seiko

```

21784 197abd2b5741a96ff5b83121f760ed52 panasonic
 21784 197abd2b5741a96ff5b83121f760ed52 olympus
 21784 197abd2b5741a96ff5b83121f760ed52 null
 21784 197abd2b5741a96ff5b83121f760ed52 nikon
 21784 197abd2b5741a96ff5b83121f760ed52 minolta
 21784 197abd2b5741a96ff5b83121f760ed52 kodak
 21784 197abd2b5741a96ff5b83121f760ed52 hewlett-packard
 21784 197abd2b5741a96ff5b83121f760ed52 fujifilm
 21784 197abd2b5741a96ff5b83121f760ed52 casio
 21784 197abd2b5741a96ff5b83121f760ed52 canon
 19561 5524f5067b8475f1e7a29fd90c1d98ef sony
 19561 5524f5067b8475f1e7a29fd90c1d98ef seiko
 19561 5524f5067b8475f1e7a29fd90c1d98ef panasonic
 19561 5524f5067b8475f1e7a29fd90c1d98ef olympus
 19561 5524f5067b8475f1e7a29fd90c1d98ef null
 19561 5524f5067b8475f1e7a29fd90c1d98ef nikon
 19561 5524f5067b8475f1e7a29fd90c1d98ef minolta
 19561 5524f5067b8475f1e7a29fd90c1d98ef kodak
 19561 5524f5067b8475f1e7a29fd90c1d98ef hewlett-packard
 19561 5524f5067b8475f1e7a29fd90c1d98ef fujifilm
 19561 5524f5067b8475f1e7a29fd90c1d98ef casio

H: q-table statistics by model

21784 197abd2b5741a96ff5b83121f760ed52 sony dsc-w7
 21784 197abd2b5741a96ff5b83121f760ed52 sony dsc-t100
 21784 197abd2b5741a96ff5b83121f760ed52 sony dsc-p92
 21784 197abd2b5741a96ff5b83121f760ed52 sony dsc-p100
 21784 197abd2b5741a96ff5b83121f760ed52 sony dsc-m1
 21784 197abd2b5741a96ff5b83121f760ed52 sony dsc-h9
 21784 197abd2b5741a96ff5b83121f760ed52 sony cybershot
 21784 197abd2b5741a96ff5b83121f760ed52 sony cd-mavica
 21784 197abd2b5741a96ff5b83121f760ed52 seiko photopc-850z
 21784 197abd2b5741a96ff5b83121f760ed52 panasonic dmc-lc50
 21784 197abd2b5741a96ff5b83121f760ed52 panasonic dmc-fz20
 21784 197abd2b5741a96ff5b83121f760ed52 olympus x550
 21784 197abd2b5741a96ff5b83121f760ed52 olympus u10d
 21784 197abd2b5741a96ff5b83121f760ed52 olympus e-10
 21784 197abd2b5741a96ff5b83121f760ed52 olympus c960z
 21784 197abd2b5741a96ff5b83121f760ed52 olympus c900z
 21784 197abd2b5741a96ff5b83121f760ed52 olympus c5060wz
 21784 197abd2b5741a96ff5b83121f760ed52 olympus c5050z
 21784 197abd2b5741a96ff5b83121f760ed52 olympus c3040z
 21784 197abd2b5741a96ff5b83121f760ed52 null null
 21784 197abd2b5741a96ff5b83121f760ed52 nikon e995
 21784 197abd2b5741a96ff5b83121f760ed52 nikon e950
 21784 197abd2b5741a96ff5b83121f760ed52 nikon e8700
 21784 197abd2b5741a96ff5b83121f760ed52 nikon e5700
 21784 197abd2b5741a96ff5b83121f760ed52 nikon e4600
 21784 197abd2b5741a96ff5b83121f760ed52 nikon e3100
 21784 197abd2b5741a96ff5b83121f760ed52 nikon e2000

21784 197abd2b5741a96ff5b83121f760ed52 nikon d50
21784 197abd2b5741a96ff5b83121f760ed52 nikon d200
21784 197abd2b5741a96ff5b83121f760ed52 nikon d1x
21784 197abd2b5741a96ff5b83121f760ed52 nikon d1h
21784 197abd2b5741a96ff5b83121f760ed52 nikon d100
21784 197abd2b5741a96ff5b83121f760ed52 nikon d1

I: q-table statistics by photoshop version

21784 197abd2b5741a96ff5b83121f760ed52 null
21784 197abd2b5741a96ff5b83121f760ed52 doc
21784 197abd2b5741a96ff5b83121f760ed52 cswin
21784 197abd2b5741a96ff5b83121f760ed52 7.0
21784 197abd2b5741a96ff5b83121f760ed52 5.2
21784 197abd2b5741a96ff5b83121f760ed52 5.0
21784 197abd2b5741a96ff5b83121f760ed52 4.0
21784 197abd2b5741a96ff5b83121f760ed52 3.0
19561 5524f5067b8475f1e7a29fd90c1d98ef null
19561 5524f5067b8475f1e7a29fd90c1d98ef doc
19561 5524f5067b8475f1e7a29fd90c1d98ef cswin
19561 5524f5067b8475f1e7a29fd90c1d98ef 7.0
19561 5524f5067b8475f1e7a29fd90c1d98ef 5.2
19561 5524f5067b8475f1e7a29fd90c1d98ef 5.0
19561 5524f5067b8475f1e7a29fd90c1d98ef 4.0
19561 5524f5067b8475f1e7a29fd90c1d98ef 3.0

J: h-table statistics by make

80402 50a73d7013e9803e3b20888f8fcafb15 vivitar
80402 50a73d7013e9803e3b20888f8fcafb15 sound
80402 50a73d7013e9803e3b20888f8fcafb15 sony
80402 50a73d7013e9803e3b20888f8fcafb15 seiko
80402 50a73d7013e9803e3b20888f8fcafb15 samsung
80402 50a73d7013e9803e3b20888f8fcafb15 research
80402 50a73d7013e9803e3b20888f8fcafb15 pentax
80402 50a73d7013e9803e3b20888f8fcafb15 panasonic
80402 50a73d7013e9803e3b20888f8fcafb15 panaso
80402 50a73d7013e9803e3b20888f8fcafb15 olympus
80402 50a73d7013e9803e3b20888f8fcafb15 null
80402 50a73d7013e9803e3b20888f8fcafb15 nikon
80402 50a73d7013e9803e3b20888f8fcafb15 newssoft
80402 50a73d7013e9803e3b20888f8fcafb15 msm6100
80402 50a73d7013e9803e3b20888f8fcafb15 motorola
80402 50a73d7013e9803e3b20888f8fcafb15 minolta
80402 50a73d7013e9803e3b20888f8fcafb15 logitech
80402 50a73d7013e9803e3b20888f8fcafb15 kyocera
80402 50a73d7013e9803e3b20888f8fcafb15 konica
80402 50a73d7013e9803e3b20888f8fcafb15 kodak
80402 50a73d7013e9803e3b20888f8fcafb15 jvc
80402 50a73d7013e9803e3b20888f8fcafb15 hp
80402 50a73d7013e9803e3b20888f8fcafb15 hewlett-packard

80402 50a73d7013e9803e3b20888f8fcafb15 fujifilm
 80402 50a73d7013e9803e3b20888f8fcafb15 fuji
 80402 50a73d7013e9803e3b20888f8fcafb15 casio
 80402 50a73d7013e9803e3b20888f8fcafb15 canon
 80402 50a73d7013e9803e3b20888f8fcafb15 asahi
 42616 1274c9b90c56e7f0e589412cbad79aa2 vivitar
 42616 1274c9b90c56e7f0e589412cbad79aa2 sound
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony
 42616 1274c9b90c56e7f0e589412cbad79aa2 seiko
 42616 1274c9b90c56e7f0e589412cbad79aa2 samsung
 42616 1274c9b90c56e7f0e589412cbad79aa2 research
 42616 1274c9b90c56e7f0e589412cbad79aa2 pentax
 42616 1274c9b90c56e7f0e589412cbad79aa2 panasonic
 42616 1274c9b90c56e7f0e589412cbad79aa2 panaso
 42616 1274c9b90c56e7f0e589412cbad79aa2 olympus
 42616 1274c9b90c56e7f0e589412cbad79aa2 null

K: h-table statistics by model

80402 50a73d7013e9803e3b20888f8fcafb15 canon ixy--50
 80402 50a73d7013e9803e3b20888f8fcafb15 canon ixy--200a
 80402 50a73d7013e9803e3b20888f8fcafb15 canon ixy
 80402 50a73d7013e9803e3b20888f8fcafb15 canon ixus-400
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos--rebel-xt
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos--rebel
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-kiss
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-d60
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-5d
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-40d
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-350d
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-30d
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-20d
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-1d-mark-ii
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-1d
 80402 50a73d7013e9803e3b20888f8fcafb15 canon eos-10d
 80402 50a73d7013e9803e3b20888f8fcafb15 asahi optio-330
 42616 1274c9b90c56e7f0e589412cbad79aa2 vivitar vivicam-3715
 42616 1274c9b90c56e7f0e589412cbad79aa2 sound null
 42616 1274c9b90c56e7f0e589412cbad79aa2 sound .
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony picturegear
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony mvc-cd500
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony mavica
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony fdmavica
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony dslr-a100
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony dsc-w7
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony dsc-w1
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony dsc-v3
 42616 1274c9b90c56e7f0e589412cbad79aa2 sony dsc-v1

L: h-table statistics by photoshop version

80402	50a73d7013e9803e3b20888f8fcfb15	null
80402	50a73d7013e9803e3b20888f8fcfb15	ele5.0
80402	50a73d7013e9803e3b20888f8fcfb15	ele2.0
80402	50a73d7013e9803e3b20888f8fcfb15	doc
80402	50a73d7013e9803e3b20888f8fcfb15	cswin
80402	50a73d7013e9803e3b20888f8fcfb15	csmac
80402	50a73d7013e9803e3b20888f8fcfb15	cs3mac
80402	50a73d7013e9803e3b20888f8fcfb15	cs2win
80402	50a73d7013e9803e3b20888f8fcfb15	cs2mac
80402	50a73d7013e9803e3b20888f8fcfb15	alb2.0
80402	50a73d7013e9803e3b20888f8fcfb15	7.0
80402	50a73d7013e9803e3b20888f8fcfb15	5.2
80402	50a73d7013e9803e3b20888f8fcfb15	5.0
80402	50a73d7013e9803e3b20888f8fcfb15	4.0
80402	50a73d7013e9803e3b20888f8fcfb15	3.0
42616	1274c9b90c56e7f0e589412cbad79aa2	null
42616	1274c9b90c56e7f0e589412cbad79aa2	ele5.0
42616	1274c9b90c56e7f0e589412cbad79aa2	ele2.0
42616	1274c9b90c56e7f0e589412cbad79aa2	doc
42616	1274c9b90c56e7f0e589412cbad79aa2	cswin
42616	1274c9b90c56e7f0e589412cbad79aa2	csmac
42616	1274c9b90c56e7f0e589412cbad79aa2	cs3mac
42616	1274c9b90c56e7f0e589412cbad79aa2	cs2win

VITA

The author was born in Kathmandu, Nepal. He obtained his bachelor's degree in computer engineering from Pulchowk Campus, Nepal in 2004. He joined the University of New Orleans to pursue masters degree in computer science in Spring 2009, and worked for Dr. Vassil Roussev as a research assistant.