CS 476 Programming Assignment 1

Joshua Gleason

September 9, 2010

In this project I wrote two simple image processing programs. The first is designed to change the spacial resolution of a 256 x 256 image to 128 x 128, 64 x 64, and 32 x 32. It then needs to resize the image for viewing purposes to determine how much quality was lost in the spacial reduction. The other program performs quantization on an image, changing the number of bits per pixel from 8 to 6, 4, and 2. In other words, the number colors in the image is reduced from 256 to 64, 16, and 4.

1 Method

The reduction of an images spacial resolution can be done using a number of methods. The method I choose to use was to average the pixels in the area which was to be sampled down, then use that average as the pixel value in the reduced image. I choose this method over simply sampling, because much more information is retained, and this also helps reduce the amount of noise in the image.

For the second program, I wanted to quantize the image, so first I had to sample the develop an intensity transformation function which would map the original 256 colors into a smaller domain. Knowing that $[0, 255] \rightarrow [0, f_{max}]$ where f_{max} is the maximum value in the new domain; we are able to solve for a linear intensity transformation $y = ax + b$ by solving the following system.

$$\begin{aligned} 0 &= a(0) + b \\ f_{max} &= a(255) + b \end{aligned}$$

Solving for a and b

$$\begin{aligned} a &= \frac{f_{max}}{255} \\ b &= 0 \end{aligned}$$

2 Implementation

I choose to implement this program using the OpenCV library. I choose to do this because it saved time, and I didn't see any reason why I should risk building a new Image class which could potentially introduce bugs, when a very well build library already exists.

For the first program, rather than simple shrink the image, and then enlarge it again, I decided to improve the algorithm by calculating everything without actually shrinking the image. To do this I first calculated the size of the area in the original image which would be mapped to a single pixel. Next I iterated across each region and calculated the average, then on the new image filled that region in with the average value. This method is slightly faster than shrinking the image, then enlarging it because it does not require the extra memory allocation for the smaller image.

For the second program, I mapped each pixel into the new domain (where $f_{max} = 3, 15,$ or 63). I was able to do this by dividing the current value by $\frac{256}{f_{max}+1}$. Dividing by a^{-1} is used here to make up for integer truncation errors and to remove the need for floating point calculations. After this step the image was normalized back to $[0, 255]$ for viewing purposes.

3 Results

For the first program, it is obvious when the dimensionality is reduced to 32 x 32 and 64 x 64. It is much harder to tell that the 128 x 128 image has been reduced by half. This is a rather interesting results, as it shows that even with a substantial loss in memory, for many purposes lower resolution images may be worthwhile. Images tend to take up a lot of space in a computer, reducing the resolution by half, reduces the size of the image to a quarter of it's original, and it does it without losing a great deal of quality.

The second program, it is also obvious that the depth of the image has been reduced in the 2 bit and 4 bit images, but with the 6 bit image it is identical to the original according to my eyes. This brings up another interesting observation. It it possible to reduce the depth of the image to 3 quarters of its original without any visible loss of quality. This quality could also be used to reduce to size of the images. In fact if we were to reduce the size of an image by half, and reduce its depth to 6 bits, the size of the image would reduce to three sixteenths of its original size.

4 Source Code

Listing 1: resample.cc

```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace std;

string err( const char* err )
{
  string retVal = err;
  return err;
}

void spacial( string fileName )
{
  // open image
  cout << "Loading " << fileName << "...";
  IplImage* img = cvLoadImage( fileName.c_str(), 0 );

  if ( img == 0 )
    throw err("Invalid file or image format");
  if ( img->width != 256 || img->height != 256 )
    throw err("Width and height of image must be 256");

  cout << "Complete" << endl;

  // remove file extension
  string extention;
  istringstream sin(fileName);
  getline( sin, fileName, '.' );
  sin >> extention;

  // will be used to construct output file name
  ostringstream sout;

  // create the resized image
  IplImage* resized = cvCreateImage( cvSize( 256, 256 ), IPL_DEPTH_8U, 1 );

  // display original
  cvNamedWindow( "Image" );
  cvShowImage( "Image", img );
  cvWaitKey();

  // size is new resolution (32, 64, or 128) and pix is the size of a block
  int size, pix, offsetX, offsetY, avg;

  // 2^5 2^6 2^7
  for ( int dim = 5; dim < 8; dim++ )
  {
    // calculate the size
    size = static_cast<int>(pow(2,dim));
    pix = 256/size;

    cout << "Changing spacial resolution to " << size << "..." << flush;
```

```
    // change spacial resolution and resize using nearest neighbor
       simultaniously
    for ( int i = 0; i < size; i++ )
      for ( int j = 0; j < size; j++ )
      {
        // location of upper left corner of block
        offsetX = i*pix;
        offsetY = j*pix;

        // get average value in block
        avg = 0;
        for ( int k = 0; k < pix; k++ )
          for ( int l = 0; l < pix; l++ )
            avg += cvGetReal2D( img, offsetX+k, offsetY+l );
        avg /= pix*pix;

        // set all pixels in block to that average
        for ( int k = 0; k < pix; k++ )
          for ( int l = 0; l < pix; l++ )
            cvSetReal2D( resized, offsetX+k, offsetY+l, avg );
      }

    cout << "Complete" << endl;

    // save file
    sout.str("");
    sout << fileName << '_' << size << '.' << extention;

    cout << "Saving " << sout.str() << "..." << flush;
    cvSaveImage(sout.str().c_str(),resized);
    cout << "Complete" << endl;

    // display image
    cvShowImage("Image",resized);
    cvWaitKey();
  }

  cvReleaseImage( &img );
  cvReleaseImage( &resized );
}

int main(int argc, char *argv[])
{
  try
  {
    if ( argc < 2 )
      throw err("Please supply an image filename that is 256x256");
    spacial( argv[1] );
  }
  catch(string err)
  {
    cout << "\n\tError : " << err << endl;
  }

  return 0;
}
```

Listing 2: quantize.cc

```
#include <cv.h>
#include <highgui.h>
#include <iostream>
#include <sstream>
#include "generic.h"

using namespace std;

string err( const char* err )
{
  string retVal = err;
  return err;
}

void quantize( string fileName )
{
  // open image
  cout << "Loading " << fileName << "...";
  IplImage* img = cvLoadImage( fileName.c_str(), 0 );

  if ( img == 0 )
    throw err("Invalid file or image format");

  cout << "Complete" << endl;

  // remove file extension from name for later
  string extention;
  istringstream sin(fileName);
  getline( sin, fileName, '.' );
  sin >> extention;

  // will be used to construct output file name
  ostringstream sout;

  // create the object to hold the quantized image
  IplImage* quantized = cvCreateImage( cvGetSize(img), IPL_DEPTH_8U, 1 );

  // display original
  cvNamedWindow( "Image" );
  cvShowImage( "Image", img );
  cvWaitKey();

  // 2, 4, and 6 bits
  for ( int dim = 2; dim < 8; dim+=2 )
  {
    // calculate the number of colors
    int colors = static_cast<int>( pow(2,dim) );

    // quantize image
    cout << "Quantizing Image..." << flush;
    jdg::quantize( img, quantized, 0, 255, colors );
    cout << "Complete" << endl;

    // save file
    sout.str("");
    sout << fileName << '_' << dim << "bit" << '.' << extention;
```

```
    cout << "Saving " << sout.str() << "..." << flush;
    cvSaveImage(sout.str().c_str(),quantized);
    cout << "Complete" << endl;

    // display image
    cvShowImage("Image",quantized);
    cvWaitKey();
  }

  cvReleaseImage( &img );
  cvReleaseImage( &quantized );
}

int main(int argc, char *argv[])
{
  try
  {
    if ( argc < 2 )
      throw err("Please supply an image filename");
    quantize( argv[1] );
  }
  catch(string err)
  {
    cout << "\n\tError : " << err << endl;
  }

  return 0;
}
```

Listing 3: generic.h

```
#include <cv.h>
#include <highgui.h>

namespace jdg
{
// quantize from range [<min>, <max>] into [0,<vals>] then back
void quantize( IplImage* src, IplImage* dest, int min, int max, int vals )
{
  int width = src->width,
      height = src->height,
      old_range = max-min+1,
      val;
  int step = old_range/vals;

  for ( int i = 0; i < height; i++ )
    for ( int j = 0; j < width; j++ )
    {
      val = static_cast<int>( cvGetReal2D( src, i, j ) ) + min;
      val /= step;
      val = val*(old_range-1)/(vals-1);
      cvSetReal2D( dest, i, j, static_cast<double>(val) );
    }
}
}
```

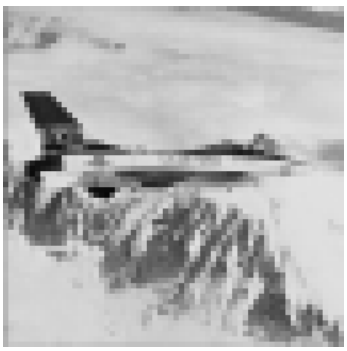
5 Images

(a) Original

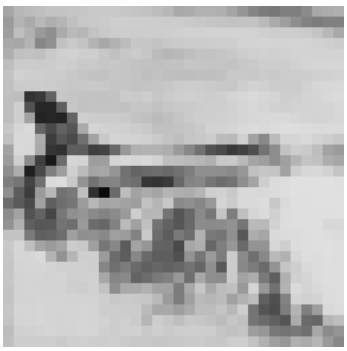
(b) 128x128 at 4 $\frac{\text{bit}}{\text{pix}}$ (12.5% mem)

(c) 128x128 at 6 $\frac{\text{bit}}{\text{pix}}$ (18.8% mem)

(d) 128x128



(e) 64x64



(f) 32x32

(g) 6 bit/pix

(h) 4 bit/pix

(i) 2 bit/pix

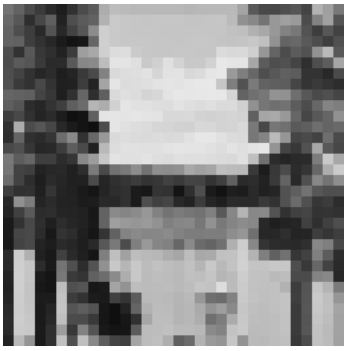
(j) Original

(k) 128x128 at 4 $\frac{\text{bit}}{\text{pix}}$ (12.5% mem)

(l) 128x128 at 6 $\frac{\text{bit}}{\text{pix}}$ (18.8% mem)

(m) 128x128

(n) 64x64



(o) 32x32

(p) 6 bit/pix

(q) 4 bit/pix

(r) 2 bit/pix