Histogram Specifications Operating Under Various Discrete Transformations

Team Members:

Casey Kuball

Glenn Craver

Steve Carr

Wesley Rose

Dawne Flanagan

**Abstract**:  
 Representing the distribution of colors in an image can most commonly be accomplished using a color histogram. This is performed by determining the number of pixels in a given image and associate the colors in those pixels with a fixed list of color ranges that cover the image's color space, as well as the set of all possible colors. This project was designed to breakdown a set of images into different histogram representations as well as breakdown features within the image, given a user specified color space. Using several open source python libraries located on the internet our program performs all of the necessary functions as dictated in the assignment’s requirements. By completing this project our group can now represent images in a variety of histogram specifications, store transformation of the histograms into database files and later extract significant features from the database files.  
  
  
  
**Keywords**:

Python 2.7.3  
 Python Imaging Library (PIL)  
 RGB

YUV

HSV

Color model

Color space

Histogram

Discrete Cosine Transform (DCT)

Discrete Wavelet Transform (DWT)

Sobel Operator

**Introduction**

Terminology

Throughout this report three different color models will be referenced using their acronym, RGB, YUV, HSV, which describes the major components in the color model. Three different methods of image data discrete transformations are discussed; DCT, Sobel, and DWT. When the term ‘API’ is referenced, it is used to denote Application Programming Interface. For this project our team used several third party API’s, such as SciPy and PyWavelets, to implement various requirements for the project

Goal

The goal of this project is to implement a program which will take a set of images as input, and after the user selects a specified color space, will implement the following tasks:

1. Divide the selected color space into 16 bins based on the color pixels in all the image files using the median-cut algorithm. The resulting histogram specification (consisting of color instance boundaries) is written into a file.
2. Given an image, divide it into 8-by-8 regions. Then, for each cell, create a 16 bin color histogram, based on the histogram specification generated in Task I. The output of the form

⟨image id, cell coord, color instance id, value⟩

are written into a file.

1. Given an image, divide it into 8-by-8 image cells and apply 2D-DCT on the three color channels (in the selected color space) of each cell of the image. The program selects the 16 most significant frequency components from each of the three channels. The outputs, of the form

⟨image id, cell coord, channel id, freq bin, value⟩

are written into a file.

1. Given an image, divide it into 8-by-8 image cells. Then create a 16-bin gradient-angle histogram, using the Sobel operator, for each cell in the image. The outputs, of the form

⟨image id, cell coord, channel id, angle bin, value⟩

are written into a file.

1. Given an image, divide it into 8-by-8 image cells. Then create a 16-bin gradient-amplitude histogram, using Sobel operator, for each cell in the image. The outputs, of the form

⟨image id, cell coord, channel id, amplitude bin, value⟩

are written into a file.

1. Given an image, divide it into 8-by-8 regions image cells. The program then applies 2D-DWT on the three color channels (in the selected color space) for each cell in the image. Then select the 16 most significant wavelet components from each of the three channels. The outputs, of the form

⟨image id, cell coord, channel id, wavelet bin, value⟩

are written to a file

1. Given a set of images and a color histogram specification, extract the features from all images and populates a feature database.
2. Given a 9-cell (3 cells by 3 cells) query image region consisting of 3 cells by 3 cells), a color histogram specification (for color-based retrieval), and a channel id (for the rest of the features), identify the best 10 matching image regions for each feature and display the matches.

Assumptions

The program assumes that the user is aware of the functions that are

implemented and has previous knowledge of the various color models as well as the mechanics of discrete image transformations.

**Proposed Solution**

The proposed solution to the goals and requirements was to implement a program utilizing the Python Imaging Library and various open source APIs available on the internet. Given a set of images provided by the user, the program will perform the functions described in the ‘Goal’ portion above and produce a SQLite file with extension .db as output. The program is run locally and the pathname to a folder containing a set of image is supplied by the user at runtime, as well as an additional argument specifying the color space to be used in the functions described in the goals section.

After reading in the image paths found in the supplied image directory then task 1 will proceed to evaluate all pixels found in the images and convert them to the user supplied color space. The converted color space is then passed as a list of pixels to the median cut algorithm where, using a predefined number of bins, the most similar pixels are grouped together into 16 individual bins. The data stored in the bins are then written to a file names ‘Task\_I\_histogram\_boundaries’. Algorithms used for this task are listed in appendix A1.

When task II starts it will ask the user to specify a path to an image. After reading that image in it will convert the pixels in that image to the user specified color space then divide the image into a 8x8 grid and save the grid as a list of values. Then it will begin to evaluate each cell in the 8x8 grid and using the values stored in Task\_I\_histogram\_boundaries the program will compare each pixel in that specific cell and get the index for the bin in which the pixel should be stored in. After evaluating all cells the individual 16 bin histograms are written to a file named ‘Task\_II\_histogram.txt’. Algorithms used for this task are listed in appendix A2.

Task III begins by asking the user to specify a path to an image. It then begins to split the image into 8x8 image cells. Then using each cell in the 8x8 grid the individual color channels are obtained for each individual cell and passed to the SciPy 2D-DCT function for processing. After the color channels are passed through the 2D-DCT function the first 16 frequency components for each color channel are appended to the output file. After all the cells in the 8x8 image have been evaluated the final values are written to a file named ‘Task\_III\_out.txt’. Algorithms used for this task are listed in appendix A3.

Task IV starts in a similar fashion to task II where it will ask the user to specify a path to an image. Then it will convert the pixels in that image to the user specified color space then divide the image into a 8x8 grid and save the grid as a list of values. Then using the user supplied image path a function will calculate two lists of 16 elements each, using the Sobel operators provided by the SciPy API. The first list has the lower bounds for each bin and the second list contains the number of pixels in each bin. After calculating the horizontal and vertical Sobel approximations they are passed to an arctangent function to calculate the gradients direction (i.e., angle). These two lists represent the angle bin and value parameters found in task IV in the Goals section. The final Sobel operations are written to a file named ‘Task\_IV\_out’. Algorithms used for this task are listed in appendix A4.

Task V is similar to task IV in that it also uses the SciPy Sobel operator except that after calculating the horizontal and vertical Sobel approximations they are not passed to the arctangent function. Just like task IV the user supplied image path is passed to a function which calculates the two lists of 16 elements each. The final Sobel operations are written to a file named ‘Task\_ V\_out’. Algorithms used for this task are listed in appendix A5.

Task VI takes advantage of a function in the PyWavelets API to perform the 2 dimensional discrete wavelet transformation. The tasks starts off by asking the user to specify a path to an image. It then begins to split the image into 8x8 image cells. . Then using each cell in the 8x8 grid the individual color channels are obtained for each individual cell and passed to the PyWavelets 2D-DWT function for processing. After the color channels are passed through the 2D-DWT function the first 16 frequency components for each color channel are appended to the output file. After evaluating all cells the results are written to a file named ‘Task\_VI\_out’. Algorithms used for this task are listed in appendix A6.

Task VII

Task VIII

**Interface Specifications**

Phase one and two of the project relied heavily on the Python Image Library for various tasks such as returning the contents of an image as a list of pixel values and opening and identifying a given image file. In this phase PIL was still used but in order to achieve functionality for all of tasks listed in the Goals our team used five separate python libraries. Those libraries are listed below with their functionality with respect to the task III, IV & V, VI and VII:

SciPy – used to return the Discrete Cosine Transform of arbitrary type sequence x

Matplotlib - a python 2D plotting library which generates plots, histograms, power spectra, bar charts, error charts, and scatterplots.

NumPy – a suite that provided multidimensional array objects, masked arrays and matrices, and a set of routines for fast operations on arrays, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations.

PyWavelets – function for calculating 1D and 2D Forward and Inverse Discrete Wavelet Transform

SQLite – python implementation that provides a software library that implements a self-contained, server less, zero-configuration, transactional SQL database engine

The interface for this project is accessed through the command prompt or terminal. The functionality of each task is stored in individual class file’s in which all the computation behind that specific task is performed by the methods in that class file with the output written to a database file stored in a folder names ‘Outputs’. The tasks are automatically executed in a linear fashion and will prompt the user for the image path when working on individual image files. After execution of task VI the SQLite engine will query the results from the individual database files from the previous tasks and merge them into one large database file.

**System Requirements**

The system requirements are as follows:

* + Mac OS X 10.6+, Windows XP or Later (any OS with Python is suitable but installation instructions are only given for Windows)
  + Root permission level to change Path variables
  + Python 2.7.3 (This is the latest stable release)
  + Python Imaging Library 1.1.7
  + Matplotlib 1.2.0
  + NumPy 1.6.2
  + SciPy 0.11
  + PyWavelets
  + SQLite 3.7.14.1

**Installation Instructions**

1. Download and install the x86 (32-bit) version of python 2.7 (<http://www.python.org/download/>).
2. Add the python installation directory (e.g. C:\Python27) to your $PATH

environmental variable.

1. Download and install the latest version of PIL for Python 2.7 (<http://www.pythonware.com/products/pil/>).
2. Download and install Numpy

(<https://sourceforge.net/projects/numpy/files/>)

1. Download and install Matplotlib

(<https://github.com/matplotlib/matplotlib/downloads>)

1. Download and install Scipy

(<https://sourceforge.net/projects/scipy/files/>)

1. Download and install PyWavelets

(<https://github.com/nigma/pywt>)

1. Download and install SQLite

(<http://sqlite.org/releaselog/3_7_14_1.html>)

**Execution Instructions**

1. Open up a DOS or Terminal prompt and navigate to the folder where the program is located
2. Type ‘python main.py’ to run the program. Note: All of the supplied .py files must be in the same folder.

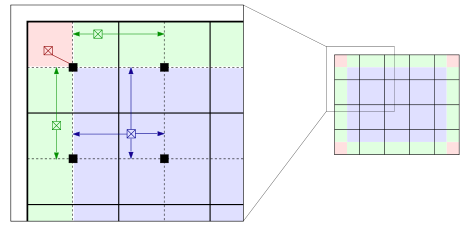
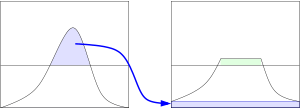


1. As each task is executed it will ask for any parameters that the user should provide.

**Related Work**

Histograms are well understood in image processing and photography but recent advancements have led to techniques used to greatly improve the contrast in images. A recent example of this is contrast limited adaptive histogram equalization (CLAHE) which works in a similar fashion as the tasks completed in the project. Adaptive histogram equalization diverges from regular histogram equalization such that the adaptive method computes several histograms, each corresponding to a specific section of the image, and uses them to rearrange the lightness values of the image (Figure 2). CLAHE improves upon this by introducing contrast limiting which removes the noise oversampling found in traditional adaptive histogram equalization (Figure 1). An applied example of this can be seen in Stephan Saalfeld’s paper discussing CLAHE and it’s operation on stacks of images.

Figure Figure 2



**Conclusions**

**Bibliography**

"Discrete Wavelet Transform." PyWavelets. Python, 01 Dec. 2012. Web. 03 Dec. 2012. <http://www.pybytes.com/pywavelets/>.

"SQL for Python." SQLite Home Page. N.p., 03 Dec. 2012. Web. 03 Dec. 2012. <http://sqlite.org/>.

DeBuyl, Pierre. "Scientific Tools for Python." SciPy. N.p., 12 Oct. 2012. Web. 03 Dec. 2012. <http://www.scipy.org/SciPy>.

Drake, Fred L., Jr. "Python V2.7.3 Documentation." *Overview â”*. Python Software Foundation, 19 Sept. 2012. Web. 20 Sept. 2012. <[http://docs.python.org/>.](about:blank)  
  
"Python Imaging Library Handbook." *Python Imaging Library Handbook*. Pythonware.com, 6 May 2005. Web. 20 Sept. 2012. <http://www.pythonware.com/library/pil/handbook/index.htm>.  
  
Li, Ze-Nian, and Mark S. Drew. *Fundamentals of Multimedia*. Upper Saddle River, NJ: Pearson Prentice Hall, 2004. Print.

Saalfeld, Stephan. "Enhance Local Contrast (CLAHE)." Process / Enhance Local Contrast. Fiji, 01 Sept. 2010. Web. 03 Dec. 2012. <http://fiji.sc/Enhance\_Local\_Contrast\_(CLAHE)>.

**Appendix**:

***A1***: *Median cut algorithm, pixels to converted color space*

class Bin(object):

"""A bin object contains pixel values, and a range of pixel values covered by the bin."""

attrs = {'pixels', 'mins', 'maxes'}

def \_\_str\_\_(self):

return str([self.mins, self.maxes])

def \_\_len\_\_(self):

return len(self.pixels)

\_\_repr\_\_ = \_\_str\_\_

def \_\_init\_\_(self, pixels=None, mins=(0, 0, 0), maxes=(255, 255, 255)):

if hasattr(pixels, '\_\_dict\_\_') and Bin.attrs.issubset(set(pixels.\_\_dict\_\_.keys())):

# Copy constructor

self.pixels = list(pixels.pixels)

self.mins = pixels.mins

self.maxes = pixels.maxes

else:

if pixels is None:

self.pixels = []

else:

self.pixels = pixels

self.mins = mins

self.maxes = maxes

def get\_median(pixels, component):

"""Given a list of pixels and a component, find the median value."""

left = int(len(pixels) / 2)

if len(pixels) % 2 == 0:

return sorted(pixels, key=itemgetter(component))[left][component]

else:

return sum(pixel[component] for pixel in sorted(pixels, key=itemgetter(component))[left:left + 1]) / 2

def split\_box(box, component):

"""Given a list of colors, median-split based on the given component."""

sorted\_box = sorted(box, key=itemgetter(component))

box1 = sorted\_box[int(len(sorted\_box) / 2):]

box2 = sorted\_box[:int(len(sorted\_box) / 2)]

return box1, box2

def split\_bin(bin, component):

"""Given a bin and a component, median split the bin on the given component."""

mid = get\_median(bin.pixels, component)

new\_bin = Bin(bin)

bin.pixels, new\_bin.pixels = split\_box(bin.pixels, component)

bin.mins = bin.mins[:component] + (mid, ) + bin.mins[component+1:]

new\_bin.maxes = new\_bin.maxes[:component] + (mid, ) + new\_bin.maxes[component+1:]

if any(bin.mins[i] > bin.maxes[i] for i in range(3)):

import pdb; pdb.set\_trace()

if any(new\_bin.mins[i] > new\_bin.maxes[i] for i in range(3)):

import pdb; pdb.set\_trace()

return bin, new\_bin

def component\_range(bin, component):

"""

Given a bin, find the range of values for the given component.

Component is an index, e.g. in RGB, red is 0, green is 1, blue is 2.

"""

colors = bin.pixels

channels = zip(\*colors)

return max(channels[component]) - min(channels[component])

def median\_cut(pixels, n):

"""Given a list of pixels, get a list of n bins, grouping the most similar pixels together."""

# We start with one box containing all the pixels

bins = [Bin(pixels)]

while len(bins) < n:

# Split the largest box

largest\_box = max(bins, key=len)

ranges = [component\_range(largest\_box, i) for i in range(3)]

# Get the color component with the largest range

max\_range = max(ranges)

component\_index = ranges.index(max\_range)

# Median split largest box based on the color component with largest range

box1, box2 = split\_bin(largest\_box, component\_index)

# Replace the largest box with the two that it split into

i = bins.index(largest\_box)

bins[i:i+1] = [box1, box2]

return bins

def get\_all\_the\_pixels(images):

return it.chain.from\_iterable(img.getdata() for img in images)

def median\_cut\_histogram(images, color\_space):

# Get all the pixels

pixels = get\_all\_the\_pixels(images)

# Convert to target operational color space

pixels = [convert\_pixel(pixel, "rgb", color\_space) for pixel in pixels]

bins = median\_cut(pixels, BINS)

with open(os.path.join(OUTPUT\_FOLDER, "Task\_I\_histogram\_boundaries.txt"), 'w') as out:

out.write(str(bins))

***A2***: *Task II implementation*

from \_\_future\_\_ import division, print\_function, generators

import matplotlib.pyplot as plt

import os

import operator as op

import itertools as it

import imagedata as id

from pixel\_converter import convert\_pixel

from divider import get\_image\_cells

OUTPUT\_FOLDER = os.path.join(os.path.split(\_\_file\_\_)[0], "../", "Outputs")

def pixel\_le(p, q):

# perform elementwise subtraction

sub = list(it.starmap(op.sub, zip(q, p)))

# p is less than or equal to q iff all elements are >= 0

if all(s >= 0 for s in sub)

return True

return False

def pixel\_ge(p, q):

# perform elementwise subtraction

sub = list(it.starmap(op.sub, zip(q, p)))

# p is greater than or equal to q iff all elements are < 0

if all(s <= 0 for s in sub):

return True

return False

def bin\_pixel(pixel, hist\_spec):

for i, bin in enumerate(hist\_spec):

if pixel\_ge(pixel, bin[0]) and pixel\_le(pixel, bin[1]):

return i

return -1

def get\_histogram\_spec():

"""

Get the histogram specification output by task I.

"""

# Get histogram specification

with open(os.path.join(OUTPUT\_FOLDER, "Task\_I\_histogram\_boundaries.txt")) as hist\_spec\_file:

hist\_spec = eval(hist\_spec\_file.read())

return hist\_spec

def histogram\_generator(image, image\_id, color\_space):

"""

Given an image, target operational color space, and the histogram spec from task I,

divide the image into 8x8 image cells, and generate a histogram for each cell,

according to the specification generated by task I.

"""

pixels = image.getdata()

width = image.size[0]

# Convert to target operational color space

pixels = [convert\_pixel(pixel, "rgb", color\_space) for pixel in pixels]

# Split the image into 8x8 image cells

image\_cells = list(get\_image\_cells(pixels, width, 8, 8))

# Get histogram specification

hist\_spec = get\_histogram\_spec()

histogram\_output = []

for cell\_coord, cell in enumerate(image\_cells):

id.add\_cell(image\_id, cell\_coord)

bin\_counter = dict(zip(range(16), [0]\*16))

for pix\_coord, pixel in enumerate(cell):

# Get bin for pixel

bin\_counter[bin\_pixel(pixel, hist\_spec)] += 1

# Output

for color\_instance\_id, value in bin\_counter.items():

histogram\_output.append((image\_id, cell\_coord, color\_instance\_id, value))

with open(os.path.join(OUTPUT\_FOLDER, "Task\_II\_histogram.txt"), 'w') as histogram\_file:

histogram\_file.write('\n'.join(str(s) for s in histogram\_output))

***A3*:** *Task III implementation*

from \_\_future\_\_ import division, print\_function, generators

from scipy.fftpack import dct as sp\_dct, idct as sp\_idct

import os

from divider import get\_image\_cells

OUTPUT\_FOLDER = os.path.join(os.path.split(\_\_file\_\_)[0], "../", "Outputs")

def dct\_freq(image, image\_id, color\_space):

pixels = list(image.getdata())

width = image.size[0]

# Split the image into 8x8 image cells

image\_cells = list(get\_image\_cells(pixels, width, 8, 8))

output = []

for cell\_coord, cell in enumerate(image\_cells):

channels = zip(\*cell)

for channel\_id, channel in enumerate(channels):

freq\_components = dct(channel)

most\_significant = freq\_components[:16]

for freq\_bin, value in enumerate(most\_significant):

output.append((image\_id, cell\_coord, channel\_id, freq\_bin, value))

with open(os.path.join(OUTPUT\_FOLDER, "Task\_III\_out.txt"), 'w') as f:

f.write('\n'.join(str(s) for s in output))

def dct(channel):

return sp\_dct([float(x) for x in channel], type=2, norm='ortho')

def dct\_inverse(channel):

return sp\_idct(channel, type=2, norm='ortho')

***A4***: *Task IV implementation*

from scipy import misc

from scipy import ndimage

import numpy

import matplotlib.pyplot as plt

import pylab as pl

from divider import get\_image\_cells

from PIL import Image

from numpy import ndarray

from pixel\_converter import convert\_pixel

import math

import os

import pdb

OUTPUT\_FOLDER = os.path.join(os.path.split(\_\_file\_\_)[0], "../", "Outputs")

def PIL2array(img):

'''

converts a python image to a numpy array

'''

return numpy.array(img.getdata(),numpy.uint8).reshape(img.size[1], img.size[0], 3)

def get\_hist\_angle\_bins(img):

'''

given a python image, return 2 lists of 16 elements each. 1 list (bin\_lowers) has the lower bounds for each bin, the other list (hist\_vals)

has the number of pixels in each bin.

'''

im = numpy.array(img)

im = numpy.resize(im,(8,8))#reshape array to model 8x8 cell

sx = ndimage.sobel(im, axis=0, mode = 'constant')#apply sobel operator in x-direction

sy = ndimage.sobel(im, axis=1, mode = 'constant')#apply sobel operator in y-direction

sx = list(numpy.array(sx).reshape(-1,))#reshape sobel output into 1-d list for easy manipulation

sy = list(numpy.array(sx).reshape(-1,))

sobel = []

for x in sx:

for y in sy:

x = float(x)

y = float(y)

if x != 0:#Avoid a divide-by-zero error

angle = math.degrees((math.atan2(y,x))) #does this give the angle we're looking for? trying to get the full 360-degree range

else:

angle = math.degrees((math.atan(200)))#what should the value be if x is 0?

if angle<0:

angle +=360

sobel.append(angle)

hist, bin\_edges = numpy.histogram(sobel, bins = 16)

bin\_lowers = list(numpy.array(bin\_edges).reshape(-1,))#unnecessary because i've already reshaped the data?

bin\_lowers.pop()#gets rid of the high side of the highest bin

hist\_vals = list(numpy.array(hist).reshape(-1,))#also unnecessary?

return bin\_lowers, hist\_vals

def angle\_histogram\_generator(image, image\_id, color\_space):

'''

given a pil image, the name of that image and a colorspace to work in:

splits the image into 8x8 cells, generates a histogram for each cell.

'''

pixels = image.getdata()

width = image.size[0]

#pixels = [convert\_pixel(pixel, color\_space, "yuv") for pixel in pixels]

c1,c2,c3 = zip(\*pixels)#separate out luminance

if color\_space == "RGB" or "rgb":

n1 = 'R'

n2 = 'G'

n3 = 'B'

elif color\_space == "YUV" or "yuv":

n1 = 'Y'

n2 = 'U'

n3 = 'V'

else:

n1 = 'H'

n2 = 'S'

n3 = 'V'

histogram\_output = []

#for c1:

image\_cells = list(get\_image\_cells(c1, width, 8, 8))

for cell\_coord, cell in enumerate(image\_cells):

#pdb.set\_trace()

color\_instance\_id\_list, value\_list = get\_hist\_angle\_bins(cell)

for i in range (0,15):

histogram\_output.append((image\_id, cell\_coord, n1, color\_instance\_id\_list[i], value\_list[i]))

return histogram\_output

#for c2:

image\_cells = list(get\_image\_cells(c2, width, 8, 8))

for cell\_coord, cell in enumerate(image\_cells):

#pdb.set\_trace()

color\_instance\_id\_list, value\_list = get\_hist\_angle\_bins(cell)

for i in range (0,15):

histogram\_output.append((image\_id, cell\_coord, n2, color\_instance\_id\_list[i], value\_list[i]))

return histogram\_output

#for c3:

image\_cells = list(get\_image\_cells(c3, width, 8, 8))

for cell\_coord, cell in enumerate(image\_cells):

color\_instance\_id\_list, value\_list = get\_hist\_angle\_bins(cell)

for i in range (0,15):

histogram\_output.append((image\_id, cell\_coord, n3, color\_instance\_id\_list[i], value\_list[i]))

return histogram\_output

def do\_task\_4(image, image\_id, color\_space):

output = angle\_histogram\_generator(image, image\_id, color\_space)

with open(os.path.join(OUTPUT\_FOLDER, "Task\_IV\_out.txt"), 'w') as output\_file:

output\_file.write('\n'.join(str(s) for s in output))

**A5**: *Task V implementation*

from scipy import misc

from scipy import ndimage

import numpy

import matplotlib.pyplot as plt

import pylab as pl

from divider import get\_image\_cells

from PIL import Image

from numpy import ndarray

from pixel\_converter import convert\_pixel

import os

import pdb

OUTPUT\_FOLDER = os.path.join(os.path.split(\_\_file\_\_)[0], "../", "Outputs")

def PIL2array(img):

'''

converts a python image to a numpy array

'''

return numpy.array(img.getdata(),numpy.uint8).reshape(img.size[1], img.size[0], 3)

def get\_hist\_amp\_bins(img):

'''

given a python image, return 2 lists of 16 elements each. 1 list (bin\_lowers) has the lower bounds for each bin, the other list (hist\_vals)

has the number of pixels in each bin.

'''

im = numpy.array(img)

im = numpy.resize(im,(8,8))#reshape array to model 8x8 cell

sx = ndimage.sobel(im, axis=0, mode = 'constant')#apply sobel operator in x-direction

sy = ndimage.sobel(im, axis=1, mode = 'constant')#apply sobel operator in y-direction

sob = numpy.hypot(sx,sy)

hist, bin\_edges = numpy.histogram(sob, bins = 16)#sob or im?

bin\_lowers = list(numpy.array(bin\_edges).reshape(-1,))

bin\_lowers.pop()

hist\_vals = list(numpy.array(hist).reshape(-1,))

return bin\_lowers, hist\_vals

def amplitude\_histogram\_generator(image, image\_id, color\_space):

'''

given a pil image, the name of that image and a colorspace to work in:

splits the image into 8x8 cells, generates a histogram for each cell.

'''

pixels = image.getdata()

width = image.size[0]

pixels = [convert\_pixel(pixel, color\_space, "yuv") for pixel in pixels]

c1,c2,c3 = zip(\*pixels)#pull out luminance

if color\_space == "RGB" or "rgb":

n1 = 'R'

n2 = 'G'

n3 = 'B'

elif color\_space == "YUV" or "yuv":

n1 = 'Y'

n2 = 'U'

n3 = 'V'

else:

n1 = 'H'

n2 = 'S'

n3 = 'V'

histogram\_output = []

#for c1:

image\_cells = list(get\_image\_cells(c1, width, 8, 8))

for cell\_coord, cell in enumerate(image\_cells):

color\_instance\_id\_list, value\_list = get\_hist\_amp\_bins(cell)

for i in range (0,15):

histogram\_output.append((image\_id, cell\_coord, n1, color\_instance\_id\_list[i], value\_list[i]))

return histogram\_output

#for c2:

image\_cells = list(get\_image\_cells(c2, width, 8, 8))

for cell\_coord, cell in enumerate(image\_cells):

color\_instance\_id\_list, value\_list = get\_hist\_amp\_bins(cell)

for i in range (0,15):

histogram\_output.append((image\_id, cell\_coord, n2, color\_instance\_id\_list[i], value\_list[i]))

return histogram\_output

#for c3:

image\_cells = list(get\_image\_cells(c3, width, 8, 8))

for cell\_coord, cell in enumerate(image\_cells):

color\_instance\_id\_list, value\_list = get\_hist\_amp\_bins(cell)

for i in range (0,15):

histogram\_output.append((image\_id, cell\_coord, n3, color\_instance\_id\_list[i], value\_list[i]))

return histogram\_output

def do\_task\_5(image, image\_id, color\_space):

output = amplitude\_histogram\_generator(image, image\_id, color\_space)

with open(os.path.join(OUTPUT\_FOLDER, "Task\_V\_out.txt"), 'w') as output\_file:

output\_file.write('\n'.join(str(s) for s in output))

**A6:** *Task VI implementation*

from \_\_future\_\_ import division, print\_function, generators

import pywt

import os

from itertools import izip\_longest

from divider import get\_image\_cells

def grouper(n, iterable, fillvalue=None): # this is an itertools recipe function

"Collect data into fixed-length chunks or blocks"

# grouper(3, 'ABCDEFG', 'x') --> ABC DEF Gxx

args = [iter(iterable)] \* n

return izip\_longest(fillvalue=fillvalue, \*args)

OUTPUT\_FOLDER = os.path.join(os.path.split(\_\_file\_\_)[0], "../", "Outputs")

def dwt\_freq(image, image\_id, color\_space):

pixels = list(image.getdata())

width = image.size[0]

# Split the image into 8x8 image cells

image\_cells = list(get\_image\_cells(pixels, width, 8, 8))

output = []

for cell\_coord, cell in enumerate(image\_cells):

channels = zip(\*cell)

for channel\_id, channel in enumerate(channels):

freq\_components = sum([list(x) for x in dwt(list(grouper(width, channel)))], [])

most\_significant = freq\_components[:16]

for freq\_bin, value in enumerate(most\_significant):

output.append((image\_id, cell\_coord, channel\_id, freq\_bin, value))

with open(os.path.join(OUTPUT\_FOLDER, "Task\_VI\_out.txt"), 'w') as f:

f.write('\n'.join(str(s) for s in output))

def dwt(channel):

cA, (cH,cV,cD) = pywt.dwt2(channel, 'db1')

return list(cD)

**A7:** *Task VII implementation*

""

schema: (items marked with \* denote primary key)

cells(\*cell\_id, image\_id, cell\_coord)

color\_instance(cell\_id, ci\_id, ci\_value)

channels(channel\_id, channel\_value)#RGB, YUV, or HSV

dct(cell\_id, channel\_id, freq\_bin\_id, freq\_bin\_value)

grad\_angle(cell\_id, channel\_id, angle\_bin\_id, angle\_bin\_value)

grad\_amp(cell\_id, channel\_id, amplitude\_bin\_id, amplitude\_bin\_val)

dwt(cell\_id, channel\_id, wavelet\_bin\_id, wavelet\_bin\_value)

"""

import sqlite3

from os import remove as delete

from os import path as path

import os

class myDB(object):

def \_\_init\_\_(self, filename):

default\_db\_location = os.path.realpath(path.join(path.split(\_\_file\_\_)[0], "../", "Databases"))

self.conn = sqlite3.connect(path.join(default\_db\_location, filename))

self.cursor = self.conn.cursor()

def make\_db(self):

self.cursor.execute("CREATE TABLE cells (cell\_id integer NOT NULL, image\_id text NOT NULL, cell\_coord text NOT NULL, PRIMARY KEY (cell\_id))")

self.cursor.execute("CREATE TABLE color\_instance (cell\_id integer, ci\_id text, ci\_value integer)")

self.cursor.execute("CREATE TABLE channels (channel\_id text, channel\_value text)")

self.cursor.execute("CREATE TABLE dct (cell\_id integer, channel\_id text, freq\_bin\_id text, freq\_bin\_value integer)")

self.cursor.execute("CREATE TABLE grad\_angle(cell\_id integer, channel\_id text, angle\_bin\_id text, angle\_bin\_value integer)")

self.cursor.execute("CREATE TABLE grad\_amp(cell\_id integer, channel\_id text, amplitude\_bin\_id text, amplitude\_bin\_value integer)")

self.cursor.execute("CREATE TABLE dwt(cell\_id integer, channel\_id text, wavelet\_bin\_id text, wavelet\_bin\_value integer)")

self.conn.commit()

def add\_cell(self, image\_id, cell\_coord):

info = [image\_id, cell\_coord]

self.cursor.execute("INSERT INTO cells VALUES (NULL, ?, ?)", info)

self.conn.commit()

def add\_channels(self,colorspace):

if colorspace == 'RGB':

c1 = 'R'

c2 = 'G'

c3 = 'B'

elif colorspace == 'YUV':

c1 = 'Y'

c2 = 'U'

c3 = 'V'

elif colorspace == 'HSV':

c1 = 'H'

c2 = 'S'

c3 = 'V'

else:

#colorspace = RGB

c1 = 'R'

c2 = 'G'

c3 = 'B'

channels = [(1, c1),(2, c2), (3, c3)]

self.cursor.executemany("INSERT INTO channels VALUES (?,?)", channels)

self.conn.commit()

def clear\_db(fullfilepath):

self.conn.close()

delete(fullfilepath)

def add\_dct(self, cell\_id, channel\_id, freq\_bin\_id, freq\_bin\_value):

info = [cell\_id, channel\_id, freq\_bin\_id, freq\_bin\_value]

self.cursor.execute("INSERT INTO dct VALUES (?,?,?,?)", info)

self.conn.commit()

def add\_grad\_angle(self, cell\_id, channel\_id, angle\_bin\_id, angle\_bin\_value):

info = [cell\_id, channel\_id, angle\_bin\_id, angle\_bin\_value]

self.cursor.execute("INSERT INTO grad\_angle VALUES (?,?,?,?)", info)

def add\_grad\_amp(self, cell\_id, channel\_id, amplitude\_bin\_id, amplitude\_bin\_value):

info = [cell\_id, channel\_id, amplitude\_bin\_id, amplitude\_bin\_value]

self.cursor.execute("INSERT INTO grad\_amp VALUES (?,?,?,?)", info)

self.conn.commit()

def add\_dwt(self, cell\_id, channel\_id, wavelet\_bin\_id, wavelet\_bin\_value):

info = [cell\_id, channel\_id, wavelet\_bin\_id, wavelet\_bin\_value]

self.cursor.execute("INSERT INTO dwt VALUES (?,?,?,?)", info)

self.conn.commit()

def query\_db(self, qrystring):

self.cursor.execute(qrystring)

self.conn.commit()

**A8:** *Task VIII implementation*

Contributions  
Steven Carr – Documentation and code review   
Casey Kuball – Task I, II & VI  
Glenn Craver – Task III & VI  
Wesley Rose - Task III, IV & V

Dawne Flanagan - Task VII & Task VIII