Richard Ngo

HW 4

Code Used:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 14 08:48:10 2019
@author: crjones4
"""
# CompVisHw4.py - Updated code included in hw4 document
# Created on 12/5/19
# @Updated: Richard Ngo
# Leung-Malik filter bank generated from code from Tony Joseph's (CVDLBOT) Github
# https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/

#####
#delete this line below to revert the code at the bottom to it's default state
doMakeTexels=True
#####

import numpy as np
import cv2
import time
import scipy.ndimage as ndimg
import math
from sklearn.cluster import KMeans
import argparse
NROT = 6
NPER = 8
NFILT = NROT*NPER
FILTSIZE = 49
```

```
NCLUSTERS = 4
TEXELSIZE = 4
#pathName = "C:\\Data\\"
pathName = ""
#fileName = "aerial-houses"
fileName = "texture"
#fileName = "richardme"
#fileNames = ["texture","aerial-houses","richardme"]

def showimg(img):
    cv2.namedWindow('showimg', flags=cv2.WINDOW_NORMAL)
    cv2.imshow('showimg', img)
    #cv2.resizeWindow('showimg', (int(len(img[0])*2), int(len(img)*2)))
    print(img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

##########################################################################
# Leung-Malik filter bank generated from code from Tony Joseph's (CVDLBOT) Github
# https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/
# code within this section is all from cited source

def gaussian1d(sigma, mean, x, ord):
    x = np.array(x)
    x_ = x - mean
    var = sigma**2

    # Gaussian Function
    g1 = (1/np.sqrt(2*np.pi*var))*(np.exp((-1*x_*x_)/(2*var)))
```

```
    if ord == 0:
        g = g1
        return g
    elif ord == 1:
        g = -g1*((x_)/(var))
        return g
    else:
        g = g1*(((x_*x_) - var)/(var**2))
        return g

def gaussian2d(sup, scales):
    var = scales * scales
    shape = (sup,sup)
    n,m = [(i - 1)/2 for i in shape]
    x,y = np.ogrid[-m:m+1,-n:n+1]
    g = (1/np.sqrt(2*np.pi*var))*np.exp( -(x*x + y*y) / (2*var) )
    return g

def log2d(sup, scales):
    var = scales * scales
    shape = (sup,sup)
    n,m = [(i - 1)/2 for i in shape]
    x,y = np.ogrid[-m:m+1,-n:n+1]
    g = (1/np.sqrt(2*np.pi*var))*np.exp( -(x*x + y*y) / (2*var) )
    h = g*((x*x + y*y) - var)/(var**2)
    return h

def makefilter(scale, phasex, phasey, pts, sup):
```

```
    gx = gaussian1d(3*scale, 0, pts[0,...], phasex)
    gy = gaussian1d(scale,   0, pts[1,...], phasey)

    image = gx*gy

    image = np.reshape(image,(sup,sup))
    return image

# This function will compute and return the Leung-Malik filters
# the filters are in a 3D array of floats, F(FILTSIZE, FILTSIZE, NFILT)
def makeLMfilters():
    sup     = 49
    scalex  = np.sqrt(2) * np.array([1,2,3])
    norient = 6
    nrotinv = 12

    nbar  = len(scalex)*norient
    nedge = len(scalex)*norient
    nf    = nbar+nedge+nrotinv
    F     = np.zeros([sup,sup,nf])
    hsup  = (sup - 1)/2

    x = [np.arange(-hsup,hsup+1)]
    y = [np.arange(-hsup,hsup+1)]

    [x,y] = np.meshgrid(x,y)

    orgpts = [x.flatten(), y.flatten()]
```

```
orgpts = np.array(orgpts)


count = 0
for scale in range(len(scalex)):
    for orient in range(norient):
        angle = (np.pi * orient)/norient
        c = np.cos(angle)
        s = np.sin(angle)
        rotpts = [[c+0,-s+0],[s+0,c+0]]
        rotpts = np.array(rotpts)
        rotpts = np.dot(rotpts,orgpts)
        F[:,:,count] = makefilter(scalex[scale], 0, 1, rotpts, sup)
        F[:,:,count+nedge] = makefilter(scalex[scale], 0, 2, rotpts, sup)
        count = count + 1

count = nbar+nedge
scales = np.sqrt(2) * np.array([1,2,3,4])

for i in range(len(scales)):
    F[:,:,count]   = gaussian2d(sup, scales[i])
    count = count + 1

for i in range(len(scales)):
    F[:,:,count] = log2d(sup, scales[i])
    count = count + 1

for i in range(len(scales)):
    F[:,:,count] = log2d(sup, 3*scales[i])
    count = count + 1
```

```
    return F
#end of cited code
##########################################################################################
def saveFilters(img):
 (height, width, depth) = img.shape
 count = 0
 for row in range(NPER):
     for col in range(NROT):
         tempImg = img[:, :, count]
         filename = "Filters\\LM_" + str(row) + "_" + str(col)
         normedFilter = normImg(tempImg)
         saveImage(normedFilter, filename)
         count = count + 1
 return
##########################################################################################
# this function will apply the filter bank in the 3D array filt to
# the inputImg; the result is an array of results res(height, width, NFILT)
def applyLMfilters(inputImg, filt):
    imgH = np.size(inputImg, axis=0)
    imgW = np.size(inputImg, axis=1)
    padding=np.uint8((np.size(filt,axis=0)-1)/2)
    imgSlice = np.zeros((np.size(filt,axis=0),np.size(filt,axis=1),np.size(filt,axis=2)),dtype=np.float64)
    imgHold =
np.zeros((np.size(inputImg,axis=0),np.size(inputImg,axis=1),np.size(filt,axis=2)),dtype=np.float64)
    res = np.copy(imgHold)
    for d in range(np.size(filt,axis=2)):
        imgHold[:,:,d] = inputImg[:,:]
```

```
   for y in range(imgH):
      for x in range(imgW):
         if((y*imgW+x)%(imgW*10)==0):
            print(x+y*imgW,"(",(x+y*imgW)/(imgH*imgW),")","out of",imgH*imgW,"done")
         voidU = min(y-padding,0)
         voidD = max(y+padding+1-imgH,0)
         voidL = min(x-padding,0)
         voidR = max(x+padding+1-imgW,0)
         imgSlice[-voidU:2*padding+1-voidD,-voidL:2*padding+1-voidR,:] = imgHold[y-padding-
voidU:y+padding+1-voidD,x-padding-voidL:x+padding+1-voidR,:]
         res[y,x,:] =  np.sum(np.multiply(imgSlice, filt),axis=(0,1))
         imgSlice*=0
   print("Filters Applied")
   return res
##############################################################################
def normImg(img):
 tempImg = np.zeros_like(img)
 tempImg = (cv2.normalize(img, tempImg, 0.0, 127.0, cv2.NORM_MINMAX))
 res = (tempImg+128.0).astype(np.uint8)
 return res
##############################################################################

def makeMosaic(img):
 (height, width, depth) = img.shape
 res = np.zeros((height*8, width*6), np.float64)
 count = 0
 for row in range(8):
    for col in range(6):
       res[row*height:(row+1)*height, col*width:(col+1)*width] = \
```

```
        normImg(img[:, :, count])
        count = count + 1
 return res
##########################################################################
def saveImage(img, name):
 cv2.imwrite(pathName + name + ".png", img)
 return
##########################################################################
# this function will take a 3D array of filter bank responses and form texels
# by combining the feature vectors in nonoverlapping squares of size sz
# the result newR is an array of floats the same size as R, but within
# texels all feature vectors are identical
def formTexels(R, sz):
   imgH = np.size(R, axis=0)
   imgW = np.size(R, axis=1)
   newR = np.zeros((imgH,imgW,np.size(R, axis=2)),dtype=np.float64)
   for y in range((math.ceil(imgH/sz))):
      for x in range((math.ceil(imgW/sz))):
         voidD = max(y+sz+1-imgH,0)
         voidR = max(x+sz+1-imgW,0)

         newR[y*sz:(y+1)*sz-voidD,x*sz:(x+1)*sz-voidR,:] = np.average(R[y*sz:(y+1)*sz-
voidD,x*sz:(x+1)*sz-voidR,:],axis=(0,1))
   print("Texels Formed")
   return newR
##########################################################################

# this function will take an image-sized collection of filter bank responses
# and use the KMeans algorithm to find the best segments
```

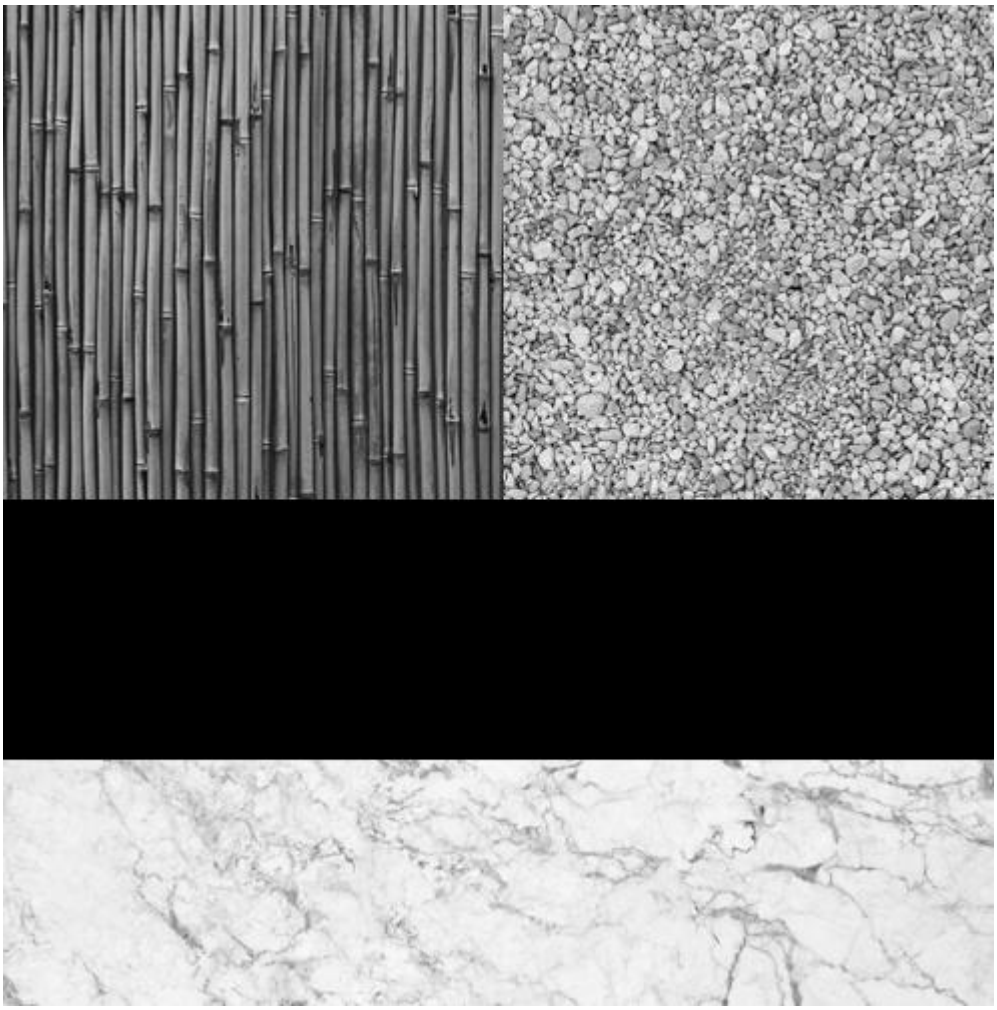
```
# it returns a pseucolor rendition of the original image, where each color
# corresponds to a separate cluster (a separate type of texture)
def segmentKMeans(R, nclus):
   imgH = np.size(R, axis = 0)
   imgW = np.size(R, axis = 1)
   imgD = np.size(R, axis = 2)
   Kcenter = np.zeros((1,nclus,imgD), dtype=np.float64)
   KcenterP = np.copy(Kcenter)
   Ksum = np.zeros((1,nclus,imgD), dtype=np.float64)
   Kcount =np.zeros((1,nclus,imgD), dtype=np.float64)
   distMin = 0
   indMin = 0
   stop = 0
   for k in range(nclus):
      Kcenter[0,k,:] =
R[np.int32(imgH/nclus*k+imgH/(nclus*2)),np.int32(imgW/nclus*k+imgW/(nclus*2)),:]


   while stop == 0:
      for y in range(imgH):
         for x in range(imgW):
            if((y*imgW+x)%(imgW*10)==0):
               print(x+y*imgW,"(",(x+y*imgW)/(imgH*imgW),")","out of",imgH*imgW,"done")
            for k in range(nclus):
               checkmin=pow(np.sum(np.power(np.subtract(R[y,x,:],Kcenter[0,k,:]),2)),1/2)
               if distMin>checkmin or k==0:
                  distMin = checkmin
                  indMin = k
            Ksum[0,indMin,:]+=R[y,x,:]
            Kcount[0,indMin,:]+=1
```

```
            distMin=0
            indMin=0
        print("Group's member count: ",Kcount[0,:,0])
        Kcount[Kcount==0]=1
        Kcenter=np.divide(Ksum,Kcount)
        Ksum*=0
        Kcount*=0

        changed = np.sum(np.abs(np.subtract(Kcenter,KcenterP)))
        print("Change in centers: ",changed)

        if (changed==0):
           stop = 1
        #KcenterP=np.copy(Kcenter)
        KcenterP[:,:,:]=Kcenter[:,:,:]
    print("Kmeans Selected")
    pcolor = np.zeros((imgH,imgW,3), dtype=np.uint8)

    for y in range(imgH):
        for x in range(imgW):
           for k in range(nclus):
              checkmin=pow(np.sum(np.power(np.subtract(R[y,x,:],Kcenter[0,k,:]),2)),1/2)
              if distMin>checkmin or k==0:
                 distMin = checkmin
                 indMin = k
           pcolor[y,x,:] = (np.array([np.uint8(50+((255-50)/nclus)*indMin), np.uint8(0+((255-
50)/nclus)*indMin), np.uint8(50+0*indMin)]))
           #pcolor[y,x,0] = np.uint8(50+((255-50)/nclus)*indMin)
           #pcolor[y,x,1] = np.uint8(0+((255-50)/nclus)*indMin)
```

```
        #pcolor[y,x,2] = np.uint8(50)
        distMin=0
        indMin=0
    return pcolor
################################################################################
# This code sets the pathname from a command line option
# add the following as a command line option: --image_path="C:\\Data\\"
# replace C:\\Data with the proper path on your system
# Do NOT change this code – it's used for grading and you WILL lose points!!!!
parser = argparse.ArgumentParser();
parser.add_argument('--image_path', required=True, help='Absolute path of the image to be used.');
if __name__ == '__main__':
    args = parser.parse_args();
    pathName = args.image_path;
    print('IMAGE PATH: ', pathName);
currTime = time.time()
# Call the make filter function
F = makeLMfilters()
saveFilters(F)
saveImage(makeMosaic(F), "allFilters")
# load an image
inputImage = cv2.cvtColor(cv2.imread(pathName + fileName + ".png"), cv2.COLOR_BGR2GRAY)
# find filter responses
rawR = applyLMfilters(inputImage, F)
if (doMakeTexels):
    R = formTexels(rawR, TEXELSIZE)
else:
    R = rawR
```

```
# try segmenting
pcolor = segmentKMeans(R, NCLUSTERS)
saveImage(pcolor, fileName+"_Seg_"+str(NCLUSTERS))
elapsedTime = time.time() - currTime
print("Completed; elapsed time = ", elapsedTime)
```

textures.png



Richard Ngo

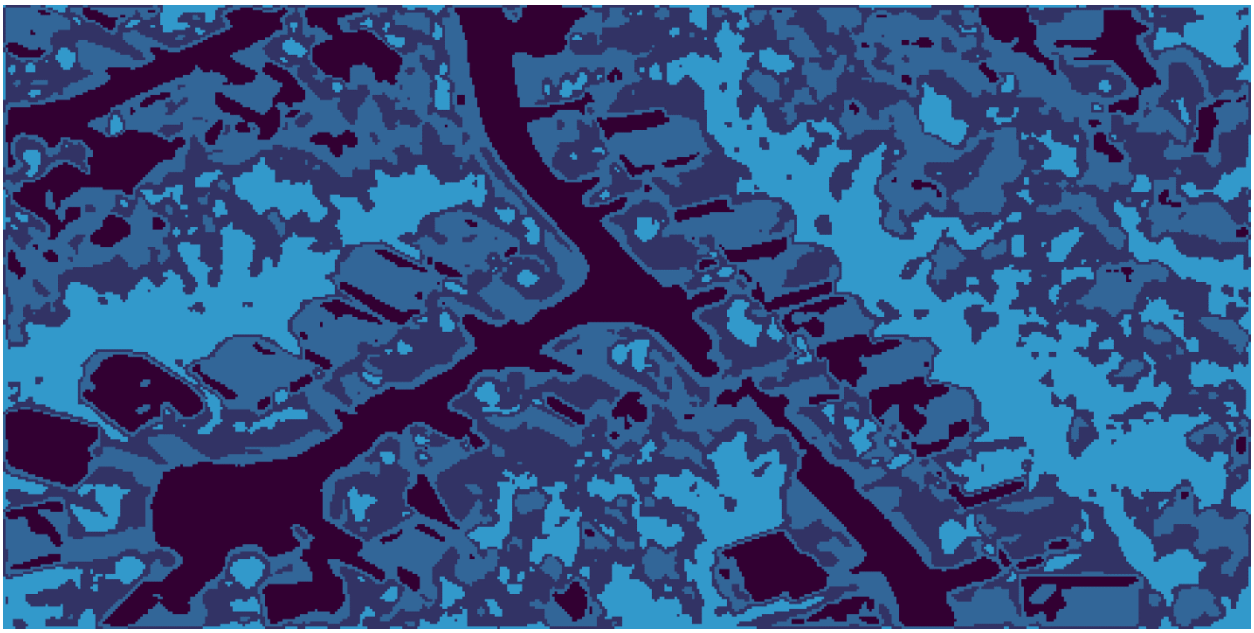
Best number of K means: 4 (each texture is mostly their own color)

Richard Ngo

aerial-houses.png

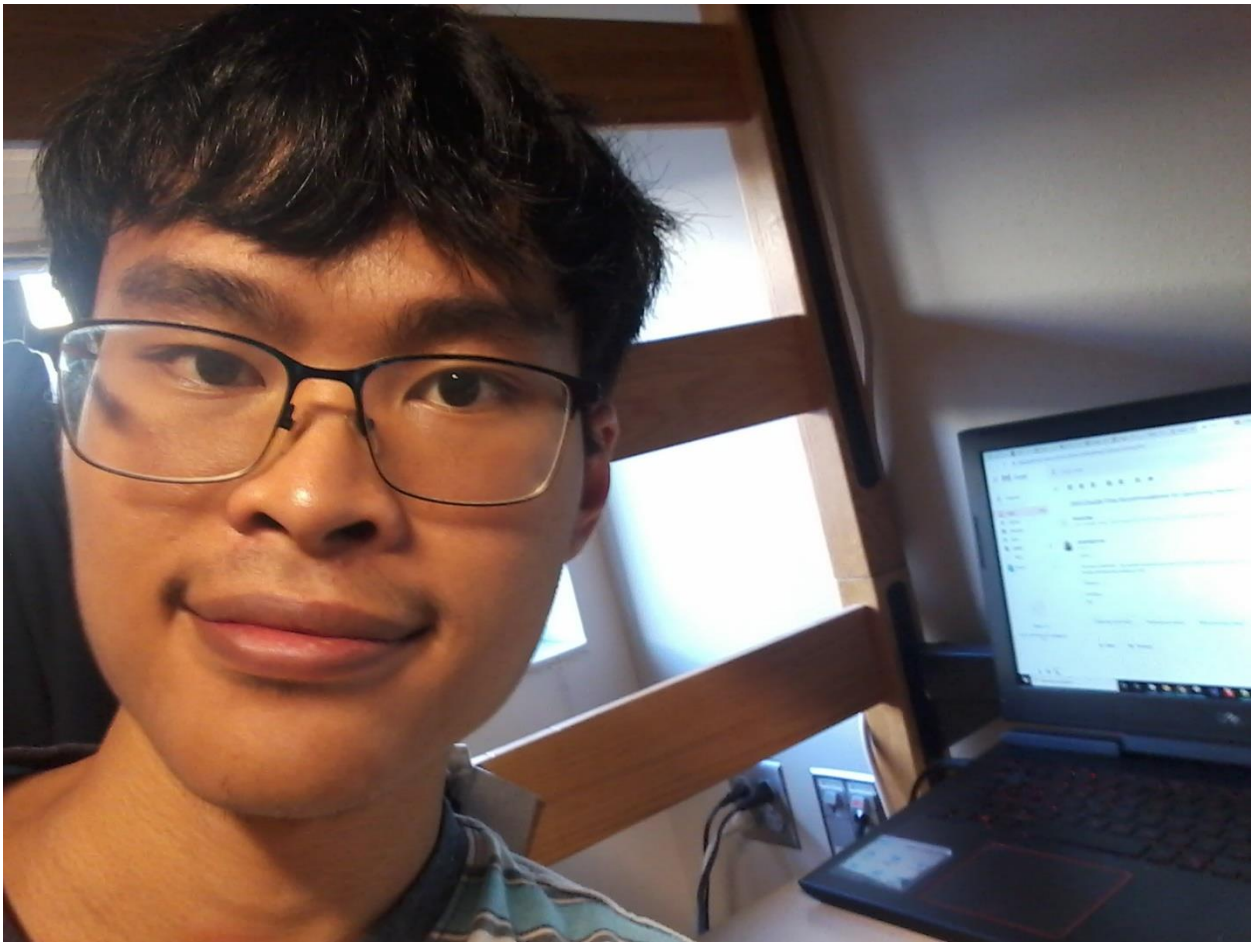
Best number of K means: 4 (road, main houses and trees mostly separated)

Richard Ngo



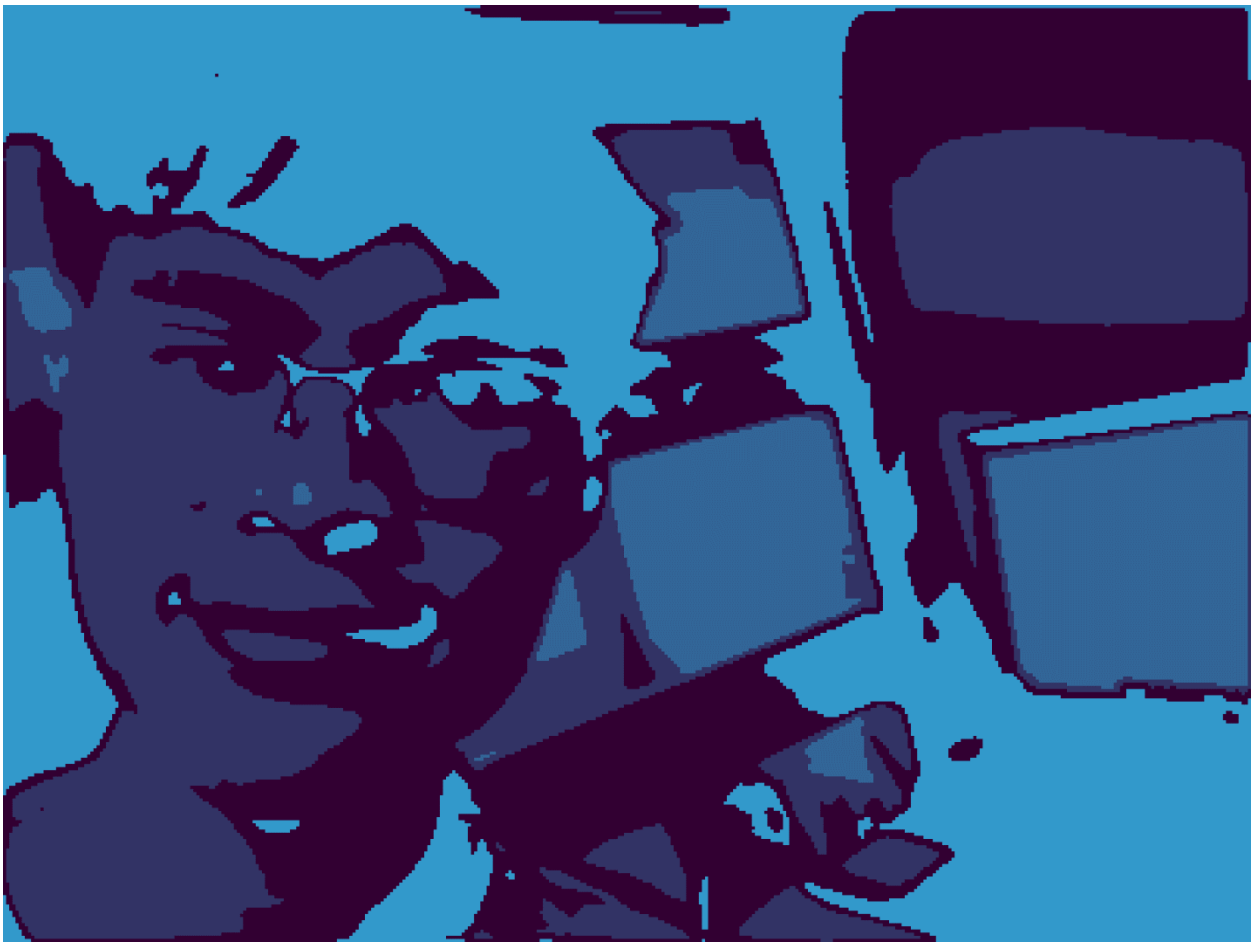
richardme.png

Richard Ngo



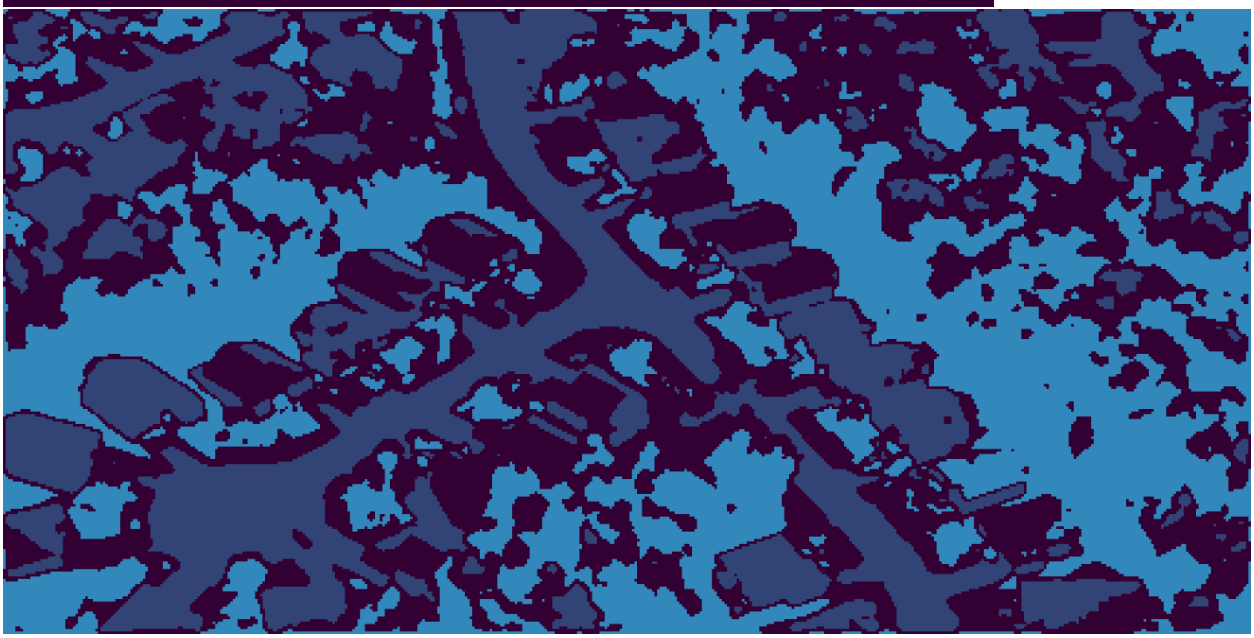
Richard Ngo

Best number of K means: 4 (Face is mostly a single color, and still enough differentiation between the background and me)

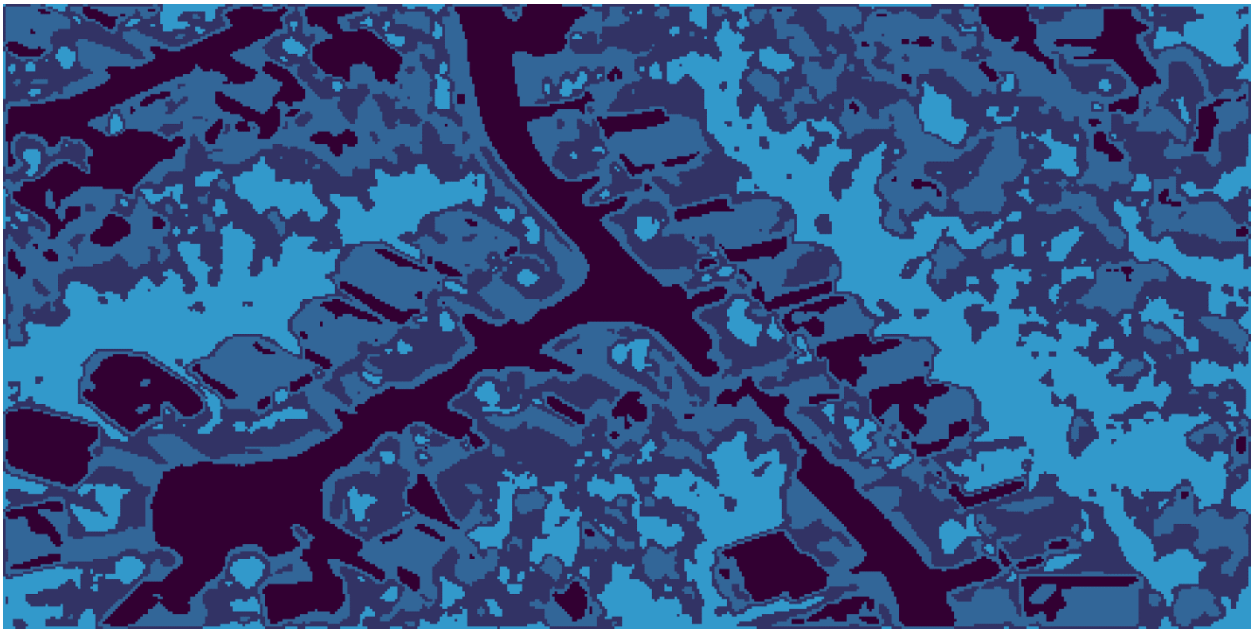


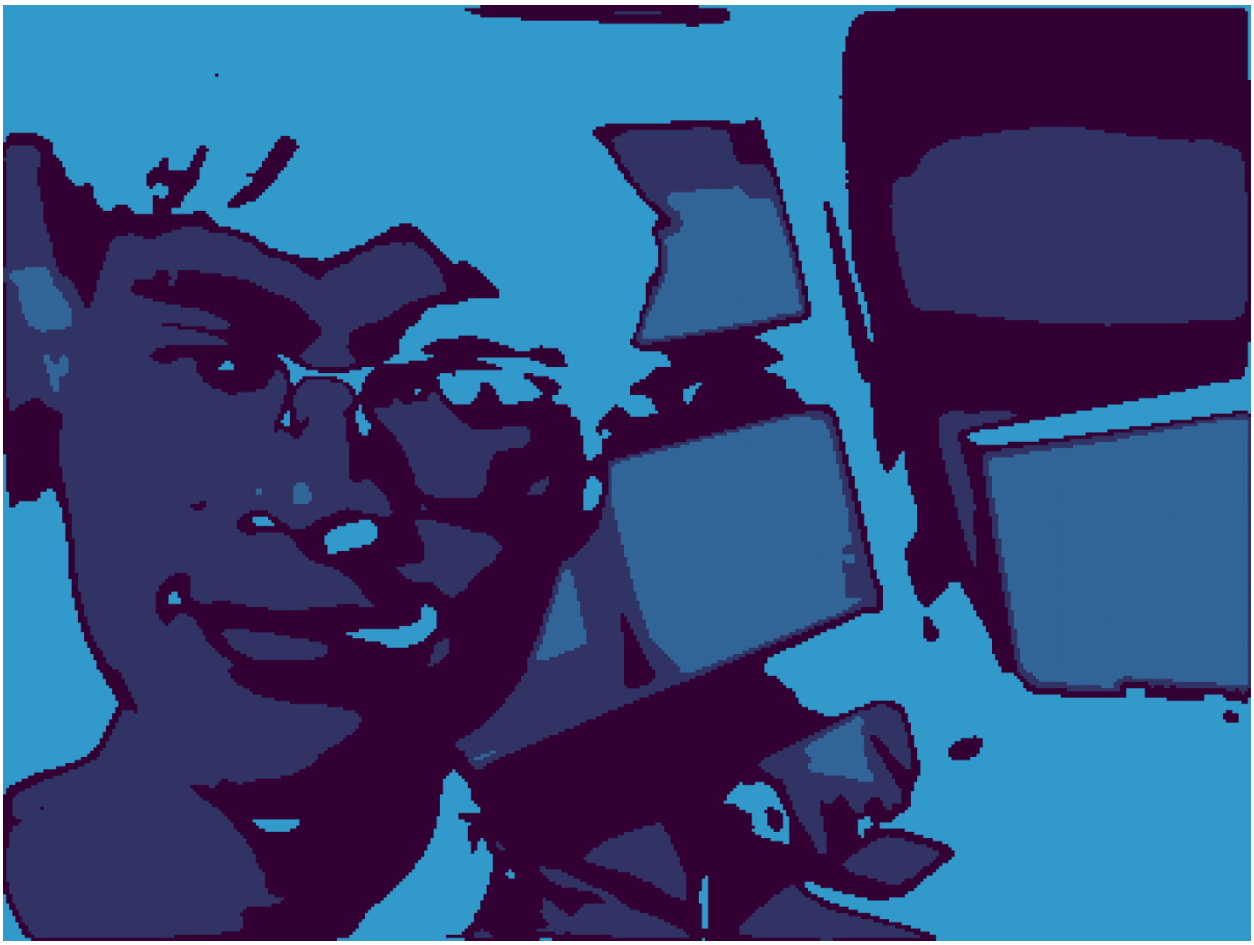
All number of K means tested:

K = 3



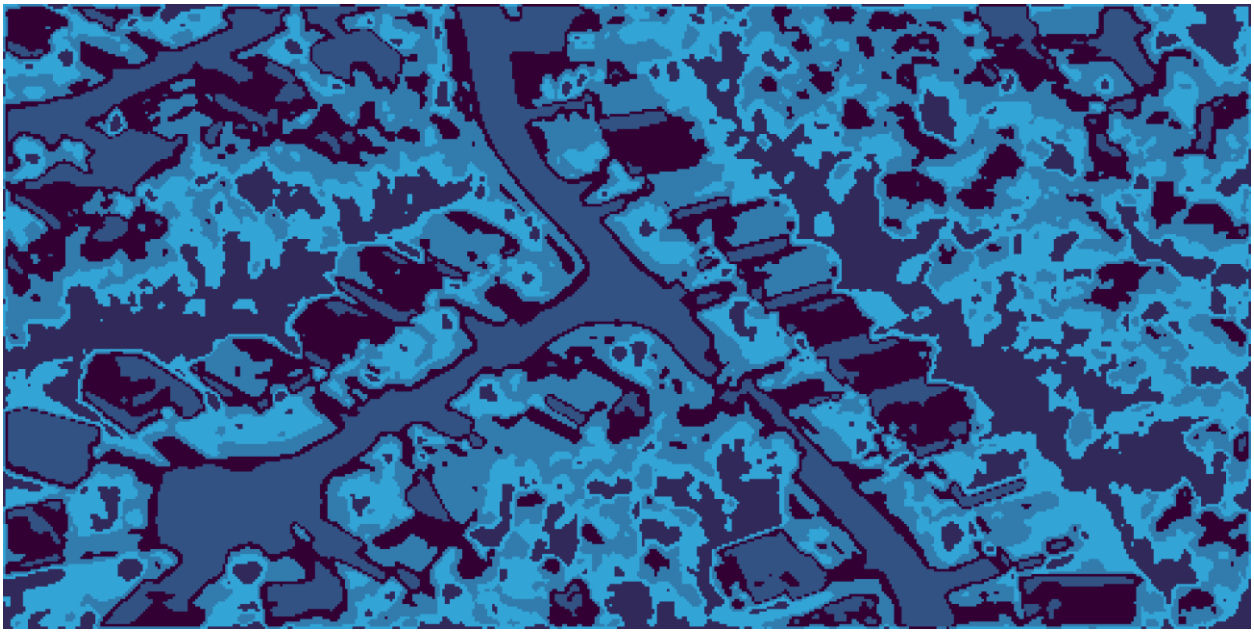
K=4



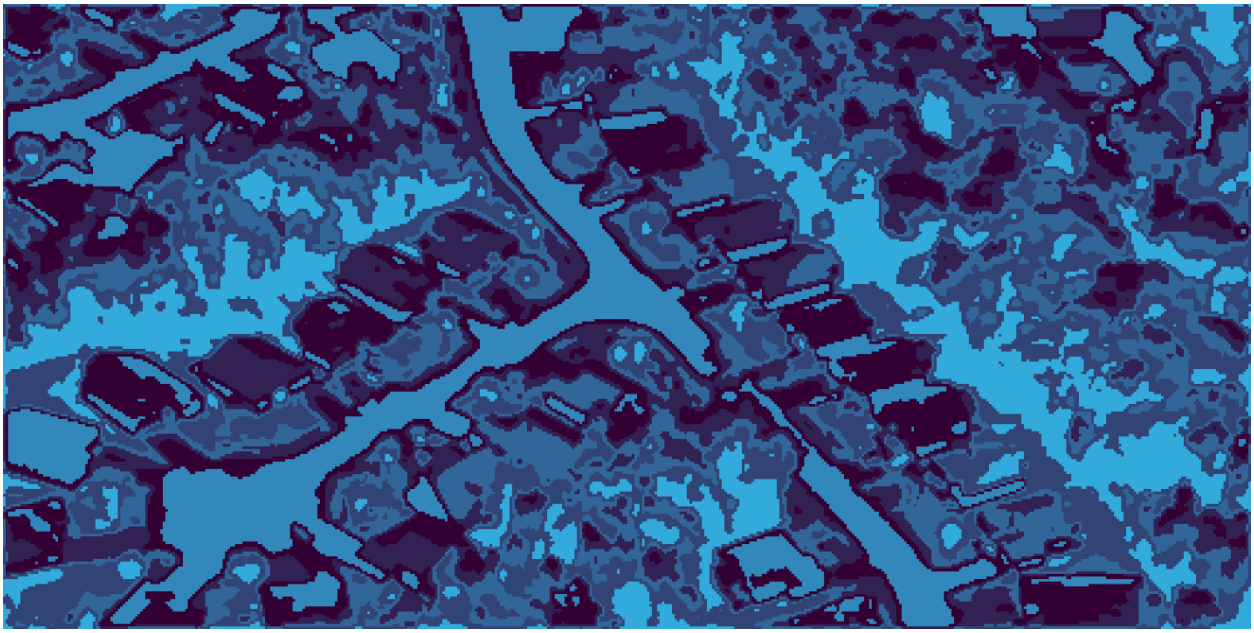


K=5

Richard Ngo



K=6



K=7