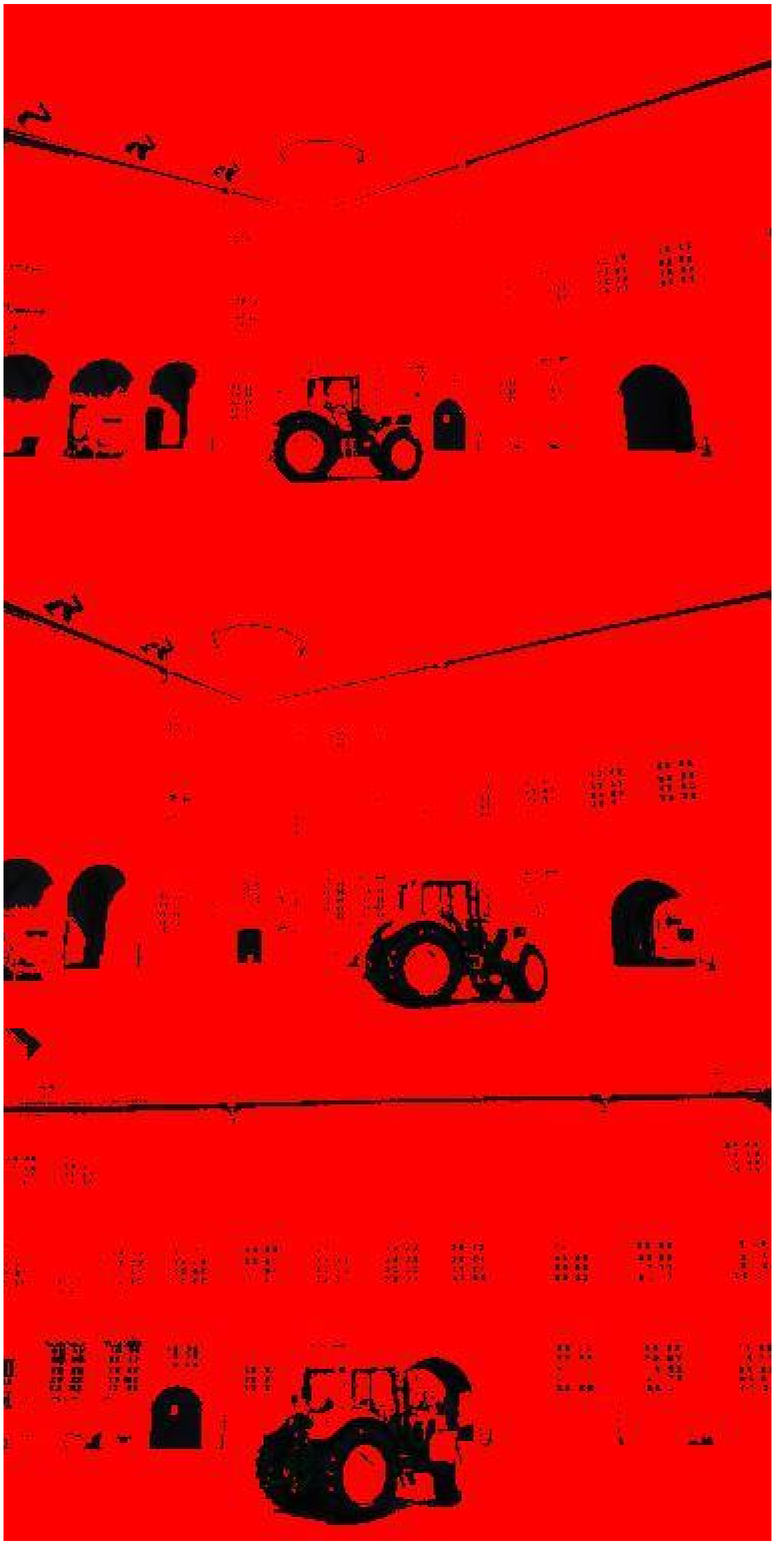
Kunal Dhaimade - CS 558 Optional Problem 2

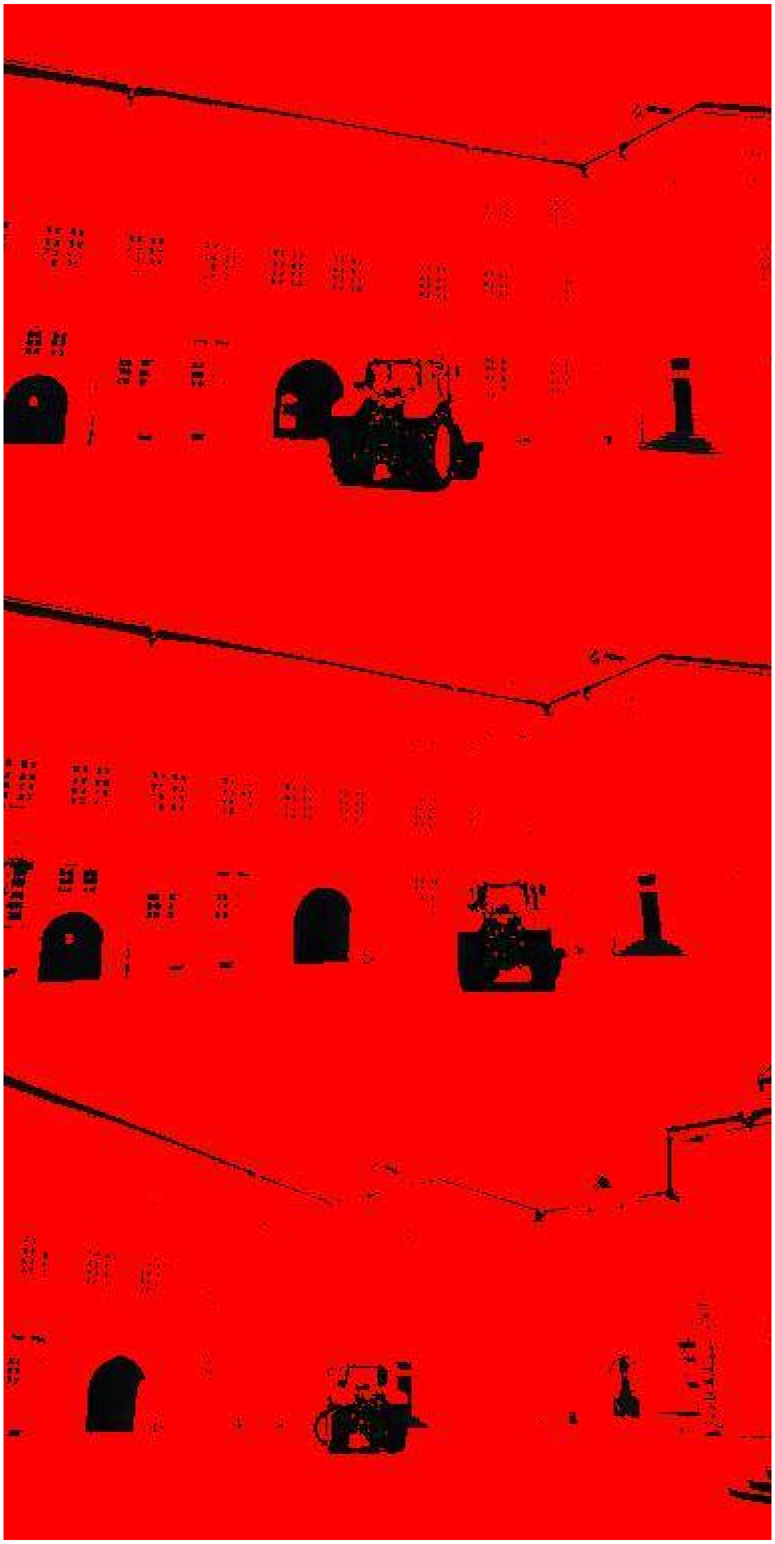
Source Code:

```
# -*- coding: utf-8 -*-
"""
Created on Wed May  3 03:23:17 2017

@author: Kunal
"""

from PIL import Image
import numpy as np
import math as mt
import random as rn
import os

'''To train the Pixel classifier'''
def PC_Train(file_path, k):
    og_im = Image.open(file_path+'train.png')
    mask_im = Image.open(file_path+'train_mask_blue.png')
    og_im_arr = np.asarray(og_im)
    row, col, x = og_im_arr.shape
    mask_im_arr = np.asarray(mask_im)
    mask_color = mask_im_arr[0][0]
    bg_set = []
    nonbg_set = []
    for i in range(row):
        for j in range(col):
            if(mask_im_arr[i][j][0] == mask_color[0] and mask_im_arr[i][j][1] == mask_color[1] and
mask_im_arr[i][j][2] == mask_color[2]):
                bg_set.append([int(og_im_arr[i][j][0]), int(og_im_arr[i][j][1]), int(og_im_arr[i][j][2])])
            else:
                nonbg_set.append([int(og_im_arr[i][j][0]), int(og_im_arr[i][j][1]), int(og_im_arr[i][j][2])])
    print('\nThe K-means Algorithm for the background set starts here: \n')
    bg_words = k_mean(bg_set, k)
    print('\nThe K-means Algorithm for the foreground set starts here: \n')
    nonbg_words = k_mean(nonbg_set, k)
    return bg_words, nonbg_words

'''To perform K-means clustering to obtain k words'''
def k_mean(ip_set, k):
    no_iter = 1
    converge_flag = 0
    centers = []
    set_len = len(ip_set)
    while(len(centers) != k):
        ran_i = rn.randint(0, set_len-1)
        ran_c = ip_set[ran_i]
        if(ran_c not in centers):
            centers.append(ran_c)
    while(converge_flag == 0):
        converge_flag = 1
        print("Iteration "+str(no_iter))
        clusters = [[] for i in range(len(centers))]
        for i in range(set_len):
            dist = 9999.0
            for k in range(len(centers)):
                c_dist = dist_color(centers[k][0], centers[k][1], centers[k][2], ip_set[i][0], ip_set[i][1],
ip_set[i][2])
                if(c_dist < dist):
                    dist = c_dist
                    c_ind = k
            clusters[c_ind].append([ip_set[i][0], ip_set[i][1], ip_set[i][2]])
        no_new_c = 0
        for i in range(len(clusters)):
            no_pts = len(clusters[i])
            r = 0; g = 0; b = 0
            for pt in clusters[i]:
                r = r + (int(pt[0])*int(pt[0]))
                g = g + (int(pt[1])*int(pt[1]))
                b = b + (int(pt[2])*int(pt[2]))
            r = r/no_pts; g = g/no_pts; b = b/no_pts
            r = int(mt.sqrt(r)); g = int(mt.sqrt(g)); b = int(mt.sqrt(b))
            new_color = [r, g, b]
            old_color = [centers[i][0], centers[i][1], centers[i][2]]
            if(new_color != old_color):
                no_new_c = no_new_c + 1
                centers[i][0] = r; centers[i][1] = g; centers[i][2] = b
                converge_flag = 0
        print("New centers:", no_new_c)
        print()
        no_iter = no_iter + 1
    return centers

'''To calculate the color distance for K-means'''
def dist_color(r1, g1, b1, r2, g2, b2):
    r_mean = (int(r1) + int(r2)) / 2
    r = int(r1) - int(r2)
    g = int(g1) - int(g2)
    b = int(b1) - int(b2)
    dist = float(mt.sqrt( (((512+r_mean)*r*r)/256) + (4*g*g) + (((767-r_mean)*b*b)/256) ))
    return dist

'''To classify and color pixels according to the Pixel Classifier'''
def PC_Test(file_path, bg_words, nonbg_words, output_path):
    for file in os.listdir(file_path):
        im = Image.open(file_path + file)
        im_arr = np.asarray(im)
        im_cl = np.asarray(im)
        im_cl.setflags(write = 1)
        row, col, x = im_arr.shape
        for i in range(row):
            for j in range(col):
                s_dist = 999999.0
                for k in range(len(bg_words)):
                    if(dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], bg_words[k][0], bg_words[k][1],
bg_words[k][2]) < s_dist):
                        s_dist = dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], bg_words[k][0],
bg_words[k][1], bg_words[k][2])
                ns_dist = 999999.0
                for k in range(len(nonbg_words)):
                    if(dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], nonbg_words[k][0],
nonbg_words[k][1], nonbg_words[k][2]) < ns_dist):
                        ns_dist = dist_color(im_arr[i][j][0], im_arr[i][j][1], im_arr[i][j][2], nonbg_words[k][0],
nonbg_words[k][1], nonbg_words[k][2])
                if(s_dist < ns_dist):
                    im_cl[i][j] = [255, 0, 0]
                else:
                    if(im_cl[i][j][0] > 25 or im_cl[i][j][1] > 25 or im_cl[i][j][2] > 25):
                        im_cl[i][j] = [255, 0, 0]
        op_im = Image.fromarray(im_cl)
        file_temp = file.split('.')
        save_name = file_temp[0]
        op_im.save(output_path + save_name+'_output_k'+str(len(bg_words))+'.jpg')
        op_im.show()

def main():
    PC_train_filepath = "./Train/"
    k = 4
    bg_words, nonbg_words = PC_Train(PC_train_filepath, k)
    PC_test_filepath = "./Test/"
    PC_op_path = "./Output/"
    PC_Test(PC_test_filepath, bg_words, nonbg_words, PC_op_path)

main()
```

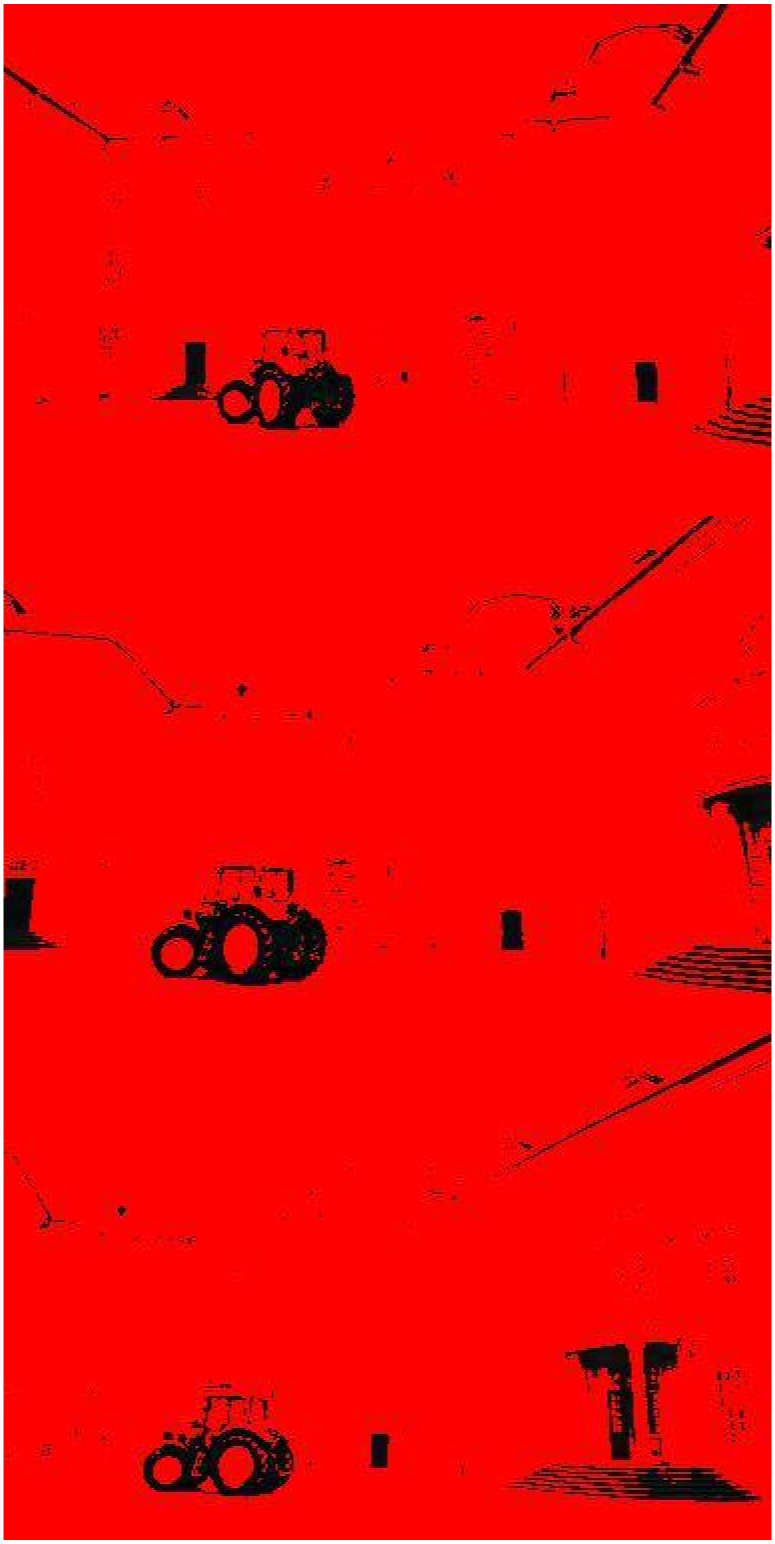
Kunal Dhaimade - CS 558 Optional Problem 2

Output Images:

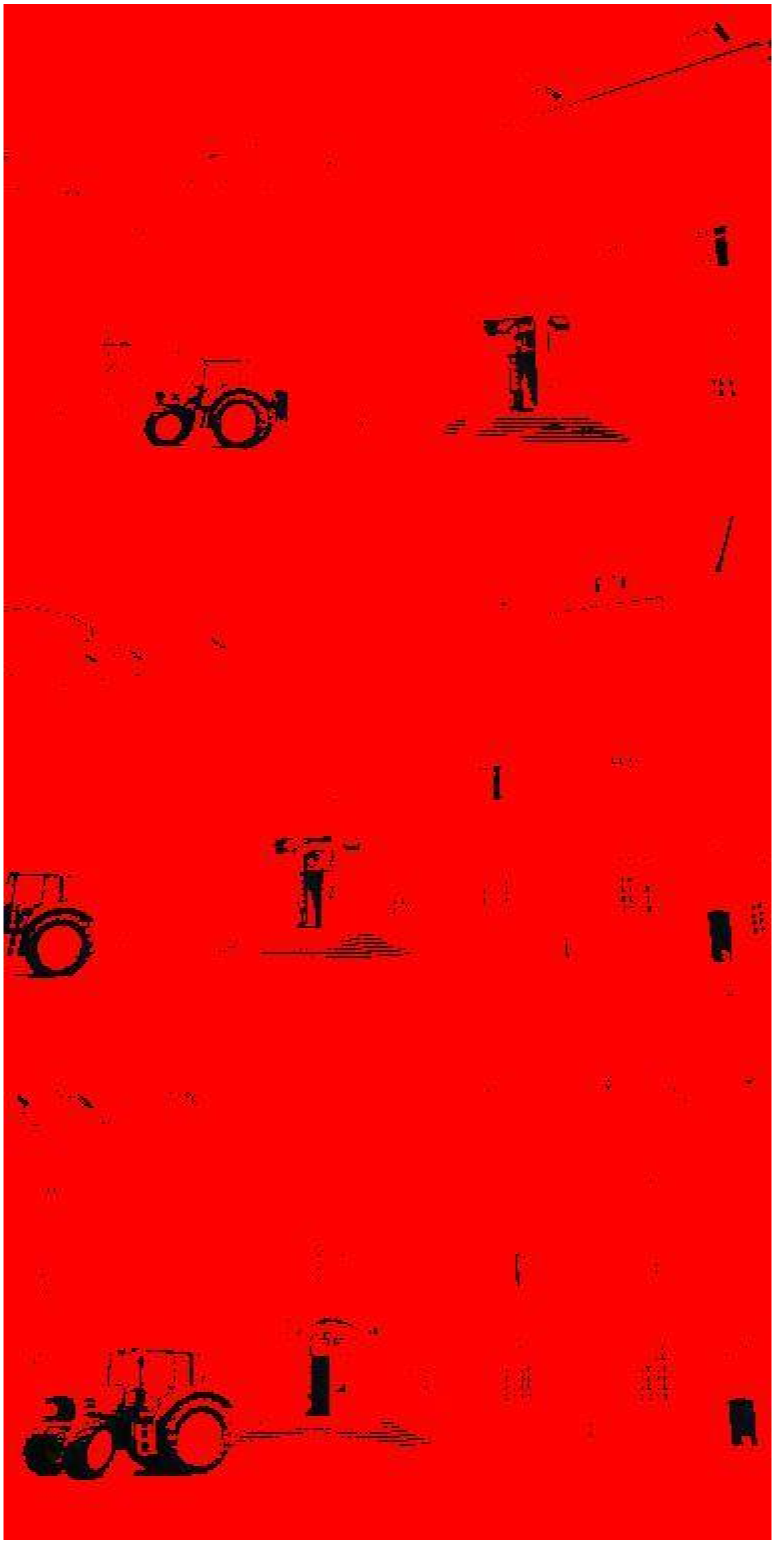




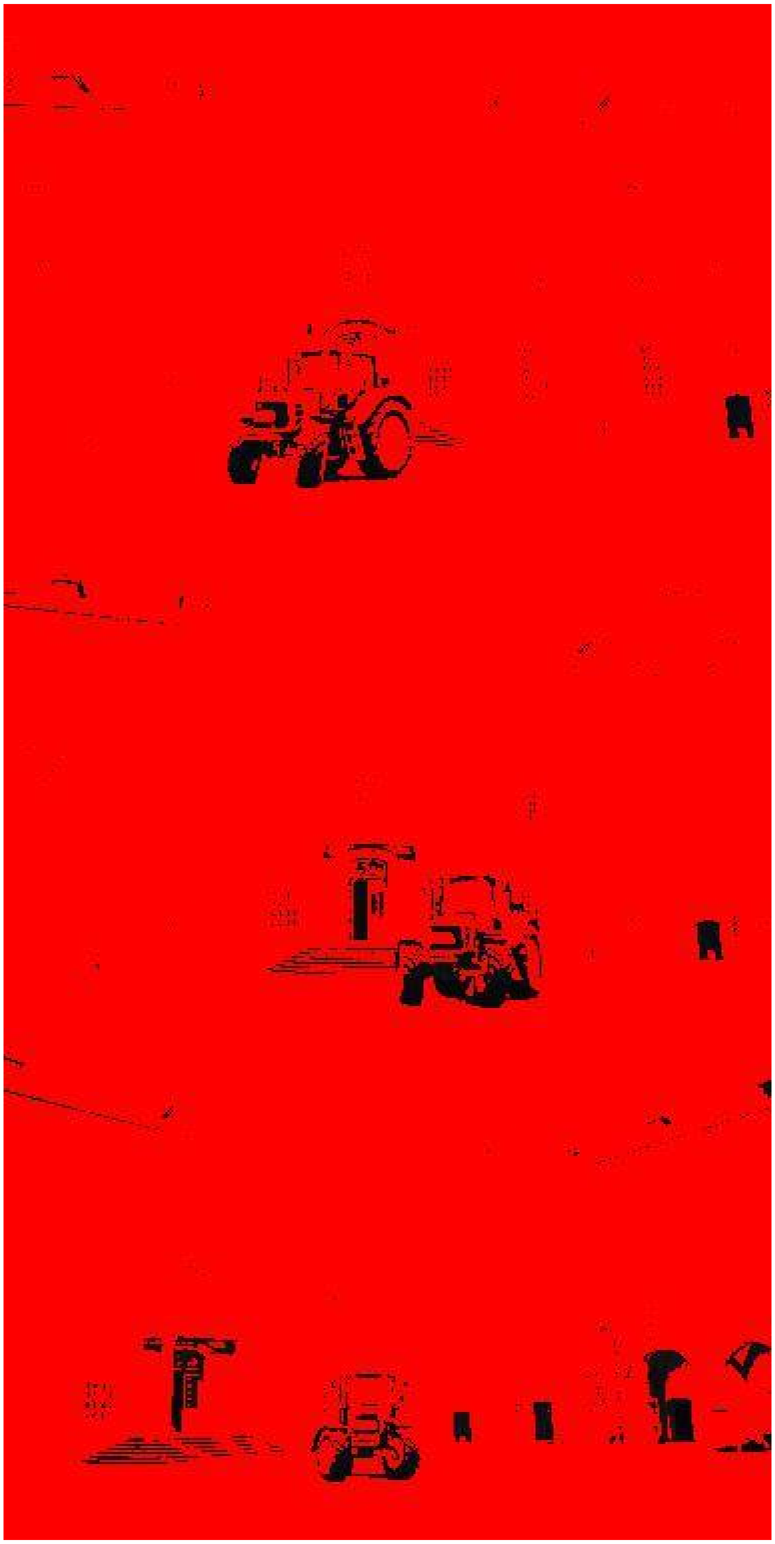
Kunal Dhaimade - CS 558 Optional Problem 2



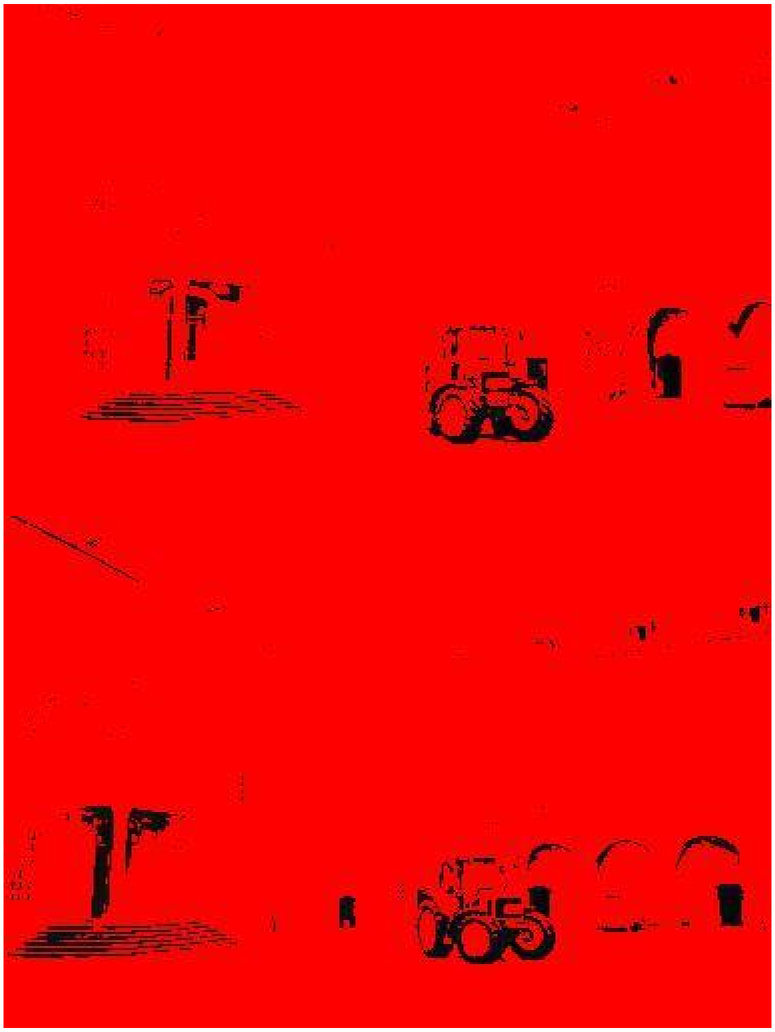
Kunal Dhaimade - CS 558 Optional Problem 2



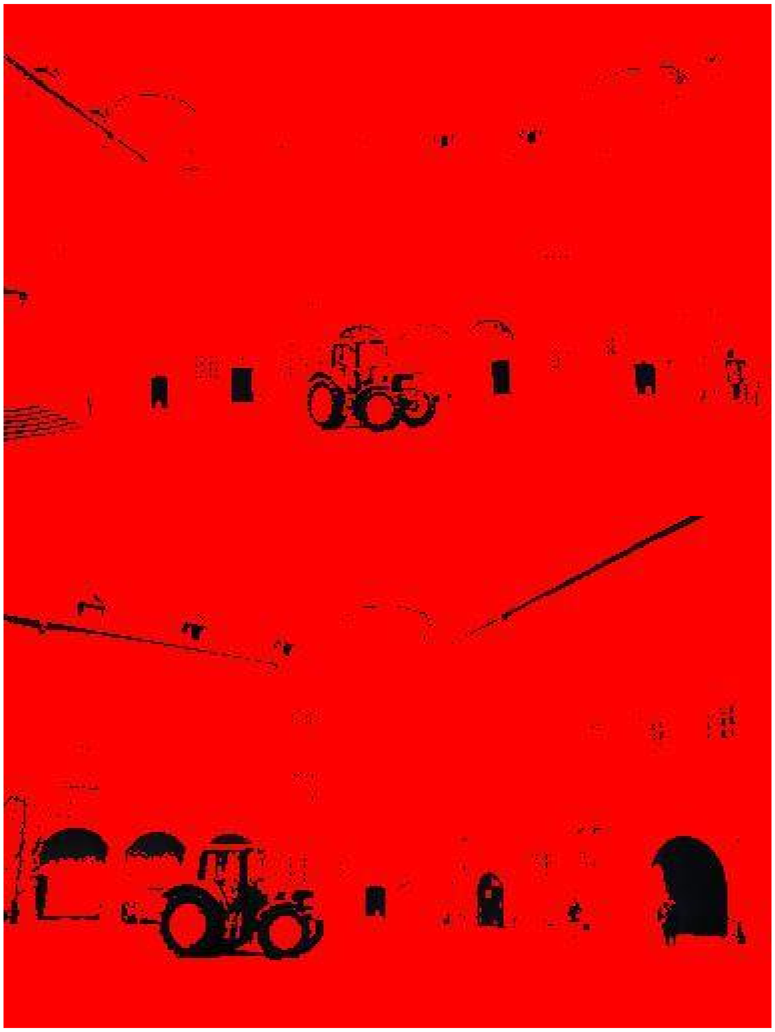
Kunal Dhaimade - CS 558 Optional Problem 2



Kunal Dhaimade - CS 558 Optional Problem 2



Kunal Dhaimade - CS 558 Optional Problem 2

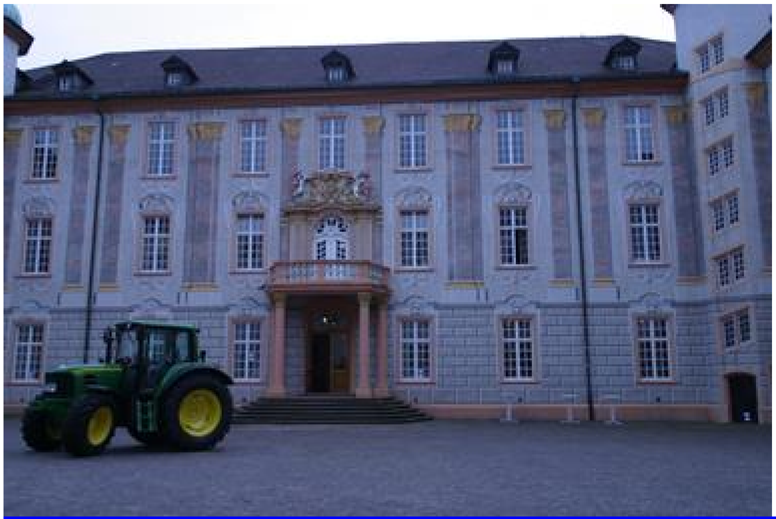


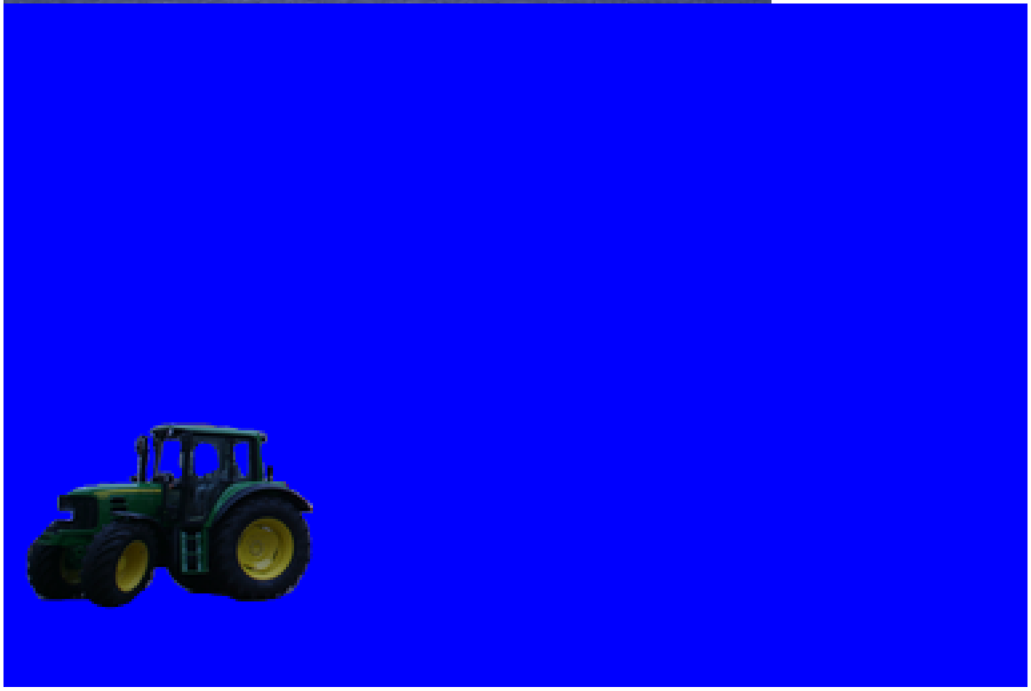
Kunal Dhaimade - CS 558 Optional Problem 2

Notes:

Object Segmentation:

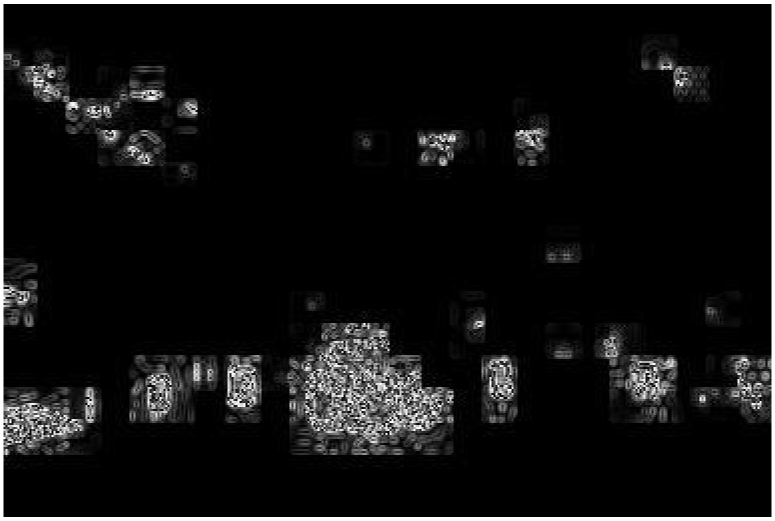
I have created a framework as same as the Pixel Classifier in the HW4. I have created a mask image for the background in order to highlight the tractor object in the foreground. The Pixel Classifier is trained using this training image, and the rest of the images are tested using it. Following are the training images:

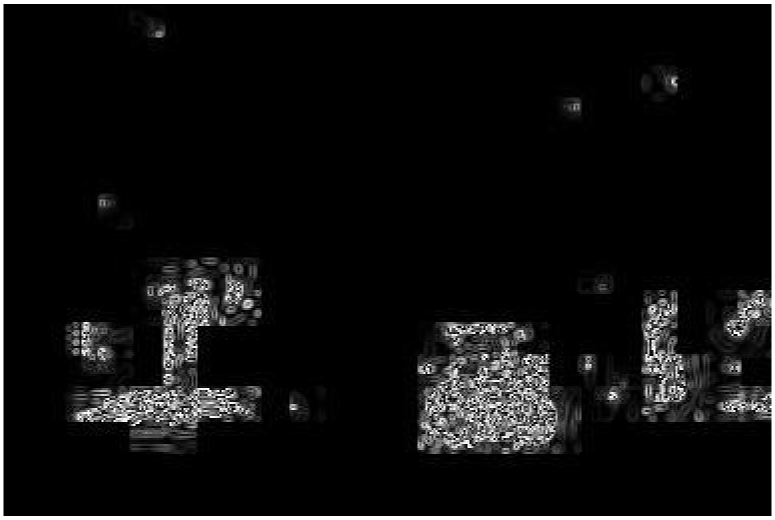




K-means algorithm is used to form visual words where k = 4. Accordingly, the pixels are colored red if they are close to the background words, else they are maintained as they are. This lead to many dark areas in the background being left colored as the black wheels of the tractor contribute to a dark color in the corresponding bag of words. To minimize these regions, pixels are checked for their value, and only the darkest of pixels are allowed, while the rest are colored red. However, as noticed in the output images, the solution obtained is far from perfect. However, there is a lot of scope for improvement here.

The gradient image for a few of the output images looks as follows:





As noticed, the region where the tractor is in the image, forms a dense contiguous region in the gradient image. This region can be detected using a blob detector to identify the location of tractor pixels, and can be used as a mask to color the rest of the pixels (background) with any other color such as red.