Pepito, Alyssa Mae & Soleño, Keziah Antonette 06-21-2022

CS171 | BM1

Assignment 3

Full Code

```
from random import random
from statistics import median, variance
from skimage import measure, io, img_as_ubyte
from matplotlib import pyplot as plt
from skimage.color import label2rgb, rgb2gray
import numpy as np
import cv2
import pandas as pd

# The input image
image = cv2.imread("images/image01/image01.jpg", 0)
# plt.imsave("images/image01/image01_gray.jpg", image, cmap = 'gray')

scale = 0.6 # microns/pixel

# display image measurements as histogram
# plt.hist(image.flat, bins=100, range=(0,255)) #.flat returns the flattened
numpy arra (1D)

from skimage.filters import threshold_otsu # binarization of image
threshold = threshold_otsu(image)

# Generate thresholded image
thresholded_img = image < threshold
plt.imshow(thresholded_img, cmap = 'gray')

# Command to save thresholded image
plt.imsave("images/image01/thresholded_gray_image01.jpg", thresholded_img, cmap =
'gray') # cmap = 'gray' may or may not be added
plt.imsave("images/image01/thresholded_image01.jpg", thresholded_img)

from skimage.segmentation import clear_border

# Remove edge touching regions
edge_touching_removed = clear_border(thresholded_img)
plt.imshow(edge_touching_removed, cmap = 'gray')

# Command to save removed border image
```

```python
plt.imsave("images/image01/edge_touching_removed_gray_image01.jpg",
edge_touching_removed, cmap = 'gray') # cmap = 'gray' may or may not be added
plt.imsave("images/image01/edge_touching_removed_image01.jpg",
edge_touching_removed)

# Label connected regions of an integer array using measure.label
# Labels each connected entity as one object
# Connectivity = Maximum number of orthogonal hops to consider a pixel/voxel as a
neighbor.
# If none, a full connectivity of input.ndim is used, number of dimension of the
image
# For 2D image it would be 2

label_image = measure.label(edge_touching_removed, connectivity=image.ndim)
plt.imshow(label_image)
plt.imsave("images/image01/labeled_image01_without_rgb.jpg", label_image, cmap =
'gray')
plt.imsave("images/image01/labeled_image01_with_rgb.jpg", label_image)

# Return an RGB image where color-coded labels are painted over the image.
# Using label2rgb

image_label_overlay = label2rgb(label_image, image=image)
plt.imshow(image_label_overlay)

# Commands to save the labeled image
plt.imsave("images/image01/labeled_image01.jpg", image_label_overlay)

# Compute image properties and return them as a pandas-compatible table.
# Available regionprops: area, bbox, centroid, convex_area, coords, eccentricity,
# equivalent diamater, euler number, label, intensity image, major axis length,
# max intensity, mean intensity, moments, orientation, perimeter, solidity, and
many more.

props = measure.regionprops_table(label_image, image,
                      properties=['label',
                                  'area', 'equivalent_diameter',
                                  'mean_intensity', 'solidity'] )

# .DataFrame() converts image properties above to a .csv file
df = pd.DataFrame(props)
print(df.head())
df.to_csv("images/image01/image_properties.csv")

# To delete small regions
```

```
#df2 = df[df['area'] > 20] # 20 pixels
#print(df2.head())
#df2.to_csv("deleted_small_regions.csv")

# Convert to micron scale
df['area_sq_microns'] = df['area'] * (scale**2)
df['equivalent_diameter_microns'] = df['equivalent_diameter'] * (scale)
print(df.head())

# Blob Detector for segmentation based on feature properties

# Set up the SimpleBlobDetector with default parameters
params = cv2.SimpleBlobDetector_Params()

# Define thresholds
params.minThreshold = 0
params.maxThreshold = 255

# Filter by area
params.filterByArea = True
params.minArea = 20
params.maxArea = 1000

# Filter by color (black = 0)
params.filterByColor = True #set True for image01.jpg as we'll be detecting black
regions
params.blobColor = 0

# Filter by Circularity
# params.filterbyCircularity = True
# params.minCircularity = 0.5
# params.maxCircularity = 1

# Filter by Convexity
params.filterByConvexity = True
params.minConvexity = 0.5
params.maxConvexity = 1

# Filter by InertiaRatio
# params.filterByInertia = True
# params.minIntertiaRatio = 0
# params.maxIntertiaRatio = 1

# Distance Between Blobs
params.minDistBetweenBlobs = 0
```

```
# Setup the detector with parameters
detector = cv2.SimpleBlobDetector_create(params)

# Detect blobs
keypoints = detector.detect(image)

print("Number of blobs detected are : ", len(keypoints))

# Draw blobs
img_with_blobs = cv2.drawKeypoints(image, keypoints, np.array([]), (0,0,255),
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.imshow(img_with_blobs)
cv2.imshow("Keypoints", img_with_blobs)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Save result
cv2.imwrite("images/image01/image01_blobs.jpg", img_with_blobs)

# Extracting features using Gabor filters

image1 = cv2.imread("for-instance-segmentation/Gabor95_rawprediction.ome.tiff")
image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY) # to convert image to grayscale

# Save original image pixels into a data frame. This is Feature #1.
image2 = image1.reshape(-1)
df3 = pd.DataFrame()
df3['Original Image'] = image2

# Generate Gabor features
num = 1 #To count numbers up to give Gabor features a label in the data frame
kernels = [] # Create empty list to hold all kernels that we will generate in a
loop
for theta in range(4): # Define number of thetas. Here only 2 theta values 0 and
1/4 . pi
    theta = theta / 4. * np.pi
    for sigma in (1, 3, 5): # Sigma with values of 1 and 3
        for lamda in np.arange(0, np.pi, np.pi / 4): # range of wavelengths
            for gamma in (0.05, 0.5): # Gamma values of 0.05 and 0.5

                gabor_label = 'Gabor' + str(num) # label Gabor columns as Gabor1,
Gabor2, etc.
                print(gabor_label)
                ksize = 9 # Try 15 for hidden image (or 9 for others)
```

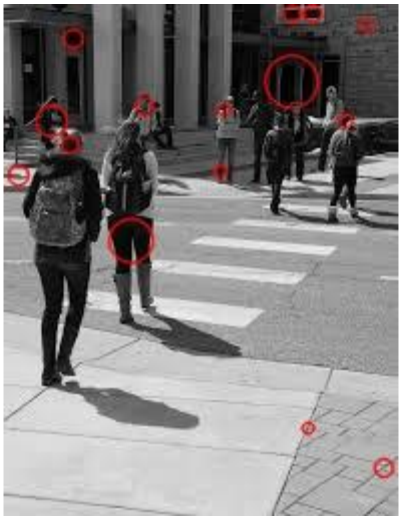
```
                phi = 0 # 0.8 for hidden image, otherwise leave it at 0
                kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda,
gamma, phi, ktype = cv2.CV_32F)
                kernels.append(kernel)

                # Now filter the image and add values to a new column
                fimg = cv2.filter2D(image1, cv2.CV_8UC3, kernel) # originally
image2 but was checked
                filtered_img = fimg.reshape(-1)

                #
cv2.imwrite("images/image01/gabor_filtered_image01/"+gabor_label+".jpg",
filtered_img.reshape(image1.shape))

                df3[gabor_label] = filtered_img # labels columns as Gabor1,
Gabor2, etc.
                print(gabor_label, ': theta=', theta, ': sigma=', sigma, ':
lamda=', lamda, ': gamma=', gamma)

                num += 1 # to increment Gabor column label

#print(df3.head())

#df3.to_csv("images/image01/Gabor3.csv")

######

# Generate other features and add them to the data frame

# CANNY EDGE
edges = cv2.Canny(image1, 100,200) # image, min. and max values
edges1 = edges.reshape(-1)
df3['Canny Edge'] = edges1 # add column to original dataframe

from skimage.filters import roberts, sobel, scharr, prewitt

# ROBERTS EDGE
edge_roberts = roberts(image1)
edge_roberts1 = edge_roberts.reshape(-1)
df3['Roberts'] = edge_roberts1

# SOBEL
edge_sobel = sobel(image1)
edge_sobel1 = edge_sobel.reshape(-1)
df3['Sobel'] = edge_sobel1
```

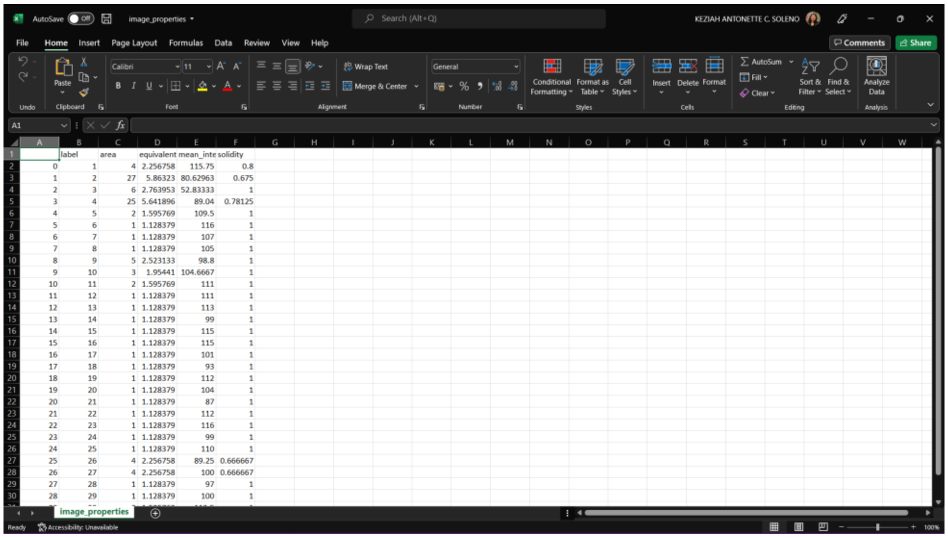
```python
# SCHARR
edge_scharr = scharr(image1)
edge_scharr1 = edge_scharr.reshape(-1)
df3['Scharr'] = edge_scharr1

# PREWITT
edge_prewitt = prewitt(image1)
edge_prewitt1 = edge_prewitt.reshape(-1)
df3['Prewitt'] = edge_prewitt1

# GAUSSIAN with sigma3
from scipy import ndimage as nd
gaussian_img = nd.gaussian_filter(image1, sigma=3)
gaussian_img1 = gaussian_img.reshape(-1)
df3['Gaussian s3'] = gaussian_img1

# GAUSSIAN with sigma7
gaussian_img2 = nd.gaussian_filter(image1, sigma=7)
gaussian_img3 = gaussian_img2.reshape(-1)
df3['Gaussian s7'] = gaussian_img3

# MEDIAN with size = 3
median_img = nd.median_filter(image1, size=3)
median_img1 = median_img.reshape(-1)
df3['Median s3'] = median_img1

# VARIANCE with size=3
#variance_img = nd.generic_filter(image1, np.var, size=3)
#variance_img1 = variance_img.reshape(-1)
#df3['Variance s3'] = variance_img1 # Add comlumn to original datafram

# Now, add a column in the data frame for the labels
# For this, we need to import the labeled image (mask)

labeled_img = cv2.imread('for-instance-segmentation/Gabor95_binarymask.tiff')

# Remember that you can load an image with partial labels
# But, drop the rows with unlabeled data

labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_BGR2GRAY)
labeled_img1 = labeled_img.reshape(-1)
df3['Labels'] = labeled_img1 # originally labeled_img1

print(df3.head())
```

```
original_img_data = df3.drop(labels = ["Labels"], axis=1) # Use for prediction
#df3.to_csv("Gabor_image01.csv")
df3 = df3[df3.Labels != 0]

#################################

# Define the dependent variable that needs to be predicted (labels)
Y = df3["Labels"].values

# Encode Y values to 0, 1, 2, 3... (Not necessary but makes it easy to use other
tools..)
# from sklearn.preprocessing import LabelEncoder # import necessary library first
# Y = LabelEncoder().fit_transform(Y)

# Define the independent variables
X = df3.drop(labels = ["Labels"], axis = 1)

# Split data into train and test to verify accuracy after fitting the model.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,
random_state = 20)

# Import the model we are using
# RandomForestRegressor is for regression type of problem.
# For classification we use RandomForestClassifier.
# Both yield similar results except for regressor the result is float
# and for classifier it is an integer.

from sklearn.ensemble import RandomForestClassifier
# Instantiate model with n number of decision trees
model = RandomForestClassifier(n_estimators = 20, random_state = 42)

# Train the model on training data
model.fit(X_train, y_train)

# Get numerical featuyre importances
# importances = list(model.feature_importances_)

# let us print them into a nice format.

feature_list = list(X.columns)
feature_imp = pd.Series(model.feature_importances_,
index=feature_list).sort_values(ascending = False)
print(feature_imp)
```

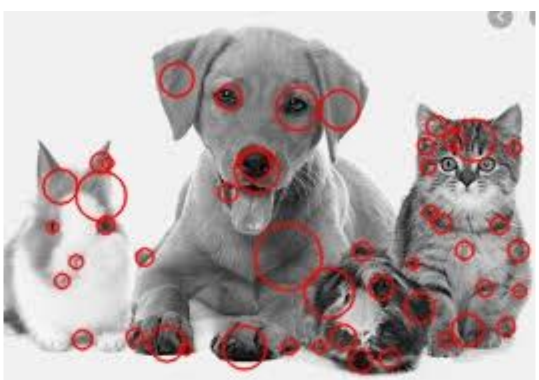
```
################################################

# MAKE PREDICTION
# You can store the model for future use. In fact, this is how you do machine
learning.
# Train on training images, validate on test images and deploy the model on
unknown images

import pickle

# Save the trained model as pickle string to disk for future use
filename = "animals_model"
pickle.dump(model, open(filename, 'wb'))

# To test the model on future datasets
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.predict(original_img_data)

segmented = result.reshape((image1.shape))

plt.imshow(segmented, cmap='jet')
plt.imsave('images/image01/image01_feature_extraction4.jpg', segmented,
cmap='jet')

            ########### INSTANCE SEGMENTATION USING BBOXES ###########

import pixellib
from pixellib.instance import instance_segmentation

segment_image = instance_segmentation()
segment_image.load_model("mask_rcnn_coco.h5")
segment_image.segmentImage("images/image05.jpg", show_bboxes = True,
output_image_name = "images/image05_bbox.jpg")
```

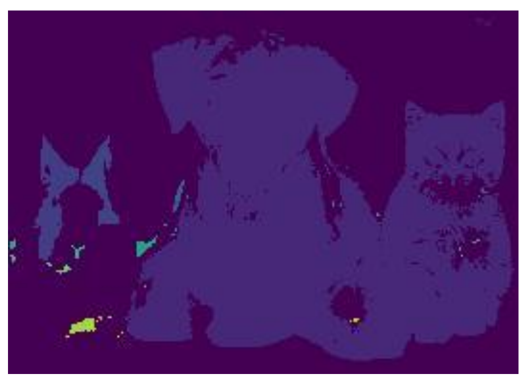
Outputs

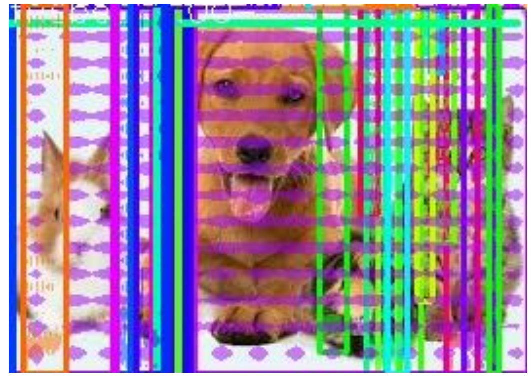


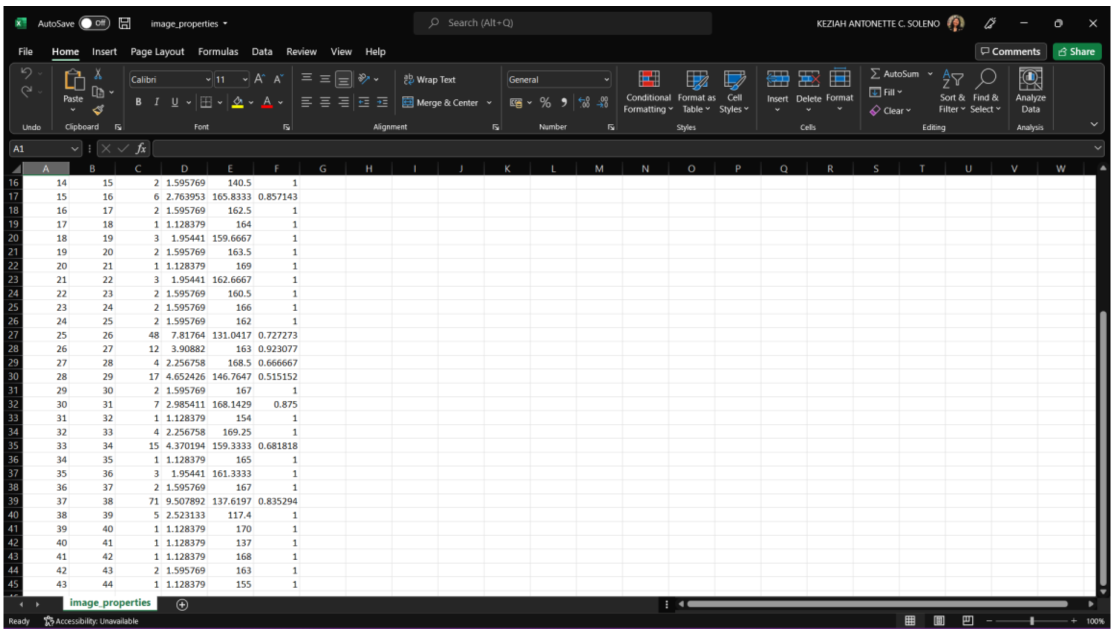


	label	area	equivalent	mean_inte	solidity
0	1	4	2.256758	115.75	0.8
1	2	27	5.86323	80.62963	0.675
2	3	6	2.763953	52.83333	1
3	4	25	5.641896	89.04	0.78125
4	5	2	1.595769	109.5	1
5	6	1	1.128379	116	1
6	7	1	1.128379	107	1
7	8	1	1.128379	105	1
8	9	5	2.523133	98.8	1
9	10	3	1.95441	104.6667	1
10	11	2	1.595769	111	1
11	12	1	1.128379	111	1
12	13	1	1.128379	113	1
13	14	1	1.128379	99	1
14	15	1	1.128379	115	1
15	16	1	1.128379	115	1
16	17	1	1.128379	101	1
17	18	1	1.128379	93	1
18	19	1	1.128379	112	1
19	20	1	1.128379	104	1
20	21	1	1.128379	87	1
21	22	1	1.128379	112	1
22	23	1	1.128379	116	1
23	24	1	1.128379	99	1
24	25	1	1.128379	110	1
25	26	4	2.256758	89.25	0.666667
26	27	4	2.256758	100	0.666667
27	28	1	1.128379	97	1
28	29	1	1.128379	100	1



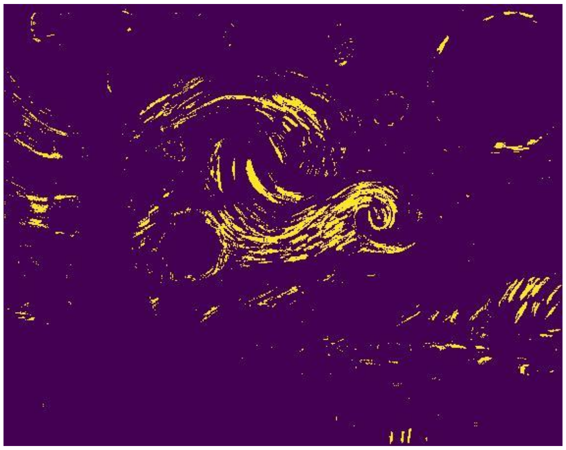


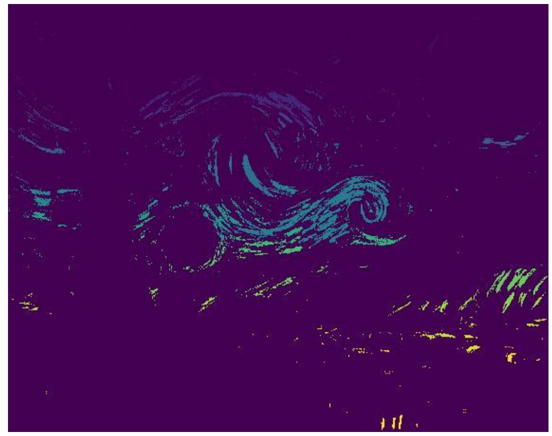


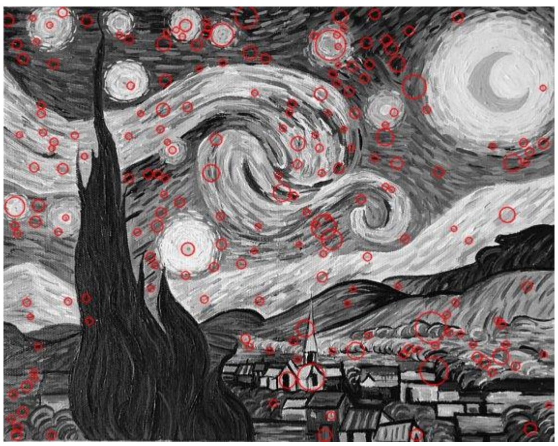


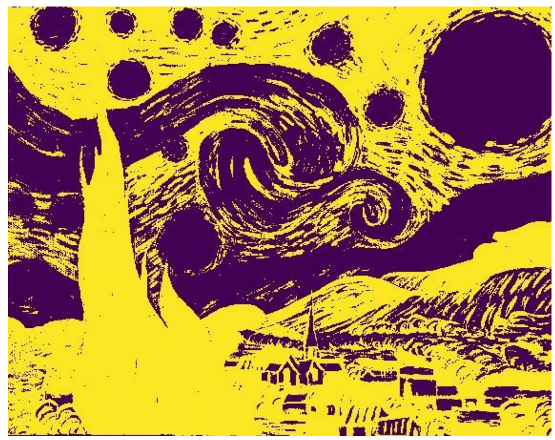
	A	B	C	D	E	F
16	14	15	2	1.595769	140.5	1
17	15	16	6	2.763953	165.8333	0.857143
18	16	17	2	1.595769	162.5	1
19	17	18	1	1.128379	164	1
20	18	19	3	1.95441	159.6667	1
21	19	20	2	1.595769	163.5	1
22	20	21	1	1.128379	169	1
23	21	22	3	1.95441	162.6667	1
24	22	23	2	1.595769	160.5	1
25	23	24	2	1.595769	166	1
26	24	25	2	1.595769	162	1
27	25	26	48	7.81764	131.0417	0.727273
28	26	27	12	3.90882	163	0.923077
29	27	28	4	2.256758	168.5	0.666667
30	28	29	17	4.652426	146.7647	0.515152
31	29	30	2	1.595769	167	1
32	30	31	7	2.985411	168.1429	0.875
33	31	32	1	1.128379	154	1
34	32	33	4	2.256758	169.25	1
35	33	34	15	4.370194	159.3333	0.681818
36	34	35	1	1.128379	165	1
37	35	36	3	1.95441	161.3333	1
38	36	37	2	1.595769	167	1
39	37	38	71	9.507892	137.6197	0.835294
40	38	39	5	2.523133	117.4	1
41	39	40	1	1.128379	170	1
42	40	41	1	1.128379	137	1
43	41	42	1	1.128379	168	1
44	42	43	2	1.595769	163	1
45	43	44	1	1.128379	155	1

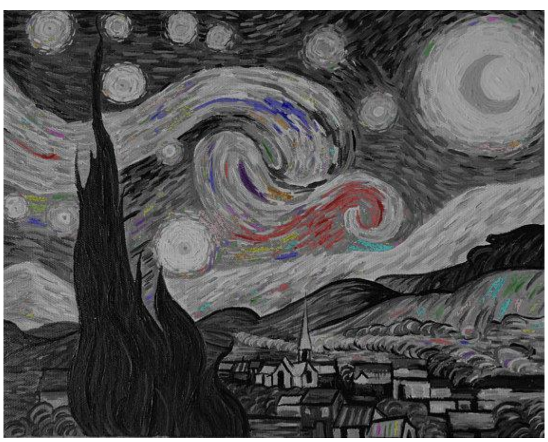
image_properties

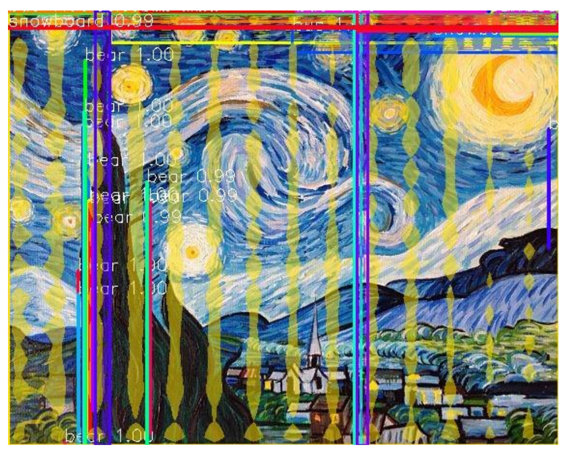


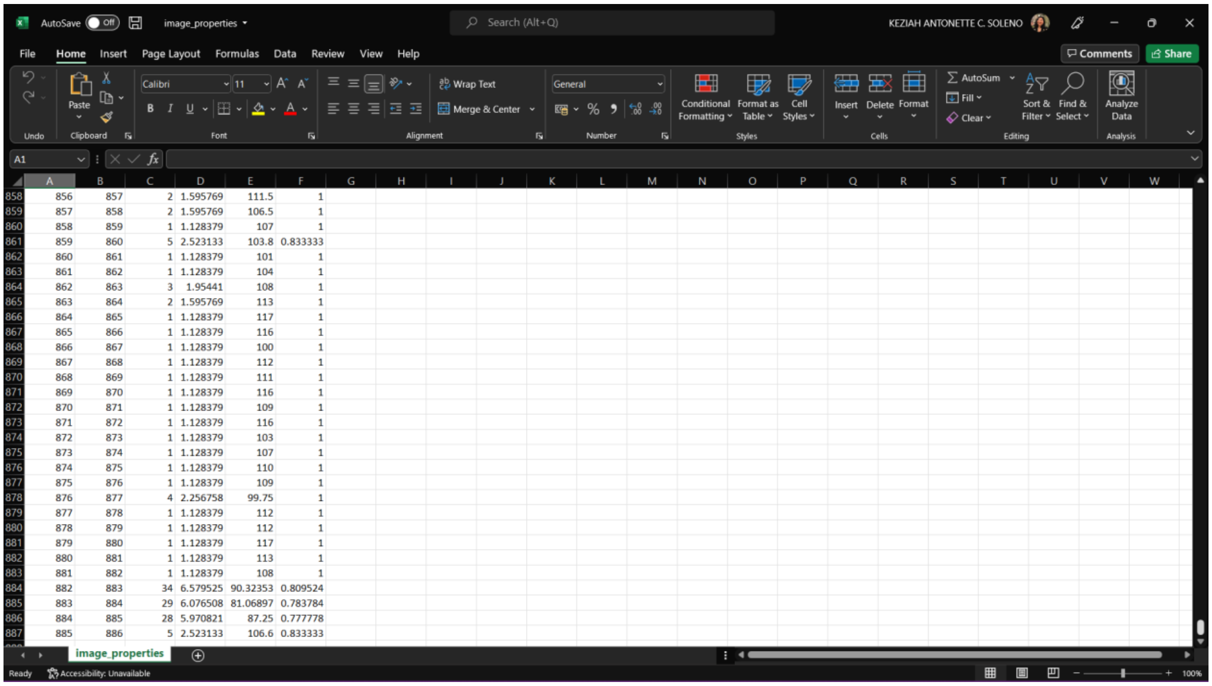









snowboard 0.99
bear 1.00
bear 0.99



	A	B	C	D	E	F
858	856	857	2	1.595769	111.5	1
859	857	858	2	1.595769	106.5	1
860	858	859	1	1.128379	107	1
861	859	860	5	2.523133	103.8	0.833333
862	860	861	1	1.128379	101	1
863	861	862	1	1.128379	104	1
864	862	863	3	1.95441	108	1
865	863	864	2	1.595769	113	1
866	864	865	1	1.128379	117	1
867	865	866	1	1.128379	116	1
868	866	867	1	1.128379	100	1
869	867	868	1	1.128379	112	1
870	868	869	1	1.128379	111	1
871	869	870	1	1.128379	116	1
872	870	871	1	1.128379	109	1
873	871	872	1	1.128379	116	1
874	872	873	1	1.128379	103	1
875	873	874	1	1.128379	107	1
876	874	875	1	1.128379	110	1
877	875	876	1	1.128379	109	1
878	876	877	4	2.256758	99.75	1
879	877	878	1	1.128379	112	1
880	878	879	1	1.128379	112	1
881	879	880	1	1.128379	117	1
882	880	881	1	1.128379	113	1
883	881	882	1	1.128379	108	1
884	882	883	34	6.579525	90.32353	0.809524
885	883	884	29	6.076508	81.06897	0.783784
886	884	885	28	5.970821	87.25	0.777778
887	885	886	5	2.523133	106.6	0.833333

image_properties