CSC 410 Assignment 1: Raw Data to Feature Space

Kidest Mamo
UNC–Greensboro

1 INTRODUCTION

In this assignment, a set of raw images were transformed into a feature space.

2 TASK 1

I already had the Anaconda Python Environment on my PC, which comes with the Jupyter Notebook. I didn't install Spyder as I prefer to use the Notebook. I installed openCV with the following command:

```
conda install cv2
```

Image of setup is depicted in Figure 1.

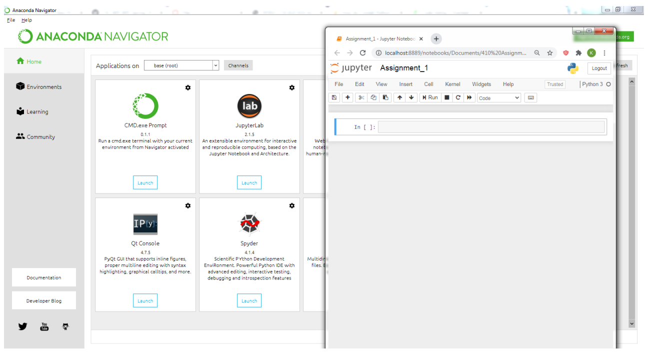


Figure 1: Programming Environment Setup.

3 TASK 2

My three chosen types of fruit are cherry, banana, and Lime. I chose these because they are very distinct from each other in shape and color, and will hopefully be easier to train on. These images were chosen from the FIDS30 dataset (1).

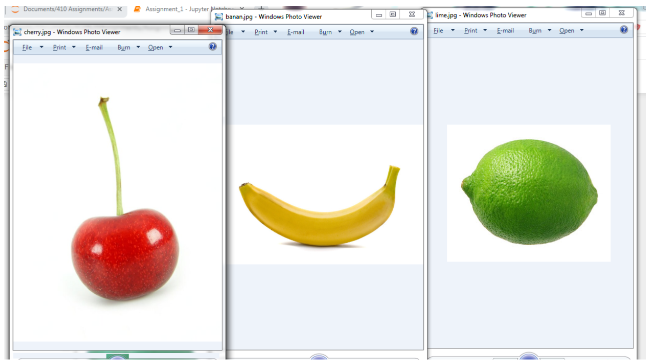
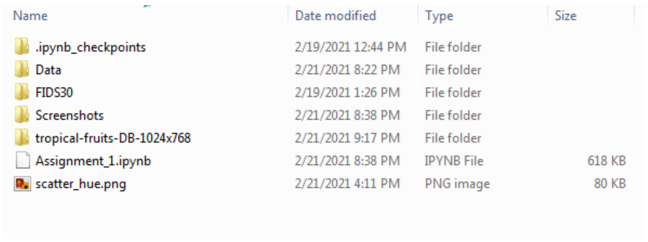

cherry.jpg - Windows Photo Viewer
banan.jpg - Windows Photo Viewer
lime.jpg - Windows Photo Viewer

Figure 2: Image Data-sets used in this project

Conference'17, July 2017, Washington, DC, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn



Name	Date modified	Type	Size
.ipynb_checkpoints	2/19/2021 12:44 PM	File folder	
Data	2/21/2021 8:22 PM	File folder	
FIDS30	2/19/2021 1:26 PM	File folder	
Screenshots	2/21/2021 8:38 PM	File folder	
tropical-fruits-DB-1024x768	2/21/2021 9:17 PM	File folder	
Assignment_1.ipynb	2/21/2021 8:38 PM	IPYNB File	618 KB
scatter_hue.png	2/21/2021 4:11 PM	PNG image	80 KB

Figure 3: Exact images used in this assignment. Left to right: cherry, banana, lime.

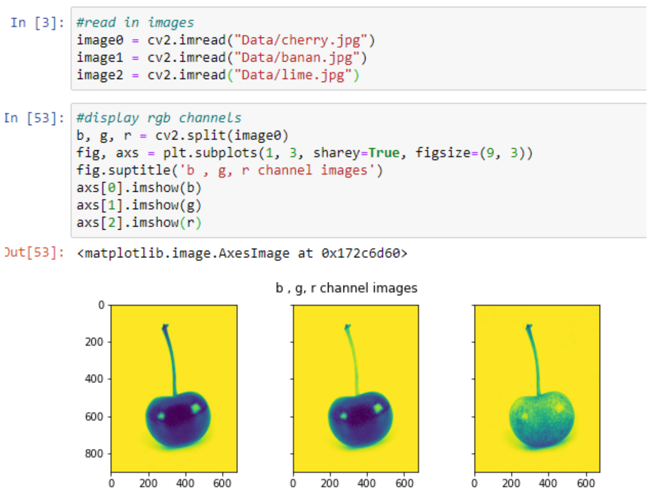
4 TASK 3

In this section suitable Python and OpenCV libraries were added to perform this task and the rest of this assignment. The libraries used are numpy, pandas, matplotlib, and openCV.

```
import numpy as np
import cv2
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap
```

Figure 4: Imported Libraries

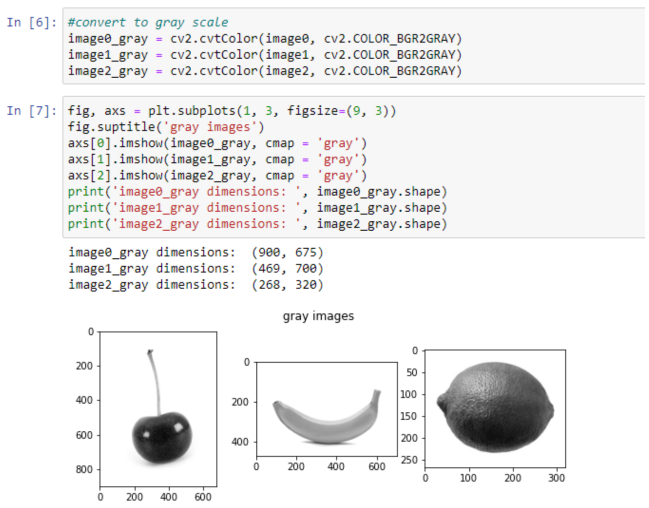
Then, the three chosen color images were read in and their R, G, and B channel images were displayed. An example result is shown in Figure 10.



```
In [3]: #read in images
        image0 = cv2.imread("Data/cherry.jpg")
        image1 = cv2.imread("Data/banan.jpg")
        image2 = cv2.imread("Data/lime.jpg")

In [53]: #display rgb channels
         b, g, r = cv2.split(image0)
         fig, axs = plt.subplots(1, 3, sharey=True, figsize=(9, 3))
         fig.suptitle('b , g, r channel images')
         axs[0].imshow(b)
         axs[1].imshow(g)
         axs[2].imshow(r)

Out[53]: <matplotlib.image.AxesImage at 0x172c6d60>
```

Figure 5: Reading in images and displaying the r,b,g color channels of image0.

They were then converted into gray-scale and displayed while printing out their dimensions as shown in Figure 6.

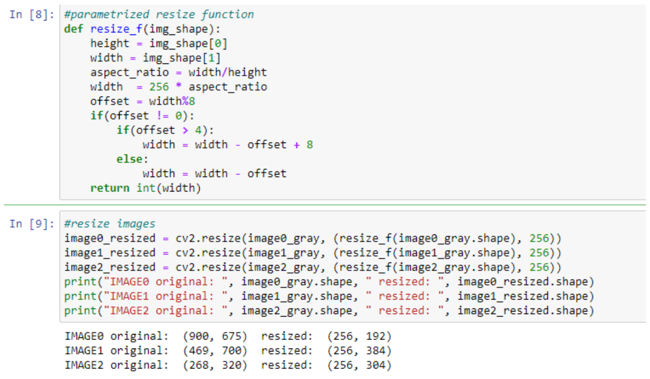


```
In [6]: #convert to gray scale
        image0_gray = cv2.cvtColor(image0, cv2.COLOR_BGR2GRAY)
        image1_gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
        image2_gray = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

In [7]: fig, axs = plt.subplots(1, 3, figsize=(9, 3))
        fig.suptitle('gray images')
        axs[0].imshow(image0_gray, cmap = 'gray')
        axs[1].imshow(image1_gray, cmap = 'gray')
        axs[2].imshow(image2_gray, cmap = 'gray')
        print('image0_gray dimensions: ', image0_gray.shape)
        print('image1_gray dimensions: ', image1_gray.shape)
        print('image2_gray dimensions: ', image2_gray.shape)

        image0_gray dimensions:  (900, 675)
        image1_gray dimensions:  (469, 700)
        image2_gray dimensions:  (268, 320)
```

Figure 6: Display of gray-scale images and their dimensions. From left to right: image0, image1, and image2.

5 TASK 4

In this section, parametric function was written to reduce the size of the gray-scale images to have height of 256 pixels while maintaining the aspect ratio for the width keeping it divisible by 8. The function and the resulting dimensions of the resized images is shown in Figure 7.



```
In [8]: #parametrized resize function
        def resize_f(img_shape):
            height = img_shape[0]
            width = img_shape[1]
            aspect_ratio = width/height
            width  = 256 * aspect_ratio
            offset = width%8
            if(offset != 0):
                if(offset > 4):
                    width = width - offset + 8
                else:
                    width = width - offset
            return int(width)

In [9]: #resize images
        image0_resized = cv2.resize(image0_gray, (resize_f(image0_gray.shape), 256))
        image1_resized = cv2.resize(image1_gray, (resize_f(image1_gray.shape), 256))
        image2_resized = cv2.resize(image2_gray, (resize_f(image2_gray.shape), 256))
        print("IMAGE0 original: ", image0_gray.shape, " resized: ", image0_resized.shape)
        print("IMAGE1 original: ", image1_gray.shape, " resized: ", image1_resized.shape)
        print("IMAGE2 original: ", image2_gray.shape, " resized: ", image2_resized.shape)

        IMAGE0 original:  (900, 675)  resized:  (256, 192)
        IMAGE1 original:  (469, 700)  resized:  (256, 384)
        IMAGE2 original:  (268, 320)  resized:  (256, 304)
```

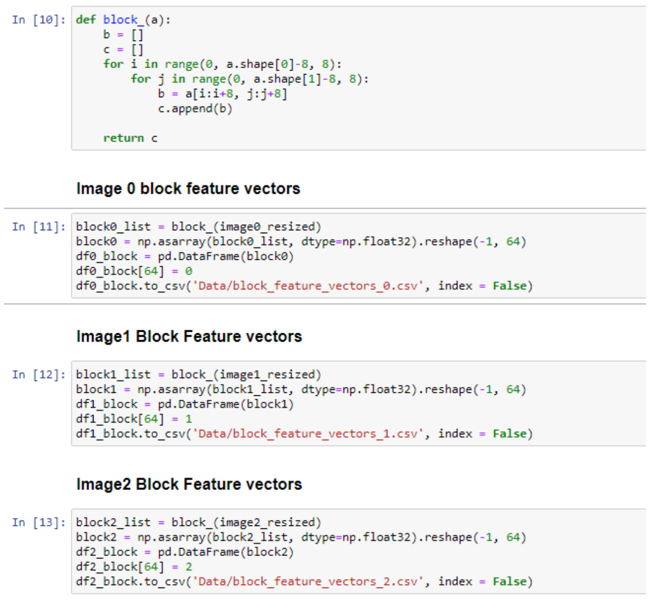
Figure 7: Resize function and resulting dimensions.

6 TASK 5

In this section, a function made to divide each image into blocks of 8x8 pixels. The blocks were then transformed to vectors of size 64. A label was assigned to each feature vector with 0, 1, and 2 for the first, second, and third images, respectively. A spreadsheet was then generated by storing a feature vector per row in the spreadsheet for each image as shown in Figure 8.

7 TASK 6

In this section, a function was made to divide each image into overlapping blocks of 8x8 pixels. The blocks were then transformed to vectors of size 64. A label was assigned to each feature vector with 0, 1, and 2 for the first, second, and third images, respectively.



```
In [10]: def block_(a):
             b = []
             c = []
             for i in range(0, a.shape[0]-8, 8):
                 for j in range(0, a.shape[1]-8, 8):
                     b = a[i:i+8, j:j+8]
                     c.append(b)

             return c
```

Image 0 block feature vectors

```
In [11]: block0_list = block_(image0_resized)
         block0 = np.asarray(block0_list, dtype=np.float32).reshape(-1, 64)
         df0_block = pd.DataFrame(block0)
         df0_block[64] = 0
         df0_block.to_csv('Data/block_feature_vectors_0.csv', index = False)
```

Image1 Block Feature vectors

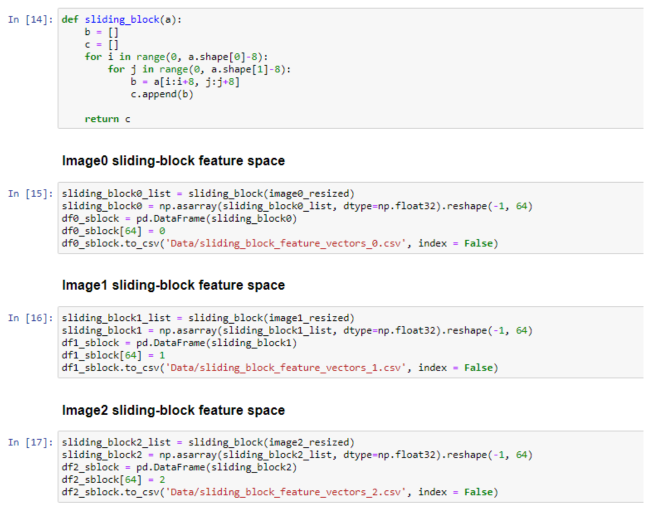
```
In [12]: block1_list = block_(image1_resized)
         block1 = np.asarray(block1_list, dtype=np.float32).reshape(-1, 64)
         df1_block = pd.DataFrame(block1)
         df1_block[64] = 1
         df1_block.to_csv('Data/block_feature_vectors_1.csv', index = False)
```

Image2 Block Feature vectors

```
In [13]: block2_list = block_(image2_resized)
         block2 = np.asarray(block2_list, dtype=np.float32).reshape(-1, 64)
         df2_block = pd.DataFrame(block2)
         df2_block[64] = 2
         df2_block.to_csv('Data/block_feature_vectors_2.csv', index = False)
```

Figure 8: Function block. Block feature Vector generation for each image.

A spreadsheet was then generated by storing a feature vector per row in the spreadsheet for each image as shown in Figure ??.



```
In [14]: def sliding_block(a):
             b = []
             c = []
             for i in range(0, a.shape[0]-8):
                 for j in range(0, a.shape[1]-8):
                     b = a[i:i+8, j:j+8]
                     c.append(b)

             return c
```

Image0 sliding-block feature space

```
In [15]: sliding_block0_list = sliding_block(image0_resized)
         sliding_block0 = np.asarray(sliding_block0_list, dtype=np.float32).reshape(-1, 64)
         df0_sblock = pd.DataFrame(sliding_block0)
         df0_sblock[64] = 0
         df0_sblock.to_csv('Data/sliding_block_feature_vectors_0.csv', index = False)
```

Image1 sliding-block feature space

```
In [16]: sliding_block1_list = sliding_block(image1_resized)
         sliding_block1 = np.asarray(sliding_block1_list, dtype=np.float32).reshape(-1, 64)
         df1_sblock = pd.DataFrame(sliding_block1)
         df1_sblock[64] = 1
         df1_sblock.to_csv('Data/sliding_block_feature_vectors_1.csv', index = False)
```

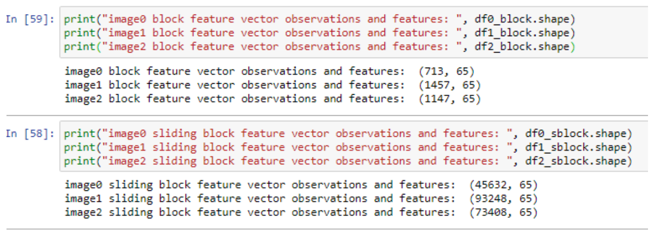
Image2 sliding-block feature space

```
In [17]: sliding_block2_list = sliding_block(image2_resized)
         sliding_block2 = np.asarray(sliding_block2_list, dtype=np.float32).reshape(-1, 64)
         df2_sblock = pd.DataFrame(sliding_block2)
         df2_sblock[64] = 2
         df2_sblock.to_csv('Data/sliding_block_feature_vectors_2.csv', index = False)
```

Figure 9: Sliding block function. Sliding feature Vector generation for each image.

8 TASK 7

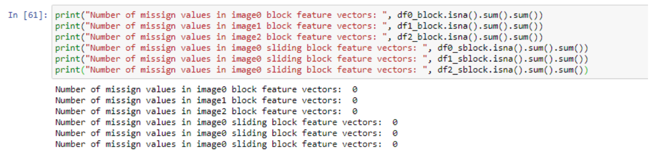
- Is the dataset imbalanced, inaccurate or incomplete?
  - The data set is imabalanced due to sized differences in the images, resulting in different number of observations in the feature vectors of each image.And if is specially so for the sliding block feature vectors where the number of observations for image1 were roughly double that of image0. The exact number of observations for each is given in Figure 10



```
In [59]: print("image0 block feature vector observations and features: ", df0_block.shape)
         print("image1 block feature vector observations and features: ", df1_block.shape)
         print("image2 block feature vector observations and features: ", df2_block.shape)

         image0 block feature vector observations and features:  (713, 65)
         image1 block feature vector observations and features:  (1457, 65)
         image2 block feature vector observations and features:  (1147, 65)

In [58]: print("image0 sliding block feature vector observations and features: ", df0_sblock.shape)
         print("image1 sliding block feature vector observations and features: ", df1_sblock.shape)
         print("image2 sliding block feature vector observations and features: ", df2_sblock.shape)

         image0 sliding block feature vector observations and features:  (45632, 65)
         image1 sliding block feature vector observations and features:  (93248, 65)
         image2 sliding block feature vector observations and features:  (73408, 65)
```

Figure 10: Number of observations and features of the feature vectors

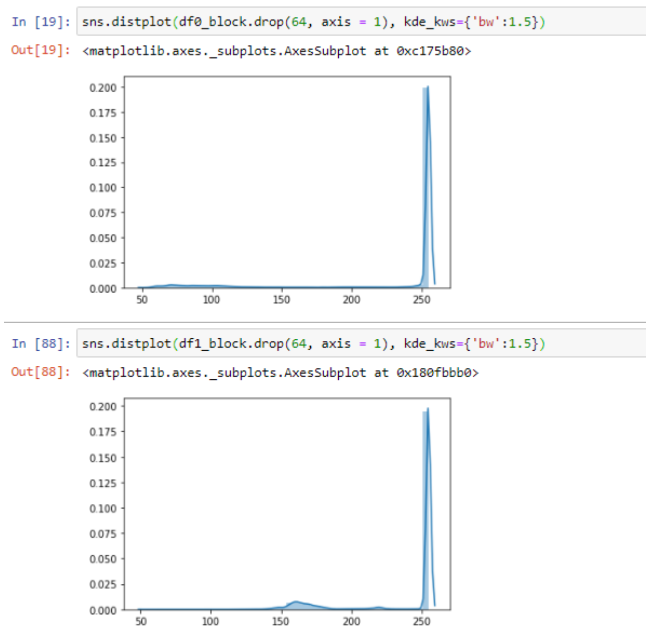
- The dataset is not inaccurate because all vectors corresponding to each image were labeled with the same value, separate from the others. Thus, there's no inaccurately labeled data.
- The dataset is not incomplete. There're no missing values as shown Figure 11



```
In [61]: print("Number of missign values in image0 block feature vectors: ", df0_block.isna().sum().sum())
         print("Number of missign values in image1 block feature vectors: ", df1_block.isna().sum().sum())
         print("Number of missign values in image2 block feature vectors: ", df2_block.isna().sum().sum())
         print("Number of missign values in image0 sliding block feature vectors: ", df0_sblock.isna().sum().sum())
         print("Number of missign values in image0 sliding block feature vectors: ", df1_sblock.isna().sum().sum())
         print("Number of missign values in image0 sliding block feature vectors: ", df2_sblock.isna().sum().sum())

         Number of missign values in image0 block feature vectors:  0
         Number of missign values in image1 block feature vectors:  0
         Number of missign values in image2 block feature vectors:  0
         Number of missign values in image0 sliding block feature vectors:  0
         Number of missign values in image0 sliding block feature vectors:  0
         Number of missign values in image0 sliding block feature vectors:  0
```

Figure 11: Number of observations and features of the feature vectors

- Is it a trivial data or possibly a big data? This is a big data. Although it's big in size it's not anything that a modern computer can't handle. It has a decent amount of features, but it's not high dimensional. However, it's very complex and the classes aren't easily separable. Classification can't be done with non-ML methods. Thus this can be considered to be a big data.
- Does it have scalability problem? There aren't/won't be scalability problems in the datasets that are and will be generated in this project because we're keeping the number of features constant. However, if a dataset generated by somehow who sampled their data differently, then there will be some issues.
- Are they high dimensional? This dataset is not high dimensional. As shown previously in Figure 10 each dataset has 64 features and the number of observations in each dataset is much greater than 64.
- Do you need to standardize? Do you need to normalize? How do they affect the data characteristics? The data does need to be transformed one of two ways since as shown in the Figure 14, Figure 15, Figure 16, Figure 17 the classes aren't easily separable. Transforming the data might bring out some useful patterns.
  - Do you need to normalize? Yes, as shown in the distributions of image0 and image1's distributions in Figure 12, the data doesn't follow a Gaussian distribution so normalization could be useful.
  - Do you need to standardize? No, standardization wouldn't be of use since the data doesn't follow a Gaussian distribution.
  - How do they affect the data characteristics? Standardization and normalization affect the observation characteristics by scaling the values of the observations. Size and volume, as well as any feature and class characteristics remain unaffected.

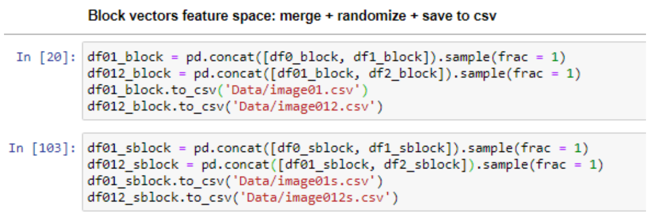


```
In [19]: sns.distplot(df0_block.drop(64, axis = 1), kde_kws={'bw':1.5})
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0xc175b80>

In [88]: sns.distplot(df1_block.drop(64, axis = 1), kde_kws={'bw':1.5})
Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x180fbbb0>
```

Figure 12: image0 and image1 distribution plots.

9 TASK 8

In this section, the feature vectors in image0.csv and image1.csv were first merged to create a feature space for these images. Each feature and label columns were align vertically to generate the correct feature space for these image classes. The placement of the feature vectors were then randomized and saved as a spreadsheet, image01.csv. Similarly the feature vectors in image0.csv, image1.csv, and image2.csv were merged to create a feature space for these images. Again, the placement of the feature vectors was randomized and the result was saved as a spreadsheet, image012.csv. This was repeated to create feature spaces from those of the sliding block feature vectors. A total of four feature spaces were constructed.



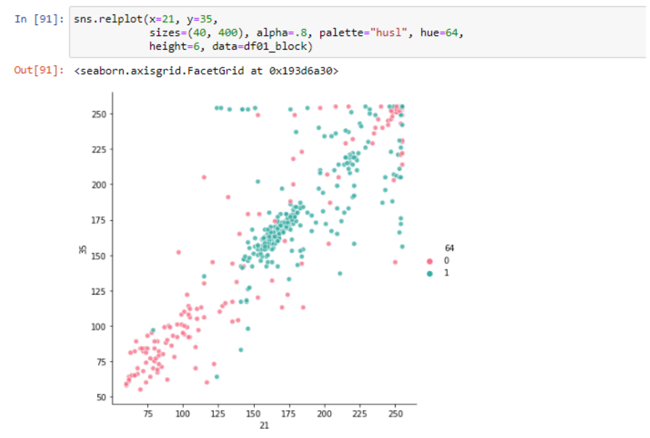
Block vectors feature space: merge + randomize + save to csv

```
In [20]: df01_block = pd.concat([df0_block, df1_block]).sample(frac = 1)
         df012_block = pd.concat([df01_block, df2_block]).sample(frac = 1)
         df01_block.to_csv('Data/image01.csv')
         df012_block.to_csv('Data/image012.csv')

In [103]: df01_sblock = pd.concat([df0_sblock, df1_sblock]).sample(frac = 1)
          df012_sblock = pd.concat([df01_sblock, df2_sblock]).sample(frac = 1)
          df01_sblock.to_csv('Data/image01s.csv')
          df012_sblock.to_csv('Data/image012s.csv')
```

Figure 13: Construction of the four feature spaces.

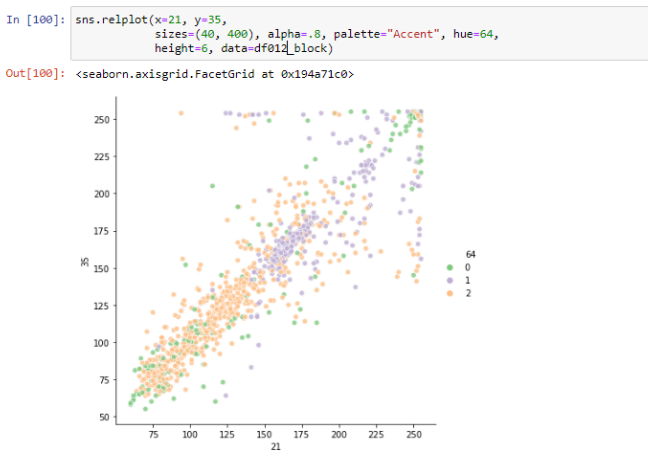
10 TASK 9

Two features were selected and the two-dimensional feature space was plotted, labeling the observations by color based on class(feature 64). One plot for 2 classes. Another for 3 classes. Three features were then selected and the three-dimensional feature space was plotted, labeling the observations by color based on class(feature 64). One plot for 2 classes. Another for 3 classes. None of the features as plotted in the above figures seem easily separated by class. There are too many overlaps.



```
In [91]: sns.relplot(x=21, y=35,
                     sizes=(40, 400), alpha=.8, palette="husl", hue=64,
                     height=6, data=df01_block)

Out[91]: <seaborn.axisgrid.FacetGrid at 0x193d6a30>
```

Figure 14: 2 feature plot for 2 classes 0 and 1.



```
In [100]: sns.relplot(x=21, y=35,
                      sizes=(40, 400), alpha=.8, palette="Accent", hue=64,
                      height=6, data=df012_block)

Out[100]: <seaborn.axisgrid.FacetGrid at 0x194a71c0>
```

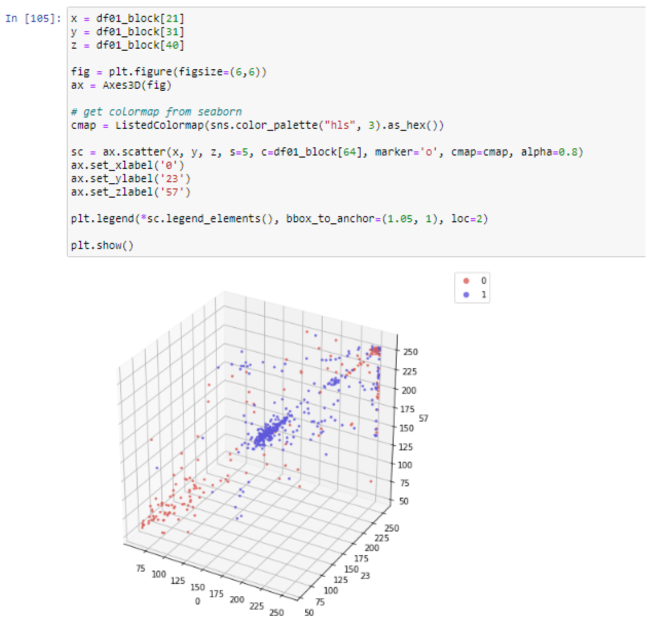
Figure 15: 2 feature plot for 3 classes 0, 1, and 2

11 TASK 10

Appropriate changes were made to the Python code such that it can read any number of images from a folder that consists of many similar images, generate a feature spaces, and generate a spreadsheets for the feature spaces. This function is depicted in Figure 18. An example is shown in Figure 19.

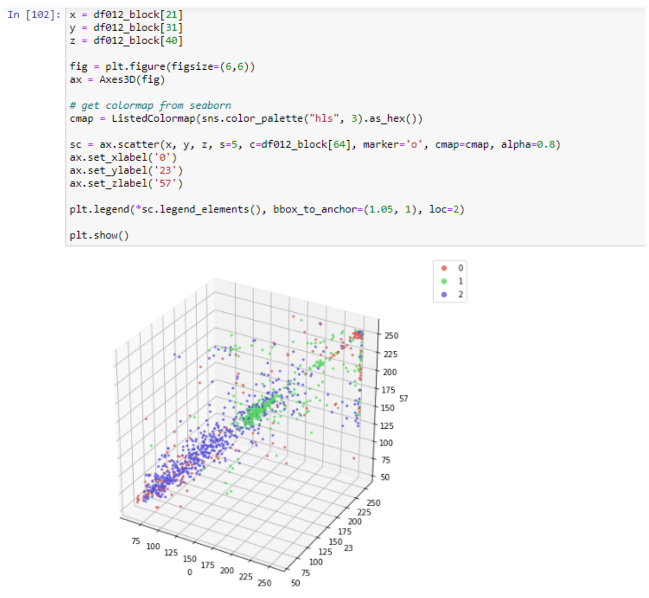
12 TASK 11

- Describe the effects of block size on the dimensionality of the feature space and the number of vectors. As block size increases, the higher the dimension(more features) and the lower the number of vectors that the feature space will have. The opposite is the case as block size decreases.
- Describe how these effect may influence the classifier that divides the domain First of all, as we increase the the block size, thus decrease the number of vectors, we're decreasing the number of vectors we have for training the classifier, which isn't ideal. Also, the increase in features, especially when it result in a high dimensional dataset will raise scalability issues and possibly render the results of the classifier useless. And thus, the dataset will first need to be processed to reduce its dimension before feeding it to the classifier.



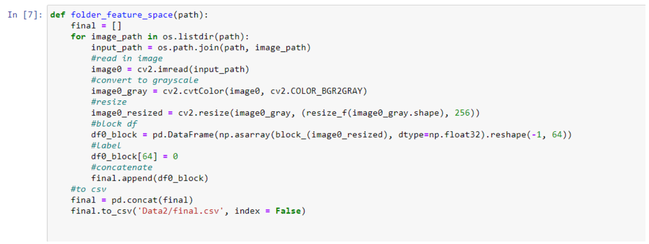
```
In [105]: x = df01_block[21]
          y = df01_block[31]
          z = df01_block[40]

          fig = plt.figure(figsize=(6,6))
          ax = Axes3D(fig)

          # get colormap from seaborn
          cmap = ListedColormap(sns.color_palette("hls", 3).as_hex())

          sc = ax.scatter(x, y, z, s=5, c=df01_block[64], marker='o', cmap=cmap, alpha=0.8)
          ax.set_xlabel('0')
          ax.set_ylabel('23')
          ax.set_zlabel('57')

          plt.legend(*sc.legend_elements(), bbox_to_anchor=(1.05, 1), loc=2)

          plt.show()
```

Figure 16: 3 feature plot for 2 classes 0 and 1.



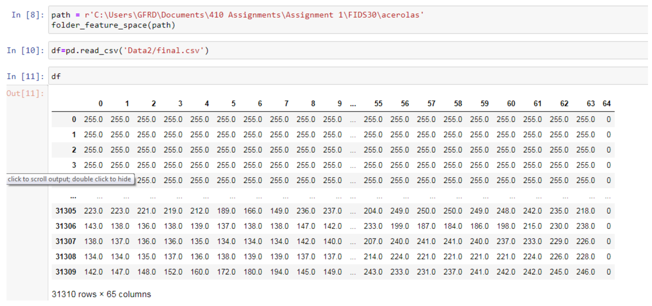
```
In [102]: x = df012_block[21]
          y = df012_block[31]
          z = df012_block[40]

          fig = plt.figure(figsize=(6,6))
          ax = Axes3D(fig)

          # get colormap from seaborn
          cmap = ListedColormap(sns.color_palette("hls", 3).as_hex())

          sc = ax.scatter(x, y, z, s=5, c=df012_block[64], marker='o', cmap=cmap, alpha=0.8)
          ax.set_xlabel('0')
          ax.set_ylabel('23')
          ax.set_zlabel('57')

          plt.legend(*sc.legend_elements(), bbox_to_anchor=(1.05, 1), loc=2)

          plt.show()
```

Figure 17: 3 feature plot for 3 classes 0, 1, and 2



```
In [7]: def folder_feature_space(path):
            final = []
            for image_path in os.listdir(path):
                input_path = os.path.join(path, image_path)
                #read in image
                image0 = cv2.imread(input_path)
                #convert to grayscale
                image0_gray = cv2.cvtColor(image0, cv2.COLOR_BGR2GRAY)
                #resize
                image0_resized = cv2.resize(image0_gray, (resize_f(image0_gray.shape), 256))
                #block df
                df0_block = pd.DataFrame(np.asarray(block_(image0_resized), dtype=np.float32).reshape(-1, 64))
                #label
                df0_block[64] = 0
                #concatenate
                final.append(df0_block)
            #to csv
            final = pd.concat(final)
            final.to_csv('Data2/final.csv', index = False)
```

Figure 18: Function for creating feature space from a folder of similar images.



```
In [8]:  path = r'C:\Users\GFRD\Documents\410 Assignments\Assignment 1\FIDS30\acerolas'
          folder_feature_space(path)

In [10]: df=pd.read_csv('Data2/final.csv')

In [11]: df
Out[11]:
```

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	64
0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	0
1	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	0
2	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	0
3	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	0
click to scroll output; double click to hide			255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
31305	223.0	223.0	221.0	219.0	212.0	189.0	166.0	149.0	236.0	237.0	...	204.0	249.0	250.0	250.0	249.0	248.0	242.0	235.0	218.0	0
31306	143.0	138.0	136.0	138.0	139.0	137.0	138.0	138.0	147.0	142.0	...	233.0	199.0	187.0	184.0	186.0	198.0	215.0	230.0	238.0	0
31307	138.0	137.0	136.0	136.0	135.0	134.0	134.0	134.0	142.0	140.0	...	207.0	240.0	241.0	241.0	240.0	237.0	233.0	229.0	226.0	0
31308	134.0	134.0	135.0	137.0	136.0	138.0	139.0	139.0	137.0	137.0	...	214.0	224.0	221.0	221.0	221.0	221.0	224.0	226.0	228.0	0
31309	142.0	147.0	148.0	152.0	160.0	172.0	180.0	194.0	145.0	149.0	...	243.0	233.0	231.0	237.0	241.0	242.0	242.0	245.0	246.0	0

31310 rows × 65 columns

Figure 19: Example of creating a feature space from a folder of similar images.

13 ASSIGNMENT 2 INTRODUCTION

In this assignment, classifiers were developed under the four categories of datasets and evaluated by using suitable qualitative measures, namely, precision and accuracy.

14 TASK 1

Here, the data domain of the datasets was divided into 80:20, where 80% is assigned to training datasets and 20% is assigned to testing datasets. Then, two features were selected in each category of training and testing datasets, and their histograms were plotted to see if they follow the same distribution. The same two features were used again to generate scatter plots, color coded by class label. The resulting plots for image01's training dataset is show in Figure **??** and Figure **??**. The mean and variance of the selected features are also listed in Figure **??**. Looking at both the plots and mean and vaariance values, the features in this case do follow very similar distributions.

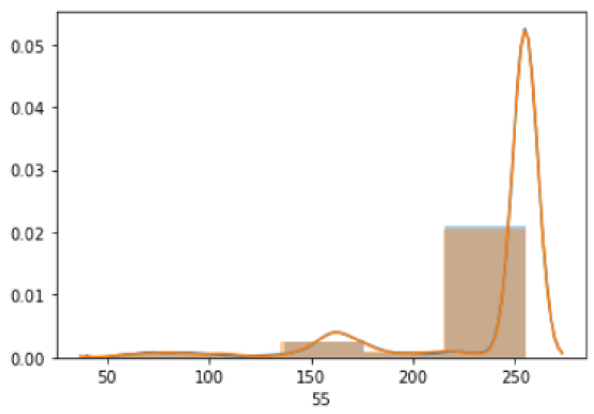


Figure 20: Distribution plots for the two features of image01$_t raindataset$.

15 TASK 2

In this task, elastic-net regression was trained and implemented as a two-class classifier using the training sets of the overlapping and non-overlapping feature vectors created in the previous assignment. Only the 2-class datasets were used since this is a 2-class classifier. This was achieved using the built in sklearn LogisticRegression model and applying the elastic net penalty on it. The initialization

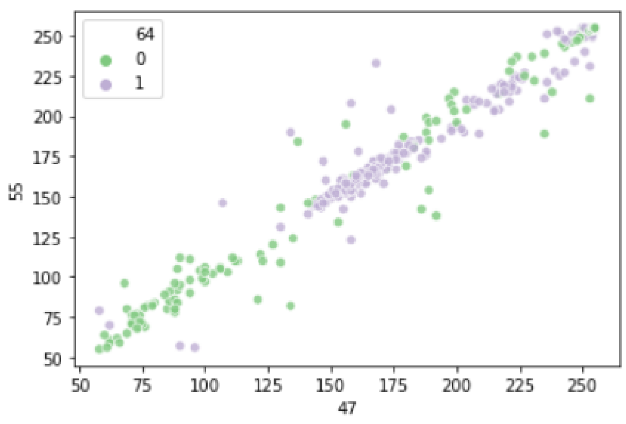
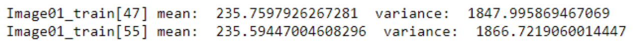


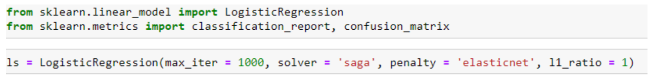
Figure 21: scatter plot of the two features of image01$_t raindataset$.



```
Image01_train[47] mean:  235.7597926267281  variance:  1847.995869467069
Image01_train[55] mean:  235.59447004608296  variance:  1866.7219060014447
```

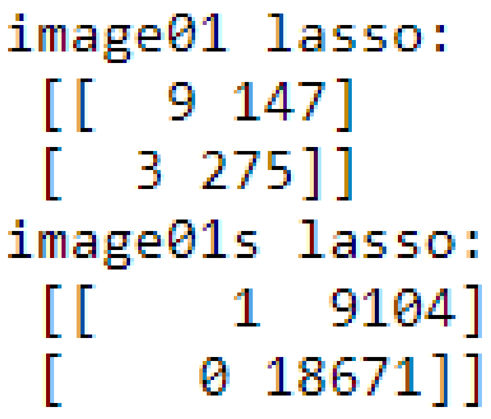
Figure 22: mean and variance for the two features of image01$_t raindataset$.

of the models is shown in Figure 23. After applying the model in the two datasets, a confusion matrix, as shown in Figure 24 was constructed. From it, the precision and accuracy for each was calculated. The model applied on the overlapping dataset, 'image01s' had a higher accuracy and precision score. Exact values are given in Figure 19.



```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

ls = LogisticRegression(max_iter = 1000, solver = 'saga', penalty = 'elasticnet', l1_ratio = 1)
```

Figure 23: Initialization of the lasso classifier.

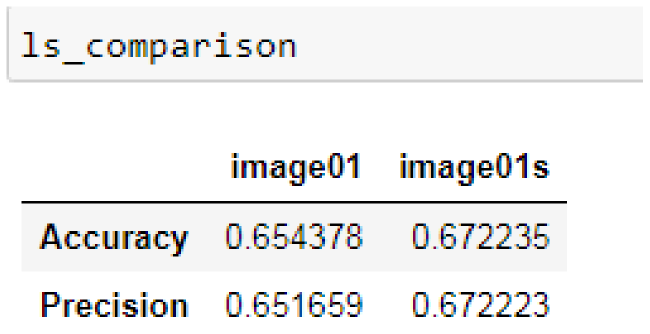


```
image01 lasso:
 [[   9  147]
 [   3  275]]
image01s lasso:
 [[     1   9104]
 [     0  18671]]
```

Figure 24: Confusion matrices from applying the lasso classifier to the 2-class datasets.

16 TASK 3

In this section, a random forest classifier was trained and implemented as both a 2-class and 3-class classifier using the sklearn.ensemble library. Initialization of the classifier is shown in Figure 26. The

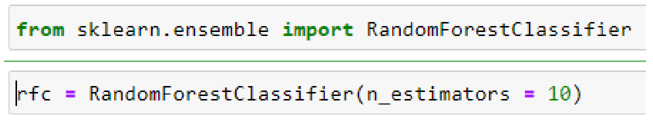


```
ls_comparison
```

	image01	image01s
Accuracy	0.654378	0.672235
Precision	0.651659	0.672223

Figure 25: Precision and accuracy scores for the lasso classifier.

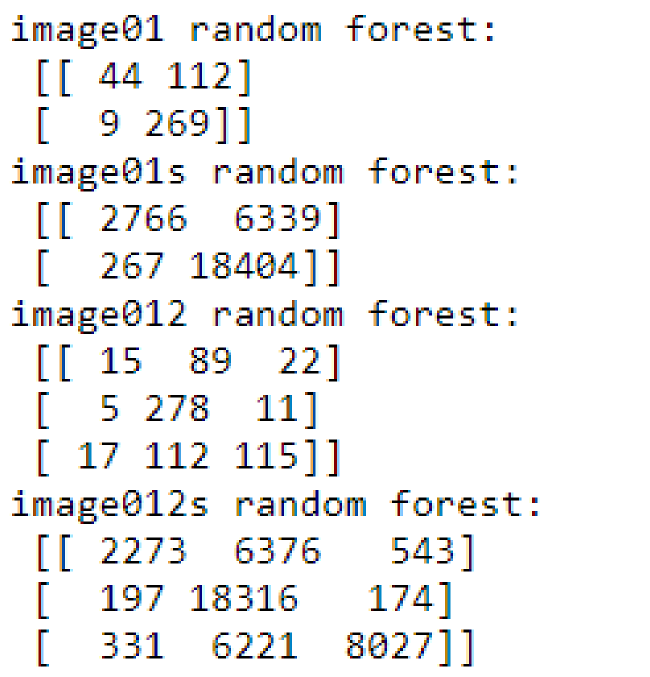
classifier was trained and applied on all four datasets and a confusion matrix was constructed for each case using the actual and predicted labels. The confusion matrix of each is given in Figure 27. From there, the precision and accuracy in each was calculated. For the sake of comparison and consistency, the precision on the 3-class datasets was calculated using class 1 as the 'positive' class was the case for the 2-class datasets. The exact values are given in Figure 28. Again the classifier performed best on the overlapping 2-class dataset, image01s; it had both the highest accuracy and precision score in this case.



```
from sklearn.ensemble import RandomForestClassifier
```

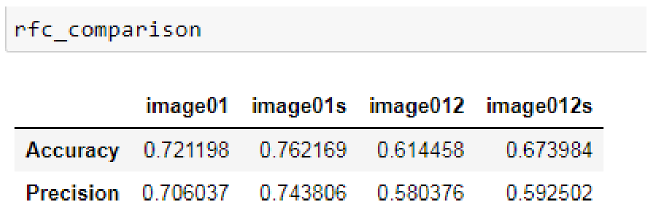
```
rfc = RandomForestClassifier(n_estimators = 10)
```

Figure 26: Initialization of the random forest classifier.



```
image01 random forest:
 [[ 44 112]
 [  9 269]]
image01s random forest:
 [[ 2766  6339]
 [  267 18404]]
image012 random forest:
 [[ 15  89  22]
 [  5 278  11]
 [ 17 112 115]]
image012s random forest:
 [[ 2273  6376   543]
 [  197 18316   174]
 [  331  6221  8027]]
```

Figure 27: Confusion matrices from applying the random forest classifier to the 2-class datasets.



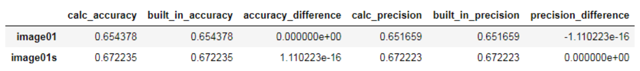
```
rfc_comparison
```

	image01	image01s	image012	image012s
Accuracy	0.721198	0.762169	0.614458	0.673984
Precision	0.706037	0.743806	0.580376	0.592502

Figure 28: Precision and accuracy scores for the random forest classifier.

17 TASK 4

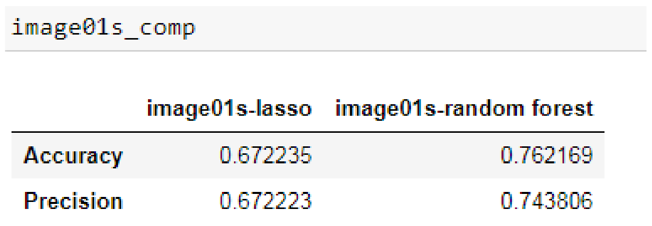
The first part of task 4 was examining the difference between the confusion matrix based quantitative measures with the built in measures. The built in measures, accuracy score and precision score were imported from the sklearn.metrics libarary. Figure **??** shows how these compare with those calculated from the confusion matrix of the lasso regression models. The biggest difference between the two is in the magnitude of 10e-16, which is insignificant. Thus, it's safe to conclude that there's no significant quantitative difference between the two methods of calculating the accuracy and precision scores.



	calc_accuracy	built_in_accuracy	accuracy_difference	calc_precision	built_in_precision	precision_difference
image01	0.654378	0.654378	0.000000e+00	0.651659	0.651659	-1.110223e-16
image01s	0.672235	0.672235	1.110223e-16	0.672223	0.672223	0.000000e+00

Figure 29: Built-in vs confusion$_{m}atrixbasedmeasurescomparison$.

The second part of task 4 was looking at the performance of all the classifiers across all the datasets to see which pair of classifier and dataset performed best. In part 2, we saw that the lasso regression classifier performed best on the 2-class overlapping dataset, image01s. Same thing in the case of the random forest classifier. Now to determine the better classifier on the image01s dataset, we can see in Figure 30 that both the accuracy and precision scores are higher for the random forest classifier. Thus, the best classifier-dataset pair is Random Forest Classifier - Image01s Dataset.



```
image01s_comp
```

	image01s-lasso	image01s-random forest
Accuracy	0.672235	0.762169
Precision	0.672223	0.743806

Figure 30: random forest classifier performance vs lasso classifier performance on image01s dateset.