

CHAPTER 4

ANALYSIS AND DESIGN

This study project is proposed to estimate an individual object's real mass by using 2D images with the help of DBSCAN clustering and k-NN classification. By using object detection and image processing, the volume of the object can be calculated and matched with the density table as the final result, the object's mass can be estimated. This goal is approached with the following steps :



















































4.1. Dataset Preparation

There are two sections to this model, the first one is image identification and the second one is volume and mass estimation. To perform this study, the fruits and vegetable dataset from Kaggle take major roles as the first section's main dataset. To identify the success of the first section which is image identification, 131 kinds of fruits and vegetables from Kaggle are collected but since the author is using google colab and due to lack of colab 12GB ram, only a few kinds of fruit and vegetables will be used. The author chooses a few kinds of fruit and vegetables based on common items which are easily found in daily life. 10, 20, 30, and 40 kinds of fruits will be collected for further analysis to see if the more dataset affects the time and object detection result. These images dataset are typically 100x100 pixels and easier for image processing especially since each image has a white background already consisting of 360-degree images.

Table 4.1: Fruits and Vegetables Kinds Dataset Table

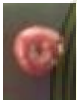
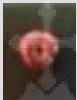
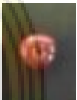
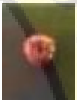
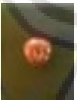
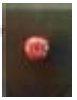
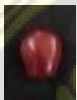
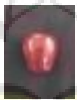
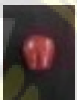
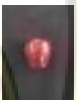
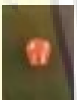

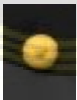
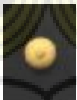
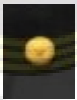



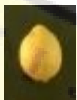
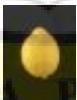
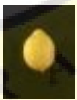




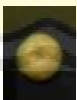
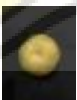
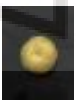

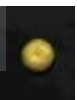






No.	Kinds	Fruits and Vegetables
1.	10	Apple Red Delicious, Blueberry, Cantaloupe 2, Cocos, Dragon Fruit, Guava, Lemon, Onion White, Orange, Tomato
2.	20	Apple Granny Smith, Apple Red, Apple Red Delicious, Blueberry, Cantaloupe 2, Cherry, Cocos, Corn, Dragon Fruit, Guava, Lemon, Limes, Mulberry, Onion White, Orange, Pepper Green, Pepper Red, Pepper Yellow, Tomato, Watermelon

Table 4.2: Kaggle Dataset Examples

Fruits and Vegetables Kinds		Images Dataset Examples				
<i>No.</i>	<i>Name</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
1.	Apple Red					
						
2.	Apple Red Delicious					
						
3.	Cantaloupe 2					
						
4.	Corn					
						
5.	Guava					
						

As for the second section, selected fruits and vegetable images that were taken by the iPhone X camera will be used for the second section's main dataset which the author calls by input image one from the object's top view and the other one from the object's side view. The top view is to calculate the object's area and the side view to calculate the object's height where the author is going to multiply them in the end to find the object's mass. Images with different kinds, shapes, colors of fruit will be taken to identify the effectiveness of these programs. Each with a different image shooting distances which are 15, 17, 19, 21 cm. Each image taken by iPhone X consists of 3024 x 4032 pixels. As for the result, we can see these models can estimate an object's volume and mass accurately with particular shapes or colors or shooting distances that the author is going to analyze.

Table 4.3: Input Image Examples

Fruits and Vegetables Kinds		Image Distances					
<i>No.</i>	<i>Name</i>	<i>15</i>	<i>17</i>	<i>19</i>	<i>21</i>	<i>23</i>	<i>25</i>
1.	Apple Red Delicious						
							
2.	Lemon						
							
3.	Orange						
							

4.2. Dataset Preprocessing

4.2.1. Input Image Preprocessing

As for the first step, the author stored the dataset and input image on google drive, while the author run the code in google colab so integration between the two platforms is required. After both platforms have mounted, the next thing to do is to call and extract needed files to google colab. The next thing to do is to copy input images from the google drive path to the tmp folder in google colab with a determined name. In other words, the input images are copied and renamed. The reason behind this is for easier to call images on the model later which are `inputtopview.jpg` and `inputsideview.jpg`.

To improve the area calculation of the input images, as for the first step input images' background will be turned into white color. The reason why the author does not just take objects with an HSV background for the input image is that the object's shadow will be terminated and calculated for the area and produce inaccurate results considering taking an image without even slight shadow is hard. This step is used for both input image top view and side view.

The first step is to read the selected image which the first one is the top view image later same steps are used for the side view. After the author stored the image copy on a new variable in order to create an image mask. The goal of this model is to extract an object from its background accurately and the result is extracted object which has the most perfect edge. By converting image color space to another color it makes the object have significant color difference so the object's edge can be seen clearly and can be extracted precisely.

The image needs to be transformed into HSV first to have a clear color difference between the object and the background. After HSV, input images are turned into RGB and then grayscale to have an increased sight of color difference. The output has a slight difference by using HSV the object tends to have a more precise edge cut. Afterward, gaussian blur is added to give a blur effect where it will reduce. After gaussian blur, the object's image requires to be eroded because by using gaussian blur only and canny edge detection, the object output has a black border around the cropped object as the effect of the blurred edge. Next, after the object has successfully eroded, the object's edge is obtained by using Canny edge detection. Followed by dilation either because by using canny only, a particular part of the object gets cut off too so dilation increases the preciseness of the object's edge.

Afterwards the result will be transformed to a threshold. By using Otsu's Thresholding, the optimal global threshold value will be determined from an image histogram. Since most of the objects tend to have elliptical/circular shapes, then an elliptical/circular kernel is needed here cv2 is provided a structuring element function instead of using NumPy which has a rectangular-shaped kernel. After by using closing from cv2 morphologyEx function which consists of a dilation followed by erosion in order to remove noises on the object. The result is still in edge form where only the edge line has a different color from the background like in Figure 1, to be used as a mask, it needs to have a different color inside the edge which is the object so looping here and replacing pixel is needed where pixel inside the edge has a bright color and the background have a dark color like in Figure 2 as the result.

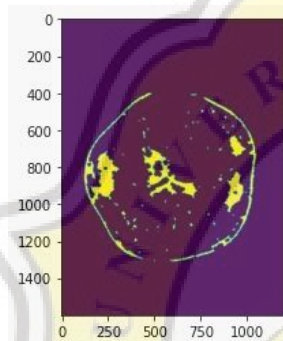


Figure 4.1: Data Preprocessing Result

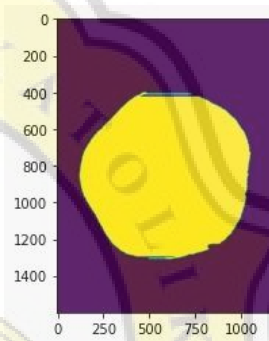


Figure 4.2: Mask Transformation

The following result will be masked into the input image so only the interesting part of the image is left and the rest will be in white color. After the image with white background is saved in the tmp folder with jpg format so that the next step is to replace all white pixels with transparent can be done easily by opening the image and converting to RGBA followed by converting the list into a buffer then recreating the image. Last is to save and show the image result. Result image is saved in the tmp folder provided with google colab with a determined name for easier to call later which is topviewextractedr.png. Exact same steps happen for the side view image either but with a different saved image result name which is sideviewextracted.jpg for the side view image with white background and sideviewextractedr.png for a transparent background and both of them in the tmp folder too.

4.2.2. Dataset Preprocessing

After we have done the input images preprocessing, it's the object's dataset turn which is extracted from the google drive path that has been mounted before to the provided tmp folder with "Training" as for the subfolder name for easier callables that consist of sub

subfolders with each fruits and vegetables kinds name as the sub subfolders name. After the zipped dataset has been successfully extracted, the next step is to have a side view image for the fruits and vegetables identification by adding new sub subfolders 'Input Image' inside the folder Training. Inside of the sub subfolders 'Input Image' where we place the side view of the input image by using a copy file command from google drive path to the new path and rename it with 'inputimage.jpg'. For easier processing of the data, the input side view image is stored within the same subfolder with the dataset but with a different sub subfolder which is "Input Image".

To read all the files within the specified folder, the author uses the glob module provided by python and stores them in the tuple. The tuple consists of each image's dataset path and name. For easier readable format data frame from pandas is used. Here the author can see the path with the dataset name easier and eliminate irrelevant data paths.

The dataset consists of raw images data that needs to be preprocessed for an understandable format. Since the DBSCAN algorithm works best with dataset in coordinate form, to acquire coordinates, first all dataset images need to be transformed in hash form. Dataset that is used for this research consists of images. The author chose hash instead of calculating the image RGB or pixel value because similar images tend to have similar hash value unlike other options. Imagehash package is required to transform the images dataset to hash form where we need to install it first in google colab.

The author creates 2 functions, one to calculate the hash and the other one to calculate the distance from the hash. The first function name is hashes_calculation which the parameter will consist of an image path and to call whash (wavelet hashing) from imagehash which is stored in hashfunc variable. Wavelet hashing using DWT (Discrete Wavelet Transformation) instead of DCT (Discrete Cosine Transform) like other options and whash is famous for it's great result either. First, declare the used variable which is hashes and names. Next with looping of each image dataset so every image in the dataset folder can be calculated to hash form. Inside of the looping is to open the image and use the hashfunc to calculate image hash and store them in hashes variable while names variable stored each of images name, both in list form. As for the result, the function returns hashes and names.

After all the images dataset has successfully turned into hash form, distance matrix will be calculated. The next function is distance calculation to find the distance between each image. First to declare a 'matrix' variable which has a matrix form with the length of the column and rows is the same as the hash length. Afterward looping is required to access every

image hash in the list. By using combinations from python the loop can access every possible combination of the list so distances between each of two images can be found and stored in the matrix variable that we have declared earlier and return the matrix.

From here with the help of PCA from sklearn, coordinates of each image can be acquired by using PCA with components of 2 and the result ready to enter the DBSCAN algorithm. By using coordinates clustering and visualization can be done easier either.

4.3. DBSCAN

To enter the DBSCAN clustering algorithm, the dataset needs to be transformed into list form and normalized. DBSCAN algorithm depends on eps and minPts value where eps stands for distance measure to find neighborhood within a point and minPts stands for a minimum number of points to form a cluster. The first step in the DBSCAN algorithm is to initialize all data points as outliers. After, the algorithm needs to find the neighbors for each data point. The core point is initialized where the neighborhood point is equal to or greater than minPoints. From here, core points, border points, and outliers can be found.

The DBSCAN algorithm is started by creating an object class named DBSCAN where it can be called easier later. After as for the constructor which is init where this method is called when an object is created from the class which consists of initializing core and border points with an arbitrary value.

The first function is find_neighbour where this function will find all the neighbors which arbitrary points in the dataset have. In this function, the author uses Euclidean Distance to find the distance between two points. By looping it will access all the data points and calculate the distance between them. The function will return neighborhoods points that particular points have and the distance between them is less than equal to determined epsilon along with storing them in list form on variable points.

The second function is fit which is the core of DBSCAN where this function will generate clusters based on processed data earlier. After finding all neighbors which each data point has now it's time to find all the clusters which this dataset has. The first step is to create a list to store which cluster each of the data points has. Zero here stands for outliers so as mentioned before all data points are considered as outliers with cluster zero. Next, initializing core and border as an empty list variables to store core and border points are needed.

There are 3 important sections on this function. The first one is to call the function we have made earlier which is `find_neighbour`. With the help of looping, each point can find all of its neighborhood points. In the second section, looping is utilized to find the core and border points by if any point has more or equal to determined `minPts` then the point is considered as a core point if the point doesn't satisfy the requirement then the point is considered as border points/outliers. The third section is to cluster each of the data points. Here the author performs queue data structure and chooses Breadth First Search (BFS) to find each data point that has been separated by epsilon distance and all the indirectly reachable points, the neighborhood points of the core can be obtained. If a point is considered as a core then it will be labeled as a cluster and find all the neighborhoods of the point which the author has grouped with the previous section and so on with the next cluster with a different core point. Now all points have been assigned to a cluster or cluster 0 for the outliers. This function returns `point_labels` which consist of a list of cluster labels of each point and number of clusters.

The last section is to visualize the clusters. A Scatter plot from `matplotlib` will help to visualize all the data points. Each cluster will be color-coded here the author chooses to have 3 different colors to identify each of the data points where the cluster they belong to but all the outliers will be colored as black. As for the outcome, each image is a member of a cluster.

The final step is to create a DBSCAN object and to call all of the functions we have made before which are `DBSCAN.fit` and completed by `DBSCAN.visualize` to visualize the outcome easier.

After we find which cluster our input image belongs to, the rest of the results outside the input image's cluster can be dropped. Now the dataset used only consists of images within the same cluster with the input image. For easier to process the next algorithm input image is being put at the first row as being the reference to find other images which have the closest distance to the input image.

4.4. K-NN

After the dataset is filtered and set up, it's ready for k-NN training data. The first step is to preprocess the data again by using image hash to distance matrix and with the help of PCA, coordinate form is obtained. These dataset coordinates will be k-NN training data. k-NN's main job is to find the closest point with the input image. k-NN uses euclidean distance to calculate the distance between each point as for the k value to decide which points have the

closest relationship with the input image. The first function consists of Euclidean Distance formula calculation to calculate distance between two data points which return the final result of square root. With euclidean distance formula as the follows:

$$d(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

The second function is to get neighbors which have 3 params consisting of train data, test data or selected points (input image point), and a number of neighbors the author is looking for. First to initialize variables to save each of the distances that have been calculated between two points by looping and calling the euclidean_distance function that we have made earlier in list form starting from the closest one. Next is to initialize a new variable to store the final result which is based on the k value from the param that has been determined. Here the author chooses the k value of two where only two points will be stored which are the input image point as the reference and the closest point with the input image. With k value, the algorithm will determine what the object is based on the closest points with the input image point. The result of k-NN will determine the success of object detection.

After the object has successfully been detected, the result is matched with the density table. The density table consists of all fruits and vegetable densities in CSV form. Dataframe from pandas is used for easier to read and find the matched density of fruit with the object detection result. Now the object's density is obtained and required for volume estimation.

4.5. Object's Volume Estimation

Object volume is needed for mass estimation. Mass and volume of fruits and vegetables have a linear relationship with their volume. The bigger the object then the heavier the object is. To calculate the object's volume top view and side view of the object are needed. 2 input images consist of a top view image and a side view image will be used. Object's area calculation from top view image is calculated by using image processing.

In the first section, data preprocessing images with transparent backgrounds have been acquired. Now the author is going to use those preprocessed images for area calculation and height estimation.

As for the area calculation, first, we use skimage.io to read the preprocessed top view image which has a transparent background because skimage will read and display in RGB. Next, turn the image into RGB and blur the image. After masking the image so that the rest

have a black background and only the interesting part of the image is selected. Reasons why not just turn the background black not using the transparent background first because it won't have a clean cut on the object edges. It helps to remove unwanted noises and increase the precision of edge detection.

Next is to calculate the area by obtaining the height and width of the image first then convert the image to gray and threshold it. To calculate the area is to count the non-zero pixel of the image which is the object. Now by counting the non-zero pixels the top area has been calculated.

Thereafter is height calculation, the same as previous which the image background needs to be converted to black by reading the image using skimage.io, convert to gray, and blur to have a clean object with a black background. Next is to convert it to gray and have the threshold the same as previous but not to calculate area instead to draw bounding boxes around the non-zero pixel. By creating a rectangle bounding box, the height of the object can be acquired.

For the side view image, the same steps will be used. The step is a bit different, grayscale and gaussian blur is still required but the threshold is used to calculate the object height by using a bounding box of the object. The area calculation result needs to be multiplied by 100 for the result in centimeters. Object's height needs to be divided by 100 to get the result in centimeters.

Last but not least is to calculate the volume, the author will use the following formula since the object's area and height have been calculated.

$$Volume = Area \times Height$$

The area needs to be multiplied by 1000 to achieve the real size of the area because earlier calculations provide a decimal result and the height needs to be divided by 100 to achieve the real size of object height. Next is to multiply them we have the object's volume.

4.6. Object's Mass Estimation

Now the object's density and object's volume have been found. As for the final step, by using the following formula

$$M = \rho \times V$$

An object's mass can be calculated. M here stands for mass of the object, ρ is for object's density which is the result of object detection matching with the stored density table and V stands for volume. With these mentioned steps, the object's mass is estimated.

4.7. Analysis

Conducted analysis in order to measure the success of this study is about accuracy comparison between real mass and estimated mass. The degree comparison result stands for the closeness between an object's real mass and obtained mass. Accuracy degree formula as the following.

$$Error_{percentage} = \frac{(Real_{mass} - Obtained_{mass}) * 100}{Real_{mass}}$$

$$Accuracy_{percentage} = 100 - Error_{percentage}$$



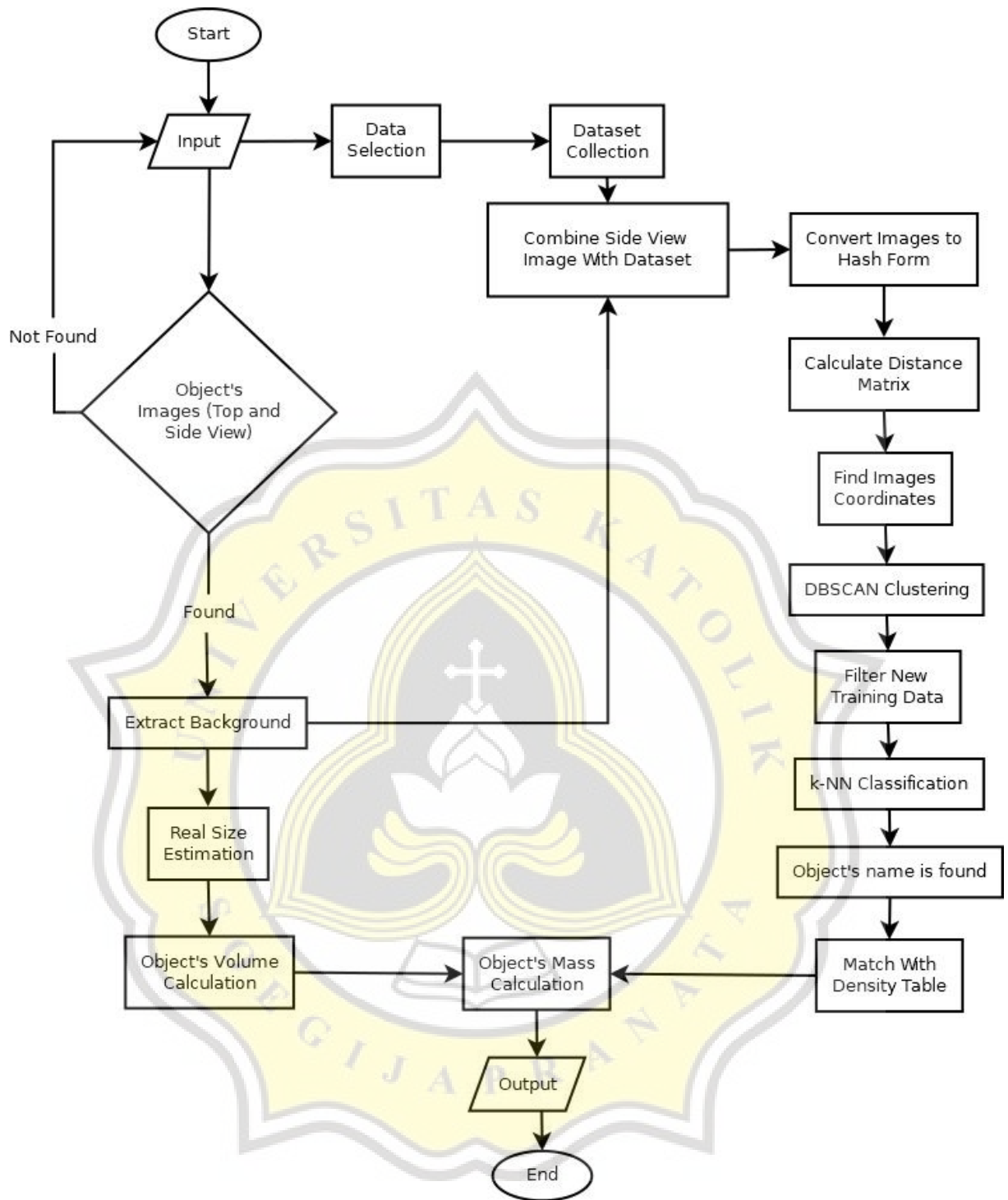


Figure 4.3: Real Mass Estimation Flowchart