CHAPTER 5
IMPLEMENTATION AND RESULT

5.1. Implementation

In this chapter, the implementation and testing of the plant classification research based on leaves using a Support Vector Machine and Canny Edge Detection will be described. For this project, the programming language used is python. First of all, the code below is part of the data input process, segmentation using Canny Edge Detection, and labeling on each segmented image.

```
1. dir  =  'C:\\Users\\Joseph-PC\\Downloads\\Compressed\\Dataset\\leaf-
   edge\\leaf dataset\\Train'
2. categories = ['Daun Ara Suci','Daun Bayam Hijau',...,'Daun Sirih',
   'Daun Srigading']
3. data = []
4. for category in categories1:
5.     path = os.path.join(dir,category)
6.     label = categories1.index(category)
7.
8.     for img in os.listdir(path):
9.         imgpath = os.path.join(path,img)
10.        leaf_img=cv2.imread(imgpath, cv2.IMREAD_GRAYSCALE)
11.        try:
12.            resize = cv2.resize(leaf_img,(200,200))
13.            edged = cv2.Canny(resize, 400, 400)
14.            img = np.array(edged).flatten()
15.            data.append([img,label])
16.        except Exception as e:
17.            pass
18.
19. random.shuffle(data)
20.
21. pick_in = open('data1.pickle','wb')
22. pickle.dump(data,pick_in)
23. pick_in.close()
```

In the first line, the variable dir is used to declare the location of the leaf dataset. In the second line, we prepare an array with the name category to apply labels to each binary. On the third line, an empty array is prepared to store the image binary and labels that have been attached. On the 10th line the image is read using a method called cv2.imread() then on the 12th line the image is resized to a size of 200 x 200, after being resized on the 13th line the image goes through the canny edge detection process, and the threshold is set at 400 to get the edges of the leaf pattern. On line 14 the image array goes through a flattening process to convert multi-dimensional arrays into a 1-dimensional array. Then on line 15 the image data and labels are entered into an empty

array that has been prepared previously. On line 19 the data is randomized so that the data spread. On lines 21-23, the data will be saved in a ***pickle*** file using the ***pickle*** library.

```
24.   pick_in = open('data1.pickle','rb')
25. data = pickle.load(pick_in)
26. pick_in.close()
27.
28. features = []
29. labels = []
30.
31. for feature, label in data:
32.     features.append(feature)
33.     labels.append(label)
34.
35. xtrain, xtest, ytrain, ytest = train_test_split(features, labels,
    test_size=0.25)
36.
37. model = SVC(C=100, kernel='rbf')
38. model.fit(xtrain,ytrain)
39.
40. pick = open('model1.sav','wb')
41. pickle.dump(model,pick)
42. pick.close
43.
44. prediction = model.predict(xtest)
45. accuracy = model.score(xtest, ytest)
46.
47. print(f'Akurasi: {accuracy*100}%')
48. print('Prediksi: ',categories1[prediction[0]])
49.
50. leaf = xtest[0].reshape(200,200)
51. plt.imshow(leaf,cmap='gray')
52. plt.show()
```

The coding above is part of model construction, model training, and model testing. On lines 24-26 the data that has been stored before is loaded again for reuse. On lines 28-29, we make available empty arrays with the names features and labels where features store the binary image segmentation results and labels store the class of the image by looping through the data files that were previously loaded (lines 31-33). Then on line 35, we split the dataset into training data and testing data with a ratio of 75:25. Then on line 37 we build a model with a support vector machine, the parameter we used is Cost 100 and radial basis function kernel. On line 38 the model is trained and stored in the model1.sav file so that it can be used for model evaluation (lines 40-42) and on lines 44-52 it is used to test the model that has been made and on lines 50-52 it is used to display the predicted image.

5.2. Result

Below are the results of research that has been made using 2 parameters to get the best accuracy results. The parameters used are threshold parameters from canny, kernel tricks and cost parameters on SVM. Threshold on canny is set with different sizes, specifically (100,200) to detect more edges such as leaf bones and (400,400) to detect only the edges of the leaves. Kernel tricks on SVM are using Poly and RBF, both kernels are analyzed by comparing the results of the confusion matrix.

5.2.1. Canny Threshold

This research uses 2 different threshold values when detecting the edges. The difference in the threshold value makes the edge read not only the pattern of the leaf but also the leaf bone. The graph below is the results of the classification process with different threshold values.

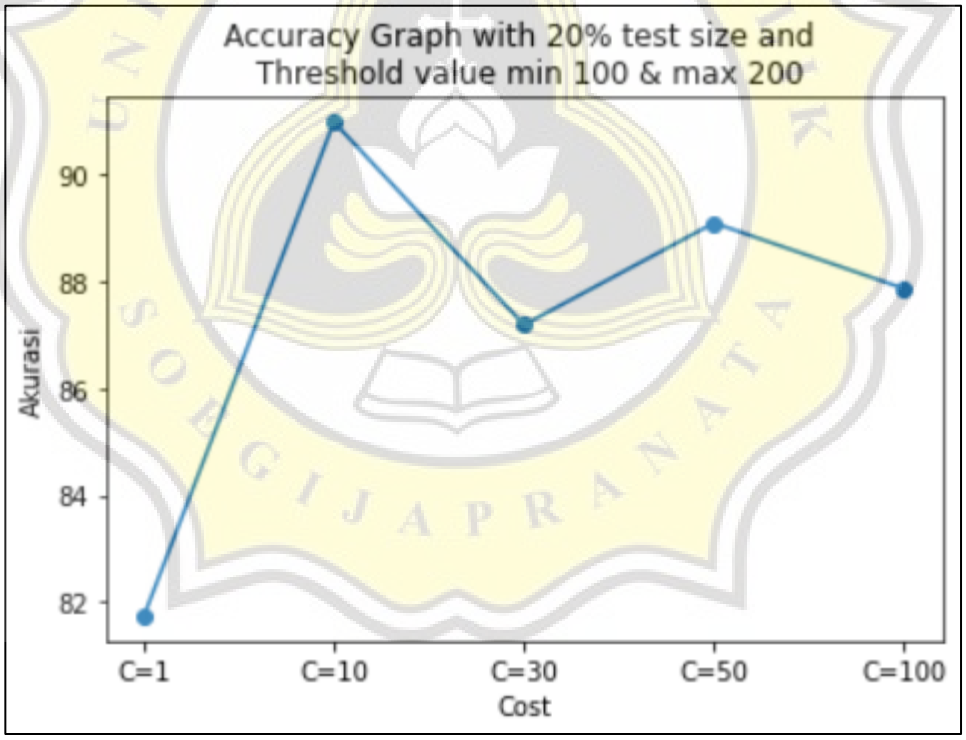

Figure 5. 1 Test size 20% and Canny threshold 100 & 200

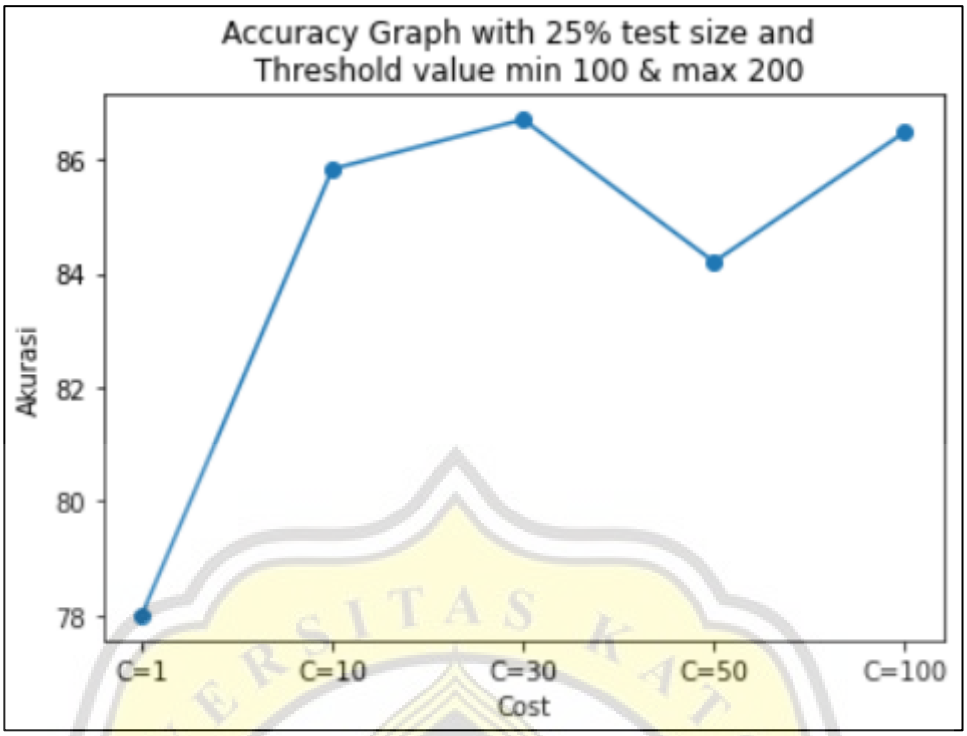

Figure 5. 2 Test size 25% and Canny threshold 100 & 200

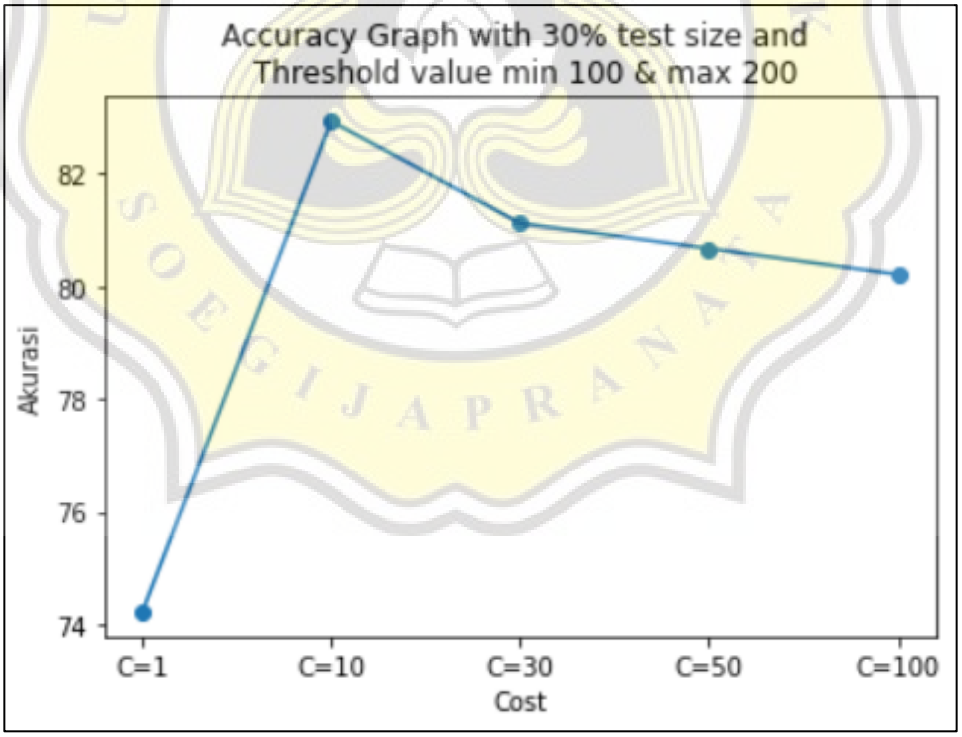

Figure 5. 3 Test size 30% and Canny threshold 100 & 200

From all the graphs above, it can be concluded that the division of the dataset and the threshold value affect the accuracy of results in the classification. for the lower threshold value of 100 and the upper threshold of 200 has a significant effect on the binary results of image segmentation. The greatest accuracy according to the graph above is when using a test size of 20%, a cost value of 10%, and using the RBF kernel.

5.2.2. SVM Cost Parameter

Below are the results of plant classification research by combining SVM and canny edge detection algorithms with different test data ratio comparisons.

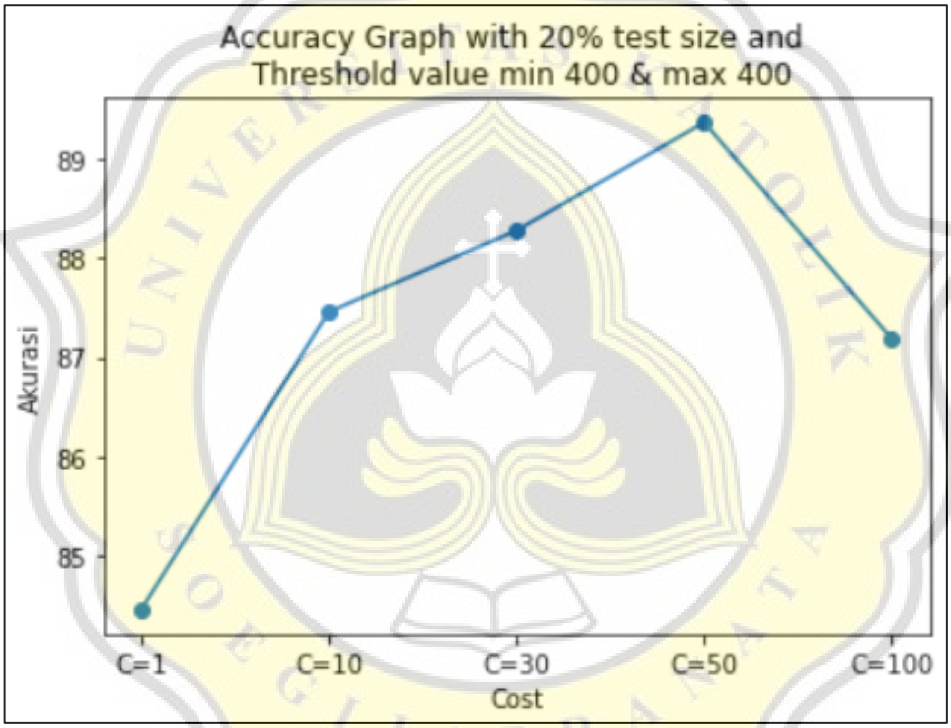

Figure 5. 4 Splitting dataset with test size 20%

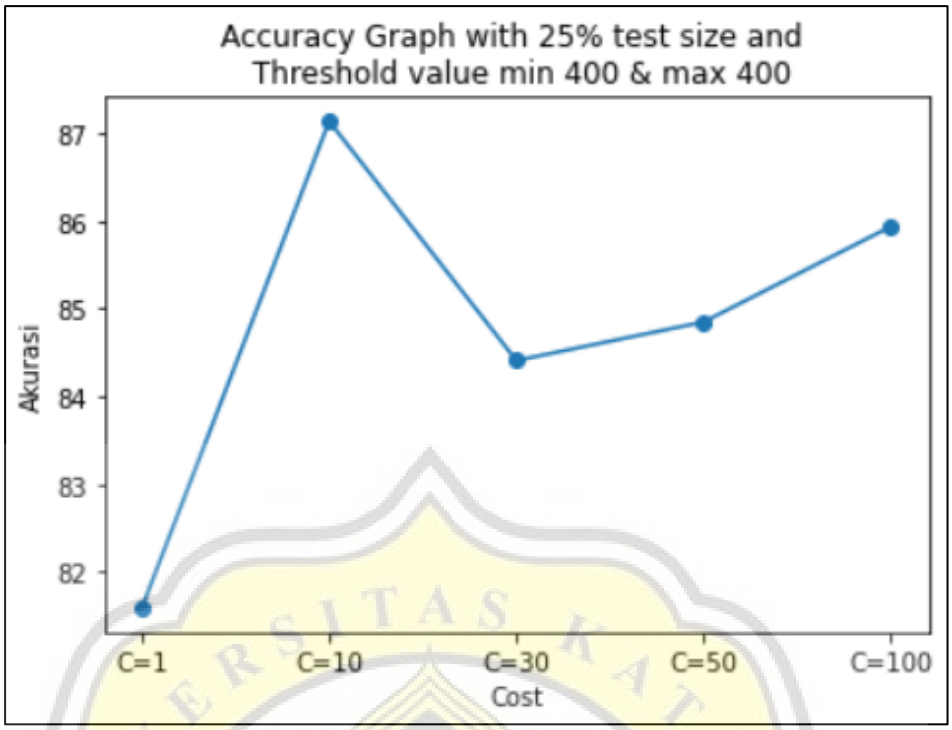

Figure 5. 5 Splitting dataset with test size 25%

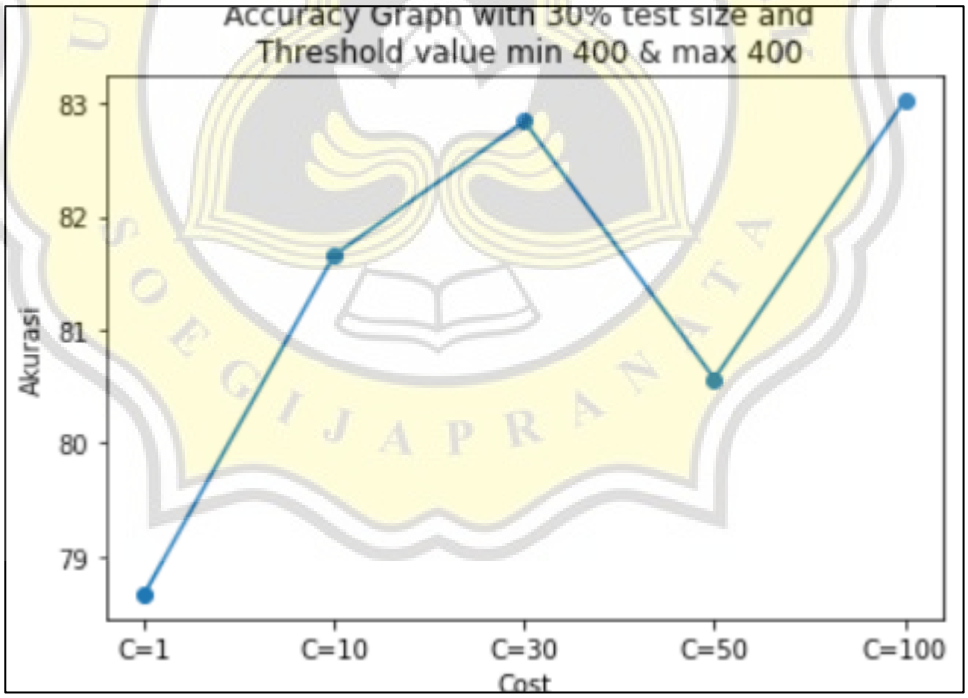

Figure 5. 6 Splitting dataset with test size 30%

From the three graphs above, it can be concluded that the distribution of test data and training data has a huge impact on the overall accuracy of the classification. Not only does the distribution from the dataset have an effect, but SVM parameters are also very important in the accuracy value. From each ratio and cost parameter compared, the highest accuracy results are when using C = 50 and the test size is at 20% with an accuracy of 89.37%. It can also be seen in figure 5.1, when using a cost parameter of 100 there is a decrease in classification results. This happens from overfitting that caused by the model only focusing on the errors in the training data so that the model becomes too complex and has a very small margin.

5.2.3. Kernel Trick SVM

There are many kinds of kernel tricks in SVM, they are RBF and Polynomial. The RBF kernel is often used because the results of the kernel are better than the polynomial kernel. Below are the results of the confusion matrix using the polynomial kernel and a cost value of 50 with an accuracy of 81.47%.

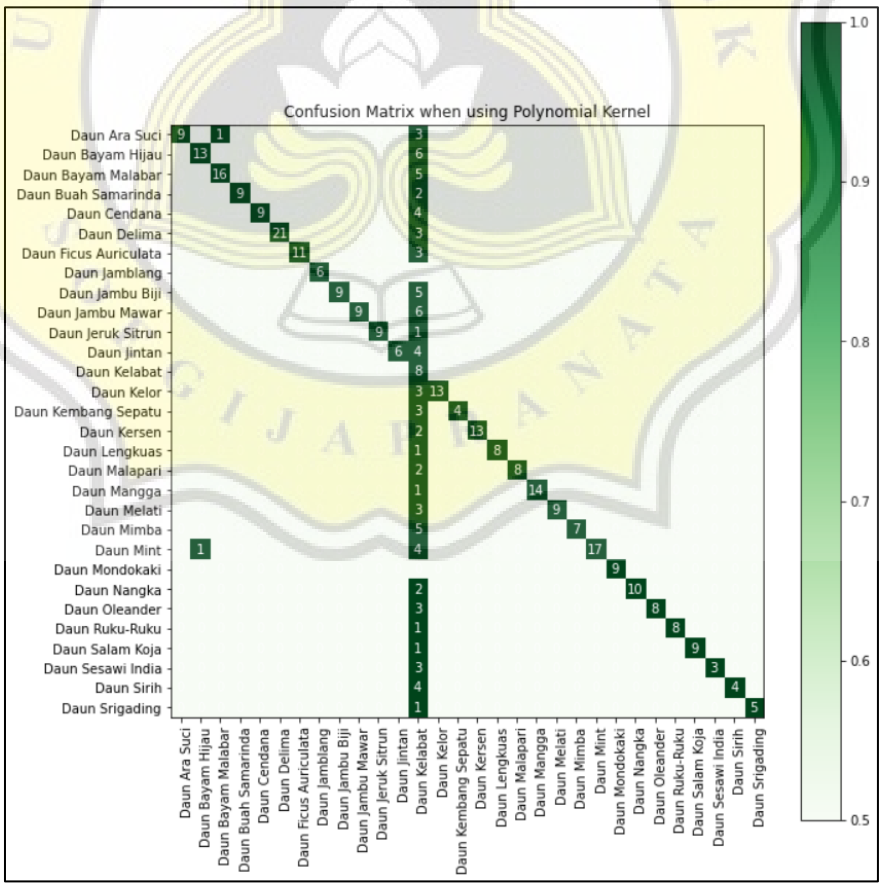

Figure 5. 7 Confusion Matrix using Kernel Polynomial

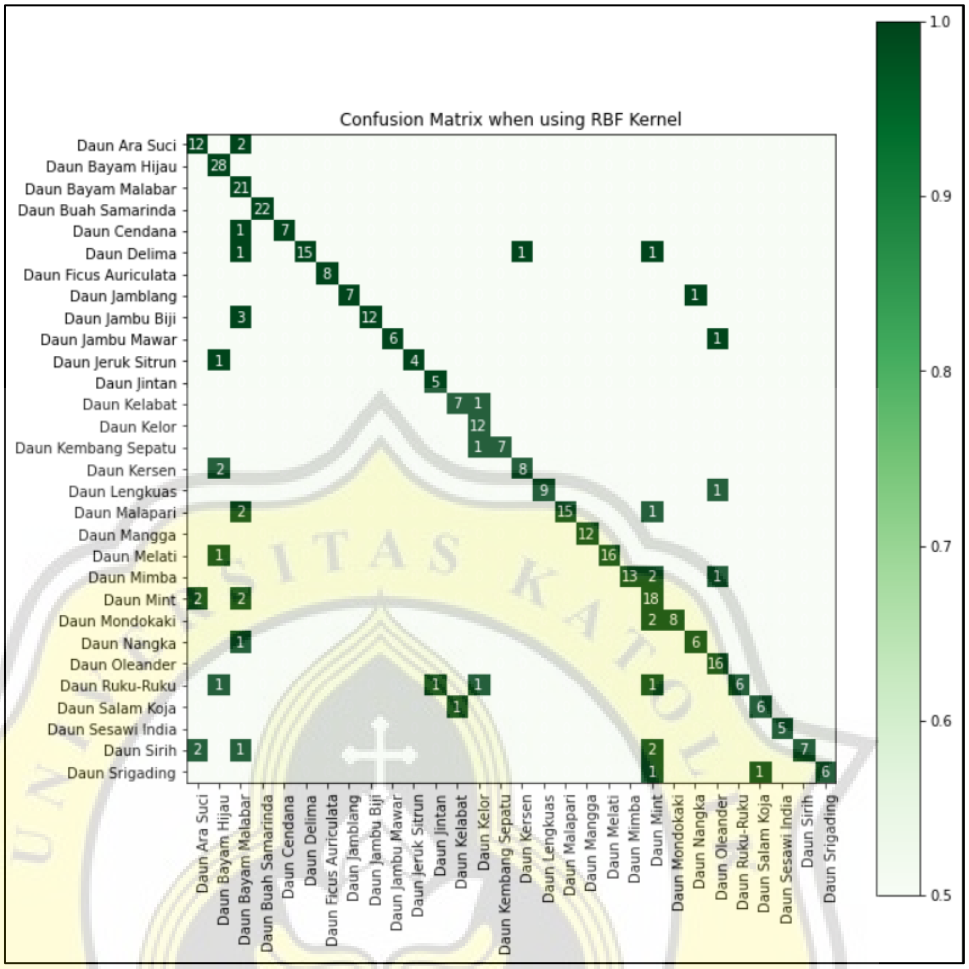

Figure 5. 8 Confusion Matrix when using kernel RBF

The confusion matrix above is the result of using the Radial Basic Function kernel with an accuracy of 89.37%. The parameters used are similar when using the Poly kernel. The value for the cost parameter is 50, the test size is 20% and for the canny parameter, the size of the threshold is min 100 and max 200. Based on the two confusion matrix results above, it can be concluded that the poly kernel has less performance so it only produces an accuracy of around 81% - 82%.

5.2.4. Comparison when applying Canny Edge Detection and when not using it

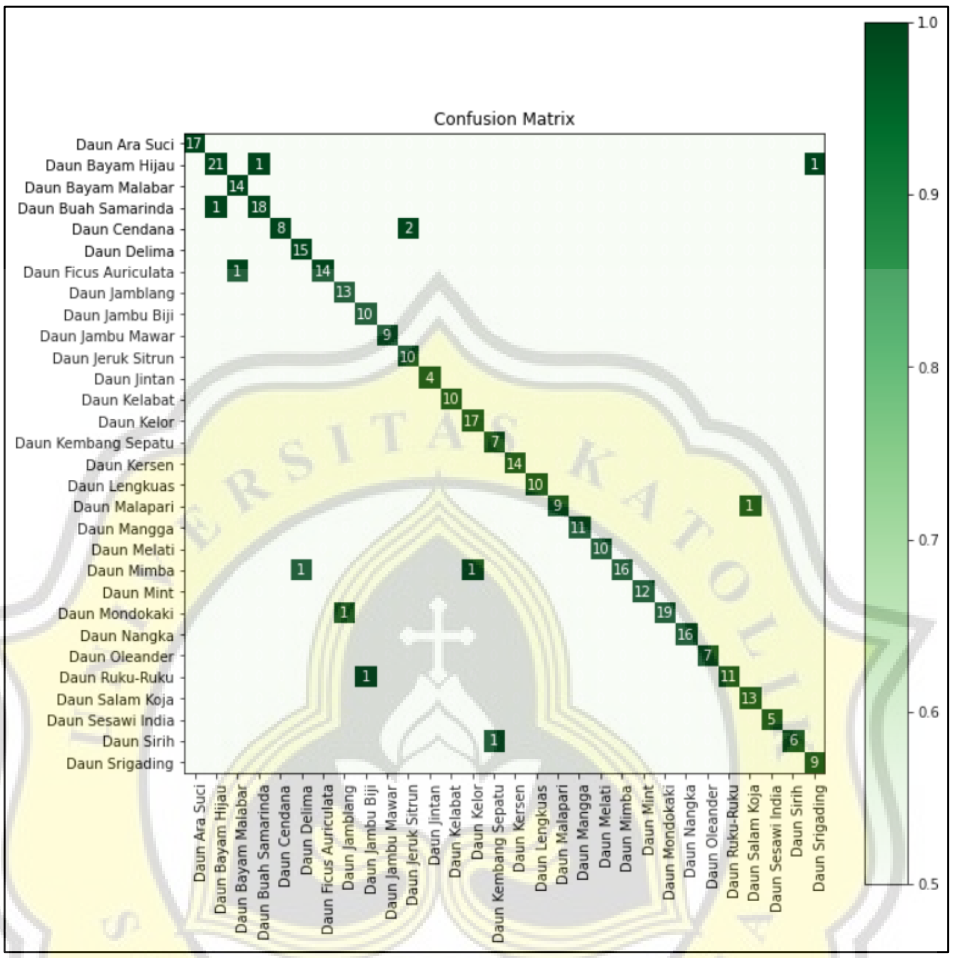

Figure 5. 9 Confusion Matrix while not using canny edge detection

The image above is a confusion matrix result when the support vector machine classifies plants based on their leaf images without using Canny Edge Detection. From the confusion matrix, we can conclude that without the help of Canny Edge Detection, SVM itself can classify leaf images very well. The result of the accuracy of the classification is 96.59%.

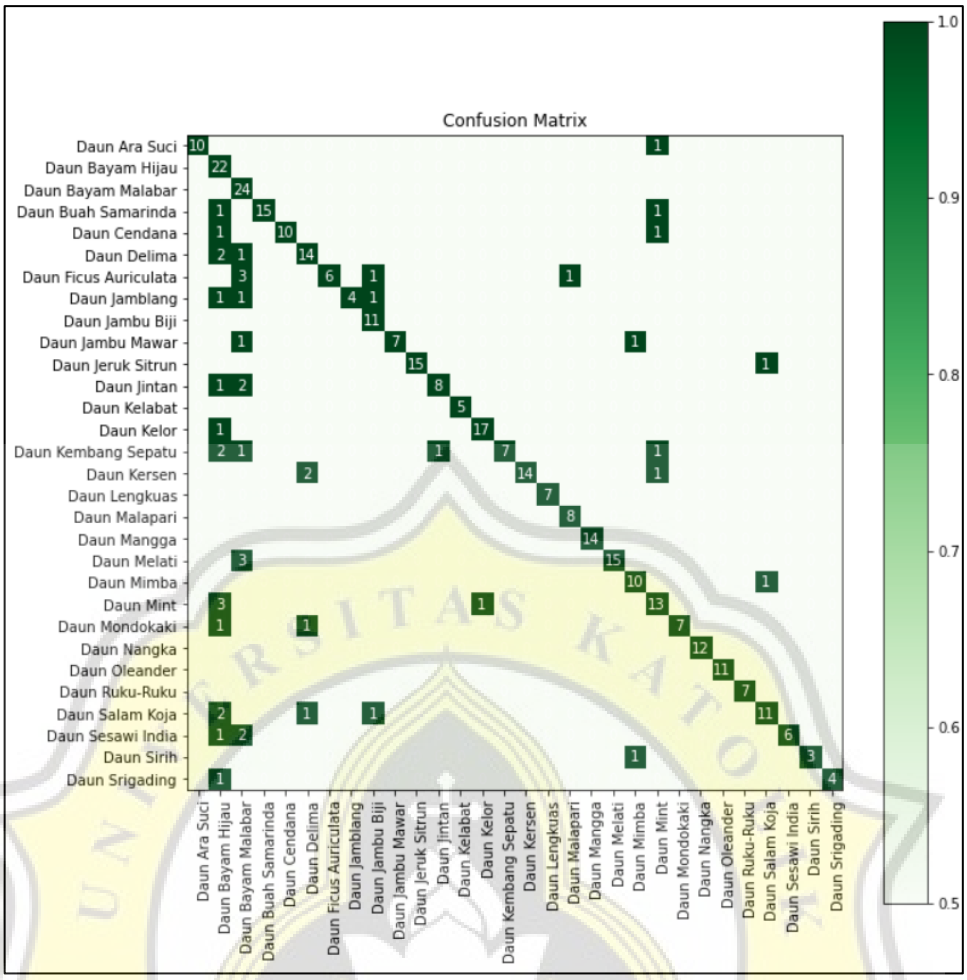

Figure 5. 10 Confusion Matrix while using Canny Edge Detection

The image above is the result of the confusion matrix when the support vector machine classifies plants based on leaf images using canny edge detection. The accuracy result of the classification when combining SVM and Canny Edge Detection is 89.37%. From the two confusion matrix above, it can be concluded that using canny edge detection will not improve the accuracy of the support vector machine classification.