BAB IV

HASIL DAN PEMBAHASAN

Proses pembuatan sistem pada penelitian ini menggunakan beberapa *library* atau modul yang tersedia dalam bahasa *python*. *Library* yang digunakan adalah math, mathplotlib, numpy, pandas, os, glob, dan BytesIO. *Library* tersebut digunakan untuk membantu serta memudahkan pembuatan sistem dan pemanggilan modul tersebut dapat dilakukan dengan menggunakan perintah *import*.

Kode 4. 1 Import Modul yang Dibutuhkan.

```
import math
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as nm
import pandas as pd
```

Pada Kode 4.1 terdapat modul, untuk modul *math* adalah module yang digunakan untuk mengakses fungsi perhitungan matematikan dengan tipe data *float*. Tipe data *float* tersendiri adalah tipe data angka yang terdapat bagian titik di akhir angka seperti (3,14) yang menandakan nilai desimal.

Untuk modul *mathplotlib* merupakan modul yang digunakan untuk visualisasi data dan mempunyai fitur *scripting interface* untuk merepresentasikan dengan *pyplot* untuk memudahkan dalam membuat grafik.

numpy merupakan modul pada python. *Import numpy* digunakan untuk melakukan perhitungan operasi matriks dan vector untuk mengolah angka dalam bentuk *array* atau *array* multidimensional yang berarti bentuknya dapat berupa 1

dimensi atau lebih dari 2 dimensi, dan biasanya digunakan dalam menganalisis sebuah data.

pandas adalah modul pada python untuk mempermudah dalam mengolah data serta menganalisa data-data yang tersetruktur. Selain itu *pandas* digunakan untuk membuat Tabel, mengubah dimensi data, mengecek data dan sebagainya. *pandas DataFrame* digunakan untuk membaca file dalam format *.txt*, *.csv*, dan *.tsv*. Fitur ini dijadikan Tabel dan dapat diolah menggunakan operasi *join*, *distinct*, *group by* dan lain sebagainya.

4.1 Pembuatan Class GLCM Untuk Tiap Matriks

Pada metode GLCM (Gray Level Co-ocurrence Matrix) mempunyai beberapa fungsi untuk mengekstrasi Gambar dengan menggunakan matriks 0°, matriks 45°, matriks 90°, dan matriks 135°. Fungsi tersebut dimasukkan dalam *class* GLCM.

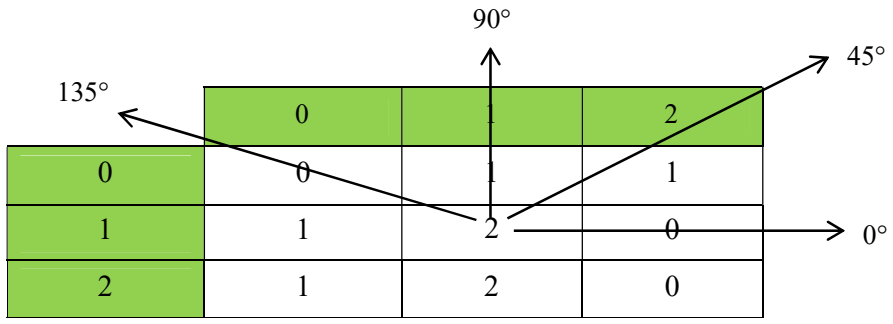
Kode 4. 2 Membuat *class* GLCM dan Perhitungan Matriks 0°

```
class Glcm:
    def __init__(self):
        self.result =[]
    def forOder(self,a, b):
        size = 0
        for i in a[0]:
            size=size+1
        for i in range(len(a)):
            for j in range(len(a[i])-1):
                p=a[i][j]
                q=a[i][j+1]
                b[p][q]=b[p][q]+1
        matriksIterasi1=list(map(list,zip(*b)))
        for i in range(len(b)):
            for j in range(len(b)):
                b[i][j]=b[i][j]+matriksIterasi1[i][j]
        count=0
        for i in b:
            for j in i:
                count=count+j
```

```
        return self.normalisasi(b, count)
```

Pada Kode 4.2 adalah rumus untuk menghitung matriks 0°, dengan melakukan perulangan data pixel pada matriks. Untuk melakukan perulangan tersebut harus sesuai arah matriks seperti pada Tabel berikut ini.

Tabel 4. 1 Menentukan Matriks 0°, Matriks 45°, Matriks 90° dan Matriks 135° Dengan Pixel Matriks 3x3.



Pada Kode 4.1 salah satu rumus dari matriks 0°, untuk matriks 0° mengambil *index* [2][0], untuk matriks 45° mengambil *index* [2][1], untuk matriks 90° mengambil *index* [2][1], dan untuk matriks 135° mengambil *index* [2][0].

Kode 4. 3 Perhitungan Matriks 45°

```
def forOder45(self,a ,b):
        size = 0
        for i in a[0]:
            size=size+1
        for i in range(len(a)-1):
            for j in range(size-1):
                p=a[i][j+1]
                q=a[i+1][j]
                b[p][q]=b[p][q]+1
        matriksIterasi1 =list(map(list,zip(*b)))
        for i in range(len(b)):
            for j in range(len(b)):
                b[i][j]=b[i][j]+matriksIterasi1[i][j]
        count = 0
        for i in b:
```

```
            for j in i:
                count=count+j
        return self.normalisasi(b, count)
```

Pada Kode 4.3 merupakan rumus perhitungan untuk matriks 45°, cara nya sama seperti matriks 0° dengan perbedaan posisi index i dan j yang sudah dijelaskan pada Tabel 4.1 dan melakukan perhitungan normalisasi. Begitupula sama seperti pada Kode 4.5.

Kode 4. 4 Perhitungan Matriks 90°

```
def forOder90(self,a, b):
        size = 0
        for i in a[0]:
            size=size+1
        for i in range (len(a)-1):
            for j in range(size):
                p=a[i][j]
                q=a[i+1][j]
                b[p][q]=b[p][q]+1
        matriksIterasi1 =list(map(list,zip(*b)))
        for i in range (4):
            for j in range(4):
                b[i][j]=b[i][j]+matriksIterasi1[i][j]
        count=0
        for i in b:
            for j in i:
                count=count+j
        return self.normalisasi(b, count)
```

Kode 4. 5 Perhitungan Matriks 135°

```
def forOder135(self,a, b):
        size = 0
        for i in a[0]:
            size=size+1
        for i in range(len(a)-1):
            for j in range(size-1):
                p=a[i][j]
                q=a[i+1][j+1]
                b[p][q]=b[p][q]+1

        matriksIterasi1 =list(map(list,zip(*b)))
        for i in range(len(b)):
            for j in range(len(b)):
```

```
                b[i][j]=b[i][j]+matriksIterasi1[i][j]
        count=0
        for i in b:
            for j in i:
                count=count+j
        return self.normalisasi(b, count)
```

4.2 Pembuatan File *.csv*

Setelah melakukan perhitungan matriks 0°, 45°, 90°, dan 135° akan dilakukan pemyimpanan data dengan menggunakan modul *pandas* pada python, yang akan diolah dengan manambahkan fungsi *insertoCsv* untuk membuat file *.csv*, dengan nama file "DataSetGLCM.csv" terlihat dalam Kode 4.6.

Kode 4. 6 Pembuatan File *.csv*

```
def insertoCsv(self,data):
        row = data;
        with open ('DataSetGLCM.csv', 'a') as csvFile:
            writer = csv.writer(csvFile)
            writer.writerow(row)
        print('succes')
        csvFile.close()
```

Setelah sukses membuat file *.csv* untuk menjadi output ekstrasi fitur, maka akan dikelompokkan sesuai masing-masing fitur GLCM seperti, energi, kontras, korelasi dan homogenitas yang terdapat pada Kode 4.7.

Kode 4. 7 Membuat Fitur GLCM Pada File *.csv*

```
row  =["Citra",  "Class",  "energy0",  "contras0",  "homogenity0",
"correlation0","energy45","contras45","homogenity45","correlation4
5","energy90","contras90","homogenity90","correlation90","energy13
5","contras135", "homogenity135", "correlation135", "energy_mean",
"contras_mean", "homogenity_mean", "correlation_mean"]
with open('DataSetGLCM.csv', 'a') as csvFile:
    writer = csv.writer(csvFile)
    writer.writerow(row)
    print("Succes")
csvFile.close()
```

Tabel 4. 2 Fitur GLCM dalam File *.csv*

Citra	Class	Energy0	Contras0	Homogenity0	Correlation0

Pada Tabel 4.2 terdapat kolom Citra, pada kolom citra akan berisi nama file dari Gambar Apel yang sudah di ekstraksi, kemudian terdapat class, pada kolom class akan berisi nama variabel untuk folder apel seperti, Apel Hijau 1 diberikan nama variable H1, untuk Apel Hijau 2 diberikan nama variable H2, dan Apel Hijau 3 diberikan nama variabel H3. Kemudian untuk kolom energi berisi nilai hasil konsentrasi intensitas pada matriks co-occurrence, nilai energi akan membesar jika pasangan piksel terkonsentrasi pada beberapa koordinat, dan jika sebaliknya akan mengecil apabila letaknya menyebar. Untuk kolom kontras, nilainya akan membesar apabila variasi dalam citra tinggi, dan sebaliknya jika variasinya rendah. Untuk kolom homogenitas, sebagai ukuran kehomogenan variasi intesitas dalam citra, nilai pada homogenitas akan membesar jika variasi intensitas dalam citra rendah dan sebaliknya apabila variasinya tinggi. Dan untuk korelasi sebagai mengukur ketergantungan linear dari aras keabuan dalam ketetanggaan piksel citra.

Setelah semua fitur sudah masuk kedalam file *.csv*, maka akan melakukan perhitungan normalisasi dan ekstrasi fitur pada energi, kontras, homogenitas, dan korelasi.

Kode 4. 8 Fungsi untuk Normalisasi

```
def normalisasi(self,data, counter):
   for i in range(len(data)):
        for j in range(len(data)):
           data[i][j]=data[i][j]/counter
   return data
```

Nilai dari normalisasi didapatkan dari matriks dan matriks co-ocurrence, kemudian dapat digunakan untuk melakukan perhitungan fitur untuk Energi, Kontras, Homogenitas, dan Korelasi, sesuai dengan rumusnya masing-masing pada Persamaan 2.1

Kode 4. 9 Fungsi untuk Energi, Kontras, Homogenitas, dan Korelasi.

```
def egyMethod(self, data):
    sume =0
    for i in data:
        for j in i:
            sume = sume + (j*j)
    return sume

def egyMethod(self, data):
    sume =0
    for i in data:
        for j in i:
            sume = sume + (j*j)
    return sume

def contrasMethod(self, data):
   result = 0
   for i in range(len(data)-1):
       for j in range(len(data[i])-1):
           result = result+data[i][j]*((i-j)*(i-j))
   return result

def hgnMethod(self, data):
    result = 0
    for i in range(len(data)-1):
        for j in range(len(data[i])-1):
            kiri = 1/(1+((i-j)**2))
            tot = kiri*data[i][j]
            result = result + tot
    return result

def correlation(self, data):
    result = []
    u = self.getMean(data)
    mean = self.getStd(data, u)
    for i in range(len(data)-1):
        for j in range(len(data[i])-1):
            temp = ((i-u)*(j-u))/mean
```

```
            temp2 = data[i][j]*temp
            result.append(temp2)
    return sum(result)
```

4.3 Ekstraksi Fitur GLCM

Setelah membuat file untuk menyimpan parameter GLCM, selanjutnya adalah melakukan ekstraksi fitur pada setiap folder apel hijau dimulai dari apel hijau 1, apel hijau 2 dan apel hijau 3.

Kode 4. 10 Modul *Glob* Pada Ekstraksi Fitur GLCM

```
import glob
x = 1
for filename in glob.glob('Apel Hijau 1/*.jpg'):
```

Pada Kode 4.10 terdapat modul *glob* yang berfungsi untuk membuat daftar urutan file dari hasil pencarian pada folder atau file dalam satu folder, oleh karena itu pada code ini akan menghasilkan output urutan file yang ada dalam folder apel hijau 1.

Kode 4. 11 Kode Ekstraksi Fitur GLCM Pada Folder Apel Hijau

```
for filename in glob.glob('Apel Hijau 1/*.jpg'):
    if x <= 49:
        data = Data(filename)
        glcm = Glcm()
        feture = [filename,"H1"]
        data0 = glcm.forOder(data.citra, data.matriksIterasi)
        countfeture(data0, glcm)

        data45 = glcm.forOder45(data.citra, data.matriksIterasi)
        countfeture(data45, glcm)

        data90 = glcm.forOder90(data.citra, data.matriksIterasi)
        countfeture(data90, glcm)

        data135 = glcm.forOder135(data.citra, data.matriksIterasi)
        countfeture(data135, glcm)
```

Pada Kode 4.11 merupakan syntax untuk memanggil dan memproses data training yang ada didalam folder apel hijau 1 dan etiap citra dilakukan ekstraksi fitur dengan GLCM. Output yang didapatkan berupa urutan data training yang diproses, dapat dilihat pada Gambar 4.4. Setelah melakukan ekstraksi fitur maka selanjutnya adalah menghitung nilai rata-rata pada seluruh citra dengan sudut 0°, 45°, 90°, 135°. Nilai rata-rata yang didapat akan dimasukkan ke dalam file csv yang sudah dibuat sebelumnya.

Kode 4. 12 Source Code Untuk Mencari Mean

```
for i in range(len(data0)-1):
            for j in range(len(data0[i])-1):

data0[i][j]=(data0[i][j]+data45[i][j]+data90[i][j]+data135[i][j])/
4
        countfeture(data0, glcm)
        data.insertoCsv(feture)
        print(x)
        x=x+1
```

Pada Kode 4.12 merupakan syntax untuk mencari rata-rata dari keempat sudut yang ada dengan perulangan pada 50 data training citra yang ada dalam folder apel hijau 1. Jika berhasil maka akan menghasilkan output seperti Tabel 4.3.

Tabel 4. 3 Proses Ekstraksi Fitur GLCM

Succes 0	Succes 1	Succes 2	Succes 3	…
…	Succes 46	Succes 47	Succes 48	Succes 49

Pada Tabel 4.3 adalah informasi bahwa berhasil melakukan ekstraksi dan di simpan dalam file DataSetGLCM.csv‟ Tahap ekstraksi fitur dengan GLCM ini dilakukan berulang pada folder apel hijau 2 dan folder apel hijau 3. Dengan

syntax, tahapan dan output yang sama, hanya berubah nama folder nya saja, H1 untuk folder apel hijau 1, H2 untuk folder apel hijau 2, dan H3 untuk foler apel hijau 3.

Tabel 4. 4 Cuplikan Hasil Ekstraksi GLCM

Citra	Class	Energi0	Kontras0	Homogenitas0	Korelasi0
0_100	H1	0.020325	0.287295	148.645234	0.979015
100_100	H1	0.013494	0.141198	181.097843	0.972568
108_100	H1	0.013043	0.128931	216.547866	0.964354
117_100	H1	0.013072	0.120726	215.600191	0.961599
124_100	H1	0.013351	0.125035	198.907192	0.963649

Pada Tabel 4.3 adalah cuplikan hasil output yang tersimpan dalam file csv yang berisi 150 data yang berhasil di ekstraksi.

4.4 Klasifikasi KNN

Kode 4. 13 *Import* Modul

```
import math
import random
```

Pada Kode 4.13 adalah *import* modul yang dibutuhkan pada klasifikasi KNN. Untuk yang pertama terdapat modul *import math* merupakan fungsi menghitung dalam bentuk angka. Kemudian untuk *import random* digunakan untuk pengacakan bilangan dengan tipe data *float*

Kode 4. 14 Hitung Euclidean Antara Dua Vektor.

```
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(1, len(row1)):
        distance += (float(row1[i]) - float(row2[i]))**2
    return math.sqrt(distance)
```

Pada Kode 4.14 digunakan untuk menghitung rumus *Euclidean Distance* atau bisa disebut dengan rumus *pythagoras* yang merupakan rumus dari KNN seperti Persamaan 2.5 yang digunakan untuk menghitung jarak antara 2 vetor. Pada fungsi e*uclidean_distance* terdapat parameter row1 dan row2. Pertama membuat variabel *distance* untuk menetapkan nilai yang akan di hitung. Kemudian melakukan perhitungan dengan cara melakukan perulangan yang menggunakan parameter row1 dan row2. Dan untuk fungsi sqrt untuk mengembalikan akar kuadrat pada *distance*. Fungsi sqrt ini hanya bisa menggunakan tipe data float dan integer.

Pada apel di atas merupakan jenis apel yang sama yaitu apel hijau, dan memiliki perbedaan pada bentuknya, pada Apel H1, memiliki ciri bentuk yang bulat dengan sedikit warna yang lenih tua dan terdapat seperti keriput kecil pada ujung atasnya, untuk apel H2 dengan bentuk yang lonjong dan sedikit lebih cerah warnanya, dan yang terakhir untuk apel H3 bentuknya sama dengan apel H1 berbentuk bulat tetapi memiliki warna yang berbeda yaitu warna sedikit lebih mida dan tidak ada keriput pada ujung atasnya.

Kode 4.15 Mencari ketetanggan yang Sama

```
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist      =      euclidean_distance(test_row,
train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()

    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
```

Pada Kode 4.15 adalah untuk mencari objek tetangga yang terdekat agar dapat menentukan nilai pada data yang uji dan data yang di tes. Pada fungsi ini harus menentukan ketetanggaannya yang, karena ketika melakukan pengecekan harus ada batasannya, jika tidak ditetapkan ketetanggaannya makan fungsi ini akan terus mengecek seluruh objek terdekatnya. Fungsi *get_neighbors* terdapat parameter *train*, *test_row*, dan *num_neighbors*.

Kode 4.16 Melakukan Prediksi dengan Ketetanggaan

```
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[0] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

Pada Kode 4.16 fungsi dari *predict_classification* ini terdapat parameter *num_neighbors* yang berarti *num_neighbors* ini harus ditetapkan untuk mencari objek atau label yang jarak maksimal nya sesuai dengan *num_neighbors* tersebut.

Kemudian terdapat variabel *neighbors* untuk mendapatkan nilai objek atau label berdasarlan *train* dan *test_row* yang terdekat dengan total *num_neighbors* yang telah di tentukan.

Untuk mendapatkan nilai dari variabel *prediction* adalah jumlah maksimal atau banyaknya label atau objek yang terdekat dan tidak melebih batas jumlah *num_neighbors*.

Kode 4.17 Memisaahkan Kumpulan Data di Dataset

```
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
```

```
        while len(fold) < fold_size:
            index = random.randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
```

Pada Kode 4.17 adalah membagi data sesuai dengan *n_folds* yang sudah ditetapkan, serta membagi data sejumlah *n_folds* bagian dan melakukan pengujian sebanyak berapa kali *n_folds*.

Jika terdapat 100 data kemudian *n_folds* di tetapkan *n_folds*=5, maka data panjang data adalah 20 dan akan melakukan perhitungan perulangan sebanyak 5 kali sesuai dengan Gambar 2.3 yang sudah dijelaskan.

Kode 4. 18 Hitung Persentase Akurasi

```
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

Pada Kode 4.18 untuk menghitung nilai akurasi dari nilai prediksi dan nilai yang benar kemudian di jadikan nilai persen.

Kode 4. 19 Evaluasi algoritma menggunakan *cross validation*

```
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    actual_pred_pairs = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
        predicted = algorithm(train_set, test_set, *args)
```

```
        actual = [row[0] for row in fold]
        actual_pred_pairs.append((actual, predicted))
    return actual_pred_pairs
```

Pada Kode 4.19 evaluasi ini terdapat variabel *folds* yang nilai dari *dataset_split* kemudian terdapat variabel *score* yang berisi *list* yang akan berisi masing-masing nilai *folds* yang sudah melakukan perhitungan perulangan. Pada *train_set* merupakan *array* dari *folds* yang sudah ditetapkan dan *test_set* merupakan literasi dari *folds*, jika *n_folds*=5 dalam bentuk *array* maka *test_set* yang pertama adalah [1] dan *train_set* nya adalah [2, 3, 4, 5], ini adalah litarasi pertama. Untuk literasi selanjutnya *test_set* nya adalah [2] dan *train_set* nya adalah [1, 3, 4, 5], literasi ini akan terus dilakukan hingga *test_set* nya [5] sesuai *n_folds* yang di tetapkan. Dan yang terakhir menghitung nilai prediksi dan nilai akurasinya yang akan dimasukkan dalam variabel *scores*.

Kode 4. 20 Algoritma KNN

```
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)
    return (predictions)
```

Pada Kode 4.20 adalah menyipkan variabel *predictions* untuk menyimpan nilai saat di *return*, kemudian melakukan perulangan yang *output* nya berisi *predict_classification* untuk masing-masing *test* yang berupa *array* yang akan menghasilkan prediksi dari *train*.

Kode 4. 21 *Import* Modul

```
import pandas as pd
```

Pada Kode 4.21 adalah modul yang salah satunya digunakan untuk membuat tabel, mengecek data dan lain sebagainya. *Dataframe* merupakan struktur dasar pada modul *pandas* yang memudahkan untuk membaca file dalam format .txt, .csv, dan .tsv.

Kode 4. 22 Menampilkan File Data Pelatihan Dalam Format .csv

```
df = pd.read_csv("DataSetGLCM_Update.csv", header=None)
df_list = df.to_numpy().tolist()
df
```

Pada Kode 4.22 digunakan untuk menampilakan file data pelatihan dengan format *dataframe* dijadikan menjadi list.

Tabel 4. 5 Cuplikan Isi Data Pelatihan

	0	1	2	3	...	17	18	19	20
0	H1	0.016101	0.187527	113.281350	...	0.011747	0.003503	4209.053415	0.235560
1	H1	0.016334	0.198823	98.460434	...	0.011801	0.004020	4436.487876	0.235513
2	H1	0.016550	0.204267	104.765258	...	0.011803	0.002553	4580.037071	0.253141
3	H1	0.016737	0.212947	151.958087	...	0.011878	0.002163	4769.757642	0.268946
...	...	...	...	...	...	...	...	...	...
116	H3	0.021167	0.305499	158.020276	...	0.013133	0.031351	2934.669755	0.557107
117	H3	0.020887	0.299003	150.077363	...	0.013109	0.030129	2905.501285	0.557613
118	H3	0.020951	0.299726	144.175954	...	0.013080	0.031245	2908.210815	0.550348
119	H3	0.020741	0.298059	141.309383	...	0.012940	0.031525	2916.043806	0.541407

Pada Tabel 4.4 merupakan cuplikan dari Data Pelatihan yang berjumlah 120 data yang terdiri dari 3 kelas apel yang masing-masing kelas berjumlah 40 *dataset*.

Kode 4. 23 Menampilkan File Data Pengujian Dalam Format .csv

```
df_test = pd.read_csv("DataSetGLCM_Update-Test.csv", header=None)
df_test_list = df_test.to_numpy().tolist()
df_test
```

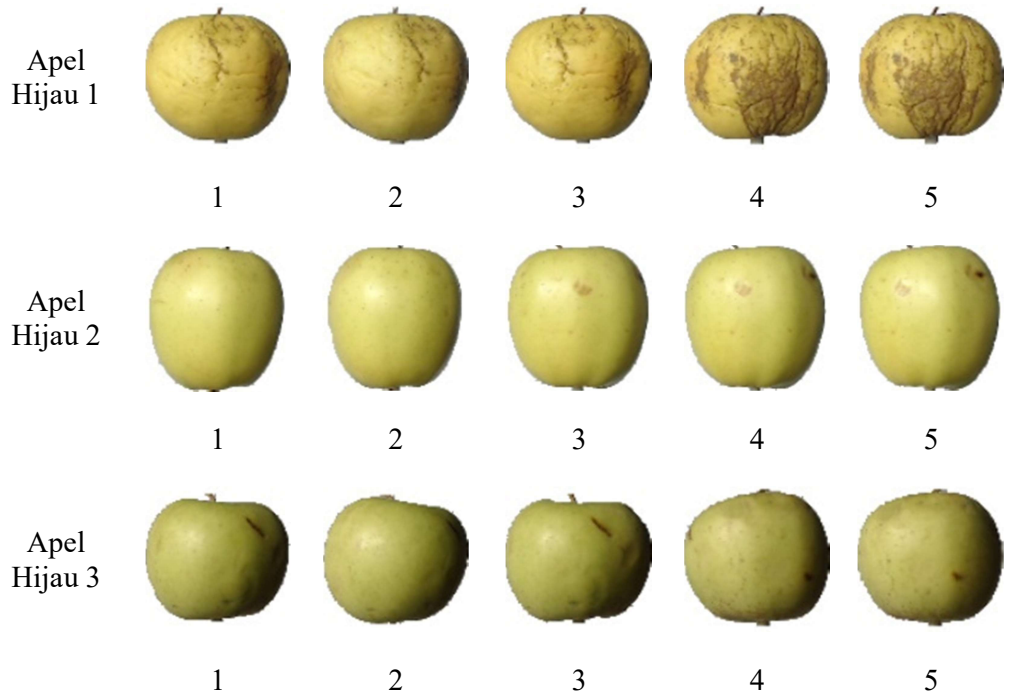
Pada Kode 4.23 digunakan untuk menampilakan file data pelatihan dengan format *dataframe* dijadikan menjadi list.

Tabel 4. 6 Cuplikan Isi Data Pengujian

	0	1	2	3	...	17	18	19	20
0	H1	0.020325	0.287295	148.645234	...	0.013250	0.003394	4087.395363	0.354632
1	H1	0.013494	0.141198	181.097843	...	0.010152	0.005374	3465.112826	0.420846
2	H1	0.013044	0.128931	216.547867	...	0.009958	0.005788	3433.980241	0.365963
3	H1	0.013073	0.120726	215.600190	...	0.010031	0.005512	3379.612763	0.313861
...	...	...	...	...	...	...	...	...	...
26	H3	0.023902	0.326096	202.746955	...	0.015057	0.007324	3164.690486	0.416168
27	H3	0.021819	0.311135	228.042049	...	0.014124	0.037804	3322.467476	0.524070
28	H3	0.023898	0.320715	190.256379	...	0.015542	0.007164	3198.108803	0.421703
29	H3	0.023387	0.302549	197.835409	...	0.015613	0.006859	3227.676675	0.448475

Pada Tabel 4.5 merupakan cuplikan dari Data Pengujian yang berjumlah

30 data yang terdiri dari 3 kelas apel yang masing-masing kelas berjumlah 10 *dataset*.



Gambar 4. 1 Contoh Beberapa Dataset Apel Hijau Semua Kelas

Kode 4. 24 Kode Untuk Menetapkan *folds* dan *num_neighbors*

```
n_folds = 5
num_neighbors = 4

actual_pred_pairs = evaluate_algorithm(df_list,
k_nearest_neighbors, n_folds, num_neighbors)
```

Pada Kode 4.24 di gunakan untuk menetapkan *folds* nya = 5 dan *neighbors* nya = 4 Serta membuat fungsi *evaluate* untuk mengevaluasi nilai prediksi dan di cocokan dengan nilai aktual.

4.5 Pengujian Matrix

Kode 4. 25 Menetapkan Nilai untuk Pengujian Matrix

```
def tp_fp_tn_fn_count(actual, pred, label):
    actual_is_label_list = []
    pred_is_label_list = []

    for a in actual:
        actual_is_label_list.append(a == label)
    for p in pred:
        pred_is_label_list.append(p == label)

  # Menghitung:
    tp = 0 # True Positive
    fp = 0 # False Positive
    tn = 0 # True Negative
    fn = 0 # False Negative

    for i in range(len(actual_is_label_list)):
        a = actual_is_label_list[i]
        p = pred_is_label_list[i]
        if (a and p):
            tp += 1
        elif (not a and not p):
            tn += 1
        elif (a and not p):
            fn += 1
        else:
            fp += 1
    return (tp, fp, tn, fn)

# Menghitung true postive rate / sesitivity
def tp_rate_metric(actual, pred, label):
    tp, fp, tn, fn = tp_fp_tn_fn_count(actual, pred ,label)
    return tp/(tp+fn)

# Menghitung false postive rate / specificity
def fp_rate_metric(actual, pred, label):
    tp, fp, tn, fn = tp_fp_tn_fn_count(actual, pred ,label)
    return fp/(fp+tn)

# Menghitung accuracy
def accuracy_metric(actual, pred, label):
    tp, fp, tn, fn = tp_fp_tn_fn_count(actual, pred ,label)
    return (tp + tn)/(tp + fp + tn + fn)

# Menghitung misclassification / error rate
def misclassification_metric(actual, pred, label):
```

```
    tp, fp, tn, fn = tp_fp_tn_fn_count(actual, pred ,label)
    return (fp + fn)/(tp + fp + tn + fn)
```

Pada Kode 4.25 untuk menghitung *True Positive*, *False Positive*, T*rue Negative*, dan *False Negative*. Kemudian mengubah nilai *actual* dan *pred* menjadi tipe data *boolean* dengan cara membuat variabel dan melakukan perulangan serta melakukan pengecekan.

Setelah melakukan semua proses, selanjutnya menetapkan rumus akurasi, *error rate, sensitivity* dan *, specificity*. Perhitungan nya sesuai dengan Tabel 2.5 dan rumus nya sesuai dengan Persamaan 2.8 hingga Persamaan 2.11.

Kode 4. 26 *Import* Modul

```
from sklearn.metrics import confusion_matrix
```

Pada Kode 4.26 untuk menngimport modul dari *sklearn.metrics* yang digunakan untuk menggunakan Tabel matriks seperti pada Tabel 2.5.

Kode 4. 27 Untuk *Output Confusion Matrix*

```
pred = k_nearest_neighbors(df_list, df_test_list, num_neighbors)
actual = [df_test_list[i][0] for i in range(len(df_test_list))]

confusion = confusion_matrix(actual, pred)
display(confusion)
```

Pada Kode 4.27 adalah digunakan untuk menampilkan hasil *matrix* dari variabel *actual* dan variabel pred. yang pertama terdapat variabel *pred* yang berisi data latih yang diambil dari fungsi *k_nearest_neighbors*, kemudian yang kedua terdapat variabel *actual* yang berisi seluruh data pelatihan.

Tabel 4. 7 Nilai *Matrix* Dari Data *Actual* Dan Data *Pred*

```
array([[ 3,  2,  5],
       [ 5,  5,  0],
       [ 0,  0, 10]]
```

Tabel 4. 8 Detail *Confusion Matrix* 3x3

		PREDIKSI		
		Apel H1	Apel H2	Apel H3
AKTUAL	Apel H1	3	2	5
	Apel H2	5	5	0
	Apel H3	0	0	10

Setelah memperoleh nilai *Confusion Matrix* nya sesuai dengan Tabel 4.7 selanjutnya mencari nilai TP, FP, FN, dan TN seperti Tabel 2.5 yang sesuai masing-masing kelas Apel Hijau.

Tabel 4. 9 *Confusion Matrix* Apel Hijau 1

H1		PREDIKSI	
		POSITIF	NEGATIF
AKTUAL	POSITIF	3	(2 + 5) = 7
	NEGATIF	(5 + 0) = 5	(10 + 5 + 0 + 0) = 15

Tabel 4. 10 *Confusion Matrix* Apel Hijau 2

H2		PREDIKSI	
		POSITIF	NEGATIF
AKTUAL	POSITIF	5	(5 + 0) = 5
	NEGATIF	(2 + 0) = 2	(10 + 5 + 3 + 0) = 18

Tabel 4. 11 *Confusion Matrix* Apel Hijau 3

H3	PREDIKSI

		POSITIF	NEGATIF
AKTUAL	POSITIF	10	(0 + 0) = 0
	NEGATIF	(5 + 0) = 5	(5 + 5 +3 + 2) = 15

Kode 4. 28 *Import* Modul

```
import functools
```

Pada Kode 4.28 merupakan *Import* modul yang digunakan mempermudah mengambil fungsi lain sebagai argument.

Kode 4. 29 Untuk Menampilkan Uji Coba Akurasi 5-fold

```
labels = df[0].unique()

# Kolom metrics yang digunakan
cols = [
    "sensitivity",
    "specificity",
    "accuracy",
    "error rate",
]

table_arr = []

# Kalkulasi metrics setiap label
for label in labels:
    folds_arr = []
    # Hitung metric setiap fold
    for (actual, pred) in actual_pred_pairs:
        folds_arr.append([
            sensitivity_metric(actual, pred, label),
            specificity_metric(actual, pred, label),
            accuracy_metric(actual, pred, label),
            misclassification_metric(actual, pred, label),
        ])
    # Menghitung rata-rata nilai metrics setiap fold dalam satu
label
    folds_count = len(folds_arr)
    folds_arr = functools.reduce(lambda a, b: [a[i]+b[i] for i in
range(len(a))], folds_arr)
    table_arr.append([x/folds_count for x in folds_arr])
```

```
# Menampilkan dalam bentuk tabel
table = pd.DataFrame(columns=cols, data=table_arr, index=labels)
table.T
```

Pada Kode 4.29 terdapat variabel *labels* yang digunakan untuk menampilkan label yang ada di *dataset*. Kemudian menyiapkan kolom *matrix* yang digunakan seperti *accuracy*, *error rate*, *sensitivity*, dan *specificity*. Setelah itu membuat variabel *table_arr* yang digunakan untuk menampung data, kemudian melakukan kalkulasi *matrix* pada setiap label dengan nama variabel *label*. Langkah selanjutnya melakukan perulangan matrix setiap fold nya serta menghitung rata-rata nilai *matrix* setiap fold dalam satu label.

Tabel 4. 12 Hasil Uji Coba Masing-masing Kelas Apel Hijau dengan 5-fold

n-fold Cross validation	 **Apel Hijau 1**	 **Apel Hijau 2**	 **Apel Hijau 3**
	5-fold	**5-fold**	**5-fold**
Accuracy	0.825	0.80	0.958
Error Rate	0.175	0.20	0.042
Sensitivity	0.671	0.733	0.978
Specificity	0.10	0.159	0.053

Hasil akhir dari Tabel 4.11 terdapat Apel Hijau 1 Mendapatkan akurasi 0.825, kemudian tingkat kesalahannya 0.175, untuk tingkat *sensitivity* mendapatkan 0.671 dan untuk tingkat *specificity* mendapatkan 0.10.

Apel Hijau 2 Mendapatkan akurasi 0.80, kemudian tingkat kesalahannya 0.20, untuk tingkat *sensitivity* mendapatkan 0.733 dan untuk tingkat *specificity* mendapatkan 0.159.

Apel Hijau 3 Mendapatkan akurasi 0.958, kemudian tingkat kesalahannya 0.042, untuk tingkat *sensitivity* mendapatkan 0.978 dan untuk tingkat *specificity* mendapatkan 0.053.

Kode 4. 30 Kode Import Modul

```
import numpy as np
```

Pada Kode 4.30 adalah modul yang digunakan untuk proses komputasi numerik dalam bentuk *array*, kelebihan dari modul ini adalah memudahkan operasi komputasi pada data serta dapat dilakukan secara acak.

Kode 4. 31 Kode Untuk Menampilkan Akurasi Pada Data Pelatihan

```
import numpy as np

# ----------------------------------------------

def calculate_accuracy(actual, pred):
    equals = np.array(actual) == np.array(pred)
    return len(equals[equals == True])/len(actual)

def calculate_error(actual, pred):
    return 1-calculate_accuracy(actual, pred)

def calculate_sensitivity(actual, pred):
    sensitivity = 0
    for label in labels:
        sensitivity += sensitivity_metric(actual, pred, label)
        sensitivity /= len(labels)
    return sensitivity

def calculate_specificity(actual, pred):
    sensitivity = 0
```

```
    for label in labels:
        sensitivity += specificity_metric(actual, pred, label)
        sensitivity /= len(labels)
    return sensitivity

# -------------------------------------------------

accuracies = []
errors = []
sensitivitys = []
specificitys= []

for actual, pred in actual_pred_pairs:
    accuracies.append(calculate_accuracy(actual, pred))

for actual, pred in actual_pred_pairs:
    errors.append(calculate_error(actual, pred))

for actual, pred in actual_pred_pairs:
    sensitivitys.append(calculate_sensitivity(actual, pred))

for actual, pred in actual_pred_pairs:
    specificitys.append(calculate_specificity(actual, pred))

print("Accuracies:", accuracies)
print("Score:", sum(accuracies)/len(accuracies))

print("Errors:", errors)
print("Score:", sum(errors)/len(errors))

print("Sensitivitys:", sensitivitys)
print("Score:", sum(sensitivitys)/len(sensitivitys))

print("Specificitys:", specificitys)
print("Score:", sum(specificitys)/len(specificitys))
```

Kode 4.31 terdapat beberapa fungsi yaitu *calculate_accuracy*, *calculate_error*, *calculate_sensitivity*, dan *calculate_specificity* yang didalam fungsi tersebut terdapat eksekusi salah satunya melakukan perulangan yang kemudian hasilnya dikembalikan sesuai masing-masing variabel yang sudah di tetapkan dalam setiap fungsi.

Kemudian terdapat 4 variabel *accuracies*, *errors*, *sensitivitys*, dan *specificitys* untuk menyimpan nilai sesuai jumlah fold nya dengan tipe data *array*

Tabel 4. 13 Hasil Uji Coba Akurasi Seluruh Apel Hijau

```
Accuracies: [0.7083333333333334, 0.7916666666666666, 0.7916666
666666666, 0.75, 0.75]
Score: 0.7583333333333333
Errors: [0.29166666666666663, 0.20833333333333337, 0.208333333
33333337, 0.25, 0.25]
Score: 0.2416666666666667
Sensitivitys: [0.375, 0.4646464646464647, 0.441358024691358, 0
.39393939393939387, 0.4537037037037037]
Score: 0.42572951739618403
Specificitys: [0.4375, 0.4522417153996101, 0.43857116920842415
, 0.42962962962962964, 0.4398148148148148]
Score: 0.43955146581049576
```

Tabel 4. 14 Detail Hasil Uji Coba Seluruh Apel Hijau

	1-fold	**2-fold**	**3-fold**	**4-fold**	**5-fold**
Akurasi	0.708	0.792	0.792	0.75	0.75
Rata-rata	0.758				

Error	0.292	0.208	0.208	0.25	0.25
Rata-rata	0.242				

Sensitivity	0.375	0.465	0.441	0.394	0.454
Rata-rata	0.426				

Specificity	0.437	0.452	0.438	0.430	0.440
Rata-rata	0.439				

Hasil akhir dari Tabel 4.13 mendapat akurasi 75.83%, kemudian tingkat kesalahannya 24.17%, untuk tingkat *sensitivity* mendapatkan 42.57% dan untuk tingkat *specificity* mendapatkan 43.95%.