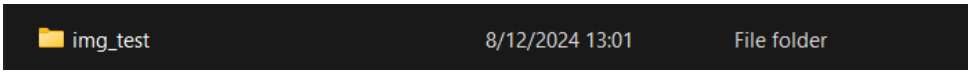
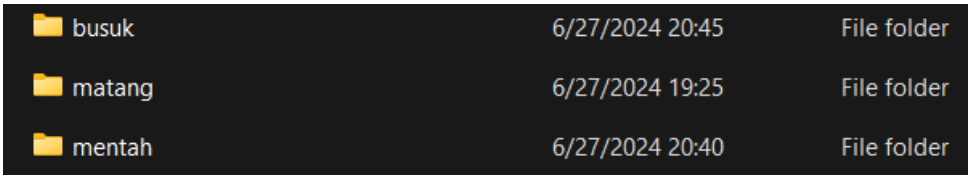
BAB IV
PEMBAHASAN

4.1 Data Collection

Setelah semua dataset yang diperlukan sudah terkumpul maka selanjutnya dilakukan penggabungan dataset dalam satu folder yang bernama img_test yang didalamnya berisikan 3 kondisi yang akan diuji yaitu matang, mentah, busuk.



Gambar 4. 1 Folder dataset

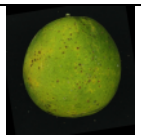
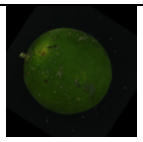


Gambar 4. 2 Folder pembagian kondisi dataset

Karena terdapat dua dataset yang berbeda yaitu dataset Orange Disease dan dataset Fresh and Stale Images of Fruits and Vegetables oleh karena itu dilakukan penggabungan dataset menjadi satu kemudian jumlah dari citra dari masing – masing kondisi berjumlah 500 dengan total keseluruhan dataset yang dipakai yaitu 1500 citra dan didapatkan hasil dari penggabungan seperti pada tabel 4.1 berikut

Tabel 4. 1 Contoh dataset yang sudah digabung

Kondisi	Contoh dataset
Busuk	
Matang	

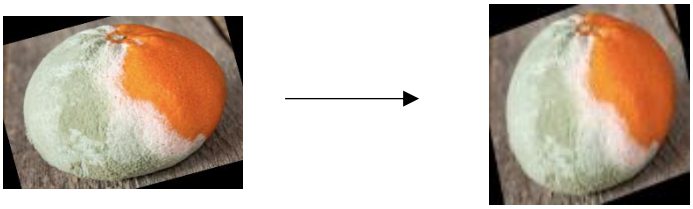
Mentah					

4.2 Preprocessing Data

Karena pada dataset yang diambil memiliki resolusi dan jenis citra yang berbeda – beda maka akan dilakukan penyesuaian citra agar semua citra sama yang bertujuan agar pada saat pemrosesannya lebih cepat. Pada penelitian ini, dilakukan cropping dengan ukuran 100x100 pixels menggunakan bantuan program python untuk proses croppingnya.

```python
import os
import cv2

def resize_and_convert_images(input_folder, output_folder, size=(100, 100)):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    image_counter = 1

    for filename in sorted(os.listdir(input_folder)):
        if filename.lower().endswith(('.jpg', '.jpeg', '.bmp', '.gif', '.tiff', '.png')):
            img_path = os.path.join(input_folder, filename)
            img = cv2.imread(img_path)
            if img is None:
                print(f"Failed to load {filename}. Skipping...")
                continue

            img_resized = cv2.resize(img, size, interpolation=cv2.INTER_AREA)

            output_filename = f'mentah{image_counter}.png'
            output_path = os.path.join(output_folder, output_filename)

            cv2.imwrite(output_path, img_resized)
            print(f"Processed {filename} -> {output_filename}")

            image_counter += 1

input_folder = 'D:/Skripsi/img_test/busukasli'
output_folder = 'D:/Skripsi/img_test/busuk'
resize_and_convert_images(input_folder, output_folder, size=(100, 100))
```

Gambar 4. 3 Program python yang dipakai



Gambar 4. 4 Contoh perubahan citra

4.3 Model Selection, Training and Validation

4.3.1 Pembuatan Model CNN dan PCA

Pada tahap awal yang dilakukan sebelum memproses dataset yaitu mengubahnya menjadi grayscale terlebih dahulu, hal ini digunakan supaya pada saat proses klasifikasi nanti dapat lebih efisen.

```
imds.ReadFcn = @(loc) imresize(rgb2gray(imread(loc)), imageSize);
```

Gambar 4. 5 kode untuk mengubah gambar rgb ke grayscale

Gambar 4. 6 Perubahan citra

Tahap selanjutnya yaitu pengaplikasian PCA untuk mereduksi dataset supaya pada saat proses klasifikasi dapat mengefisiensi waktu.

```
numComponents = 50;

[trainFeatures, trainLabels] = applyPCA(imdsTrain, imageSize, numComponents);
[testFeatures, testLabels] = applyPCA(imdsTest, imageSize, numComponents);
```

Gambar 4. 7 Kode untuk menerapkan PCA

Pada gambar 4.6 adalah kode untuk menerapkan PCA, dimana numComponents = 50; adalah jumlah komponen yang ingin dipertahankan sebesar 50% dari jumlah komponen aslinya. Baris kode applyPCA(imdsTrain, imageSize, numComponents) dan applyPCA(imdsTest, imageSize, numComponents) yaitu berfungsi untuk menerapkan PCA pada dataset pelatihan dan pengujian, mengurangi dimensi gambar. Hasilnya adalah satu set fitur (trainFeatures dan testFeatures) yang mewakili gambar dalam ruang

berdimensi lebih rendah, bersama dengan label yang sesuai (trainLabels dan testLabels).

```
layers = [
    imageInputLayer([1 1 numComponents], 'Normalization', 'none')

    convolution2dLayer(1, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(1, 'Stride', 1)

    convolution2dLayer(1, 64, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(1, 'Stride', 1)

    convolution2dLayer(1, 128, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(1, 'Stride', 1)

    fullyConnectedLayer(numel(categories))
    softmaxLayer
    classificationLayer];
```

Gambar 4. 8 Kode untuk membangun arsitektur CNN

Kode yang diberikan mendefinisikan arsitektur jaringan saraf convolutional (CNN) yang dirancang untuk klasifikasi citra setelah data mengalami proses Principal Component Analysis (PCA). Jaringan ini dimulai dengan lapisan input yang menerima data citra yang telah diproses menjadi bentuk vektor dengan dimensi numComponents, tanpa normalisasi tambahan karena data sudah diproses sebelumnya.

Selanjutnya, jaringan ini melibatkan beberapa lapisan konvolusi dengan ukuran filter 1x1 yang digunakan untuk mengekstraksi fitur dari data, diikuti oleh lapisan batch normalization untuk menormalkan output dan lapisan ReLU untuk menambahkan non-linearitas.

Setiap lapisan konvolusi diikuti oleh lapisan pooling, meskipun ukuran filter pooling adalah 1x1 sehingga tidak mengubah dimensi.

Jaringan ini mengulangi proses konvolusi, normalisasi, ReLU, dan pooling beberapa kali dengan jumlah filter yang meningkat (32, 64, dan 128) untuk menangkap fitur yang semakin kompleks.

Di akhir arsitektur, terdapat lapisan fully connected yang menghubungkan semua neuron sebelumnya ke neuron pada lapisan ini, jumlah neuron pada lapisan ini sesuai dengan jumlah kategori kelas yang ada. Lapisan softmax kemudian mengubah skor menjadi probabilitas untuk setiap kelas, dan lapisan classification layer terakhir menghitung loss untuk mengoptimalkan model selama pelatihan. Arsitektur ini dirancang untuk menangani data citra yang telah direduksi dimensinya oleh PCA, menggunakan struktur CNN yang sederhana namun efektif untuk tugas klasifikasi.

```
% Set training
options = trainingOptions('adam', ...
    'InitialLearnRate', 1e-4, ...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 32, ...
    'ValidationData', {testFeatures, testLabels}, ...
    'ValidationFrequency', 30, ...
    'Verbose', true, ...
    'Plots', 'training-progress', ...
    'OutputFcn', @(info) writeProgressToLog(info, logFilePath));
```

Gambar 4. 9 kode untuk training dataset

Training data dilakukan selama 20 epoch, di mana setiap epoch melibatkan satu kali pengulangan melalui seluruh set data pelatihan. Selain itu, data pelatihan dipecah menjadi mini-batch berukuran 32 sampel per batch, yang memungkinkan pembaruan bobot lebih sering tanpa harus menunggu seluruh dataset selesai diproses.

Untuk memantau performa model selama pelatihan, data validasi digunakan pada interval tertentu, dalam hal ini setiap 30 iterasi mini-batch. Hasil dari validasi ini tidak digunakan untuk memperbarui model, tetapi membantu dalam mengevaluasi apakah model mengalami overfitting atau masih berada pada jalur yang benar. Pengaturan `Verbose` diaktifkan, sehingga Anda akan mendapatkan

informasi terperinci tentang proses pelatihan secara real-time, termasuk nilai loss dan akurasi. Selain itu, MATLAB akan menampilkan grafik yang memvisualisasikan progres pelatihan tersebut.

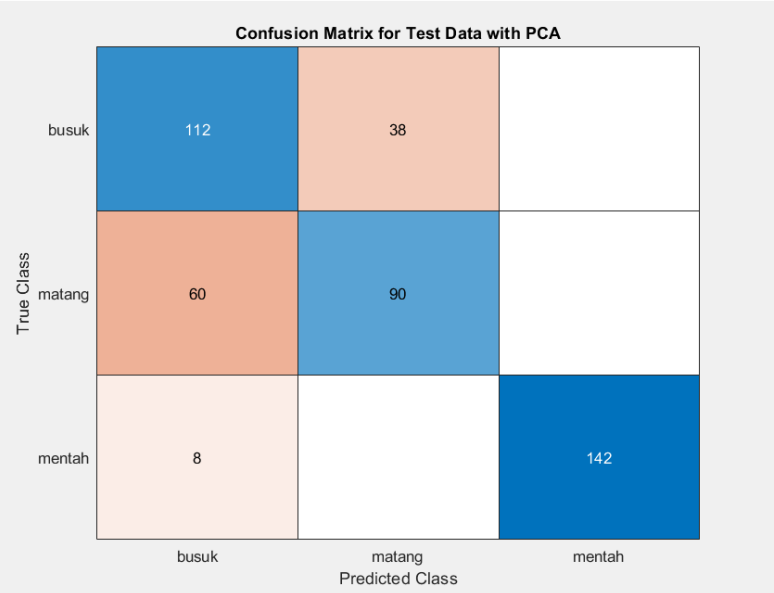
Akhirnya, terdapat fungsi kustom yang digunakan untuk mencatat perkembangan pelatihan ke dalam file log. Fungsi ini secara otomatis mencatat berbagai informasi seperti loss dan akurasi pada setiap epoch, sehingga Anda dapat melacak perkembangan pelatihan jaringan saraf tiruan Anda dengan lebih mudah. Dan berikut adalah hasil dari training datanya terdapat pada Tabel 4.2

Tabel 4. 2 Hasil training Adam

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss
1	1	00:00:02	18.75	24.67	1.6038	1.6049
1	30	00:00:02	56.25	50.89	0.9866	1.1506
2	50	00:00:03	65.62	62.0	0.8403	0.9027
2	60	00:00:03	65.62	62.0	0.767	0.9027
3	90	00:00:04	68.75	65.78	0.6875	0.7816
3	100	00:00:04	65.62	65.62	0.672	0.7313
4	120	00:00:05	78.12	68.22	0.5541	0.7136
4	150	00:00:05	81.25	69.33	0.5251	0.6713
5	180	00:00:06	75.0	70.89	0.527	0.6424
7	200	00:00:07	68.75	72.22	0.5143	0.6225
7	210	00:00:07	78.12	72.22	0.4491	0.6077
8	240	00:00:08	96.88	72.44	0.3796	0.5962
9	250	00:00:08	84.38	72.22	0.5437	0.5868
9	270	00:00:09	71.88	72.44	0.45	0.5962
10	300	00:00:09	78.12	72.22	0.3999	0.5873
11	330	00:00:10	84.38	72.44	0.4919	0.5804
11	350	00:00:10	87.5	72.0	0.3368	0.5734

12	360	00:00:11	81.25	72.0	0.389	0.5753
13	390	00:00:11	87.5	71.56	0.3368	0.5715
14	420	00:00:12	87.5	71.33	0.315	0.5681
15	450	00:00:13	78.12	71.33	0.3615	0.5681
15	480	00:00:14	90.62	71.11	0.4441	0.5634
16	510	00:00:14	87.5	71.11	0.3564	0.5634
17	540	00:00:15	87.5	71.11	0.338	0.5618
18	550	00:00:15	96.88	71.33	0.2305	0.5604
19	600	00:00:17	93.75	71.56	0.2869	0.5579
20	630	00:00:17	96.88	71.56	0.2407	0.5571
20	640	00:00:17	90.62	71.33	0.3172	0.5573

Pada proses selanjutnya akan dilakukan evaluasi model yang sudah dibuat sebelumnnya, model evaluasi yang akan dipakai berupa confusion matrix.



Gambar 4. 10 Confusion Matrix CNN dan PCA

Hasil yang didapatkan yaitu

Diagonal (warna biru): Menunjukkan jumlah data yang diklasifikasikan dengan benar oleh model.

- Busuk ke busuk: 112 data diklasifikasikan dengan benar sebagai busuk.
- Matang ke matang: 90 data diklasifikasikan dengan benar sebagai matang.
- Mentah ke mentah: 142 data diklasifikasikan dengan benar sebagai mentah.

Off-Diagonal (warna lain): Menunjukkan jumlah data yang diklasifikasikan dengan salah oleh model.

- Busuk ke matang: 28 data sebenarnya busuk, tapi diklasifikasikan sebagai matang.
- Mentah ke busuk: 8 data sebenarnya busuk, tapi diklasifikasikan sebagai mentah.
- Matang ke busuk: 60 data sebenarnya matang, tapi diklasifikasikan sebagai busuk.

Dari hasil confusion matrix tersebut selanjutnya dapat digunakan untuk mengukur kinerja model seperti akurasi, recall, f1-score, dan precision. Hasilnya seperti gambar berikut

```
Test Accuracy: 0.76444
Classification Report:
---------------------
Class      | Precision  | Recall     | F1 Score
---------------------
busuk      | 0.6222     | 0.7467     | 0.6788
matang     | 0.7031     | 0.6000     | 0.6475
mentah     | 1.0000     | 0.9467     | 0.9726
---------------------
Macro Average:
Macro Avg  | 0.7751     | 0.7644     | 0.7663
```

Gambar 4. 11 Hasil Klasifikasi

Seperti yang terlihat pada Gambar 4.10 dapat dilihat bahwa nilai rata – rata *accuracy* sebesar 76,4%, *Precision* 77,51%, *Recall* 76,44%, *F1 Score* 76,63%. Dilihat dari hasil klasifikasi diatas bahwa kombinasi antara PCA dengan CNN dengan dataset yang diubah ke

grayscale tingkat akurasi belum cukup tinggi dalam menentukan tingkat kematangan buah jeruk keprok.

4.3.2 Pembuatan model PCA dan SVM

Seperti pembuatan model sebelumnya yaitu pca dan cnn, karena dataset yang saya pakai yaitu dataset berupa gambar yang berwarna atau rgb maka hal pertama yang dilakukan yaitu mengubah dataset dari rgb ke *grayscale*. Berikut kode yang saya pakai untuk mengubahnya

```
% Convert images to grayscale
imds.ReadFcn = @(filename) rgb2gray(imread(filename));
```

Gambar 4. 12 Mengubah gambar dari rgb ke grayscale

Selanjutnya dilakukan pembagian dataset untuk dilakukan training dan test dimana data yg digunakan yaitu 70% untuk data training dan 30% untuk data testing. Pembagian data ini dilakukan dengan cara otomatis dengan menggunakan code. Setelah itu dilakukan ekstraksi feature dataset menggunakan squeezenet, supaya nantinya dapat diklasifikasikan kelasnya. Berikut codenya

```
[imdsTrain, imdsTest] = splitEachLabel(imds, 0.7, 'randomized');

% Extract features
net = squeezenet;
inputSize = net.Layers(1).InputSize;
```

Gambar 4. 13 Code untuk membagi data dan ekstraksi feature dataset

Setelah ekstraksi feature dilakukan selanjutnya dilakukan pengaplikasian PCA dengan reduksi data 50%, hal ini bertujuan supaya pada saat klasifikasinya tidak memerlukan waktu lama dan tidak terlalu banyak data yang hilang dalam dataset tersebut. Berikut codenya :

```
numComponents = round(size(featuresTrain, 2) / 2);
[coeff, scoreTrain] = pca(featuresTrain, 'NumComponents', numComponents);
scoreTest = featuresTest * coeff(:, 1:numComponents);
```

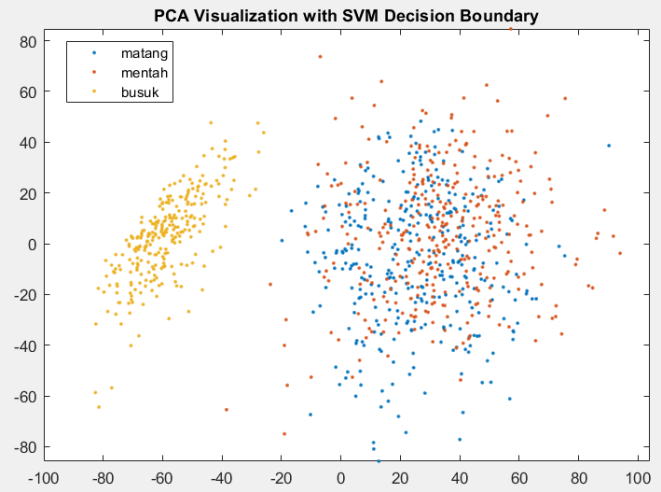
Gambar 4. 14 Code PCA

Setelah proses semua selesai selanjutnya dilakukan klasifikasi menggunakan SVM. Berikut codenya

```
% Train the SVM classifier
classifier = fitcecoc(featuresTrain, labelsTrain);

% Predict the labels of the test set
predictedLabels = predict(classifier, featuresTest);
```

Gambar 4. 15 Code klasifikasi SVM

Pada code train SVM diatas adalah code utama untuk mengklasifikasi dataset dengan mengambil ekstraksi feature dari dataset yang sudah dilakukan sebelumnya menggunakan Squeezenet. Dan selanjutnya akan dilakuan prediksi label dengan menggunakan ekstraksi feature dataset test yang nantinya akan menentukan apakah data tersebut termasuk kategori busuk, matang, mentah. Berikut decision boundary dari svm

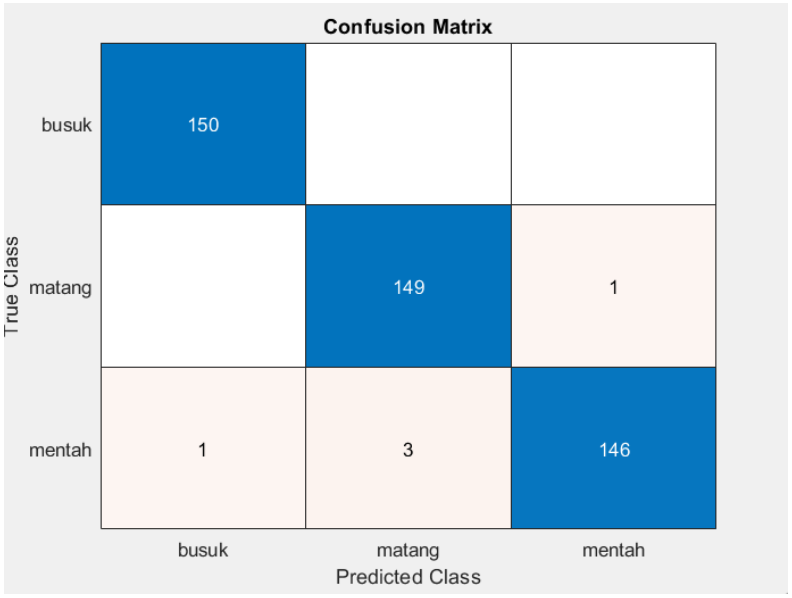


Gambar 4. 16 Decision Boundary

Dari Decision Boundry diatas bisa dilihat bahwa kategori dari dataset jeruk busuk paling terlihat perbedaannya dibanding matang dan mentah, maka bisa dipastikan akurasi untuk kategori busuk

sangat bisa dibedakan oleh svm berbeda dengan matang dan mentah hampir menyatu di satu area yang sama.

Untuk hasil klasifikasi bisa dilihat diconfusion matrix pada gambar 4.17 berikut.



Gambar 4. 17 Hasil Confusion Matrix SVM dan PCA

Sesuai dari decision boundry diatas bahwasannya kategori busuk memiliki tingkat akurasi yang sangat tinggi dengan akurasi 100%, untuk kategori matang dan busuk masih tergolong cukup tinggi juga dengan prediksi matang yang benar sebanyak 149 dan hanya 1 salah prediksi, begitu juga dengan mentah jumlah data yang benar sejumlah 146, dan 4 prediksi yang salah.

```
Test Result:
Accuracy: 98.89%

Results:
Category    Precision    Recall    F1 Score    Support
  matang     98.03%     99.33%     98.68%     150
  mentah     96.05%     97.33%     96.69%     150
  busuk     98.68%     100.00%     99.34%     150

Mean Results:
Precision: 97.59%
Precision: 100.91%
Precision: 98.23%
Recall: 98.89%
F1 Score: 98.23%
F1 Score: 99.89%
F1 Score: 98.56%
```

Gambar 4. 18 Nilai dari klasifikasi SVM dan PCA

Hasil dari klasifikasi SVM dan PCA didapat bahwa tingkat akurasi dari penggabungan dua algoritma tersebut dengan pembagian data training dan testing sebesar 70% dan 30% memiliki tingkat akurasi sebesar 98.89%, *Precision* 97.58%, *Recall* 98.88%, dan *F1-Score* 98.23%. Dari hasil tersebut bisa disimpulkan bahwa penggabungan kedua algoritma ini memiliki tingkat akurasi yang sangat tinggi untuk mengklasfikasi tingkat kematangan buah jeruk keprok.

4.3.3 Pembuatan Model CNN

Pada tahap awal seperti yang sebelumnya dilakukan yaitu mengubah dataset yang sebelumnya rgb ke grayscale terlebih dahulu.

Gambar 4. 19 perubahan dari rgb ke grayscale

Setelah melalui beberapa tahapan, maka selanjutnya data yang telah diproses akan di-training. Arsitektur sangat berpengaruh dalam pembuatan model deteksi jenis sampah dengan metode CNN ini dan sangat mempengaruh hasil dari akurasi model. Berikut layer yang dipakai :

Gambar 4. 20 Layer CNN

Layer input menerima data gambar dengan dimensi 100x100 piksel dalam format grayscale (1 channel) tanpa normalisasi otomatis. Kemudian, terdapat tiga blok yang terdiri dari layer konvolusi 2D

```
layers = [
    imageInputLayer([100 100 1], 'Normalization', 'none')

    convolution2dLayer(3, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 64, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 128, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    fullyConnectedLayer(numel(categories))
    softmaxLayer
    classificationLayer];
```

dengan ukuran kernel 3x3, diikuti oleh batch normalization untuk mempercepat pelatihan dengan menormalkan keluaran layer sebelumnya.

Fungsi aktivasi ReLU (Rectified Linear Unit) untuk menangani non-linearitas, dan max pooling 2D dengan ukuran 2x2 serta langkah (stride) 2 untuk mengurangi dimensi data sambil mempertahankan informasi penting. Jumlah filter di setiap blok bertambah secara bertahap, mulai dari 32, 64, hingga 128, untuk menangkap fitur yang semakin kompleks. Setelah prosses ini selesai selanjutnya masuk ke proses training dataset.

```
% Set training
options = trainingOptions('adam', ...
    'InitialLearnRate', 1e-4, ...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 32, ...
    'ValidationData', {testFeatures, testLabels}, ...
    'ValidationFrequency', 30, ...
    'Verbose', true, ...
    'Plots', 'training-progress', ...
    'OutputFcn', @(info) writeProgressToLog(info, logFilePath));
```

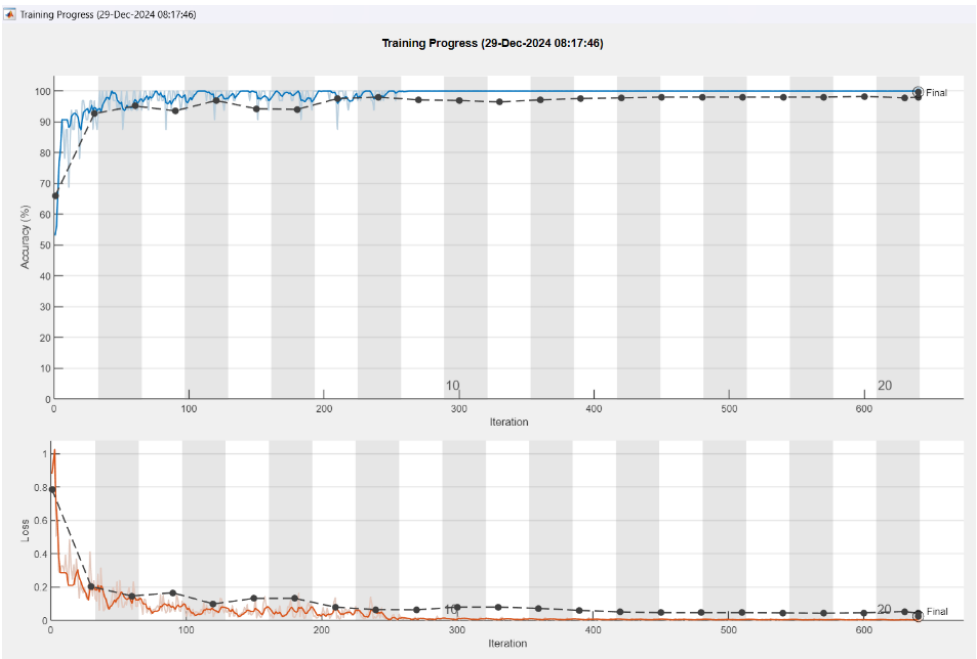
Gambar 4. 21 Code untuk training dataset

Pada proses training ini menggunakan model training adam dengan epoch yang dipakai sebanyak 20 kali.

Tabel 4. 3 Training Adam

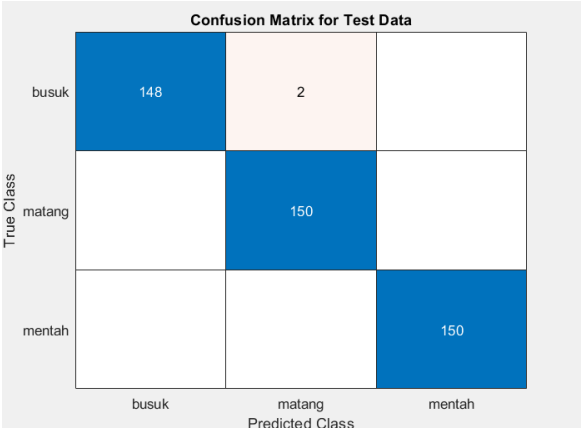
Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss
1	1	00:00:40	53.12%	66.00%	0.8801	0.7855
1	30	00:00:43	96.88%	92.67%	0.2517	0.2037
2	50	00:00:44	100.00%	95.11%	0.0828	0.1471
2	60	00:00:44	100.00%	95.11%	0.0680	0.1471
3	90	00:00:46	96.88%	93.56%	0.1087	0.1665
4	100	00:00:46	100.00%	96.89%	0.0250	0.0998
4	120	00:00:47	100.00%	96.89%	0.0224	0.0998
5	150	00:00:48	100.00%	94.22%	0.0569	0.1327
6	180	00:00:50	100.00%	94.00%	0.0190	0.1330
7	200	00:00:50	100.00%	97.56%	0.0449	0.0801
7	210	00:00:51	87.50%	97.56%	0.1361	0.0801
8	240	00:00:52	100.00%	98.00%	0.0149	0.0653
8	250	00:00:53	100.00%	98.00%	0.0105	0.0653
9	270	00:00:54	100.00%	97.11%	0.0060	0.0643
10	300	00:00:55	100.00%	96.89%	0.0168	0.0807
11	330	00:00:57	100.00%	96.44%	0.0047	0.0807
11	350	00:00:58	100.00%	97.11%	0.0087	0.0729

12	360	00:00:58	100.00%	97.11%	0.0043	0.0729
13	390	00:01:00	100.00%	97.56%	0.0062	0.0610
13	400	00:01:01	100.00%	97.56%	0.0087	0.0610
14	420	00:01:02	100.00%	97.78%	0.0075	0.0516
15	450	00:01:04	100.00%	98.00%	0.0079	0.0471
15	480	00:01:06	100.00%	98.00%	0.0069	0.0466
16	500	00:01:06	100.00%	98.00%	0.0035	0.0470
17	540	00:01:08	100.00%	98.00%	0.0044	0.0454
18	550	00:01:09	100.00%	98.00%	0.0047	0.0438
19	600	00:01:11	100.00%	98.22%	0.0030	0.0461
20	630	00:01:13	100.00%	97.78%	0.0038	0.0543
20	640	00:01:13	100.00%	98.00%	0.0046	0.0429



Gambar 4. 22 Training Progress CNN

Seperti yang terlihat pada hasil training pada tabel 4.3 bahwasannya terjadi peningkatan accuracy disetiap epoch yang dilakukan. Pada epoch ke 8 merupakan validation tertinggi dengan tingkat validation sebesar 98%, untuk epoch selanjutnya tidak ada perubahan yang cukup signifikan. Dengan training accuracy mencapai 100% yang berarti sangat akurat dalam proses training berlangsung. Setelah proses training dilakukan selanjutnya dilakukan evaluasi model dengan menggunakan confusion matrix



Gambar 4. 23 Confusion Matrix CNN

Hasil yang didapatkan yaitu

Diagonal (warna biru): Menunjukkan jumlah data yang diklasifikasikan dengan benar oleh model.

- Busuk ke busuk: 148 data diklasifikasikan dengan benar sebagai busuk.
- Matang ke matang: 150 data diklasifikasikan dengan benar sebagai matang.
- Mentah ke mentah: 150 data diklasifikasikan dengan benar sebagai mentah.

Off-Diagonal (warna lain): Menunjukkan jumlah data yang diklasifikasikan dengan salah oleh model.

- Busuk ke matang: 2 data sebenarnya busuk, tapi diklasifikasikan sebagai matang.

Dari hasil confusion matrix tersebut selanjutnya dapat digunakan untuk mengukur kinerja model seperti akurasi, recall, f1-score, dan precision. Hasilnya seperti gambar berikut

```
Test Accuracy: 0.99556
Classification Report:
----------------------
Class       | Precision  | Recall     | F1 Score
----------------------
busuk       | 1.0000     | 0.9867     | 0.9933
matang      | 0.9868     | 1.0000     | 0.9934
mentah      | 1.0000     | 1.0000     | 1.0000
----------------------
Macro Average:
Macro Avg   | 0.9956     | 0.9956     | 0.9956
```

Gambar 4. 24 Hasil Klasifikasi CNN

Seperti yang terdapat pada gambar 4.24 bisa dilihat bahwasannya untuk nilai Precision didapatkan nilai 100%, Recall 98,67%, dan F1 Score 99,33%. Hasil akurasi yang didapatkan sebesar 99,56%, akurasi yang sangat tinggi.

4.3.4 Pembuatan model SVM

Pada tahap awal seperti sebelumnya yaitu mengubah dataset yang awalnya rgb ke grayscale terlebih dahulu.

Gambar 4. 25 perubahan dari rgb ke grayscale

Proses yang selanjutnya dilakukan yaitu menentukan berapa besaran jumlah dataset yang digunakan untuk data training dan data testing, disini menggunakan 70% data training dan 30% data testing.

```
% Split the data into training (70%) and testing (30%)
[imdsTrain, imdsTest] = splitEachLabel(imds, 0.7, 'randomized');
```

Gambar 4. 26 Code untuk membagi dataset

Selanjutnya yaitu proses ekstraksi fitur pada dataset atau citra seperti nilai warna pada dataset ukuran citra.

```
net = squeezenet;
inputSize = net.Layers(1).InputSize;
```

Gambar 4. 27 Code untuk ekstraksi fitur citra

Pada penelitian ini untuk ekstraksi fitur citra untuk algoritma SVM menggunakan squeezenet. Inputsize pada code tersebut berfungsi untuk mengambil input layer pertama pada citra dan input size pada squeezenet ini hanya bisa menggunakan ukuran 227x227 dan hanya bisa mengambil warna rgb maka akan dilakukan proses berikut.

```
augimdsTrain = augmentedImageDatastore([inputSize(1:2) 1], imdsTrain, 'ColorPreprocessing', 'gray2rgb');
augimdsTest = augmentedImageDatastore([inputSize(1:2) 1], imdsTest, 'ColorPreprocessing', 'gray2rgb');
```

Gambar 4. 28 Code untuk menyesuaikan gambar

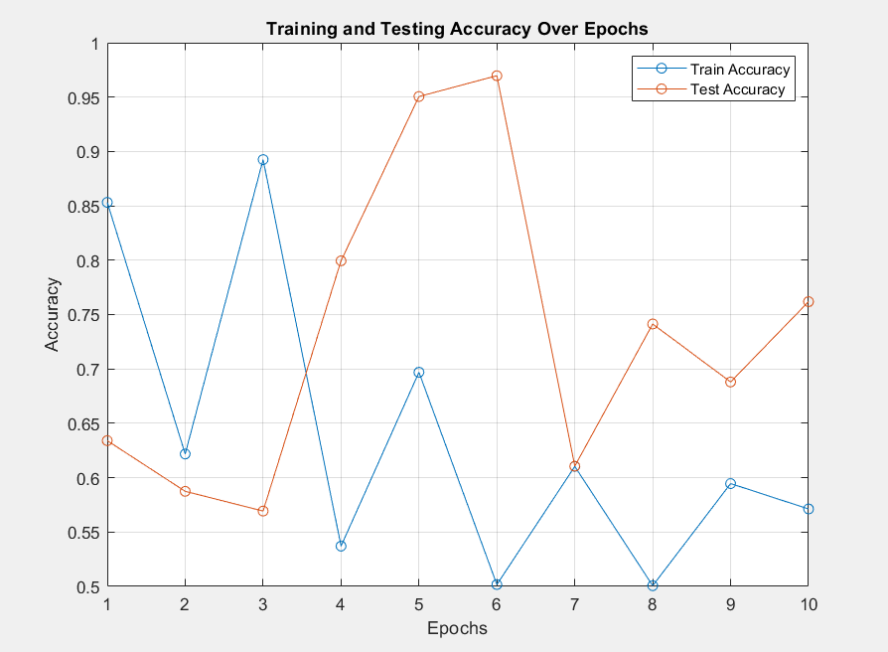
Untuk mengatasi permasalahan diatas maka ditambahkan code pada gambar 4.28 tersebut agar memenuhi syarat pada proses squeezenet sebelumnya dimana data akan diubah ke ukuran 227x227. Karena citra sebelumnya sudah diproses ke grayscale dan pada squezeenet mengharuskan untuk citra agar memiliki warna rgb maka code ‘ColorPreprocessing’ ditambahkan.

Setelah semua proses selesai dilakukan maka selanjutnya dilakukan klasifikasi menggunakan algoritma Support Vector Machine (SVM) dan berikut codenya.

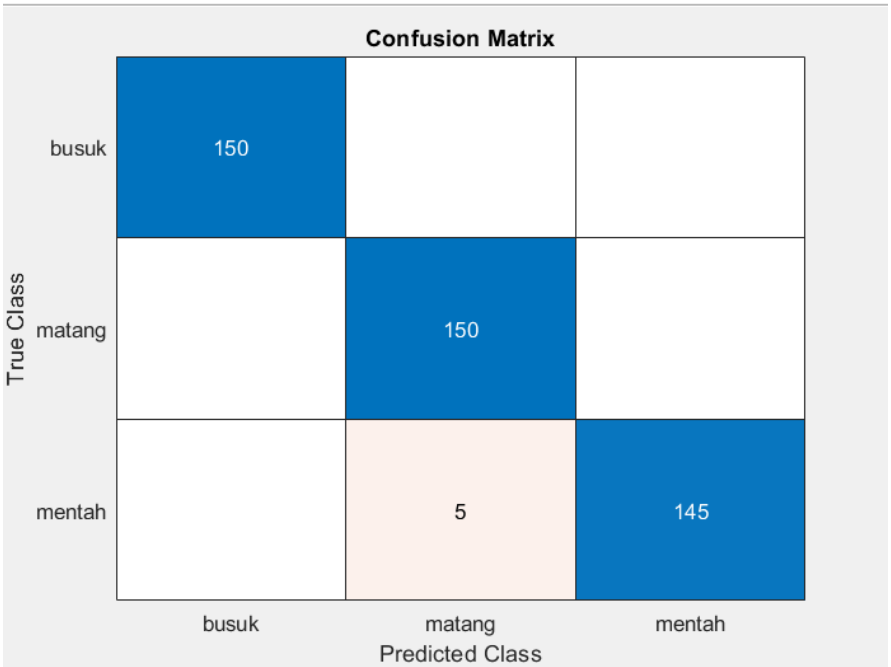
```
classifier = fitcecoc(featuresTrain, labelsTrain);
```

Gambar 4. 29 Code untuk klasifikasi menggunakan SVM

Setelah klasifikasi selesai maka didapat hasil berikut



Gambar 4. 30 Grafik proses training dan testing



Gambar 4. 31 Hasil Confusion Matrix

Hasil yang didapatkan yaitu

Diagonal (warna biru): Menunjukkan jumlah data yang diklasifikasikan dengan benar oleh model.

- Busuk ke busuk: 150 data diklasifikasikan dengan benar sebagai busuk.
- Matang ke matang: 150 data diklasifikasikan dengan benar sebagai matang.
- Mentah ke mentah: 145 data diklasifikasikan dengan benar sebagai mentah.

Off-Diagonal (warna lain): Menunjukkan jumlah data yang diklasifikasikan dengan salah oleh model.

- Mentah ke matang: 5 data sebenarnya busuk, tapi diklasifikasikan sebagai matang.

Dari hasil confusion matrix tersebut selanjutnya dapat digunakan untuk mengukur kinerja model seperti akurasi, recall, f1-score, dan precision. Hasilnya seperti gambar berikut

```
Accuracy: 98.89%
Pixel values saved to pixel_values.csv
Classification Report:

Class       Precision  Recall     F1-Score
busuk       0.97       1.00       0.98
matang      1.00       0.97       0.98
mentah      1.00       1.00       1.00

Mean        0.99       0.99       0.99
```

Gambar 4. 32 Hasil klasifikasi

Hasil yang didapatkan yaitu

- Nilai Presisi yang didapatkan yaitu 99%
- Nilai Recall yang didapatkan yaitu 99%
- Nilai F1-Score yang didapatkan yaitu 99%

- Dan hasil akurasi yang didapatkan yaitu 98.89%

4.4 Evaluation

Setelah dilakukan pengujian didapat sebuah data akurasi sebagai berikut

Tabel 4. 4 Tabel Perbandingan Hasil

	Split Data	Akurasi
CNN	70 : 30	99,56 %
CNN + PCA	70 : 30	76,4 %
SVM	70 : 30	99 %
SVM + PCA	70 : 30	98,89 %
CNN	80 : 20	99,33 %
CNN + PCA	80 : 20	74 %
SVM	80 : 20	99,67 %
SVM + PCA	80 : 20	99 %

Dari data diatas didapat kesimpulan bahwa split data antara data uji dan data test tidak berpengaruh cukup banyak seperti pada model CNN + PCA yang dari data split 70 : 30 mendapatkan akurasi sebesar 76,4 % terjadi penurunan jika dilakukan split data 80 : 20 dengan akurasi 74%, terjadi penurunan 2,4 %. Untuk perbandingan performa untuk model SVM baik ditambahkan PCA maupun tidak hasilnya tetap tinggi diatas 98 %, Untuk CNN kebalikan dari SVM setelah ditambahkan PCA justru mengalami penurunan performa yang tanpa PCA mendapatkan akurasi sebesar 99, 56 % dengan split data 70 : 30, ditambahkan PCA menjadi 76,4 % terjadi penurunan 23,16 % dengan split data yang sama.