BAB IV
PERANCANGAN

4.1 Sistem Terdahulu

Salah satu studi sebelumnya yang digunakan sebagai pembanding dalam penelitian ini adalah jurnal berjudul *"Sistem Deteksi Kualitas Kentang Segar dan Busuk Menggunakan YOLOv8n"* oleh Ocha Alfiano dan Santi Rahayu. Penelitian tersebut bertujuan untuk membangun sistem deteksi kualitas kentang secara real-time menggunakan algoritma YOLOv8n, yang merupakan salah satu varian tercepat dan paling ringan dari keluarga YOLO. Dataset terdiri dari 1.000 gambar kentang segar dan busuk yang kemudian diperluas menjadi 2.304 gambar melalui proses augmentasi menggunakan platform Roboflow. Gambar-gambar tersebut diberi anotasi *bounding box* dan dilatih menggunakan model YOLOv8n pretrained. Hasil pelatihan menunjukkan performa sangat tinggi, dengan precision sebesar 99,9%, *recall* 100%, dan *mAP50-90* sebesar 97,9%. Model ini diimplementasikan dalam aplikasi berbasis Flask untuk mendeteksi kualitas kentang langsung dari kamera secara cepat (Alfiano & Rahayu, 2024).

4.2 Spesifikasi Kebutuhan Sistem Baru

4.2.1 Spesifikasi Kebutuhan Perangkat Keras

Perangkat keras yang dipakai dalam penelitian ini mencakup beberapa komponen, seperti prosesor, *harddisk*, memori, dan *VGA*. Tabel berikut menunjukkan spesifikasi perangkat keras yang digunakan.

Tabel 4. 1 Tabel Spesifikasi kebutuhan Perangkat Keras

No	**Perangkat Keras**	**Kebutuhan Perangkat Keras**
1	Prosesor	Ryzen 5 3500U
2	Media Penyimpanan *(Storage)*	1 TB SSD
3	Memori Utama *(RAM)*	8 GB
4	VGA	Radeon Vega 8

4.2.2 Spesifikasi Kebutuhan Perangkat Lunak

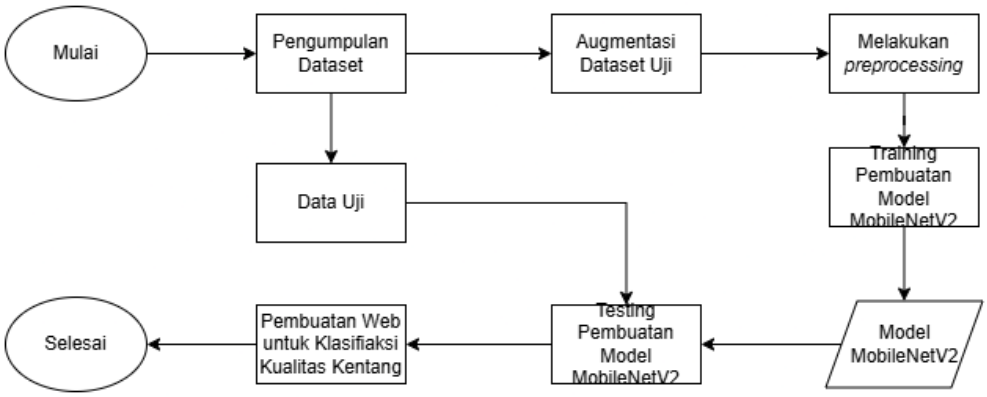
Perangkat lunak yang dimanfaatkan dalam penelitian ini meliputi sistem operasi dan aplikasi editor. Daftar perangkat lunak yang dimanfaatkan selama proses penelitian dapat dilihat pada tabel 4.2.

Tabel 4. 2 Spesifikasi Kebutuhan Perangkat Lunak

No	**Perangkat Keras**	**Kebutuhan Perangkat Keras**
1.	*Operation System*	Windows 11
2.	*IDE*	Visual Studio Code
3.	*Scripting Language*	Python
4.	*Framework*	Flask

4.3 Perancangan Sistem

Pada bagian ini, perancangan aplikasi klasifikasi kualitas kentang berdasarkan dua kelas, yaitu kentang dengan kualitas bagus dan kentang dengan kualitas rusak. Berikut penjelasan mengenai tahapan dalam pengembangan aplikasi klasifikasi kualitas kentang.



Gambar 4. 1 Perancangan Aplikasi untuk Mengklasifikasi Kualitas Kentang

4.3.1. Pengumpulan Dataset

Gambar 4. 2 Dataset Kentang Bagus dan Kentang Rusak

Pada tahap awal pengembangan aplikasi klasifikasi kualitas kentang, pengumpulan dataset dilakukan secara manual. Proses ini dilakukan langsung di lapangan, yaitu dengan mengambil gambar kentang di lokasi pasar tradisional. Gambar-gambar tersebut diambil menggunakan kamera smartphone dan mencakup berbagai kondisi pencahayaan serta sudut pengambilan. Setiap gambar kemudian melalui proses pelabelan berdasarkan klasifikasi kualitas, yaitu kelas "bagus" dan kelas "rusak". Proses pelabelan ini dilakukan berdasarkan panduan dari pakar yang memiliki pengalaman langsung dalam pemilahan kentang, yaitu para pedagang sayur di pasar. Mereka membantu mengidentifikasi ciri-ciri kentang layak konsumsi dan kentang yang sudah rusak atau tidak memenuhi standar kualitas. Jumlah dataset yang terkumpul terdiri dari 150 gambar, dengan rincian 100 gambar untuk kelas "bagus" dan 50 gambar untuk kelas "rusak". Selanjutnya, dataset ini digunakan untuk pelatihan model klasifikasi citra, dengan 10% dari total gambar dialokasikan sebagai data uji guna mengevaluasi performa model.

4.3.2. Membuat Dataset Training

```python
# Konfigurasi
input_dir = 'kentang/dataset/dataset/'  # Path dataset asli
output_dir = 'kentang/dataset/dataset_split2/'  # Output folder untuk split dataset
classes = ['bagus', 'rusak']
train_ratio = 0.7
val_ratio = 0.2
test_ratio = 0.1

Windsurf: Refactor | Explain | Generate Docstring | X
def split_dataset():
    for cls in classes:
        src_folder = os.path.join(input_dir, cls)
        all_files = [f for f in os.listdir(src_folder) if f.lower().endswith(('jpg', 'jpeg', 'png'))]
        random.shuffle(all_files)

        total = len(all_files)
        train_end = int(train_ratio * total)
        val_end = train_end + int(val_ratio * total)

        split_data = {
            'train': all_files[:train_end],
            'val': all_files[train_end:val_end],
            'test': all_files[val_end:]
        }

        print(f"\n=== Kelas: {cls} ===")
        print(f"Total gambar asli: {total}")

        for split, files in split_data.items():
            split_dir = os.path.join(output_dir, split, cls)
            os.makedirs(split_dir, exist_ok=True)

            for filename in files:
                src_path = os.path.join(src_folder, filename)
                shutil.copy(src_path, split_dir)

            print(f"[✓] {split.upper()} - {cls}: {len(files)} gambar disalin")

if __name__ == "__main__":
    split_dataset()
```

Gambar 4. 3 Pembuatan Dataset Training

Pada gambar 4.3, dataset latih dibuat dengan mengorganisir gambar sesuai dengan kelasnya masing-masing. Gambar yang telah diproses akan disalin ke direktori khusus untuk setiap kelas, yaitu "bagus" dan "rusak". Dataset ini dibagi menjadi tiga subset: train, val, dan test. Gambar dari setiap kelas dibagi berdasarkan proporsi yang telah ditentukan untuk training (70%), validation (20%), dan testing (10%). Setiap subset ini disalin ke direktori yang sesuai untuk memudahkan pelatihan dan evaluasi model.

4.3.3. Augmentasi Dataset

```python
import os
import cv2
import numpy as np

# Konfigurasi
input_dir = 'kentang/dataset/dataset_split/'  # Folder split dataset
output_dir = 'kentang/dataset/dataset_split_augmented/'  # Output folder untuk dataset augmented
classes = ['bagus', 'rusak']
target_size = (224, 224)
```

Gambar 4. 4 Cuplikan Kode Program Augmentasi Ganbar 1

Pada gambar 4.4, bagian pertama kode ini mengimpor library yang diperlukan, seperti os, cv2 (OpenCV untuk manipulasi gambar), dan numpy untuk operasi

matematika. Kemudian dilakukan konfigurasi direktori. input_dir menunjuk pada folder yang berisi dataset yang telah dibagi menjadi subset (train, val, test). output_dir adalah lokasi tempat gambar-gambar yang sudah diaugmentasi akan disimpan. Kelas yang ada pada dataset adalah bagus dan rusak, dan setiap gambar akan diubah ukurannya menjadi 224x224 agar sesuai dengan ukuran input model.

```
# Fungsi augmentasi kelas rusak (tanpa blur, zoom, atau shift)
Windsurf: Refactor | Explain | Generate Docstring | X
def augment_rusak(img):
    augmented = [img]
    augmented.append(cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE))  # Rotasi 90° CW
    augmented.append(cv2.rotate(img, cv2.ROTATE_180))  # Rotasi 180°
    augmented.append(cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE))  # Rotasi 90° CCW
    augmented.append(cv2.flip(img, 1))  # Flip horizontal
    augmented.append(cv2.flip(img, 0))  # Flip vertical
```

Gambar 4. 5 Cuplikan Kode Program Augmentasi Ganbar 2

Pada gambar 4.5, menunjukkan Fungsi *augment_rusak()* bertugas untuk menghasilkan variasi gambar dari gambar asli untuk kelas rusak. Augmentasi dilakukan dengan beberapa teknik, yaitu rotasi 90°, rotasi 180°, rotasi 270°, dan flipping (pembalikan) gambar secara horizontal dan vertikal. Fungsi ini mengembalikan sebuah daftar yang berisi gambar asli dan gambar-gambar hasil augmentasi.

```
# Fungsi augmentasi kelas bagus (tanpa blur, zoom, atau shift)
Windsurf: Refactor | Explain | Generate Docstring | X
def augment_bagus(img):
    augmented = [img]
    augmented.append(cv2.flip(img, 1))  # Flip horizontal
    augmented.append(cv2.flip(img, 0))  # Flip vertical
    augmented.append(cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE))  # Rotasi 90°

    return augmented
```

Gambar 4. 6 Cuplikan Kode Program Augmentasi Ganbar 3

Pada gambar 4.6, Fungsi augment_bagus() melakukan augmentasi pada gambar untuk kelas bagus. Teknik augmentasi yang digunakan serupa dengan kelas rusak, yaitu horizontal flipping, vertical flipping, dan rotasi 90 derajat searah jarum jam. Teknik horizontal flipping dilakukan dengan membalik gambar secara simetris terhadap sumbu vertikal, sehingga sisi kiri dan kanan objek dalam gambar dipertukarkan. Sementara itu, vertical flipping membalik gambar terhadap sumbu

horizontal, sehingga bagian atas dan bawah gambar ditukar. Selain itu, rotasi 90° searah jarum jam memutar seluruh gambar sehingga objek berpindah orientasi sebanyak seperempat putaran ke kanan. Kombinasi ketiga teknik ini menghasilkan variasi posisi dan sudut pandang objek kentang tanpa mengubah identitas visual utamanya. Hasil dari fungsi ini adalah kumpulan gambar yang terdiri dari satu gambar asli dan tiga gambar hasil augmentasi, menjadikan total empat gambar per input.

```python
# Simpan hasil augmentasi
Windsurf: Refactor | Explain | Generate Docstring | X
def save_augmented_images(img_path, save_dir, base_name, cls, is_train):
    count = 0
    img = cv2.imread(img_path)
    img = cv2.resize(img, target_size)

    if is_train:
        if cls == 'rusak':
            augmented = augment_rusak(img)
        else:
            augmented = augment_bagus(img)
    else:
        augmented = [img]

    for i, aug in enumerate(augmented):
        aug_name = f"{base_name}_aug{i}.jpg"
        cv2.imwrite(os.path.join(save_dir, aug_name), aug)
        count += 1

    return count
```

Gambar 4. 7 Cuplikan Kode Program Augmentasi Ganbar 4

Pada gambar 4.7, Fungsi save_augmented_images() bertugas untuk menyimpan gambar-gambar yang telah diaugmentasi. Fungsi ini membaca gambar dari path yang diberikan (img_path), meresize gambar ke ukuran yang diinginkan (target_size), kemudian melakukan augmentasi berdasarkan kelas (rusak atau bagus). Gambar hasil augmentasi disimpan ke dalam direktori yang telah ditentukan (save_dir) dengan nama yang unik, berdasarkan nama dasar gambar dan nomor urut augmentasi. Fungsi ini mengembalikan jumlah gambar yang berhasil disalin.

```python
def augment_and_save():
    total_original_images = 0
    total_augmented_images = 0

    for subset in ['train', 'val', 'test']:
        print(f"\n=== Subset: {subset.upper()} ===")

        for cls in classes:
            src_folder = os.path.join(input_dir, subset, cls)
            files = [f for f in os.listdir(src_folder) if f.lower().endswith(('jpg', 'jpeg', 'png'))]

            original_images = len(files)
            augmented_images = 0
            print(f"  Kelas: {cls}")
            print(f"  Total gambar asli: {original_images}")

            # Create output folder for augmented images
            split_dir = os.path.join(output_dir, subset, cls)
            os.makedirs(split_dir, exist_ok=True)

            for idx, filename in enumerate(files):
                src_path = os.path.join(src_folder, filename)
                base_name = f"{cls}_{idx}"
                is_train = (subset == 'train')  # Augment only for train subset
                augmented_images += save_augmented_images(src_path, split_dir, base_name, cls, is_train)

            total_original_images += original_images
            total_augmented_images += augmented_images

            print(f"  Gambar augmentasi: {augmented_images}")

    print(f"\nTotal gambar asli (Sebelum Augmentasi): {total_original_images}")
    print(f"Total gambar augmentasi (Sesudah Augmentasi): {total_augmented_images}")
```

Gambar 4. 8 Cuplikan Kode Program Augmentasi Ganbar 5

Pada gambar 4.8, fungsi augment_and_save() adalah fungsi utama yang mengatur seluruh proses augmentasi dan penyimpanan gambar. Fungsi ini melakukan iterasi melalui setiap subset (train, val, test) dan masing-masing kelas (bagus dan rusak). Untuk setiap gambar yang terdapat pada subset, gambar akan diproses melalui fungsi augmentasi sesuai dengan kelasnya, lalu disalin ke folder tujuan (output_dir). Proses ini memastikan bahwa hanya gambar pada subset train yang mengalami augmentasi untuk memperkaya data pelatihan, sedangkan subset val dan test tetap mempertahankan gambar aslinya. Selama proses berjalan, sistem juga mencatat dan menampilkan jumlah gambar asli serta hasil augmentasi yang dihasilkan untuk setiap kelas. Total gambar setelah proses augmentasi adalah sebanyak 280 gambar untuk kelas "bagus" dan 210 gambar untuk kelas "rusak".

4.3.4. Penerapan Preprocessing pada Dataset Gambar

Pada tahap ini, kode bertugas untuk memproses dataset gambar yang sudah diaugmentasi, yang kemudian akan digunakan untuk pelatihan model klasifikasi. Proses *preprocessing* melibatkan beberapa langkah, seperti pemotongan gambar (cropping), normalisasi, dan *resizing* agar gambar sesuai dengan format yang dibutuhkan oleh model. Berikut adalah penjelasan lebih rinci tentang setiap cuplikan kode.

```
def preprocessing(img_path):
    pic_ori = cv2.imread(img_path)
    row, col = pic_ori.shape[:2]

    row_margin = round(row_crop * row)
    col_margin = round(col_crop * col)
    pic_crop = pic_ori[row_margin:row-row_margin, col_margin:col-col_margin, :]

    pic_asal = pic_crop.astype(np.float16)
    row_asal, col_asal, _ = pic_asal.shape
    pic_res = np.zeros((m, n, ch), dtype=np.float16)
    delta_row = round(row_asal / m)
    delta_col = round(col_asal / n)

    for i_res, i_asal in enumerate(range(0, row_asal - delta_row, delta_row)):
        for j_res, j_asal in enumerate(range(0, col_asal - delta_col, delta_col)):
            patch = pic_asal[i_asal:i_asal+delta_row, j_asal:j_asal+delta_col]
            pic_res[i_res, j_res] = np.mean(patch, axis=(0, 1))

    average = np.mean(pic_res, axis=2)
    pic_gs = np.stack((average,) * 3, axis=-1)
    pic_gs_resized = cv2.resize(pic_gs.astype(np.uint8), target_size)
    return pic_gs_resized
```

Gambar 4. 9 Cuplikan Kode Program *preprocessing* 1

Pada gambar 4.9, fungsi *preprocessing* ini melakukan beberapa tahap pemrosesan pada gambar, dimulai dengan membaca gambar menggunakan cv2.imread(). Selanjutnya, gambar akan dipotong sesuai dengan margin yang telah ditentukan berdasarkan rasio row_crop dan col_crop. Setelah pemotongan, gambar diubah menjadi tipe np.float16 agar proses pengolahan lebih efisien.

Selanjutnya, gambar diproses dalam bentuk patch kecil dengan ukuran m x n (20x20), dan setiap patch dihitung rata-ratanya. Hasilnya adalah gambar yang telah diproses menjadi format yang lebih kecil dan konsisten. Gambar ini kemudian dikonversi kembali ke tipe np.uint8 dan di-resize ke ukuran target (224x224) yang dibutuhkan oleh model.

```python
def process_subset(subset):
    input_dir = os.path.join(input_base_dir, subset)
    output_dir = os.path.join(output_base_dir, subset)

    for class_name in classes:
        os.makedirs(os.path.join(output_dir, class_name), exist_ok=True)

    start_time = time.time()
    total_processed = 0

    for class_name in classes:
        class_folder = os.path.join(input_dir, class_name)
        files = [f for f in os.listdir(class_folder) if f.lower().endswith(('jpg', 'jpeg', 'png'))]

        for idx, file_name in enumerate(files):
            try:
                img_path = os.path.join(class_folder, file_name)
                processed_img = preprocessing(img_path)
                save_path = os.path.join(output_dir, class_name, file_name)
                cv2.imwrite(save_path, cv2.cvtColor(processed_img, cv2.COLOR_RGB2BGR))
                total_processed += 1
            except Exception as e:
                print(f"[X] Gagal memproses {file_name}: {e}")
    durasi = time.time() - start_time
    print(f"[✓] Preprocessing pada '{subset}' selesai: {total_processed} gambar → {durasi:.2f} detik")
```

Gambar 4. 10 Cuplikan Kode Program *preprocessing* 2

Pada gambar 4.10, fungsi *process_subset* ini memproses setiap subset dataset (train, validation, dan test) secara terpisah. Fungsi ini pertama-tama memastikan bahwa folder untuk setiap subset dan kelas sudah ada. Kemudian, untuk setiap gambar di dalam folder, fungsi akan memanggil *preprocessing* untuk melakukan pemrosesan gambar, dan hasilnya disimpan ke folder output yang sesuai. Waktu proses dihitung dan ditampilkan untuk memantau berapa lama waktu yang dibutuhkan untuk memproses setiap subset.

4.3.5. Pembuatan Model *MobileNetV2*

Dalam tahap ini, peneliti mengembangkan model untuk aplikasi mengklasifikasi kualitas kentang menggunakan arsitektur *MobileNetV2*. *MobileNetV2* adalah salah satu model jaringan saraf konvolusional (CNN) yang efisien dalam menghasilkan akurasi tinggi dengan penggunaan sumber daya komputasi yang rendah. Model ini dirancang untuk aplikasi yang membutuhkan komputasi efisien tanpa mengorbankan akurasi.

```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix
```

Gambar 4. 11 Cuplikan Kode Program Pelatihan Model 1

Pada gambar 4.11, bagian pertama berbagai pustaka yang diperlukan diimpor. os digunakan untuk operasi terkait file dan direktori, *matplotlib* dan *seaborn* untuk visualisasi grafik, *numpy* untuk operasi matematis, dan *pandas* untuk manajemen data dalam format tabel. Kelas-kelas dari *tensorflow.keras* digunakan untuk membangun dan melatih model, serta untuk evaluasi. *ImageDataGenerator* digunakan untuk memuat dan melakukan augmentasi data gambar. MobileNetV2 adalah model pra-latih yang digunakan sebagai basis untuk model klasifikasi, sementara *Adam* digunakan sebagai *optimizer*.

```
# Data generator
train_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
    os.path.join(data_dir, 'train'),
    target_size=target_size,
    batch_size=batch_size,
    class_mode='sparse'
)
```

Gambar 4. 12 Cuplikan Kode Program Pelatihan Model 2

Pada gambar 4.12, ImageDataGenerator digunakan untuk preprocessing dan augmentasi data. Data pelatihan, validasi, dan uji diproses dengan mengubah gambar menjadi format yang sesuai untuk model *MobileNetV2* (menggunakan rescaling untuk normalisasi nilai piksel ke rentang [0,1]).

```
# Model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
for layer in base_model.layers:
    layer.trainable = False

x = GlobalAveragePooling2D()(base_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
output_layer = Dense(len(classes), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output_layer)
```

Gambar 4. 13 Cuplikan Kode Program Pelatihan Model 3

Pada gambar 4,13, model dasar yang digunakan adalah *MobileNetV2*, yang dipra-latih dengan weights='imagenet' untuk memanfaatkan pengetahuan yang diperoleh dari dataset ImageNet. Di sini, bagian top layer dari *MobileNetV2* dihapus dengan include_top=False, dan model dilanjutkan dengan menambahkan GlobalAveragePooling2D, Dense, dan Dropout untuk menghasilkan output kelas. GlobalAveragePooling2D mengurangi dimensi spasial hasil dari konvolusi dan mengubahnya menjadi satu vektor. Dense digunakan untuk membuat lapisan *fully connected* dengan total neuronnya disesuaikan dengan banyaknya kelas (2 kelas: bagus dan rusak). Dropout(0.4) ditambahkan untuk mengurangi overfitting dengan menghilangkan 40% neuron selama pelatihan. Algoritma optimisasi Adam dipilih dengan kecepatan pembelajaran (learning rate) rendah (1e-5), guna memastikan proses pelatihan berjalan stabil.

```
# Training
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=100,
    verbose=1,
    callbacks=callbacks
)
```

Gambar 4. 14 Cuplikan Kode Program Pelatihan Model 4

Pelatihan model dilakukan pada gambar 4.14, menggunakan model.fit(). Data pelatihan dimuat melalui train_gen, sementara data validasi dimuat melalui val_gen. Model dilatih selama 100 epoch, dan callback yang sudah didefinisikan sebelumnya digunakan untuk memantau proses pelatihan dan menghentikannya jika diperlukan.

```
# Save final model
final_model_path = os.path.join(model_dir, 'mobilenetv2_2kelas_finetune.keras')
model.save(final_model_path)
```

Gambar 4. 15 Cuplikan Kode Program Pelatihan Model 5

Pada gambar 4.15 setelah pelatihan selesai, model yang telah dilatih disimpan dalam file mobilenetv2_2kelas_finetune.keras menggunakan model.save() untuk digunakan di kemudian hari atau untuk evaluasi lebih lanjut.

```
# Evaluasi
loss, acc = model.evaluate(test_gen)
print(f"[INFO] Akurasi test: {acc:.4f}")
y_pred = model.predict(test_gen)
y_pred_labels = np.argmax(y_pred, axis=1)
```

Gambar 4. 16 Cuplikan Kode Program Pelatihan Model 6

Pada gambar 4.16 setelah pelatihan selesai, model dievaluasi menggunakan model.evaluate() dengan data pengujian test_gen. Hasil evaluasi berupa *loss* dan *accuracy* dicetak. Prediksi model pada data uji dihitung dengan model.predict(), dan np.argmax() digunakan untuk mengambil label kelas prediksi tertinggi.

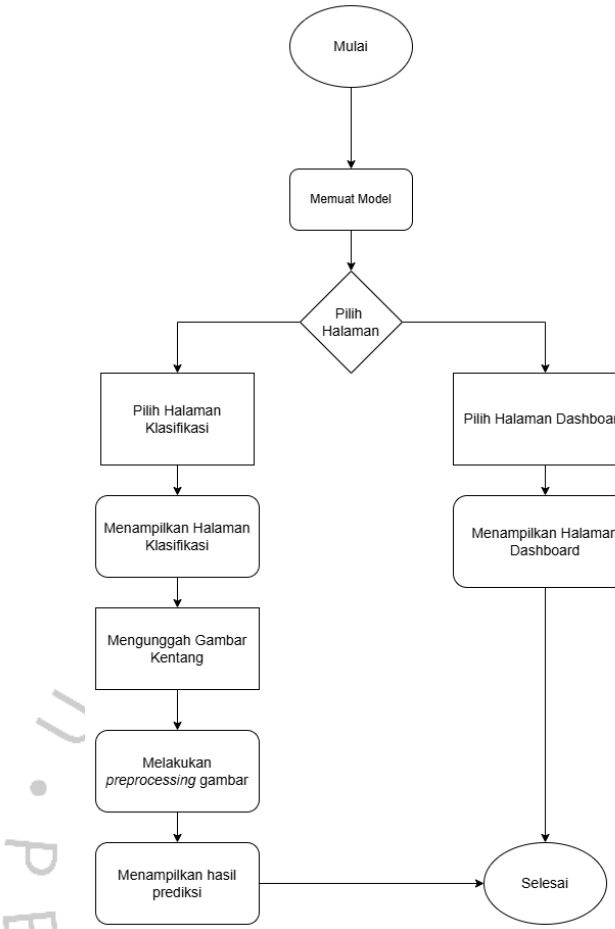
```
# Confusion Matrix & Classification Report
cm = confusion_matrix(test_gen.classes, y_pred_labels)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=classes, yticklabels=classes, cmap='Blues')
plt.xlabel("Prediksi")
plt.ylabel("Aktual")
plt.title("Confusion Matrix")
plt.tight_layout()
os.makedirs(grafik_dir, exist_ok=True)
plt.savefig(os.path.join(grafik_dir, 'confusion_matrix_finetune.png'))

report = classification_report(test_gen.classes, y_pred_labels, target_names=classes)
print(report)
```

Gambar 4. 17 Cuplikan Kode Program Pelatihan Model 7

Pada gambar 4.17 setelah evaluasi, *confusion matrix* dihitung dan divisualisasikan dengan seaborn.heatmap(), yang menunjukkan performa model dalam mengklasifikasikan data uji.

4.3.6. Alur Komputasi

Pada tahap alur komputasi ini, dijelaskan proses melalui diagram alir. Berikut adalah diagram alir aplikasi untuk klasifikasi kualitas kentang.



Gambar 4. 18 Alur Komputasi Aplikasi untuk Mengklasifikasi Kualitas Kentang

Diagram alir pada gambar 4.18, menggambarkan proses aliran komputasi aplikasi untuk klasifikasi kualitas kentang. Berikut adalah penjelasan alur yang digambarkan dalam diagram tersebut:

1. Mulai: Proses dimulai dengan memuat model yang digunakan untuk klasifikasi kualitas kentang.
2. Memilih Halaman: Setelah model dimuat, pengguna memilih halaman yang akan ditampilkan, yaitu Halaman Klasifikasi atau Halaman Dashboard.
    a) Jika memilih Halaman Klasifikasi:
        - Sistem menampilkan halaman klasifikasi.
        - Pengguna mengunggah gambar kentang yang akan diprediksi.
        - Gambar yang diunggah kemudian diproses melalui tahap preprocessing gambar untuk memastikan gambar siap digunakan dalam model prediksi.

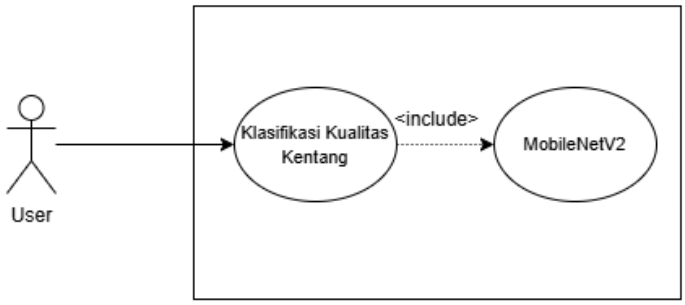
- Setelah gambar diproses, sistem menampilkan hasil prediksi, yaitu kualitas kentang apakah bagus atau rusak.

b) Jika memilih Halaman Dashboard:

- Sistem menampilkan Halaman Dashboard yang berisi informasi umum atau tampilan utama aplikasi.

3. Selesai: Setelah pengguna selesai, proses akan berakhir.

4.3.7. Use Case Diagram



Gambar 4. 19 *Use Case* Diagram Aplikasi Untuk Mengklasifikasi Kualitas Kentang

Pada Gambar 4.19, setelah membuka aplikasi untuk menentukan kualitas kentang, pengguna dapat langsung melihat halaman dashboard. Jika pengguna memilih menu klasifikasi, halaman akan muncul untuk menngunggah gambar kentang yang diprediksi. Setelah itu, pengguna dapat menekan tombol klasifikasi untuk melihat hasil prediksi kualitas kentang.

4.3.8. Skenario Use Case

Pada tabel 4.3 menampilkan beberapa skenario *use case* yang diterapkan dalam aplikasi untuk mengidentifikasi kualitas kentang.

Tabel 4. 3 Skenario *Use Case* Aplikasi untuk Mengklasifikasi Kualitas Kentang

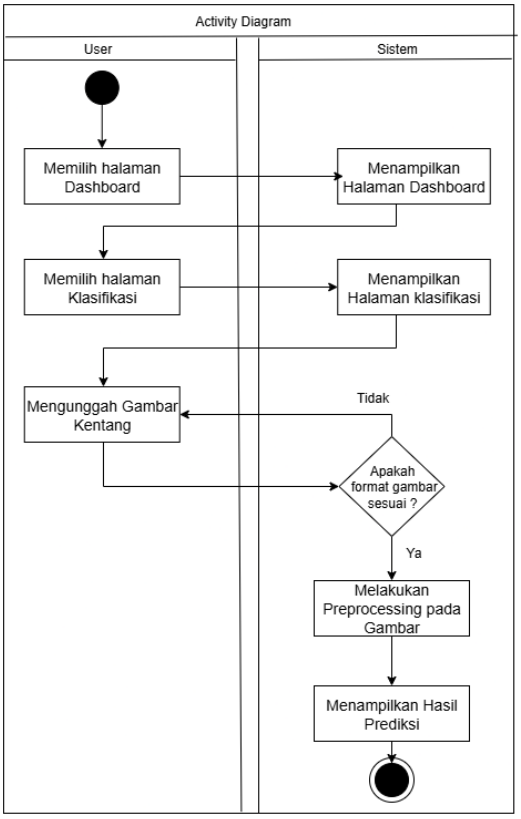
ID	UC1: Identifikasi Kualitas Kentang
Nama Use Case	Identifikasi Kualitas Kentang
Aktor	User
Deskripsi	User mengunggah gambar kentang untuk mengidentifikasi kualitasnya (bagus atau rusak) menggunakan model prediksi MobileNetV2.
Pre-Condition	1. Aplikasi sudah memuat model prediksi MobileNetV2. 2. User membuka aplikasi.
Langkah-Langkah	1. User membuka aplikasi. 2. User memilih halaman klasifikasi. 3. Sistem menampilkan halaman klasifikasi. 4. User memilih untuk mengunggah gambar. 5. Sistem memvalidasi format gambar. 6. Sistem melakukan preprocessing pada gambar. 7. Sistem menjalankan model prediksi untuk mengidentifikasi kualitas kentang. 8. Aplikasi menampilkan hasil prediksi kepada user.
Trigger	User mengunggah gambar kentang.
Post-Condition	Gambar berhasil diproses dan hasil prediksi kualitas kentang ditampilkan kepada user.

Pada tabel 4.3 menggambarkan proses interaksi pengguna dalam melakukan klasifikasi gambar kentang melalui aplikasi. Pengguna memulai dengan memilih halaman klasifikasi, kemudian mengunggah gambar kentang yang akan dianalisis. Sistem akan memvalidasi format gambar, dan apabila sesuai, proses dilanjutkan dengan *preprocessing* dan klasifikasi menggunakan model *MobileNetV2*. Hasil klasifikasi berupa label “bagus” atau “rusak” beserta nilai confidence ditampilkan kembali kepada pengguna. Jika format gambar tidak sesuai, sistem akan menampilkan pesan kesalahan. Skenario use case ini berfokus pada alur utama dan alternatif dalam proses klasifikasi citra kentang secara otomatis.

4.3.9 Activity Diagram

Activity diagram merupakan representasi visual yang memperlihatkan tahapan-tahapan proses dalam aplikasi yang telah dirancang oleh peneliti. Pada gambar 4.20,

disajikan *activity diagram* untuk aplikasi yang digunakan dalam penentuan kualitas kentang.



Gambar 4. 20 *Activity Diagram* Aplikasi untuk Mengklasifikasi Kualitas Kentang

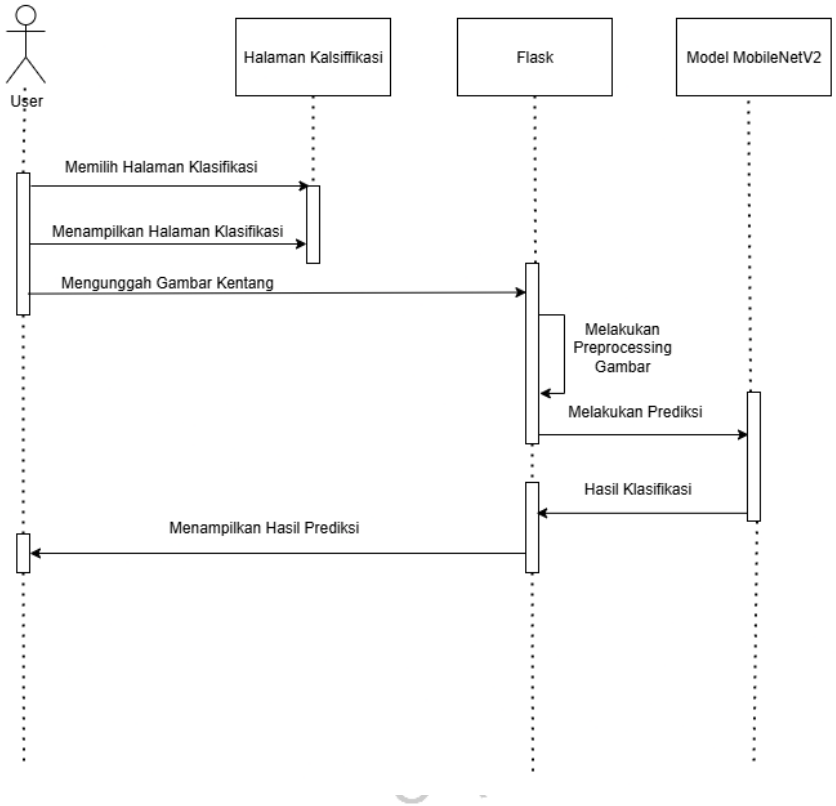
Gambar 4.20 menunjukkan *activity diagram* proses klasifikasi gambar kentang antara pengguna (user) dan sistem. Aktivitas dimulai ketika pengguna memilih halaman dashboard pada antarmuka aplikasi. Sistem merespon dengan menampilkan halaman dashboard. Selanjutnya, pengguna memilih halaman klasifikasi untuk melakukan proses klasifikasi kualitas kentang. Sistem akan menampilkan halaman klasifikasi sebagai respon dari aksi tersebut.

Setelah halaman klasifikasi terbuka, pengguna mengunggah gambar kentang yang ingin diklasifikasikan. Sistem kemudian memverifikasi apakah format gambar yang diunggah sesuai dengan format yang diizinkan (PNG, JPG, atau JPEG). Jika format tidak sesuai, proses tidak dilanjutkan dan pengguna akan menerima pesan kesalahan. Namun jika format sesuai, sistem akan melakukan proses preprocessing pada gambar tersebut. Gambar yang telah diproses selanjutnya digunakan oleh model klasifikasi untuk melakukan prediksi kualitas kentang. Hasil prediksi

kemudian ditampilkan kepada pengguna sebagai output akhir dari proses. Aktivitas berakhir setelah hasil klasifikasi ditampilkan.

4.3.10 Sequence Diagram

Sequence diagram memvisualisasikan urutan interaksi antar komponen dalam sistem, yang terjadi secara berurutan selama sistem dijalankan. Diagram ini berfungsi untuk memahami urutan proses saat aplikasi digunakan hingga menghasilkan output yang diharapkan.



Gambar 4. 21 *Sequence Diagram* Aplikasi untuk Mengklasifikasi Kualitas Kentang

Gambar 4.21 menunjukkan *sequence diagram* untuk proses klasifikasi gambar kentang. Proses diawali oleh aktor user yang memilih halaman klasifikasi melalui antarmuka aplikasi. Permintaan ini dikirimkan ke komponen halaman klasifikasi, yang kemudian merespons dengan menampilkan tampilan klasifikasi kepada user. Setelah halaman tampil, user melakukan aksi dengan mengunggah gambar kentang ke sistem.

Gambar yang diunggah dikirimkan ke komponen *Flask* sebagai *controller backend*. Pada tahap ini, Flask akan menjalankan proses preprocessing gambar agar gambar sesuai dengan format input model. Setelah proses *preprocessing* selesai, data gambar dikirimkan ke model *MobileNetV2* untuk dilakukan proses klasifikasi. Model akan memproses input dan memberikan hasil klasifikasi berupa label kualitas kentang, yaitu “bagus” atau “rusak”, disertai nilai confidence.

Hasil klasifikasi tersebut kemudian dikembalikan dari model ke Flask, dan diteruskan kembali ke halaman antarmuka. Terakhir, halaman klasifikasi akan menampilkan hasil prediksi tersebut kepada user. Dengan demikian, diagram ini menggambarkan secara lengkap interaksi dan alur data dari awal hingga akhir proses klasifikasi gambar kentang.

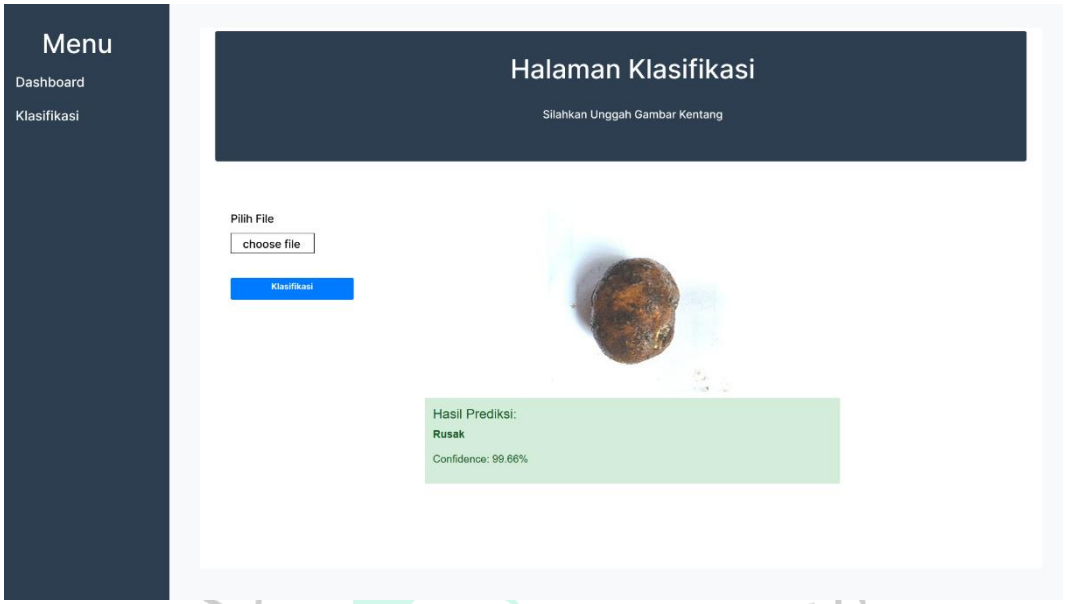
4.3.11 Perancangan Antar Muka

Rancangan tampilan aplikasi ini dibuat menggunakan *Figma* oleh peneliti, dengan fokus pada antarmuka pengguna untuk proses klasifikasi kualitas kentang.. Berikut ini merupakan beberapa rancangan tampilan aplikasi.

Gambar 4. 22 Rancang Antar Muka pada Halaman Dashboard

Berdasarkan gambar 4.22, rancangan tampilan antarmuka aplikasi untuk mengklasifikasi kualitas kentang menampilkan halaman utama dengan menu navigasi di sisi kiri yang berisi pilihan Dashboard dan Klasifikasi. Pada bagian

utama, terdapat pesan selamat datang yang menjelaskan fungsi aplikasi, yakni "Sistem Klasifikasi Kualitas Kentang".



Gambar 4. 23 Rancangan Antar Muka pada Halaman Klasifikasi

Pada gambar 4.23, terlihat rancangan tampilan antarmuka aplikasi yang menampilkan halaman klasifikasi di mana pengguna dapat mengunggah gambar kentang yang akan diprediksi kualitasnya. Pada bagian atas, terdapat pesan untuk mengunggah gambar kentang yang ingin diprediksi. Di bagian utama, terdapat tombol "*Choose file*" untuk memilih gambar kentang yang akan diproses, diikuti dengan tombol "Klasifikasikan" untuk memulai prediksi. Setelah gambar diunggah, hasil prediksi kualitas kentang akan ditampilkan di bawah gambar, menunjukkan apakah kentang tersebut bagus atau rusak, beserta tingkat *confidence* prediksi yang diperoleh. Desain ini memberikan pengalaman pengguna yang sederhana dan intuitif, dengan informasi yang jelas dan mudah diakses.

4.3.12. Perancangan Pengujian

Tahap pengujian sangat penting untuk memastikan aplikasi yang dibangun sesuai dengan desain yang telah direncanakan. Pengujian *Black Box* dan *White Box* digunakan oleh peneliti pada tahap ini.

4.3.12.1. Perancangan Pengujian Black Box

Pengujian black box dirancang untuk menguji fungsionalitas aplikasi berdasarkan spesifikasi yangsudah ditetapkan. Berdasarkan Tabel 4.4, pengujian ini dilakukan untuk memastikan bahwa fitur-fitur utama, seperti input gambar melalui unggahan, proses prediksi kualitas kentang, dan penampilan hasil prediksi berfungsi dengan baik. Fokus pengujian ini meliputi validasi input, akurasi prediksi, dan respons sistem terhadap berbagai skenario penggunaan.

Tabel 4. 4 Rancangan Pengujian *Black Box*

No	Skenario Pengujian	Input	Proses yang Diuji	Output yang Diharapkan
1.	Membuka halaman utama aplikasi	Akses ke URL aplikasi	Menampilkan halaman dashboard	Halaman dashboard berhasil ditampilkan di browser
2.	Mengakses halaman klasifikasi	Klik menu "Klasifikasi"	Menampilkan halaman klasifikasi	Halaman klasifikasi tampil dengan form upload
3.	Mengunggah gambar	File kentang.jpg	Upload file	Gambar berhasil diunggah dan ditampilkan sebagai pratinjau
4.	Melakukan klasifikasi	Klik tombol "Klasifikasikan"	Preprocessing dan prediksi menggunakan model	Label klasifikasi (“bagus” atau “rusak”) ditampilkan dengan skor
5.	Menampilkan hasil klasifikasi		Menampilkan hasil prediksi	Label dan confidence score tampil di halaman hasil

4.3.12.2. Perancangan Pengujian White Box

Pengujian *white box* bertujuan untuk memeriksa logika internal, jalur kode, dan struktur algoritma aplikasi. Dalam aplikasi untuk menentukan kualitas kentang, pengujian ini dilakukan untuk memastikan bahwa semua fungsi, seperti preprocessing gambar, pemrosesan model *MobileNetV2*, dan pengecekan hasil

prediksi, berjalan sesuai dengan desain yang diharapkan. Pengujian ini juga mencakup evaluasi jalur kontrol, penanganan kesalahan, serta optimalisasi algoritma agar sistem tetap efisien dalam memproses data dan memberikan hasil yang akurat.

Tabel 4. 5 Rancangan Pengujian *White Box*

No	Nama Fungsi	Kode Program yang Diuji	Proses yang Diuji
1.	Inisialisasi folder dan model.	os.makedirs(UPLOAD_FOLDER, exist_ok=True) model = load_model(MODEL_PATH)	Menguji apakah sistem berhasil membuat folder upload dan memuat model tanpa error.
2.	Validasi ekstensi file.	def allowed_file(filename): return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS	Mengecek apakah ekstensi file sesuai dengan format yang diperbolehkan (.jpg, .jpeg, .png).
3.	Penyimpanan gambar.	file.save(file_path)	Menguji apakah gambar berhasil disimpan dengan nama unik di folder upload.

4.	Preprocessing gambar.	def preprocess_image(*img_path*):
		img = cv2.imread(img_path)
		img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
		img = cv2.resize(img, (224, 224)) # Ukuran input model
		img = img / 255.0 # Normalisasi gambar
		return np.expand_dims(img, *axis*=0)
			Memastikan gambar dibaca, diubah format warnanya, diubah ukuran, dan dinormalisasi dengan benar.

5.	Prediksi menggunakan model.	pred = model.predict(img) prediction = CLASS_NAMES[np.argmax(pred)]	Menguji apakah model dapat menghasilkan prediksi yang sesuai berdasarkan input gambar.
6.	Penanganan error dalam proses klasifikasi.	try: img = preprocess_image(file_path) except Exception as e:	Memastikan sistem dapat menangani error ketika terjadi kesalahan dalam proses klasifikasi.
7.	Penolakan file dengan format tidak didukung	if file and allowed_file(file.filename): ... else: error_message = "Format file tidak didukung..."	Menguji apakah sistem memunculkan pesan error jika user mengunggah file dengan ekstensi yang tidak valid.

Tabel 4. 6 Perancangan Pengujian *White Box* Lanjutan

8.	Perhitungan confidence (%)	confidence = round(float(np.max(pred)) * 100, 2)	Menguji apakah nilai confidence berhasil dikalkulasi dan ditampilkan dalam bentuk persentase.