APLIKASI PENGKLASIFIKASI KEMATANGAN PEPAYA MENGGUNAKAN METODE CNN BERBASIS ANDROID

Disusun Oleh:

N a m a : Muhammad Sayyidin Hawibowo

NIM : 20523101

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2024

HALAMAN PENGESAHAN DOSEN PEMBIMBING

APLIKASI PENGKLASIFIKASI KEMATANGAN PEPAYA MENGGUNAKAN METODE CNN BERBASIS ANDROID

TUGAS AKHIR

Disusun Oleh:

N a m a : Muhammad Sayyidin Hawibowo
NIM : 20523101

Yogyakarta, 1 Mei 2024

Pembimbing,

(Izzati Muhimmah, S.T., M.Sc., Ph.D.)

HALAMAN PENGESAHAN DOSEN PENGUJI

APLIKASI PENGKLASIFIKASI KEMATANGAN PEPAYA MENGGUNAKAN METODE CNN BERBASIS ANDROID

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 29 Mei 2024

Tim Penguji

Izzati Muhimmah, S.T., M.Sc., Ph.D.

Anggota 1

Rahadian Kurniawan, S.Kom., M.Kom.

Anggota 2

Sheila Nurul Huda, S.Kom., M.Cs.

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Muhammad Sayyidin Hawibowo

NIM : 20523101

Tugas akhir dengan judul:

APLIKASI PENGKLASIFIKASI KEMATANGAN PEPAYA MENGGUNAKAN METODE CNN BERBASIS ANDROID

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 1 Mei 2024

(Muhammad Sayyidin Hawibowo)

HALAMAN PERSEMBAHAN

Dipersembahkan kepada mereka yang selalu memberikan dukungan, cinta, dan motivasi dalam setiap langkah perjalanan hidupku. Terima kasih kepada keluarga tercinta, sahabat-sahabat sejati, dan semua yang telah memberikan inspirasi.

HALAMAN MOTO

"Karena sesungguhnya sesudah kesulitan itu ada kemudahan. Sesungguhnya sesudah kesulitan itu ada kemudahan.”

(Al-Insyirah Ayat 5-6)

"Kesungguhan adalah kunci kesuksesan. Konsistensi adalah kuncinya. Kegigihan adalah jalan menuju hasil."

(Harvey Mackay)

KATA PENGANTAR

Segala puji bagi Allah SWT yang tak pernah henti memberikan kenikmatan dan rahmat kepada seluruh hamba-Nya. Shalawat dan salam semoga senantiasa tercurahkan kepada junjungan kita, Nabi Muhammad SAW.

Dengan limpahan rahmat dan petunjuk Allah SWT, tugas akhir dengan judul "APLIKASI PENGKLASIFIKASI KEMATANGAN PEPAYA MENGGUNAKAN METODE CNN BERBASIS ANDROID" telah berhasil diselesaikan. Tugas akhir ini merupakan bagian dari syarat penyelesaian studi Strata-1 pada program studi Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.

Dalam perjalanan menyelesaikan tugas akhir ini, saya sangat berterima kasih atas dukungan, bantuan, dan arahan yang diberikan oleh berbagai pihak. Dengan tulus, saya ingin mengucapkan terima kasih kepada:

1. Allah SWT, yang telah memberikan petunjuk dan kemudahan sehingga penulis dapat menyelesaikan laporan ini dengan baik.
2. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. selaku Ketua Program Studi Informatika Universitas Islam Indonesia, yang telah memberikan wawasan yang luas tentang dunia informatika.
3. Ibu Izzati Muhimmah, S.T., M.Sc., Ph.D. sebagai dosen pembimbing yang selalu memberikan arahan, bimbingan, dan dukungan dalam menjalankan penelitian ini.
4. Kedua orang tua saya yang telah memberikan kasih sayang sejak lahir hingga saat ini.
5. Teman-teman yang telah menyempatkan waktu untuk menemani saya saat mengerjakan tugas akhir ini.
6. Tidak lupa juga kepada semua pihak yang turut mendukung, meskipun tidak bisa disebutkan satu per satu.

Saya sadar bahwa tugas akhir ini masih memiliki kekurangan. Oleh karena itu, kritik, saran, dan masukan dari pembaca sangat saya harapkan agar karya ilmiah ini dapat menjadi lebih baik di masa mendatang.

Terakhir, semoga aplikasi yang telah dikembangkan dapat memberikan manfaat yang besar dalam meningkatkan efisiensi di industri pertanian, khususnya dalam hal pengelolaan panen pepaya. Harapan saya, penelitian ini dapat memberikan kontribusi positif bagi perkembangan ilmu pengetahuan dan teknologi di masa depan.

Yogyakarta, 1 Mei 2024

(Muhammad Sayyidin Hawibowo)

SARI

Pepaya adalah buah tropis yang memiliki nilai ekonomi signifikan dan menyediakan kandungan gizi penting untuk kesehatan manusia. Tingkat kematangan pepaya memiliki dampak besar terhadap rasanya, teksturnya, dan nilai jualnya. Oleh karena itu, pengembangan sistem otomatis untuk mengidentifikasi tingkat kematangan pepaya sangat penting dalam industri pertanian. Sebuah penelitian telah mengembangkan aplikasi berbasis Android yang menggunakan metode *Convolutional Neural Network* (CNN) untuk secara otomatis mengenali tingkat kematangan pepaya. Dataset pepaya yang terkumpul digunakan untuk melatih model CNN agar dapat mengklasifikasikan pepaya menjadi tiga tingkat kematangan: belum matang, setengah matang, dan matang. Selain itu, aplikasi ini juga memberikan perkiraan waktu petik pepaya berdasarkan analisis tingkat kematangan. Dengan implementasi pada platform Android, aplikasi ini memungkinkan petani atau pemilik kebun pepaya untuk dengan mudah memantau tingkat kematangan buah mereka. Hasil eksperimen menunjukkan bahwa model CNN yang diusulkan memiliki tingkat akurasi sebesar 96,97% dalam mengklasifikasikan tingkat kematangan pepaya. Diharapkan aplikasi ini dapat membantu petani dalam meningkatkan efisiensi proses petik dan pengelolaan pepaya.

Kata kunci: pepaya, klasifikasi, android, *convolutional neural network*.

GLOSARIUM

AlarmManager	Komponen Android yang digunakan untuk menjadwalkan tugas-tugas tertentu, seperti menampilkan notifikasi pada waktu tertentu.
Android	Sistem operasi untuk perangkat *mobile* yang dikembangkan oleh *Google*.
CNN	Algoritma *deep learning* yang digunakan untuk pengenalan gambar dengan memanfaatkan lapisan konvolusi.
Deep Learning	Sub-bidang dari *machine learning* yang menggunakan model *neural networks* dengan banyak lapisan untuk mempelajari representasi data yang kompleks.
Hortikultura	Cabang pertanian yang berfokus pada budidaya tanaman-tanaman hias, tanaman obat, sayuran, dan buah-buahan.
Kelas Klasifikasi	Kategori atau level yang digunakan untuk mengelompokkan objek atau data berdasarkan karakteristik tertentu.
Preprocessing	Serangkaian langkah yang dilakukan untuk mempersiapkan dan memproses data sebelum digunakan dalam analisis atau pembuatan model.
Room Database	Sistem database lokal yang disediakan oleh Android untuk menyimpan dan mengakses data secara efisien.
TensorFlow Lite	Format ringan dari *TensorFlow* yang dioptimalkan untuk perangkat *mobile*.
ViewModel	Komponen Android *Architecture Components* yang bertanggung jawab untuk menyediakan data kepada antarmuka pengguna.
Wireframe Low-Fidelity	Desain awal yang menggambarkan struktur dan tata letak elemen antarmuka pengguna tanpa detail grafis.
Notification Channel	Saluran notifikasi yang digunakan untuk mengatur pengaturan terkait notifikasi.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Industri pertanian dan perkebunan buah-buahan merupakan sektor ekonomi penting di banyak negara di seluruh dunia. Dalam lanskap pertanian Indonesia yang didominasi oleh iklim tropis, produk-produk pertanian memiliki potensi besar untuk pertumbuhan dan pengembangan. Di antara produk-produk hortikultura, buah-buahan dan sayur-sayuran memegang peran utama. Salah satu buah yang memiliki peran signifikan dalam industri ini adalah buah pepaya. Buah pepaya dikenal dengan nilai komersial yang tinggi dan beragam manfaat kesehatan. Buah pepaya memiliki kadar vitamin A dan vitamin C yang cukup tinggi, selain itu, juga mengandung mineral seperti kalsium, fosfor, magnesium, dan zat besi (Agustina dan Sukron 2022). Untuk mendapatkan hasil panen yang optimal dan memenuhi standar pasar, penting untuk mengidentifikasi kematangan buah pepaya dengan tepat. Kematangan buah pepaya dapat memengaruhi rasa, tekstur, warna, aroma, dan nilai nutrisinya.

Keunggulan dari tanaman pepaya adalah bahwa ia dapat berbuah secara berkelanjutan tanpa memperhatikan perubahan musim karena pepaya tidak terpengaruh oleh perubahan musim (Agustina dan Sukron 2022). Menurut data dari Badan Pusat Statistik (BPS), pada tahun 2022, produksi pepaya di Indonesia mencapai 1,05 juta ton, menunjukkan penurunan sebesar 10,41% dibandingkan dengan tahun sebelumnya yang mencapai 1,17 juta ton. Sebelumnya, produksi pepaya di Indonesia memiliki tren fluktuasi yang umumnya meningkat. Tahun 2021 mencatatkan rekor produksi tertinggi sebanyak 1,17 juta ton. Secara geografis, Jawa Timur merupakan produsen terbesar pepaya di Indonesia dengan produksi sebanyak 249.961-ton pada tahun 2022, yang setara dengan 23,88% dari total produksi pepaya dalam negeri. Jawa Barat berada di posisi kedua dengan produksi sebanyak 125.507 ton, diikuti oleh Jawa Tengah yang mencapai 121.482-ton pada tahun tersebut. Di sisi lain, provinsi Gorontalo merupakan produsen pepaya paling sedikit pada tahun 2022, dengan produksi sebanyak 579 ton. Papua Barat dan Kepulauan Riau memproduksi pepaya dalam jumlah yang sedikit lebih besar, yakni masing-masing sebanyak 821-ton dan 911 ton.

Proses pengendalian kualitas buah pepaya sebelum dan setelah panen melibatkan langkah-langkah yang memerlukan penilaian visual yang tepat mengenai tingkat kematangan buah tersebut. Proses penilaian kematangan buah pepaya sebelum panen adalah langkah yang krusial. Secara visual, ketika kita melihat buah papaya dengan mata telanjang, perubahan warna

kulit yang mencerminkan tingkat kematangan, yang dapat diidentifikasi secara visual tanpa perlu alat bantu; mayoritas buah papaya akan memiliki kulit berwarna hijau saat belum matang, berubah menjadi lebih kuning saat setengah matang, dan akhirnya mencapai warna oranye saat sudah matang sepenuhnya, sehingga karakteristik ini merupakan indikator penting dalam menentukan saat yang tepat untuk panen dan konsumsi (Aminudin 2019). Namun, pengendalian kualitas yang mengandalkan penilaian visual manual dapat memakan waktu dan tenaga kerja yang signifikan, dan hasilnya seringkali bervariasi tergantung pada pengalaman individu. Oleh karena itu, pengembangan metode otomatis yang dapat mendeteksi tingkat kematangan buah pepaya secara efisien menjadi sangat penting dalam sektor pertanian.

Salah satu pendekatan yang muncul untuk mengatasi masalah ini adalah menggunakan *Convolutional Neural Networks* (CNN) dalam aplikasi Android. CNN adalah algoritma *deep learning* yang telah terbukti efektif dalam pengenalan gambar dan dapat digunakan untuk mengenali pola visual pada buah pepaya yang menunjukkan tanda-tanda kematangan. Jaringan ini didesain dengan asumsi bahwa citra (gambar) akan digunakan sebagai masukan. Jaringan ini memiliki komponen khusus yang disebut lapisan konvolusi, di mana lapisan ini akan menghasilkan pola dari berbagai bagian citra *input* untuk mempermudah proses klasifikasi. Pendekatan ini membantu meningkatkan efisiensi pembelajaran citra dan membuatnya lebih mudah untuk diaplikasikan. Seperti yang umumnya diketahui, CNN memiliki tingkat akurasi yang cukup tinggi karena menggunakan jaringan yang dalam dan telah banyak digunakan dalam pemrosesan data citra (Nurmalasari dkk. 2023). Aplikasi Android yang mampu mendeteksi kematangan buah pepaya dengan metode CNN memiliki potensi untuk mengoptimalkan proses pengendalian kualitas dan meningkatkan efisiensi dalam industri pertanian.

Di era yang sudah modern ini, smartphone telah menjadi salah satu inovasi teknologi paling berpengaruh dalam kehidupan sehari-hari kita. *Smartphone*, khususnya yang menggunakan sistem operasi Android, telah membawa perubahan signifikan dalam berbagai sektor, termasuk pertanian. Teknologi *smartphone* telah menciptakan peluang baru dan membuka pintu bagi inovasi yang sangat dibutuhkan di dalam dunia pertanian. Selain itu, perkembangan teknologi *smartphone* telah membawa dampak positif yang signifikan pada sektor pertanian. Aplikasi *mobile* yang dapat digunakan oleh petani dan operator pertanian untuk mengukur kematangan buah pepaya dengan cepat dan akurat adalah sebuah inovasi yang sangat diperlukan dalam menghadapi tantangan modern di bidang pertanian. Hal ini dapat

membantu petani dalam mengambil keputusan yang lebih baik terkait waktu panen, penanganan pasca-panen, dan pengelolaan persediaan buah pepaya mereka.

Pemanfaatan teknologi aplikasi Android dan kemampuan *deep learning Convolutional Neural Networks* (CNN) dalam penelitian ini menciptakan sebuah solusi yang lebih efisien dan mudah diakses bagi semua pihak yang terlibat dalam industri pertanian pepaya. Dengan aplikasi ini, petani dapat dengan mudah mengidentifikasi tingkat kematangan buah pepaya mereka tanpa harus mengandalkan metode tradisional yang mungkin kurang akurat dan memakan waktu.

Aplikasi ini tidak hanya bermanfaat bagi para petani, tetapi juga bagi eksportir dan pemangku kepentingan lainnya. Dengan informasi yang lebih akurat tentang kematangan buah pepaya, eksportir dapat merencanakan pengiriman mereka dengan lebih baik, memastikan bahwa buah-buah yang dikirim mencapai pasar dalam kondisi terbaik. Selain itu, para pemangku kepentingan dalam industri pertanian, seperti produsen pupuk, pengecer, dan pemerintah, dapat menggunakan data yang dihasilkan oleh aplikasi ini untuk mengoptimalkan produksi dan distribusi serta mengurangi pemborosan dalam rantai pasokan.

Dengan kontribusi positifnya pada industri pertanian, aplikasi pendeteksi kematangan buah pepaya ini diharapkan dapat membantu meningkatkan kualitas hasil panen pepaya, mengurangi pemborosan makanan, dan mendukung keberlanjutan industri pertanian secara keseluruhan. Selain itu, aplikasi ini juga memungkinkan petani untuk menghemat waktu dan sumber daya yang berharga, sehingga mereka dapat fokus pada aspek-aspek lain dari pertanian yang juga memerlukan perhatian, seperti perawatan tanaman, irigasi, dan manajemen lahan. Aplikasi ini juga akan dilengkapi dengan fitur pengingat yang akan disetel oleh pengguna setelah ia mendapat informasi tingkat kematangannya sehingga waktu panen yang pas tidak terlewat. Dengan demikian, inovasi ini adalah contoh nyata bagaimana teknologi dapat memberikan manfaat besar dalam meningkatkan produktivitas dan keberlanjutan sektor pertanian.

1.2 Rumusan Masalah

Dengan merujuk pada konteks latar belakang yang telah diuraikan di atas, hal-hal berikut akan menjadi fokus utama dalam penelitian ini:

1. Bagaimana mengimplementasikan metode CNN untuk mengklasifikasikan tingkat kematangan buah papaya?

2. Bagaimana membantu petani untuk menetukan tingkat kematangan pepaya dan penctataan data waktu panen?

1.3 Batasan Masalah

Dalam rangka menjaga fokus penelitian dan menghindari penyimpangan dari tujuan utama penelitian ini, kami akan menguraikan batasan masalah berikut ini:

1. Objek papaya yang digunakan pada penelitian ini adalah pepaya jenis California dengan kelas klasifikasi sebanyak tiga, yaitu mentah, setengah matang, dan matang.
2. Pengambilan gambar dilakukan secara langsung pada beberapa kebun petani pepaya di daerah Sleman, Yogyakarta.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah:

1. Model dapat mengklasifikasikan tingkat kematangan buah papaya dengan menggunakan metode CNN.
2. Aplikasi android dapat mengklasifikasikan tingkat kematangan buah pepaya serta mengimplementasikan fitur pencatatan.
3. Aplikasi dapat memprediksi kapan waktu petik pepaya.

1.5 Sistematika Penulisan

Struktur penulisan yang digunakan dalam penyusunan tugas akhir ini dapat dijelaskan sebagai berikut:

BAB I PENDAHULUAN

Bagian pertama ini menjelaskan dengan rinci latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, serta panduan penulisan.

BAB II KAJIAN PUSTAKA

Pada Bagian ini berisikan teori-teori dan konsep-konsep yang relevan dengan penelitian dan mendukung pemecahan masalah yang diajukan. Bagian ini juga disajikan ulasan tentang penelitian-penelitian terdahulu yang terkait dengan topik penelitian ini.

BAB III METODOLOGI

Dalam bagian ini, diuraikan secara rinci mengenai teori yang relevan dalam penelitian dan bagaimana teori tersebut diaplikasikan dalam pengembangan aplikasi yang dikembangkan.

BAB IV HASIL DAN PEMBAHASAN

Bagian ini akan mengulas dengan mendalam tentang penelitian yang telah dilakukan oleh penulis. Hasil dari penelitian akan disajikan dalam berbagai bentuk seperti diagram alur sistem, potongan kode program, tabel data, gambar, atau format lain yang relevan. Semua ini akan dijelaskan secara terstruktur sesuai dengan tahapan penelitian yang telah dilakukan.

BAB V KESIMPULAN DAN SARAN

Bagian akhir ini membahas rangkuman dari hasil penelitian yang telah dilaksanakan, termasuk kesimpulan yang diperoleh, serta menyertakan saran-saran dan ide-ide untuk penelitian masa depan yang terkait dengan topik yang telah diteliti dalam laporan ini.

BAB II
KAJIAN PUSTAKA

2.1 Kajian Teori

Penelitian ini berfokus pada pengembangan sebuah aplikasi Android yang memiliki kemampuan untuk mendeteksi kematangan pepaya dengan menggunakan metode *Convolutional Neural Network* (CNN). Pendeteksian kematangan buah-buahan, seperti pepaya, memiliki potensi besar dalam industri pertanian dan distribusi makanan. Dengan memanfaatkan kecerdasan buatan, penelitian ini bertujuan untuk memberikan solusi yang akurat dan efisien dalam menilai tingkat kematangan pepaya, sehingga dapat membantu petani dan pedagang buah dalam proses panen dan pengiriman yang lebih tepat waktu. Melalui pengembangan aplikasi Android ini, kita dapat meraih kemajuan signifikan dalam meningkatkan efisiensi dan kualitas dalam rantai pasok buah-buahan, serta memungkinkan kontribusi yang lebih positif terhadap pemenuhan kebutuhan pangan global.

2.1.1 Pepaya

Pepaya (*Carica papaya L.*) adalah sebuah tumbuhan buah yang termasuk dalam keluarga *Caricaceae*. Asal-usulnya berasal dari daerah tropis Amerika dan muncul karena perpaduan alami antara *Carica peltata Hook. & Arn*. Saat ini, pepaya dapat ditemukan tumbuh secara luas di berbagai wilayah tropis dan subtropis di seluruh dunia (Febjislami, Suketi, dan Rahmi 2018). Mencari buah pepaya tidaklah menjadi hal yang sulit, karena buah ini tersedia sepanjang tahun dan mudah ditemukan di pasar. Selain itu, buah pepaya memiliki banyak nutrisi dan memberikan berbagai manfaat bagi kesehatan tubuh (Ellif, Sitorus, dan Hidayati 2021). Buah pepaya juga merupakan sebuah buah yang kaya akan nutrisi, terutama vitamin A dan vitamin C. Dalam setiap 100 gramnya, buah pepaya mengandung 3,65 mg vitamin A dan 78 mg vitamin C (Wardani, Wijaya, dan Bimantoro 2022).

Menurut (Noviani, Prambudi, dan Mulyadi 2020)sistem taksonomi, pepaya tergolong dalam:

Kerajaan : *Plantae*
Divisi : Tumbuhan berbiji
Kelas : Berbiji tertutup
Subkelas : Tumbuhan berkeping dua
Ordo : *Caricales*

Famili : *Caricaceae*
Jenis : *Carica papaya L.*

Pepaya IPB9, yang sering disebut sebagai Pepaya California atau Pepaya *Callina*, merupakan salah satu varietas unggul. Keunggulan utama dari Pepaya California adalah rasa yang lebih manis, daging buah yang lebih padat, dan kemampuan untuk menjaga kesegaran buah dalam waktu yang lebih lama (Agustin, Suyudi, dan Nuryaman 2019). Pepaya California merupakan jenis pepaya yang biasanya berbobot sekitar 1 kilogram per buahnya dan berbentuk silindris. Salah satu ciri khas dari buah pepaya California adalah dagingnya berwarna oranye dengan rasa yang manis, sedangkan kulit buahnya memiliki warna hijau yang akan berubah menjadi kuning oranye saat buah tersebut sudah matang (Dwiantara 2020).

Pada penelitian yang dilakukan oleh Suryana, fokusnya adalah pada pengaruh konsentrasi kitosan terhadap lama simpan dan mutu buah pepaya pada dua tingkat kematangan yang berbeda. Metode yang digunakan melibatkan pemantauan secara berkala setiap empat hari untuk mengevaluasi status kandungan kitosan pada setiap tingkat kematangan. Hasil penelitian menunjukkan bahwa untuk mencapai tingkat kematangan dari setengah matang hingga matang, diperlukan waktu sekitar empat hari. Hal ini menyoroti pentingnya periode waktu yang tepat untuk proses pematangan pepaya, yang bisa memengaruhi kualitas dan daya simpan buah tersebut.(Suryana dan Wiradinata 2013)

2.1.2 Citra

Citra adalah suatu gambaran, perbandingan, atau tiruan dari suatu objek. Citra ini bisa berupa hasil visual seperti foto, sinyal analog seperti tampilan pada layar televisi, atau data digital yang dapat disimpan langsung dalam media penyimpanan (Al Ghifari, Sasmito, dan Rudhistiar 2022). Sebuah gambar merupakan representasi, replikasi, atau tiruan dari suatu objek. Citra ini bisa dihasilkan dalam berbagai bentuk, seperti rekaman optik berupa foto atau sinyal analog seperti yang terlihat pada layar monitor. Selain itu, ada juga citra digital yang dapat disimpan pada perangkat televisi atau langsung di dalam media penyimpanan (Arkadia, Damayanti, dan Prasvita 2021).

2.1.3 Citra Digital

Citra didefinisikan sebagai sebuah fungsi f(x, y) yang memiliki dimensi M baris dan N kolom, di mana x dan y adalah koordinat spasial. Intensitas atau tingkat keabuan dari citra pada titik koordinat (x, y) adalah representasi dari nilai amplitudo f di titik tersebut. Jika nilai x, y, dan amplitudo f semuanya bersifat terbatas (*finite*) dan berupa nilai-nilai diskrit, maka citra tersebut dapat digolongkan sebagai citra digital (Ellif dkk. 2021).

2.1.4 Citra Warna

Menurut Ahmad Citra warna adalah sistem grafis yang menggunakan satu set nilai untuk merepresentasikan berbagai tingkat warna. Setiap piksel dalam citra warna menggambarkan warna yang terbentuk dari campuran tiga warna dasar, yaitu Merah (*Red*), Hijau (*Green*), dan Biru (*Blue*), yang biasanya diwakili dengan penyimpanan 8-bit atau 1 *byte*. Ini artinya masing-masing warna dasar memiliki 256 tingkat kecerahan yang berbeda. Sebagai hasilnya, setiap piksel pada citra warna dapat menggabungkan kombinasi warna sebanyak 256^3, atau 16.777.216 warna. Karena jumlah warnanya yang sangat besar, citra ini sering disebut sebagai "*true color*." (Ellif dkk. 2021).

2.1.5 Pengolahan Citra

Menurut Fukushima Pengolahan citra adalah proses yang melibatkan penggunaan metode khusus untuk mengolah citra guna mencapai tujuan yang diinginkan. Dalam perkembangan lebih lanjut, bidang pengolahan citra dan visi komputer digunakan untuk menggantikan peran mata manusia dengan perangkat seperti kamera dan pemindai sebagai alat penerima citra, sementara mesin komputer (dan perangkat lunaknya) berperan sebagai otak atau pusat informasi untuk pengaturan dan pengolahan citra. Oleh karena itu, beberapa aspek penting dalam bidang visi komputer mencakup pengenalan pola, pengenalan biometrik yang berdasarkan karakteristik yang muncul pada tubuh manusia, pencarian citra dan video berdasarkan konten (penarikan informasi dari gambar atau video), pengeditan video, dan lain sebagainya (Arkadia dkk. 2021).

2.1.6 *Convolutional Neural Network* (CNN)

Convolutional Neural Network (CNN) adalah sebuah algoritma dalam *Deep Learning* yang merupakan perkembangan dari *Multi Layer Perceptron* (MLP) dan dibuat khusus untuk memproses data yang disajikan dalam format *grid*, seperti gambar dua dimensi atau suara (Yanto

dkk. 2021). Metode CNN masuk ke dalam kategori *deep neural network learning* karena struktur jaringannya yang dalam, dan metode ini sangat umum digunakan ketika datanya berupa citra (Arkadia dkk. 2021). CNN dirancang untuk memahami fitur-fitur spasial yang paling relevan dalam menggambarkan kelas atau jumlah yang diinginkan. Ini dilakukan dengan menggunakan transformasi data input berurutan, seperti konvolusi, pada berbagai skala spasial. Proses ini membantu dalam mengenali dan menggabungkan fitur-fitur sederhana seperti tepi dan sudut, serta konsep yang lebih kompleks. Fungsi CNN ini dapat dibandingkan dengan cara korteks hewan memproses berbagai rangsangan visual pada skala yang berbeda untuk mengidentifikasi objek, dengan mempertimbangkan fitur-fitur spasial dan konteks spasialnya (Kattenborn dkk. 2021).

CNN terdiri dari tiga komponen utama: layer *convolutional*, layer *pooling*, dan layer *fully connected*. Layer *convolutional* berfungsi sebagai komponen pertama dalam CNN yang bertugas untuk mengidentifikasi fitur-fitur yang terdapat dalam citra. Pada layer ini, filter atau kernel yang berukuran kecil diaplikasikan pada citra input secara bertahap untuk mendeteksi pola atau fitur tertentu seperti tepi, tekstur, atau bentuk. Proses ini menghasilkan peta fitur (*feature map*) yang merepresentasikan lokasi dan kekuatan fitur yang terdeteksi dalam citra asli. Setiap filter dalam layer *convolutional* belajar mendeteksi fitur yang berbeda, dan dengan menggunakan beberapa filter, model dapat mengenali berbagai karakteristik penting dari citra input. Layer *pooling*, sering juga disebut sebagai *subsampling* atau *downsampling*, digunakan setelah layer convolutional. Tujuan utama dari layer ini adalah untuk mengurangi ukuran spatial dari peta fitur yang dihasilkan oleh layer *convolutional*. Salah satu teknik *pooling* yang umum digunakan adalah *max pooling*, di mana hanya nilai maksimum dalam setiap *patch* dari peta fitur yang dipertahankan. Proses *pooling* membantu dalam mengurangi jumlah parameter yang perlu dipelajari oleh model, sehingga mengurangi kompleksitas komputasi dan risiko *overfitting*. Selain itu, *pooling* juga memberikan sedikit translasi *invariance*, artinya model menjadi lebih *robust* terhadap perubahan kecil dalam posisi fitur dalam citra. Layer *fully connected* (FC layer) adalah komponen akhir dalam CNN yang menghubungkan *output* dari layer *convolutional* dan *pooling* ke *output* akhir. Pada layer ini, setiap neuron terhubung ke semua neuron di layer sebelumnya, mirip dengan struktur dalam jaringan saraf tiruan tradisional. FC layer berfungsi untuk menggabungkan fitur-fitur yang telah diekstraksi dan diproses oleh layer *convolutional* dan *pooling*, dan kemudian memetakan mereka ke dalam ruang keluaran yang diinginkan (Nashrullah, Wibowo, dan Budiman 2020).

Membangun dan melatih *Convolutional Neural Network* (CNN), terdapat beberapa *hyperparameter* yang perlu dipertimbangkan untuk mengoptimalkan kinerja model. Salah satunya adalah jumlah *batch*, yang dapat didefinisikan sebagai jumlah data yang diproses secara bersamaan dalam satu iterasi. Pemilihan jumlah *batch* yang tepat sangat penting karena dapat mempengaruhi waktu pelatihan dan akurasi model. Selain itu, *epoch* juga memainkan peran krusial. *Epoch* merujuk pada jumlah iterasi pelatihan yang dilakukan pada seluruh data, dan jumlah *epoch* yang tepat dapat mempengaruhi akurasi model serta waktu yang dibutuhkan untuk pelatihan. Ukuran gambar juga merupakan *hyperparameter* penting, dimana ukuran gambar yang diproses oleh model harus dipilih dengan cermat karena dapat mempengaruhi akurasi model dan waktu pelatihan. Struktur lapisan-lapisan konvolusional, yang mencakup jumlah lapisan, ukuran filter, dan fungsi aktivasi, tentunya harus dirancang dengan hati-hati. Struktur lapisan konvolusional yang optimal akan sangat mempengaruhi akurasi model dan efisiensi waktu pelatihan. Dengan memperhatikan dan mengatur *hyperparameter* ini dengan tepat, kinerja model CNN dapat ditingkatkan secara signifikan (Nurhopipah dan Larasati 2021).

2.1.7 *TensorFlow Lite*

TensorFlow Lite adalah sebuah kerangka kerja yang dirancang khusus untuk menjalankan model *deep learning* pada perangkat mobile. Kerangka kerja ini menggunakan format *file* yang lebih kompak untuk menyimpan model-model tersebut, sehingga dapat dijalankan secara efisien pada perangkat dengan sumber daya komputasi dan memori yang terbatas. *TensorFlow Lite* dikembangkan sebagai versi yang lebih ringan dan lebih cepat dari *TensorFlow*, yang dioptimalkan khusus untuk perangkat *mobile*. Dalam *TensorFlow Lite*, model *deep learning* yang telah dilatih menggunakan *TensorFlow* dapat diubah menjadi format yang lebih efisien yang dikenal sebagai *FlatBuffer*. Format ini memungkinkan model-model tersebut untuk dijalankan dengan kecepatan dan efisiensi tinggi pada perangkat mobile, sambil mengurangi penggunaan sumber daya seperti CPU, memori, dan daya baterai (Reda dkk. 2022).

2.1.8 Android

Android merupakan sebuah sistem operasi yang digunakan di perangkat mobile, khususnya smartphone. Sistem operasi ini dirancang dan dikembangkan oleh *Google* dengan tujuan memberikan pengalaman pengguna yang mudah dimengerti dan terhubung dengan beragam fitur

serta layanan. Android juga memberikan dukungan luas untuk pengembangan aplikasi melalui *Google Play Store* (Fadlilah, Mahamad, dan Handaga 2021).

2.2 Kajian Literatur

Penelitian yang telah dilakukan sebelumnya memiliki peran yang sangat signifikan dalam memahami kaitan antara penelitian saat ini dengan penelitian sebelumnya. Kemungkinan adanya tindakan meniru atau pengulangan dalam penelitian saat ini juga menjadi hal yang penting untuk ditelusuri, dengan tujuan untuk menilai kontribusi penelitian terbaru terhadap perkembangan ilmu pengetahuan.

Tabel 2. 1 Tabel ulasan kritis

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
1	Kematangan dengan Metode CNN (Nanas)	Mikrokontroler, Pendeteksi, Buah Nanas, *Convolutional Neural Network* (CNN).	Penelitian ini bertujuan mengembangkan sistem pendeteksian kulit buah nanas dengan menggunakan model *Convolutional Neural Network* (CNN) dan mikrokontroler ESP32 dengan kamera. Proses pengujian melibatkan pengambilan gambar nanas oleh kamera ESP32 dengan bantuan sensor ultrasonik. Hasil pengujian menunjukkan akurasi tertinggi 86%, terendah 80%, dengan rata-rata akurasi sekitar 83,33%. Pada penelitian ini dihasilkan alat pendeteksi kematangan buah nanas. Pendapat saya tentang penelitian ini tentunya untuk peningkatan dan validasi lebih lanjut mungkin diperlukan untuk memastikan keandalan dan efektivitas sistem ini dalam situasi dunia nyata, Jumlah data untuk di *training* juga bisa diperbanyak lagi pada penelitian ini hanya ada 40 data *training* dari setiap kelasnya.	(Bili dkk. 2022)
2	Kematangan dengan Metode CNN (Mangga Badami)	Mangga Badami, CNN, Citra Digital	Penelitian ini bertujuan untuk membuat program guna mendeteksi tingkat kematangan buah mangga Badami berdasarkan warnanya. Hal ini dicapai dengan mengimplementasikan metode *Convolutional Neural Networks* (CNN) dalam aplikasi pengolahan citra digital. Tujuannya adalah untuk memastikan bahwa buah mangga yang akan dikonsumsi telah mencapai tingkat kematangan yang sesuai. Pengujian sistem ini dilakukan dengan menggunakan 25 citra sebagai data uji dan 179 citra sebagai data latih dari total 204 citra yang tersedia. Hasil pengujian menunjukkan tingkat akurasi model sebesar 97,2%. Menurut pendapat saya dalam penelitian ini citra yang disediakan jumlah nya tidak seimbang contoh pada citra	(Arkadia dkk. 2021)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
			yang dikategorikan Mangga Badami Busuk ada 35 sedangkan pada Mangga Badami Mentah ada 75 citra, dan 94 Mangga Badami Matang untuk itu perlu disetarakan agar memudahkan dalam melatih model, sehingga model yang di buat lebih memiliki akurasi yang maksimal bukan secara angka saja.	
3	Kematangan dengan Metode *Naive Bayes* (Pepaya)	Klasifikasi, kematangan buah, warna HSV, *naïve bayes*	Penelitian ini mengembangkan sebuah sistem untuk mengklasifikasikan tingkat kematangan buah pepaya dengan menggunakan ruang warna HSV dan metode *Naïve Bayes*. Dalam proses klasifikasi citra, sistem ini memanfaatkan ruang warna HSV dan metode *Naïve Bayes* dengan menghitung probabilitas kemiripan gambar yang diuji dengan citra latih yang telah ada. Dari total 50 buah pepaya yang digunakan dalam penelitian ini (30 sebagai data latih dan 20 sebagai data uji). Menurut pendapat saya pada penelitian ini walaupun mendapatkan akurasi di angka 100% bukan berarti sistem sudah berjalan dengan baik mungkin untuk mendeteksi gambar atau citra yang sumbernya dari orang asing mungkin masih bisa terjadi kesalahan dalam memprediksi, dan juga mungkin juga bisa di buat aplikasi *mobile* untuk memudahkan pengguna mengaplikasikan aplikasi ini secara nyata.	(Ellif dkk. 2021)
4	Kematangan dengan Metode CNN (Jeruk)	*Convolutional Neural Network* (CNN), *Deep Learning*, *Sweet orange*	Penelitian ini tentang membuat sistem komputer untuk mendeteksi kematangan buah jeruk berdasarkan tingkat kecerahan warna. Penelitian menggunakan algoritma *Convolutional Neural Network* (CNN) sebagai bagian dari deep learning untuk mengklasifikasikan jeruk manis berdasarkan citra. Berikut untuk langkah-langkah yang digunakan untuk penelitian ini, yang pertama adalah Identifikasi masalah dalam klasifikasi jeruk manis, menggunakan algoritma *Convolutional Neural Network* (CNN) untuk klasifikasi citra, mengumpulkan dataset 100	(Yanto dkk. 2021)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
			gambar jeruk berkualitas dan tidak berkualitas, melatih model CNN dengan dataset tersebut, dan menguji model dengan 10 citra (5 berkualitas, 5 tidak berkualitas) untuk mengukur akurasi.Hasil penelitian ini menunjukkan tingkat akurasi klasifikasi sebesar 97.5184% pada dataset 100 gambar jeruk manis. Untuk pengujian dengan 10 citra, hasilnya adalah 96% untuk pelatihan dan 92% untuk pengujian. Grafik hasil akurasi testing mencapai 92%, yang dianggap cukup baik.Pendapat tentang penelitian ini adalah mungkin perlu diperluas dan diuji lebih lanjut dengan dataset yang lebih besar dan variasi yang lebih luas dalam citra jeruk manis untuk memastikan konsistensi dan kemampuan model ini dalam situasi dunia nyata.	
5	Klasifikasi kematangan menggunakan algoritma *K-Nearest Neighbor* (Tomat)	Kematangan, Tomat, KNN, *Thresholding*, RGB	Penelitian ini bertujuan untuk mengembangkan aplikasi deteksi kematangan tomat dengan menggunakan pengolahan citra melalui Matlab. Dalam penelitian ini, algoritma *K-Nearest Neighbor* digunakan untuk mengklasifikasikan tingkat kematangan tomat berdasarkan nilai RGB citra. Metode penelitian melibatkan pengambilan gambar tomat, proses segmentasi citra menjadi citra biner, ekstraksi fitur RGB, dan penggunaan algoritma *K-Nearest Neighbor* untuk klasifikasi. Aplikasi hasil pengembangan menampilkan gambar asli, citra biner, tabel nilai RGB, dan hasil klasifikasi. Dari hasil pengujian, aplikasi ini mencapai tingkat akurasi sebesar 80% dalam mengklasifikasikan tingkat kematangan tomat. Ini dapat membantu petani dalam memilih tomat yang baik untuk konsumsi. Secara keseluruhan, penelitian ini berkontribusi dalam pengembangan aplikasi deteksi kematangan tomat melalui pengolahan citra. Aplikasi ini berguna bagi petani, meskipun perlu perbaikan tingkat akurasi. Penelitian	(Al Ghifari dkk. 2022)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
			selanjutnya dapat fokus pada peningkatan metode klasifikasi yang lebih akurat.	
6	Kematangan buah menggunakan algoritma YOLO (Pepaya)	Pepaya, Deteksi obyek, YOLO	Penelitian ini bertujuan untuk mengembangkan metode deteksi tingkat kematangan buah pepaya menggunakan algoritma YOLO (*You Only Look Once*). Metode penelitian yang digunakan terdiri dari empat tahap, yaitu *pre-processing* data, pengumpulan gambar, *labelling*, dan *training* model. Pada tahap *pre-processing* data, citra objek pepaya dikumpulkan dan diberi label sesuai dengan kelasnya. Kemudian dilakukan pelatihan data untuk mendapatkan model yang dapat digunakan dalam deteksi menggunakan metode YOLO. Hasil penelitian ini menunjukkan bahwa deteksi tingkat kematangan buah pepaya menggunakan metode YOLO memiliki tingkat keberhasilan yang cukup tinggi. Dalam pengujian, dari total 30 data yang dideteksi, 27 data berhasil terdeteksi dengan benar. Berdasarkan hasil penelitian, model deteksi tingkat kematangan buah pepaya menggunakan metode YOLO memiliki akurasi sebesar 93%. Selain itu, penelitian ini juga mengidentifikasi beberapa kendala yang dapat mempengaruhi hasil deteksi, seperti pencahayaan yang tidak optimal, sudut kamera yang miring, pergerakan cepat objek, dan adanya objek penghalang . Secara keseluruhan, penelitian ini memberikan kontribusi dalam pengembangan metode deteksi tingkat kematangan buah pepaya menggunakan algoritma YOLO. Metode ini dapat digunakan dalam aplikasi pengolahan buah pepaya, seperti dalam industri pertanian atau pengolahan makanan. Namun, penelitian ini juga memiliki beberapa keterbatasan,	(Agustina dan Sukron 2022)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
			seperti belum adanya analisis performa secara mendalam dan belum dilakukan perbandingan dengan metode deteksi lainnya. Oleh karena itu, penelitian ini dapat menjadi dasar untuk penelitian lebih lanjut dalam pengembangan metode deteksi tingkat kematangan buah pepaya.	
7	Kematangan dengan Metode CNN (Pepaya)	*Convolutional*, Klasifikasi, Tingkat Kematangan, *Neural Network*	Mengembangkan metode klasifikasi menggunakan *Convolutional Neural Network* (CNN) untuk menentukan tingkat kematangan buah pepaya. Penelitian ini menggunakan dataset 1.500 gambar buah pepaya yang diambil dengan menggunakan ponsel. Gambar-gambar tersebut dikategorikan ke dalam tiga kelas yang mewakili tingkat kematangan yang berbeda. Untuk melakukan penelitian, penulis mengimplementasikan model CNN dan melatihnya menggunakan dataset tersebut. Kinerja metode klasifikasi dievaluasi dengan menggunakan berbagai metrik, termasuk *confusion matrix, precision, recall, accuracy, dan kappa score. Confusion matrix* memberikan informasi komparatif tentang hasil klasifikasi, sedangkan *precision*, *recall*, *accuracy*, dan *kappa score* mengukur kinerja model. Akurasi model CNN dalam penelitian ini dilaporkan mencapai 97,69%. Hasil penelitian menunjukkan keefektifan metode klasifikasi yang diusulkan dalam menentukan tingkat kematangan buah pepaya. Model CNN mencapai akurasi, presisi, *recall*, dan skor kappa yang tinggi, yang menunjukkan kemampuannya untuk mengklasifikasikan buah secara akurat berdasarkan tingkat kematangannya.	(Nurmalasari dkk. 2023)
8	Kematangan menggunakan	Buah Pepaya, Klasifikasi	Penelitian ini bertujuan untuk mengklasifikasikan buah pepaya bangkok berdasarkan tingkat kematangannya	(Aminudin 2019)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
	Metode *K-Nearest Neighbor* (Pepaya)	Buah, KNN, Pengolahan Citra, Android.	menggunakan metode *K-Nearest Neighbor* (KNN) berdasarkan warna kulit buah . Penelitian ini dilakukan dengan menggunakan 12 citra latih dan 12 citra uji. Proses pengklasifikasian dilakukan dengan metode ekstraksi statistika warna, seperti rerata *red*, rerata *green*, standar deviasi *red*, dan lain-lain . Hasil akhir dari penelitian ini ditentukan melalui metode *K-Nearest Neighbor* (KNN) dengan menggunakan nilai rata-rata sebagai penentu klasifikasi. Penelitian ini menghasilkan akurasi sebesar 75% untuk *preprosesing* menggunakan jumlah K=5 dan juga 75% untuk *preprosesing* menggunakan jumlah K=7. Dalam pengujian, peneliti menemukan bahwa ada 4 data uji yang dinyatakan sebagai buah pepaya mentah dan 3 data uji yang dinyatakan sebagai buah pepaya setengah matang. Penelitian ini bisa ditingkatkan untuk jumlah dataset yang lebih banyak lagi sehingga bisa untuk memastikan konsistensi dan kemampuan model yang lebih baik.	
9	Klasifikasi Penyakit buah Menggunakan CNN (Pepaya)	*Papaya Diseases, Deep Learning, Keras, Classification.*	Klasifikasi dan pengenalan penyakit pepaya dengan menggunakan teknik *deep learning*. Para peneliti menggunakan model CNN (*Convolutional Neural Network*) yang diimplementasikan dengan modul *Keras API*. Mereka mengumpulkan dataset gambar penyakit pepaya dan melakukan *preprocessing* dengan meminimalkan bentuk dan transformasi. Rasio pelatihan dan pengujian untuk model tersebut adalah 80:20. Hasil penelitian menunjukkan bahwa pendekatan yang diusulkan mencapai akurasi 91% dalam mendeteksi dan mengklasifikasikan penyakit pepaya. Akurasi, *recall*, presisi, dan skor F1 dari model CNN semuanya adalah 0,909. Akurasi model dan grafik *loss* juga menunjukkan akurasi rata-rata 91%.	(Hossen dkk. 2020)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
			Dibandingkan dengan teknik lain yang sudah ada seperti *K-means clustering* dan *random forests*, model CNN dengan modul *Keras* API memberikan hasil yang lebih baik dengan akurasi rata-rata 91%. Algoritma penelitian ini dianggap lebih akurat dalam mengidentifikasi penyakit pepaya. Secara keseluruhan, penelitian ini memberikan wawasan yang berharga tentang klasifikasi dan pengenalan penyakit pepaya menggunakan teknik pembelajaran mendalam, khususnya model CNN yang diimplementasikan dengan modul *Keras API*. Hasilnya menunjukkan tingkat akurasi yang tinggi dan potensi untuk identifikasi penyakit pepaya yang lebih akurat.	
10	Prediksi menentukan waktu panen buah (DNNs)	*Ripening stage prediction, deep neural network, machine learning, tomato, strawberry, small dataset*	Riset ini bertujuan untuk mengembangkan metode prediksi yang efisien untuk menentukan waktu panen buah dengan menggabungkan fitur yang didorong oleh *Deep Neural Networks* (DNNs) dengan algoritma *machine learning* tradisional. Metode ini menggunakan teknologi pengolahan citra, *machine learning*, dan *deep learning* untuk memprediksi tahap kematangan buah secara cepat dan akurat tanpa memerlukan tenaga kerja manual . Metode yang digunakan dalam penelitian ini terdiri dari beberapa langkah. Pertama, data gambar buah stroberi dan tomat dikumpulkan dari sebuah situs web. Kedua, fitur-fitur diekstraksi menggunakan berbagai DNN berdasarkan transfer learning. Ketiga, parameter optimal untuk *fine-tuning* antara berbagai DNN dan metode *machine learning* tradisional diestimasi untuk memprediksi empat tahap kematangan buah. Terakhir, hasil prediksi digunakan untuk mengidentifikasi tahap kematangan buah.	(Cho dkk. 2021)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
			Hasil penelitian menunjukkan bahwa kombinasi antara berbagai DNN dan metode MLP (*softmax*) memiliki tingkat klasifikasi tertinggi sekitar 90% dan 83% untuk gambar stroberi . Selain itu, kombinasi antara *Resnet* 50 dan SVM atau kombinasi antara *Resnet* 101 dan MLP juga menunjukkan tingkat klasifikasi terbaik sebesar 80% dan 78% . Namun, kombinasi antara berbagai DNN dan model klasifikasi statistik menunjukkan tingkat klasifikasi yang lebih rendah secara keseluruhan . Secara keseluruhan, penelitian ini memberikan kontribusi dalam pengembangan metode prediksi yang efisien untuk menentukan tahap kematangan buah dengan menggunakan DNN dan algoritma *machine learning* tradisional. Namun, penelitian ini juga menghadapi beberapa kendala, seperti kurangnya data yang memadai .	
11	Kematangan Buah (YOLOv8)	YOLOv8, *CenterNet, Visual object detection*	Riset ini bertujuan untuk melatih model deteksi buah yang mampu mengidentifikasi lokasi dan kelas buah dalam gambar. Model ini menggunakan metode *computer vision* dan *deep learning* untuk mengekstrak fitur visual dari gambar buah dan menganalisis karakteristik kulit buah untuk memprediksi kelas buah tersebut. Dalam riset ini, digunakan dua model "*anchor-free*", yaitu YOLOv8 dan *CenterNet*, yang dilatih menggunakan dataset buatan sendiri untuk menghasilkan prediksi yang akurat . Metode pelatihan model deteksi objek visual umumnya melibatkan dua tahap, yaitu proses pelatihan dan proses pengujian. Tujuan utama dari pelatihan model adalah menggunakan dataset gambar untuk mendapatkan parameter-parameter jaringan deteksi. Dataset pelatihan ini	(Xiao, Nguyen, dan Yan 2023)

No.	Sub Tema	Keywords	Ulasan Kritis	Pustaka
			mencakup informasi yang diannotasi seperti lokasi objek dan kelasnya . Hasil dari riset ini adalah berhasilnya model YOLOv8 dalam mendeteksi dan mengklasifikasikan buah dengan tingkat akurasi yang mengesankan, yaitu 99,5%. Selain itu, dataset buatan sendiri juga mencakup faktor-faktor seperti tumpang tindih dan *occlusion* buah, serta berbagai kelas buah . Pendapat tentang penelitian ini adalah bahwa model deteksi buah yang dilatih menggunakan metode *deep learning* dapat menjadi solusi yang efektif untuk mengatasi kekurangan tenaga kerja manusia dalam panen buah. Namun, penelitian ini juga mengakui perlunya mempertimbangkan pengaruh lingkungan pertanian yang ekstrem terhadap deteksi buah secara otomatis, seperti kondisi cuaca buruk atau gangguan dari burung. Penelitian ini memberikan kontribusi dengan mengembangkan model deteksi buah yang akurat dan menciptakan dataset buatan sendiri .	

Dalam tabel literatur yang disediakan, terdapat sejumlah penelitian yang telah dilakukan untuk mengkaji deteksi tingkat kematangan buah-buahan menggunakan berbagai metode. Tabel ini memberikan pemahaman yang mendalam tentang berbagai model dan tingkat akurasi yang telah diperoleh dalam konteks deteksi tingkat kematangan buah-buahan.

Tabel Tabel 2. 1 membandingkan beberapa model yang digunakan dalam penelitian. Model-model ini rata-rata digunakan untuk deteksi tingkat kematangan buah, dengan masing-masing model memiliki pendekatan yang berbeda dalam pengenalan objek. Dalam Tabel 2. 1 juga menyajikan tingkat akurasi yang dicapai oleh masing-masing model. Akurasi merupakan ukuran sejauh mana model dapat mengklasifikasikan tingkat kematangan buah dengan benar. Hasil pengujian menunjukkan bahwa beberapa model yang memiliki akurasi beragam.

Tabel Tabel 2. 1 membandingkan berbagai model dan akurasi dari setiap penelitian. Terdapat berbagai metode yang telah digunakan, seperti CNN, *Naive Bayes*, *K-Nearest Neighbor* (KNN), dan YOLO. Akurasi hasil penelitian berkisar antara 75% hingga 97,69%, dengan beberapa penelitian mencapai akurasi sekitar 90% hingga 97,69%. Pada tabel literatur yang disediakan, terlihat perbandingan antara berbagai model dan metode yang digunakan dalam penelitian-penelitian deteksi tingkat kematangan buah. Metode-metode tersebut termasuk *Convolutional Neural Network* (CNN), *Naive Bayes*, *K-Nearest Neighbor* (KNN), dan YOLO. Masing-masing metode memiliki pendekatan yang unik dalam mengenali tingkat kematangan buah berdasarkan citra visual.

Pertama, *Convolutional Neural Network* (CNN) adalah metode yang sering digunakan dalam penelitian ini. Model-model CNN cenderung memberikan akurasi yang tinggi dalam mengklasifikasikan tingkat kematangan buah. CNN bekerja dengan memproses citra secara hierarkis, memungkinkan ekstraksi fitur otomatis dari gambar dan pengambilan keputusan yang akurat berdasarkan informasi tersebut.

Kedua, metode *Naive Bayes* digunakan dalam penelitian tertentu untuk mengklasifikasikan tingkat kematangan buah berdasarkan ruang warna HSV dan probabilitas kemiripan citra uji dengan citra latih. Meskipun satu penelitian mencapai akurasi 100%, perlu diperhatikan bahwa akurasi tinggi dalam jumlah yang kecil mungkin memunculkan kekhawatiran tentang keandalan model di berbagai situasi dunia nyata.

Ketiga, *K-Nearest Neighbor* (KNN) adalah metode yang menggunakan ekstraksi fitur warna RGB untuk mengklasifikasikan tingkat kematangan buah. Meskipun beberapa penelitian

mencapai akurasi sekitar 75%, ada potensi untuk meningkatkan tingkat akurasi dengan dataset yang lebih besar dan lebih variasi dalam citra buah.

Keempat, metode YOLO (*You Only Look Once*) digunakan dalam beberapa penelitian untuk mendeteksi tingkat kematangan buah dengan akurasi sekitar 93%. Model YOLO memungkinkan deteksi lokasi dan kelas buah dalam gambar dengan efisiensi yang tinggi.

Berbagai metode yang digunakan dalam penelitian deteksi tingkat kematangan buah menawarkan pendekatan yang berbeda dan tingkat akurasi yang bervariasi. CNN seringkali menjadi pilihan utama karena akurasi tinggi yang dicapainya. Namun, setiap metode memiliki kelebihan dan kelemahan masing-masing. Dalam konteks penelitian ini, penggunaan model CNN didukung oleh hasil penelitian sebelumnya yang menunjukkan tingkat akurasi yang tinggi, sehingga menjadi pilihan utama dalam pengembangan aplikasi klasifikasi tingkat kematangan pepaya pada pohon berbasis Android.

Pada Tabel 2. 1 menunjukkan bahwa beberapa penelitian rata-rata menggunakan model CNN untuk mendeteksi kematangan buah dan melaporkan tingkat akurasi rata-rata yang mencapai angka yang tinggi menggunakan model CNN. Hal ini menunjukkan bahwa model CNN sering digunakan dalam konteks deteksi tingkat kematangan buah dan memiliki potensi untuk memberikan hasil yang baik. Terlihat bahwa model CNN memiliki rata-rata akurasi yang lebih tinggi dibandingkan dengan metode lainnya, dengan rata-rata akurasi sekitar 97%. Metode CNN juga lebih banyak digunakan untuk memprediksi kematangan buah.

Berdasarkan hasil analisis literatur yang ada dalam tabel tersebut, penelitian ini akan menggunakan metode *Convolutional Neural Network* (CNN) sebagai metode utama dalam mengembangkan aplikasi pendeteksi tingkat kematangan pepaya pada pohon berbasis Android. Dengan tingkat akurasi yang tinggi yang dicapai oleh model CNN dalam penelitian sebelumnya, penelitian ini tentunya mengambil model CNN untuk mengembangkan aplikasi. Hal ini akan memungkinkan untuk memanfaatkan keunggulan teknologi *deep learning* dalam mengidentifikasi tingkat kematangan buah pepaya secara akurat berdasarkan citra visual.

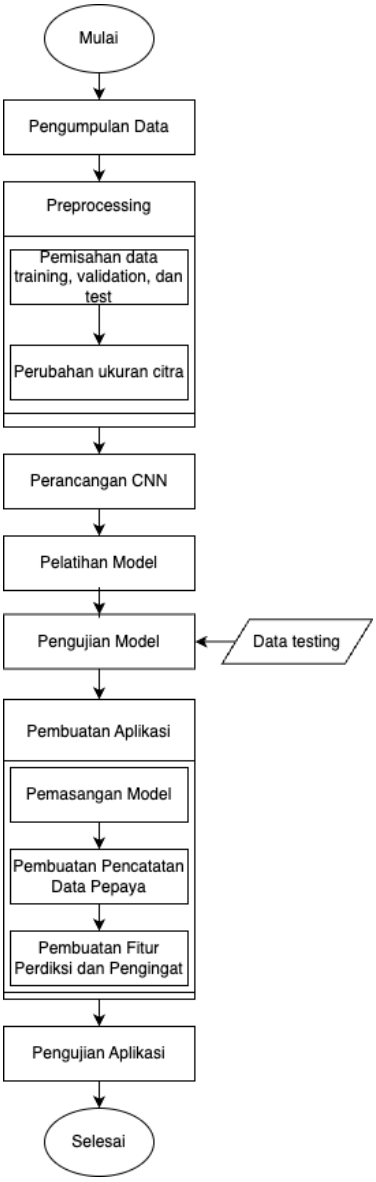
Terdapat beberapa pertimbangan mengapa menggunakan CNN, di antaranya adalah CNN telah berulang kali menunjukkan kinerja yang konsisten dalam berbagai penelitian terkait deteksi tingkat kematangan buah. Misalnya, penelitian oleh Masturoh dan Haryanti mencapai akurasi 97,69% dalam mengklasifikasikan tingkat kematangan pepaya. Selain itu, penelitian oleh Yanto juga mencatat akurasi yang tinggi pada deteksi kematangan jeruk manis menggunakan CNN. Memilih metode yang tepat untuk deteksi tingkat kematangan buah tidak hanya bergantung pada tingkat akurasi yang dicapai dalam suatu penelitian, tetapi juga pada

berbagai faktor lain seperti keandalan, konsistensi, jumlah data yang dibutuhkan, serta kemampuan metode tersebut untuk menangani variabilitas dalam data nyata. Salah satu keunggulan utama dari CNN adalah kemampuannya untuk secara otomatis mengekstrak fitur dari data mentah. CNN dapat menangkap pola-pola yang rumit dalam gambar tanpa perlu ekstraksi fitur manual, membuatnya lebih kuat dalam mendeteksi berbagai karakteristik buah dalam berbagai kondisi pencahayaan dan sudut pandang. Sedangkan pada metode *Naïve Bayes* biasanya memerlukan fitur yang telah diekstraksi secara manual, seperti ruang warna HSV dalam penelitian Ellif (2021). Ini membatasi kemampuan model untuk menemukan fitur-fitur penting lain yang mungkin tidak diketahui sebelumnya. *Framework* dan *library* modern seperti *TensorFlow* dan *Keras* memfasilitasi pengembangan model CNN dan integrasinya dengan platform Android, memudahkan implementasi sistem yang siap digunakan oleh pengguna akhir. Dengan mempertimbangkan keunggulan-keunggulan tersebut, dipilih CNN untuk mengembangkan aplikasi klasifikasi tingkat kematangan pepaya berbasis Android.

Dalam konteks penelitian ini, terdapat perbedaan dengan penelitian lainnya yang telah dilakukan sebelumnya. Perbedaan utama terletak pada penggunaan dataset yang diperoleh secara langsung dari lapangan, dimana pepaya difoto langsung dari pohon pada lahan petani. Pendekatan ini memberikan keunggulan karena data yang diperoleh lebih representatif dan sesuai dengan kondisi nyata di lapangan. Selain itu, penggunaan platform Android untuk mengintegrasikan model klasifikasi tingkat kematangan pepaya menjadikan aplikasi ini lebih mudah diakses dan digunakan, terutama oleh para petani. Aplikasi yang telah dikembangkan juga memiliki fitur-fitur tambahan seperti pencatatan data pepaya, notifikasi pengingat, dan prediksi kapan pepaya akan matang. Dengan demikian, diharapkan bahwa penelitian ini dapat memberikan kontribusi yang lebih signifikan dalam meningkatkan efisiensi dan produktivitas dalam sektor pertanian.

BAB III
METODOLOGI PENELITIAN

Proses penelitian ini mengadopsi suatu metodologi untuk memastikan pengidentifikasian langkah-langkah secara sistematis, yang nantinya dapat berfungsi sebagai panduan terstruktur dalam menangani permasalahan yang dihadapi. Rinciannya dapat diamati melalui representasi visual pada Gambar 3. 1.



Gambar 3. 1 Proses pembuatan model dan aplikasi

Pada Gambar 3. 1 menguraikan rangkaian metodologi yang akan diterapkan mulai dari tahap awal hingga tahap akhir dalam jalur penelitian ini. Setiap tahapan dalam gambar tersebut akan dijabarkan secara rinci dalam sub bab dibawah ini.

3.1 Pengumpulan Data

Pada tahapan pertama yaitu pengumpulan data atau citra dari papaya yang akan digunakan untuk membuat model machine learning. Pada pengumpulan dataset papaya mencakup berbagai variasi tingkat kematangan yaitu mentah, setengah matang, dan matang. Data yang digunakan pada penelitian ini merupakan data primer sehingga dilakukan pemotretan langsung pada kebun-kebun petani papaya yang berada di Yogyakarta.

3.2 *Preprocessing*

Setelah berhasil mengumpulkan data, langkah berikutnya adalah tahap praproses data. Pada fase ini, serangkaian langkah yang dilakukan untuk memastikan keberhasilan dan keoptimalan penggunaan dataset pada pembuatan model *machine learning* untuk pengenalan tingkat kematangan buah papaya.

3.2.1 Pemisahan data *Training*, *Validation*, dan *Testing*

Pertama-tama, dalam proses ini, dilakukan pemisahan citra papaya menjadi tiga bagian utama: data untuk pelatihan (*training*), validasi (*validation*), dan pengujian (*testing*). Pemisahan ini dilakukan dengan cermat untuk memastikan distribusi yang seimbang dari tingkat kematangan pada setiap bagian. Data pelatihan digunakan untuk melatih model, data validasi digunakan untuk mengoptimalkan parameter model selama pelatihan, dan data pengujian digunakan untuk menguji kinerja model secara keseluruhan.

3.2.2 Perubahan Ukuran citra

Selanjutnya, pada tahap ini, dilakukan perubahan ukuran citra. Proses ini mencakup penyesuaian resolusi gambar agar konsisten dan sesuai dengan kebutuhan model. Resolusi citra yang seragam memastikan bahwa model dapat mengenali pola dengan konsistensi dan presisi yang tinggi. Pengaturan ukuran citra juga membantu dalam mengoptimalkan penggunaan sumber daya komputasi selama pelatihan dan pengujian model. Adanya perubahan ukuran citra ini memungkinkan dataset yang digunakan dalam pelatihan model untuk berada dalam kondisi optimal, meningkatkan kemampuan generalisasi model terhadap data baru.

3.3 Perancangan CNN

Dalam tahap ini, model *Convolutional Neural Network* (CNN) dirancang untuk memproses dan mengenali pola pada gambar pepaya. Arsitektur CNN dipilih dan dikonfigurasi sesuai dengan kebutuhan penelitian untuk memastikan kinerja yang optimal.

3.4 Pelatihan Model

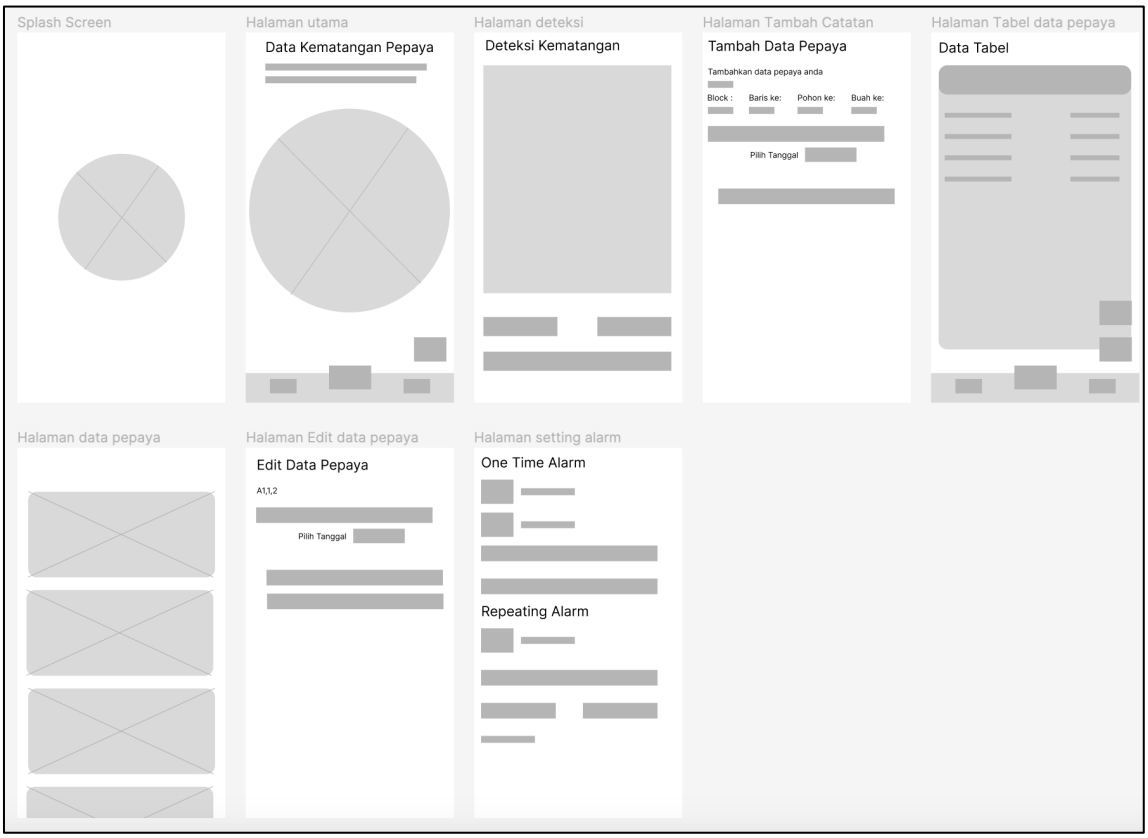
Model CNN kemudian dilatih menggunakan dataset yang telah diproses. Proses pelatihan ini memungkinkan model untuk "belajar" dan menyesuaikan diri dengan karakteristik dataset, sehingga dapat mengenali tingkat kematangan pepaya dengan akurat.

3.5 Pengujian Model

Model yang telah dilatih diuji menggunakan dataset *testing* untuk mengukur sejauh mana kemampuannya dalam mengenali tingkat kematangan pepaya. Hasil klasifikasi dan akurasi model menjadi fokus utama dalam tahap ini.

3.6 Pembuatan Aplikasi Android

Setelah berhasil mengonfirmasi keberhasilan model, langkah selanjutnya adalah memasuki fase pembuatan aplikasi Android. Proses ini memerlukan kecermatan dalam pemrograman guna memastikan fungsionalitas yang optimal dan antarmuka pengguna yang intuitif. Langkah pertama dalam tahap ini adalah pembuatan rancangan antarmuka aplikasi.



Gambar 3. 2 *Wireframe low-fidelity*

Pada Gambar 3. 2 terdapat desain *wireframe low-fidelity* yang menggambarkan halaman-halaman yang akan dibuat dalam aplikasi nantinya. *Wireframe* ini menjadi landasan awal dalam merancang antarmuka pengguna yang efektif dan efisien. Dengan menggunakan *wireframe*, kita dapat menggambarkan secara kasar struktur dan tata letak elemen-elemen antarmuka, tanpa terlalu fokus pada detail desain grafis. Hal ini memungkinkan kita untuk fokus pada pengaturan fungsi dan navigasi yang tepat sebelum melangkah ke tahap desain grafis yang lebih detail. Setelah pembuatan antarmuka, fokus diberikan pada tiga fitur utama di dalamnya, yang akan diuraikan sebagai berikut:

3.6.1 Pemasangan Model

Langkah pertama dalam pemasangan model *machine learning* ke dalam aplikasi Android melibatkan beberapa tahapan yang teliti dan esensial. Proses ini dirinci sebagai berikut:

Konversi Model ke Format *TensorFlow Lite*

Pertama-tama, model machine learning yang telah dibuat perlu diubah ke format *TensorFlow Lite*. Langkah ini sangat krusial untuk memastikan model dapat diintegrasikan

secara efisien dalam lingkungan perangkat mobile, sekaligus meminimalkan kebutuhan sumber daya yang dibutuhkan.

Impor Model ke Aplikasi

Setelah konversi, langkah selanjutnya adalah mengimpor model langsung ke dalam aplikasi Android. Pembuatan kode dilakukan secara cermat untuk memastikan bahwa integrasi model berjalan dengan mulus. Dalam hal ini, pembuatan *activity* khusus menjadi penting untuk menangani proses deteksi citra papaya. Ukuran gambar perlu disesuaikan agar sesuai dengan kebutuhan model, memastikan bahwa proses deteksi berjalan efektif tanpa mengorbankan informasi penting pada citra.

Input Gambar

Dalam langkah selanjutnya yaitu pembuatan input gambar melalui dua opsi utama: melalui kamera ponsel atau melalui galeri perangkat. Pertama, pada antarmuka aplikasi, ditambahkan tombol kamera yang memicu fungsi pengambilan gambar menggunakan kamera ponsel. Proses ini mencakup permintaan izin pengguna untuk menggunakan kamera dan pemanggilan *intent* yang membuka aplikasi kamera. Sebaliknya, opsi kedua adalah memilih gambar dari galeri. Tombol galeri ditambahkan pada antarmuka, dan pengguna diberi izin untuk mengakses galeri perangkat. Kode diperbarui untuk memanggil *intent* yang membuka galeri gambar.

3.6.2 Pembuatan Pencatatan Data Pepaya

Langkah-langkah dalam pembuatan fitur pencatatan data papaya menggunakan *Room Database* mencakup:

Desain *Entitas* dan *DAO*

Pertama-tama, entitas (*entity*) dan *Data Access Object* (DAO) dirancang untuk merepresentasikan struktur data dan akses ke database. Entitas menggambarkan objek data yang akan disimpan, sedangkan DAO menyediakan antarmuka untuk mengakses data dari database.

Pembuatan *RoomDatabase*

RoomDatabase berperan sebagai lapisan abstraksi yang memungkinkan aplikasi berinteraksi dengan database secara efisien. Desain *RoomDatabase* mencakup pembuatan

kelas yang memperluas fungsionalitas *RoomDatabase*, dan di dalamnya, entitas dan versi database diidentifikasi.

Pembuatan *Repository*

Repository bertindak sebagai perantara antara sumber data (database) dan aplikasi. Dalam langkah ini, *repository* dibuat untuk menyediakan metode akses data yang diperlukan oleh aplikasi.

Pembuatan *ViewModel*

ViewModel bertanggung jawab untuk menyediakan data kepada UI dan mengelola status data. Pembuatan *ViewModel* melibatkan desain kelas *ViewModel* yang berinteraksi dengan *repository* untuk mendapatkan dan menyimpan data.

Integrasi dengan *Activity*

Seluruh sistem *RoomDatabase* diintegrasikan dengan activity aplikasi, memungkinkan pengguna untuk menyimpan dan mengelola data tentang pepaya yang telah diproses.

3.6.3 Pembuatan fitur Prediksi dan Pengingat

Dalam tahap pembuatan fitur prediksi dan pengingat, fokus utama adalah memberikan aplikasi kemampuan untuk secara otomatis memperbarui data dalam database berdasarkan hari yang ditentukan oleh penelitian dalam jurnal acuan. Selain itu, fitur pengingat menggunakan alarm/notifikasi yang memungkinkan pengguna untuk menentukan kapan notifikasi muncul sesuai preferensi mereka.

Pertama, untuk fitur prediksi, metode update data secara otomatis diimplementasikan dengan memanfaatkan penelitian dari jurnal acuan dan pengamatan langsung. Proses ini memastikan bahwa data yang disimpan dalam database tetap akurat dan relevan dengan kondisi terkini. Hasil prediksi kemudian dapat digunakan untuk memberikan informasi yang lebih canggih kepada pengguna tentang perkembangan kematangan papaya.

Berikutnya, fitur pengingat menggunakan alarm dan notifikasi untuk memberi tahu pengguna tentang aktivitas tertentu yang terkait dengan data papaya. Langkah-langkah rinci dari implementasi fitur pengingat adalah sebagai berikut:

Pembuatan *Channel* Notifikasi

Membuat *channel* notifikasi dengan menggunakan *NotificationChannel* untuk memberikan pengaturan terkait notifikasi, seperti deskripsi dan tingkat kepentingan (*importance*).

Pengaturan Alarm

Implementasi kode untuk menentukan waktu dan detail notifikasi. Pengguna dapat memasukkan informasi judul dan pesan notifikasi melalui antarmuka pengguna aplikasi.

Penjadwalan Notifikasi

Menggunakan *AlarmManager* untuk menetapkan waktu dan menjadwalkan notifikasi. Proses ini memastikan bahwa notifikasi muncul sesuai dengan preferensi pengguna.

Menanggapi Input Pengguna

Aplikasi memperbolehkan pengguna untuk memasukkan informasi notifikasi melalui antarmuka aplikasi. Tombol "*Submit*" diaktifkan untuk menjadwalkan notifikasi berdasarkan *input* pengguna.

Penanganan Penerima Notifikasi (*Notification Receiver*)

Implementasi *class Notification* sebagai *BroadcastReceiver* yang menangani penerimaan notifikasi. Ketika notifikasi diterima, aplikasi membangun notifikasi menggunakan *NotificationCompat* dan menampilkan notifikasi sesuai dengan informasi yang telah ditentukan oleh pengguna.

3.7 Pengujian Aplikasi

Pengujian aplikasi Android melibatkan serangkaian tes untuk memverifikasi keakuratan dan keandalan aplikasi. Proses ini bertujuan untuk mengidentifikasi potensi masalah dan memastikan bahwa aplikasi berjalan sesuai dengan harapan.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Pengumpulan Data

Data atau gambar pepaya dikumpulkan langsung di kebun petani pepaya di wilayah Sleman, Yogyakarta. Proses pengambilan gambar menggunakan kamera ponsel dengan resolusi 13MP. Teknik pengambilan gambar dilakukan dengan menempatkan kamera ponsel sekitar 10 cm dari buah pepaya yang tergantung di pohonnya. Gambar pepaya diambil dalam tiga tingkat kematangan yang berbeda: setengah matang, matang, dan mentah, untuk mencakup variasi kematangan buah yang komprehensif. Penelitian ini memastikan bahwa dataset yang terkumpul mencakup beragam variasi kematangan buah pepaya, sehingga cocok untuk pengembangan model klasifikasi atau analisis yang akurat terkait kematangan buah pepaya. Total dataset yang dikumpulkan adalah 315 data, yang telah dikelompokkan berdasarkan tingkat kematangannya. Pada Gambar 4. 1 dilakukan penentuan tingkat kematangan oleh petani dari kebun ken-ken *farm* di Sanggragan, Sleman, yang memiliki pengalaman sejak tahun 2019 (peniliaian petani dapat dilihat pada lampiran).

Gambar 4. 1 Dokumentasi penilaian petani

4.2 *Preprocessing*

Setelah proses pengumpulan data gambar pepaya selesai, langkah selanjutnya adalah mempersiapkan data untuk proses pelatihan model. Ini melibatkan beberapa tahap penting yang perlu dilakukan dengan cermat untuk memastikan keberhasilan pengembangan model. Tahap-tahap tersebut antara lain menginputkan data ke *Google Colab* dengan menghubungkannya ke *Google Drive* untuk menyimpan gambar-gambar yang telah dikumpulkan sebelumnya. Setelah terhubung, langkah selanjutnya adalah menentukan direktori tempat penyimpanan data di *Google Drive*. Setelah itu, data perlu dipisahkan menjadi tiga bagian utama: data *training*, data validasi, dan data *testing*. Setelah tahap pemisahan data, langkah terakhir dalam persiapan data adalah melakukan perubahan ukuran citra agar ukurannya seragam.

4.2.1 Pemisahan data *Training*, *Validation*, dan *Testing*

Pada tahap ini dilakukan pemisahkan dataset menjadi tiga bagian utama: data latih (*train*), data uji (*test*), dan data validasi (*validation*). Ini adalah tahap penting dalam pengembangan model, di mana data latih digunakan untuk melatih model, data uji digunakan untuk menguji performa model, dan data validasi digunakan untuk mengevaluasi model serta mencegah *overfitting*. Dengan pemisahan yang tepat, kita dapat memastikan bahwa model yang dikembangkan dapat bekerja dengan baik pada data yang belum pernah dilihat sebelumnya.

```
import random
import os
from shutil import copyfile

def train_val_test_split(source, train, val, test, train_ratio, val_ratio):
    total_size = len(os.listdir(source))
    train_size = int(train_ratio * total_size)
    val_size = int(val_ratio * total_size)
    test_size = total_size - train_size - val_size

    randomized = random.sample(os.listdir(source), total_size)
    train_files = randomized[:train_size]
    val_files = randomized[train_size:train_size + val_size]
    test_files = randomized[train_size + val_size:]

    for i in train_files:
        i_file = os.path.join(source, i)
        destination = os.path.join(train, i)
        copyfile(i_file, destination)

    for i in val_files:
        i_file = os.path.join(source, i)
        destination = os.path.join(val, i)
        copyfile(i_file, destination)
```

```
    for i in test_files:
        i_file = os.path.join(source, i)
        destination = os.path.join(test, i)
        copyfile(i_file, destination)

# Rasio untuk pembagian data
train_ratio = 0.7
val_ratio = 0.1
test_ratio = 0.2

# Direktori asal data
source_00 = matang_dir
source_01 = setengahmatang_dir
source_02 = mentah_dir

# Direktori tujuan untuk train, validation, dan test
train_00 = train_matang
val_00 = validation_matang
test_00 = test_matang

train_01 = train_setengahmatang
val_01 = validation_setengahmatang
test_01 = test_setengahmatang

train_02 = train_mentah
val_02 = validation_mentah
test_02 = test_mentah

# Memanggil fungsi untuk membagi data
train_val_test_split(source_00,  train_00,  val_00,  test_00,  train_ratio,
val_ratio)
train_val_test_split(source_01,  train_01,  val_01,  test_01,  train_ratio,
val_ratio)
train_val_test_split(source_02,  train_02,  val_02,  test_02,  train_ratio,
val_ratio)
```

Gambar 4. 2 Kode pembagian data

Pada Gambar 4. 2 proses pembagian data dilakukan dengan membagi dataset menjadi tiga bagian secara proporsional. Sebanyak 70% dari total data digunakan sebagai data latih untuk melatih model, 10% dialokasikan sebagai data validasi untuk mengevaluasi performa model selama proses pelatihan, dan 20% sisanya digunakan sebagai data pengujian untuk menguji performa model setelah proses pelatihan selesai. Dengan pembagian ini, model yang dikembangkan memiliki tingkat akurasi dan generalisasi yang optimal pada data yang belum pernah dilihat sebelumnya. Hasil dari proses pembagian data dapat dilihat pada Gambar 4. 3.

```
Jumlah materi matang : 104
Jumlah train matang : 72
Jumlah val matang : 10
Jumlah test matang : 22
Jumlah materi setengah matang : 102
Jumlah train setengah matang : 71
Jumlah val setengah matang : 10
Jumlah test setengah matang : 21
Jumlah materi mentah : 109
Jumlah train mentah : 76
Jumlah val mentah : 10
Jumlah test mentah : 23
```

Gambar 4. 3 Hasil pembagian data

4.2.2 Perubahan Ukuran Citra

Citra yang sudah dikelompokkan selanjutnya dilakukan perubahan ukuran citra sesuai dengan yang diinginkan (dalam hal ini, 32x32pixel) untuk mempercepat proses pembelajaran dalam model nantinya. Penggunaan gambar berukuran 32x32 pixel juga memungkinkan aplikasi untuk berjalan lebih efisien, karena memproses gambar dengan resolusi lebih rendah membutuhkan lebih sedikit daya komputasi dan memori saat dijalankan di perangkat. Ukuran batch juga ditentukan untuk digunakan dalam pelatihan (*batch_size* = 20) penentuan ukuran *batch* ini diambil berdasarkan hasil eksplorasi *hyperparameter* yang akan dijelaskan lebih lanjut pada sub bab 4.3, dimana *batch size* sebesar 20 memberikan keseimbangan terbaik antara kecepatan pelatihan dan performa model.

```
img_height, img_width = 32, 32
batch_size = 20

train_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size = (img_height, img_width),
    batch_size = batch_size
)
val_ds = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    image_size = (img_height, img_width),
    batch_size = batch_size
)
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    image_size = (img_height, img_width),
    batch_size = batch_size
)
```

Gambar 4. 4 Contoh kode program untuk penentuan ukuran gambar dan *batch*

Pada Gambar 4. 4 menggunakan *TensorFlow* untuk memuat dataset gambar yang telah dipisahkan sebelumnya menjadi data latih, data validasi, dan data uji. Langkah pertama yaitu

mendefinisikan ukuran gambar yang diinginkan, yaitu 32x32 piksel, serta ukuran *batch* yang akan digunakan dalam proses pelatihan. Selanjutnya, memuat dataset dari direktori latih, validasi, dan uji menggunakan fungsi *image_dataset_from_directory* dari *TensorFlow*. Fungsi ini secara otomatis membaca gambar dari direktori yang diberikan dan membaginya ke dalam *batch-batch* sesuai dengan ukuran *batch* yang telah ditentukan sebelumnya. Dataset yang dihasilkan dari masing-masing direktori merupakan objek *TensorFlow tf.data.Dataset*, yang telah siap untuk digunakan dalam proses pelatihan dan evaluasi model. Dengan memuat dataset ini, kita telah menyiapkan langkah awal yang penting dalam pengembangan dan pelatihan model untuk klasifikasi gambar pepaya berdasarkan tingkat kematangan.

4.3 Perancangan CNN

Pada langkah ini dimulai merancang arsitektur model untuk *Convolutional Neural Network* (CNN) yang akan digunakan dalam proses klasifikasi gambar. Model CNN didesain dengan mempertimbangkan struktur lapisan-lapisan konvolusional, penggunaan fungsi aktivasi, serta lapisan-lapisan lain seperti *MaxPooling* dan *Dense* untuk memproses data gambar secara efektif. Dalam perancangan ini, bentuk input ditentukan, lapisan-lapisan konvolusional ditambahkan, dan lapisan-lapisan tambahan disusun untuk memastikan model dapat memahami dan mengekstraksi fitur-fitur penting dari gambar. Setiap langkah dalam perancangan CNN dipertimbangkan dengan cermat untuk memastikan bahwa model yang dihasilkan memiliki kemampuan yang optimal dalam melakukan klasifikasi berbasis gambar. Explorasi *hyperparameter* adalah langkah penting dalam pengembangan model *Convolutional Neural Network* (CNN) untuk memastikan model mencapai performa terbaik.

Tabel 4. 1 Percobaan *hyperparameter*

Percobaan ke-	***Batch Size***	***Epochs***	**Arsitektur**	***Training Accuracy***	***Validation Accuracy***	***Test Accuracy***
1	20	20	• *Rescaling layer* • *Conv2D* (32 *filters*, *kernel size* 3x3,	1.00	0.8667	0.9242

			activation "*relu*") • *MaxPooling2D* • *Flatten* • *Dense* (128 *units*, *activation* "*relu*") • *Dense* (3 *units*)			
2	10	20	• *Rescaling layer* • *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*") • *MaxPooling2D* • *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*") • *MaxPooling2D* • *Flatten* • *Dense* (128 *units*, *activation* "*relu*") • *Dense* (3 *units*)	0.9635	0.9667	0.9545
3	10	10	• *Rescaling layer*	0.9498	0.9333	0.9545

			• *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*") • *MaxPooling2D* • *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*") • *MaxPooling2D* • *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*") • *MaxPooling2D* • *Flatten* • *Dense* (128 *units*, *activation* "*relu*") • *Dense* (3 *units*)			
4	20	20	*Rescaling layer* *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*") *MaxPooling2D* *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*")	0.9863	0.9000	0.9697

			MaxPooling2D *Conv2D* (32 *filters*, *kernel size* 3x3, *activation* "*relu*") *MaxPooling2D* *Flatten* *Dense* (128 *units*, *activation* "*relu*") *Dense* (3 *units*)			

Pada Tabel 4. 1, dilakukan beberapa percobaan untuk mengeksplorasi *hyperparameter* dalam pengembangan model CNN. Beberapa aspek yang diuji meliputi *batch size*, *epochs*, dan arsitektur model. Percobaan 1 dan 4 menggunakan *batch size* 20, sementara Percobaan 2 dan 3 menggunakan *batch size* 10. *Batch size* 20 memberikan keseimbangan antara kecepatan training dan performa model. Ukuran *batch* yang terlalu kecil dapat menyebabkan estimasi gradien yang lebih bising, sedangkan ukuran batch yang terlalu besar dapat memakan memori dan memperlambat konvergensi.

Dalam hal *epochs*, Percobaan 1, 2, dan 4 menggunakan 20 *epochs*, sedangkan Percobaan 3 menggunakan 10 *epochs*. Penggunaan 20 *epochs* memberikan cukup waktu bagi model untuk belajar dari data tanpa *overfitting*, sementara 10 *epochs* terlihat kurang cukup untuk mencapai performa optimal. Arsitektur model juga berperan penting dalam performa model. *Kernel size* 3x3 pada lapisan *Conv2D* adalah pilihan umum yang efektif untuk mendeteksi fitur lokal pada gambar, sementara fungsi aktivasi *ReLU* dipilih karena membantu model belajar pola *non-linear* dengan baik dan mencegah masalah *vanishing gradient*. Pemilihan Pemilihan jumlah filter (dalam hal ini, 32 filters) pada lapisan *Convolutional Neural Network* (CNN) didasarkan pada beberapa pertimbangan penting yang dapat memengaruhi kinerja dan kemampuan model dalam mengekstraksi fitur dari data gambar, jumlah filter yang kecil dapat menyebabkan *underfitting*, di mana model tidak mampu menangkap pola dari data dengan cukup baik. Sebaliknya, jumlah filter yang terlalu banyak bisa menyebabkan *overfitting*, di mana model terlalu rumit dan belajar *noise* dari data *training*. Penggunaan 32 *filters* memberikan keseimbangan yang baik, di mana model cukup kompleks untuk belajar fitur dari data namun tidak terlalu kompleks sehingga menghindari *overfitting*. Pemilihan ukuran *kernel* 3x3 didasarkan pada praktik umum dalam literatur yang menunjukkan bahwa *kernel* kecil dapat

menangkap fitur lokal dengan lebih baik. *Kernel* 3x3 juga lebih efisien dalam hal komputasi dibandingkan dengan kernel yang lebih besar seperti 5x5 atau 7x7. Jumlah lapisan konvolusi juga bervariasi, dengan Percobaan 1 memiliki 1 *Conv2D layer*, sementara Percobaan 2 dan Final memiliki 2 dan 3 *Conv2D layers*. Tambahan *Conv2D layers* pada Percobaan 2 dan Final membantu model mengekstraksi fitur yang lebih kompleks, yang meningkatkan akurasi pada data validasi dan tes.

Fully connected (*Dense*) *layers* dengan 128 units dan fungsi aktivasi *ReLU* adalah pilihan yang sesuai untuk menangkap hubungan *non-linear* dalam fitur yang diekstraksi oleh lapisan konvolusi. Jumlah 128 units pada *Dense layer* cukup besar untuk memungkinkan jaringan saraf memodelkan hubungan yang kompleks antara fitur yang telah diekstraksi. Fungsi aktivasi *ReLU* (*Rectified Linear Unit*) digunakan karena memiliki beberapa keuntungan: pertama, memperkenalkan *non-linearitas* yang memungkinkan jaringan saraf untuk belajar dari data yang tidak linier; kedua, mengatasi masalah *vanishing gradient* yang sering terjadi pada jaringan yang dalam, karena *ReLU* tidak mengalami saturasi seperti fungsi *sigmoid* atau *tanh* pada nilai besar atau kecil; ketiga, *ReLU* mempercepat konvergensi selama training dengan memungkinkan pembelajaran yang lebih cepat dan lebih efisien. Selain itu, rescaling dengan faktor 1./255 digunakan untuk menormalkan input data, yang membantu konvergensi model lebih cepat dan stabil.

```
# Define the input shape
input_shape = (None, 32, 32, 3)

# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=input_shape[1:]),
    tf.keras.layers.Conv2D(32, 3, activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(3)
])

# Build the model with the specified input shape
model.build(input_shape)

# Print the model summary
model.summary()
```

Gambar 4. 5 Kode pembuatan arsitektur model.

Pada arsitektur model Gambar 4. 5, merupakan hasil percobaan 4 atau terakhir yang digunakan untuk melakukan klasifikasi gambar. *Input* model ditentukan dengan dimensi (32, 32, 3), yang menunjukkan bahwa gambar yang dimasukkan memiliki tinggi dan lebar 32 piksel serta tiga saluran warna (RGB). Model dimulai dengan lapisan *Rescaling*, yang digunakan untuk menormalkan intensitas piksel dalam rentang 0 hingga 1. Kemudian, ditambahkan tiga lapisan konvolusional berturut-turut, tiga lapisan ini dipilih berdasarkan hasil eksplorasi *hyperparameter* sebelumnya dan diikuti oleh fungsi aktivasi *ReLU*, yang bertujuan untuk mengekstraksi fitur-fitur penting dari gambar. Fungsi aktivasi *ReLU* dipilih karena sifat non-linearnya yang membantu dalam menangkap kompleksitas data dan menghindari masalah gradien menghilang (*vanishing gradient*) yang sering terjadi pada fungsi aktivasi lain seperti *sigmoid* atau *tanh*. *ReLU* juga lebih efisien dalam komputasi karena operasinya sederhana. Setelah setiap lapisan konvolusional, digunakan lapisan *MaxPooling* untuk mengurangi dimensi fitur. Lapisan *Flatten* digunakan untuk mengubah tensor menjadi vektor satu dimensi, sehingga dapat diteruskan ke lapisan-lapisan berikutnya yang terdiri dari lapisan *Dense*. Lapisan *Dense* ini bertujuan untuk menghubungkan semua fitur yang diekstraksi menjadi representasi yang lebih ringkas. Terdapat satu lapisan *Dense* terakhir dengan tiga unit *neuron*, yang mewakili *output* kelas yang mungkin. Setelah model dirangkai dengan bentuk *input* yang ditentukan, dicetak ringkasan model yang memberikan detail tentang arsitektur dan jumlah parameter yang terlibat dalam model tersebut.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 32, 32, 3)         0

 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 32)          9248

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 32)          0
 g2D)

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 36291 (141.76 KB)
Trainable params: 36291 (141.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Gambar 4. 6 Arsitektur CNN.

Pada Gambar 4. 6 Model ini dirancang untuk tugas klasifikasi pada gambar dengan ukuran input 32x32 piksel dan 3 saluran warna (RGB). Arsitektur model terdiri dari beberapa lapisan yang dijelaskan sebagai berikut: Pertama, ada lapisan *Rescaling* yang mereskalakan gambar masukan ke dalam rentang [0,1]. Kemudian, diikuti oleh lapisan Konvolusi (*Conv2D*) dengan 32 filter/kernel berukuran 3x3, yang diaktifkan menggunakan fungsi aktivasi *ReLU*. Lapisan *Max Pooling* (*MaxPooling2D*) yang mengikuti mengurangi dimensi spasial setengahnya dengan ukuran 2x2. Proses ini diulangi dengan tambahan lapisan Konvolusi dan *Max Pooling* dua kali lagi. Setelah itu, hasil keluaran dari lapisan *Max Pooling* terakhir diproses melalui lapisan *Flatten* untuk meratakan *output* menjadi larik satu dimensi. Dilanjutkan dengan dua lapisan *Dense* (*Fully Connected*), masing-masing dengan 128 *neuron* dan aktivasi *ReLU*. Lapisan *Dense* terakhir memiliki 3 neuron yang merupakan keluaran dari model, diasumsikan sebagai probabilitas kelas yang dihasilkan oleh model untuk tugas klasifikasi. Jumlah total parameter, parameter yang dapat dilatih, dan parameter yang tidak dapat dilatih juga ditampilkan dalam ringkasan model. Totalnya, arsitektur model ini memiliki 36.291 parameter yang dapat dilatih.

4.4 Pelatihan Model

Setelah perancangan arsitektur selesai tahap selanjutnya akan dilakukan pelatihan model. Di sini, model dikompilasi dengan menggunakan *optimizer Adam*, yang merupakan *optimizer* yang sering digunakan dalam pelatihan jaringan saraf tiruan. Kelebihan *Adam* termasuk kemampuannya untuk menyesuaikan laju pembelajaran secara adaptif untuk setiap parameter individu, serta kemampuannya untuk mengatasi masalah gradien yang menghilang. Ini membuatnya menjadi pilihan yang kuat untuk berbagai jenis tugas pelatihan. Fungsi kerugian *Sparse Categorical Crossentropy* dipilih karena tugas yang diselesaikan adalah klasifikasi multi-kelas. Fungsi ini cocok untuk kasus seperti ini karena menangani label yang diwakili secara langsung sebagai bilangan bulat, yang sesuai dengan cara label diwakili dalam dataset. Pengaturan *from_logits=True* juga digunakan untuk mengindikasikan bahwa keluaran model belum melalui fungsi *softmax*, sehingga penghitungan dilakukan secara efisien oleh *TensorFlow*. Selain itu, metrik yang dipilih untuk dipantau selama pelatihan adalah akurasi ini memberikan gambaran langsung tentang seberapa baik model dapat memprediksi label kelas yang benar dari dataset validasi. Memantau akurasi selama pelatihan membantu dalam melacak kinerja model dan memastikan bahwa model berkembang dengan baik seiring waktu.

Selanjutnya, model dilatih dengan memanggil metode *fit()* pada dataset pelatihan *train_ds*. Dataset validasi *val_ds* digunakan untuk mengevaluasi kinerja model setiap *epoch*. Parameter *epochs* menunjukkan jumlah iterasi penuh melalui dataset pelatihan.

```
model.compile(
    optimizer="adam",
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits = True),
    metrics=['accuracy']
)

history = model.fit(
    train_ds,
    validation_data = val_ds,
    epochs = 20
)
```

Gambar 4. 7 Kode untuk melatih model.

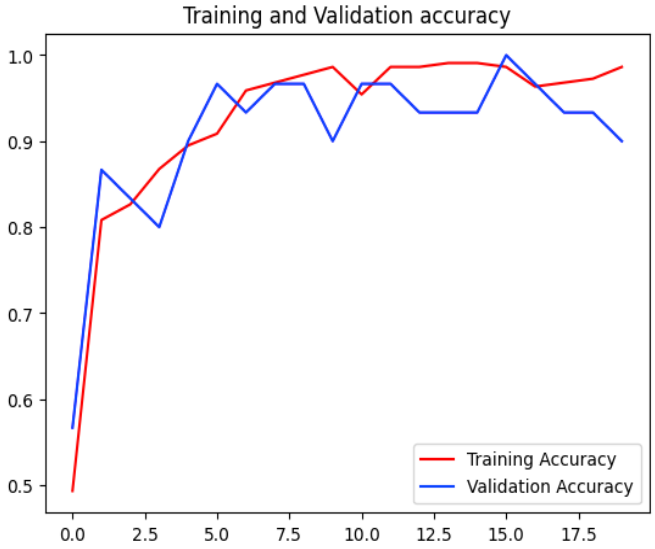
Dengan kode yang tertera pada Gambar 4. 7, model akan dikompilasi dan dilatih selama 20 *epoch* (penentuan *epoch* berdasarkan hasil eksplorasi *hyperparameter*) menggunakan *optimizer Adam*, dengan fungsi kerugian *Sparse Categorical Crossentropy*, dan akurasi sebagai metrik evaluasi.

```
Epoch 1/20
11/11 [==============================] - 21s 2s/step - loss: 1.0424 - accuracy: 0.4932 - val_loss: 0.9313 - val_accuracy: 0.5667
Epoch 2/20
11/11 [==============================] - 7s 159ms/step - loss: 0.6834 - accuracy: 0.8082 - val_loss: 0.4378 - val_accuracy: 0.8667
Epoch 3/20
11/11 [==============================] - 6s 246ms/step - loss: 0.3943 - accuracy: 0.8265 - val_loss: 0.4281 - val_accuracy: 0.8333
Epoch 4/20
11/11 [==============================] - 5s 160ms/step - loss: 0.2823 - accuracy: 0.8676 - val_loss: 0.4667 - val_accuracy: 0.8000
Epoch 5/20
11/11 [==============================] - 7s 209ms/step - loss: 0.2300 - accuracy: 0.8950 - val_loss: 0.3365 - val_accuracy: 0.9000
Epoch 6/20
11/11 [==============================] - 5s 158ms/step - loss: 0.1895 - accuracy: 0.9087 - val_loss: 0.1379 - val_accuracy: 0.9667
Epoch 7/20
11/11 [==============================] - 5s 155ms/step - loss: 0.1435 - accuracy: 0.9589 - val_loss: 0.1272 - val_accuracy: 0.9333
Epoch 8/20
11/11 [==============================] - 5s 156ms/step - loss: 0.1018 - accuracy: 0.9680 - val_loss: 0.1156 - val_accuracy: 0.9667
Epoch 9/20
11/11 [==============================] - 6s 223ms/step - loss: 0.0943 - accuracy: 0.9772 - val_loss: 0.0954 - val_accuracy: 0.9667
Epoch 10/20
11/11 [==============================] - 5s 162ms/step - loss: 0.0698 - accuracy: 0.9863 - val_loss: 0.2326 - val_accuracy: 0.9000
Epoch 11/20
11/11 [==============================] - 6s 220ms/step - loss: 0.0996 - accuracy: 0.9543 - val_loss: 0.1454 - val_accuracy: 0.9667
Epoch 12/20
11/11 [==============================] - 6s 221ms/step - loss: 0.0541 - accuracy: 0.9863 - val_loss: 0.0677 - val_accuracy: 0.9667
Epoch 13/20
11/11 [==============================] - 5s 169ms/step - loss: 0.0621 - accuracy: 0.9863 - val_loss: 0.1800 - val_accuracy: 0.9333
Epoch 14/20
11/11 [==============================] - 6s 220ms/step - loss: 0.0434 - accuracy: 0.9909 - val_loss: 0.1565 - val_accuracy: 0.9333
Epoch 15/20
11/11 [==============================] - 7s 236ms/step - loss: 0.0439 - accuracy: 0.9909 - val_loss: 0.1526 - val_accuracy: 0.9333
Epoch 16/20
11/11 [==============================] - 5s 163ms/step - loss: 0.0389 - accuracy: 0.9863 - val_loss: 0.0295 - val_accuracy: 1.0000
Epoch 17/20
11/11 [==============================] - 7s 159ms/step - loss: 0.0826 - accuracy: 0.9635 - val_loss: 0.0414 - val_accuracy: 0.9667
Epoch 18/20
11/11 [==============================] - 6s 264ms/step - loss: 0.0705 - accuracy: 0.9680 - val_loss: 0.1401 - val_accuracy: 0.9333
Epoch 19/20
11/11 [==============================] - 6s 222ms/step - loss: 0.0719 - accuracy: 0.9726 - val_loss: 0.1176 - val_accuracy: 0.9333
Epoch 20/20
11/11 [==============================] - 6s 224ms/step - loss: 0.0525 - accuracy: 0.9863 - val_loss: 0.4452 - val_accuracy: 0.9000
```

Gambar 4. 8 Hasil pelatihan model.

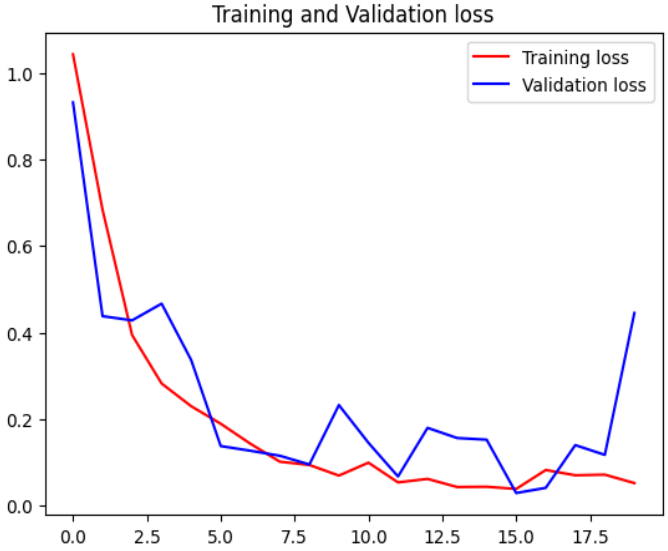
Terlihat pada Gambar 4. 8 selama proses pelatihan, model menunjukkan peningkatan kinerja dari *epoch* ke *epoch*. Pada *epoch* pertama, *loss* pada data pelatihan adalah 1.0424 dengan akurasi sekitar 49.32%, sedangkan pada data validasi, *loss* sebesar 0.9313 dengan akurasi sekitar 56.67%. Namun, pada *epoch* kedua, terjadi peningkatan yang signifikan, dengan *loss* pada data pelatihan menurun menjadi 0.6834 dan akurasi meningkat menjadi sekitar 80.82%, sedangkan pada data validasi, *loss* turun menjadi 0.4378 dengan akurasi sekitar 86.67%. Proses ini berlanjut hingga *epoch* ke-20, di mana model mencapai *loss* pada data pelatihan sebesar 0.0525 dengan akurasi sekitar 98.63%, dan *loss* pada data validasi sebesar 0.4452 dengan akurasi sekitar 90.00%.

Penurunan signifikan dalam *loss* dan peningkatan dalam akurasi menunjukkan bahwa model belajar dengan baik dari data pelatihan dan mampu melakukan generalisasi dengan baik pada data validasi. Meskipun terdapat sedikit fluktuasi dalam kinerja pada beberapa *epoch*, secara keseluruhan, model menunjukkan kemampuan yang baik dalam melakukan klasifikasi gambar sesuai dengan tujuannya.



Gambar 4. 9 Grafik akurasi pelatihan dan validasi

Dari Gambar 4. 9, terlihat bahwa baik akurasi pelatihan (garis merah) maupun akurasi validasi (garis biru) meningkat selama fase awal pelatihan. Meskipun akurasi pelatihan cenderung sedikit lebih tinggi daripada akurasi validasi sepanjang proses, hal ini menunjukkan tanda-tanda biasa dari *overfitting*. Namun, perbedaan tersebut tidak terlalu signifikan, menunjukkan bahwa model memiliki kemampuan yang baik untuk menggeneralisasi.



Gambar 4. 10 Grafik kerugian pelatihan dan validasi

Gambar 4. 10 menampilkan grafik kerugian pelatihan (*training loss*) dan kerugian validasi (*validation loss*) dari suatu model *machine learning* selama proses pelatihan. Garis merah

menunjukkan kerugian pelatihan, sementara garis biru melambangkan kerugian validasi. Kedua kurva tersebut menurun secara signifikan pada tahap awal pelatihan, mengindikasikan bahwa model dengan cepat mempelajari pola dari data. Setelah fase penurunan awal ini, baik kerugian pelatihan maupun validasi mengalami fluktuasi namun secara keseluruhan terus menurun, menunjukkan perbaikan bertahap dalam kemampuan prediksi model.

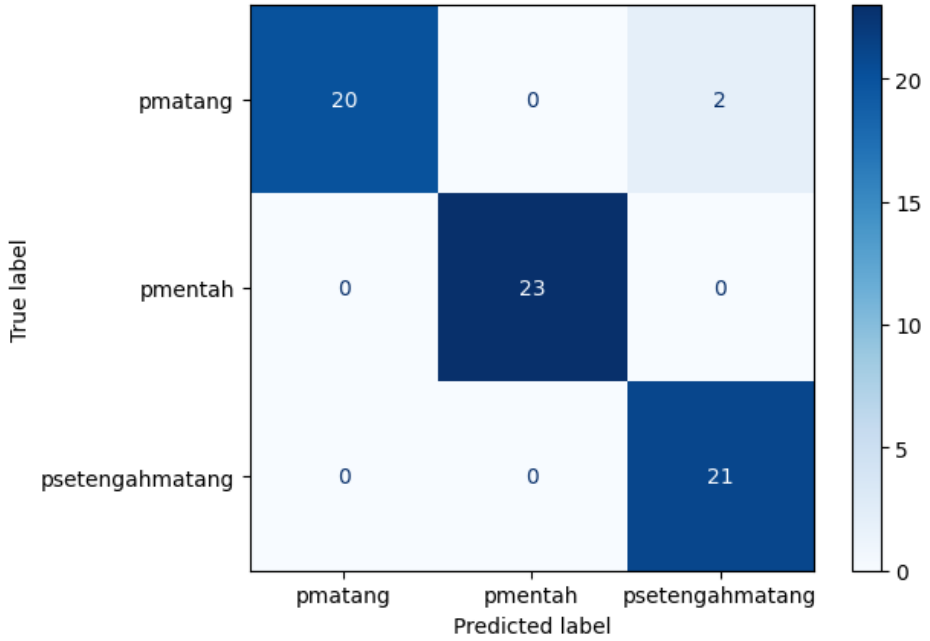
4.5 Pengujian Model

Setelah model selesai dilatih, langkah selanjutnya adalah menguji performanya menggunakan data *testing* yang belum pernah dilihat oleh model sebelumnya. Dalam kode yang terdapat pada Gambar 4. 11, *metode evaluate*() digunakan untuk mengevaluasi model menggunakan dataset *testing test_ds*. Hasil evaluasi tersebut menunjukkan bahwa model mencapai *loss* sebesar 0.0932 dan akurasi sebesar 96.97% pada data *testing*.

```
model.evaluate(test_ds)

#Hasil
4/4 [==============================] - 14s 7ms/step - loss: 0.0932 -
accuracy: 0.9697
[0.09320225566625595, 0.9696969985961914]
```

Gambar 4. 11 Kode untuk menguji model serta hasilnya

Ini berarti bahwa model mampu melakukan klasifikasi dengan baik pada data yang belum pernah dilihat sebelumnya, dengan tingkat akurasi yang cukup tinggi. Performa model ini diuji menggunakan dataset *testing* untuk memastikan bahwa model tidak hanya mampu mempelajari pola dari data pelatihan, tetapi juga mampu menggeneralisasi pengetahuannya ke data yang baru.



Gambar 4. 12 *Confusion matrix*

Pada Gambar 4. 12 terlihat bahwa *confusion matrix* yang dihasilkan dari model klasifikasi menunjukkan bahwa untuk kelas "matang" (pmatang), model berhasil mengklasifikasikan 20 sampel dengan benar, namun salah mengklasifikasikan 2 sampel sebagai "setengah matang" (psetengahmatang). Tidak ada sampel dari kelas "matang" yang salah diklasifikasikan sebagai "mentah" (pmentah). Untuk kelas "mentah" (pmentah), semua 23 sampel berhasil diklasifikasikan dengan benar tanpa ada kesalahan klasifikasi ke kelas lainnya. Demikian pula, untuk kelas "setengah matang" (psetengahmatang), semua 21 sampel diklasifikasikan dengan benar tanpa ada yang salah diklasifikasikan ke kelas lainnya. Berdasarkan *confusion matrix* ini, precision rata-rata untuk model adalah 0.97, dengan recall rata-rata juga sebesar 0.97, menunjukkan bahwa model memiliki kinerja yang baik dalam mengklasifikasikan ketiga kelas tersebut dengan tingkat akurasi sebesar 96.97%.

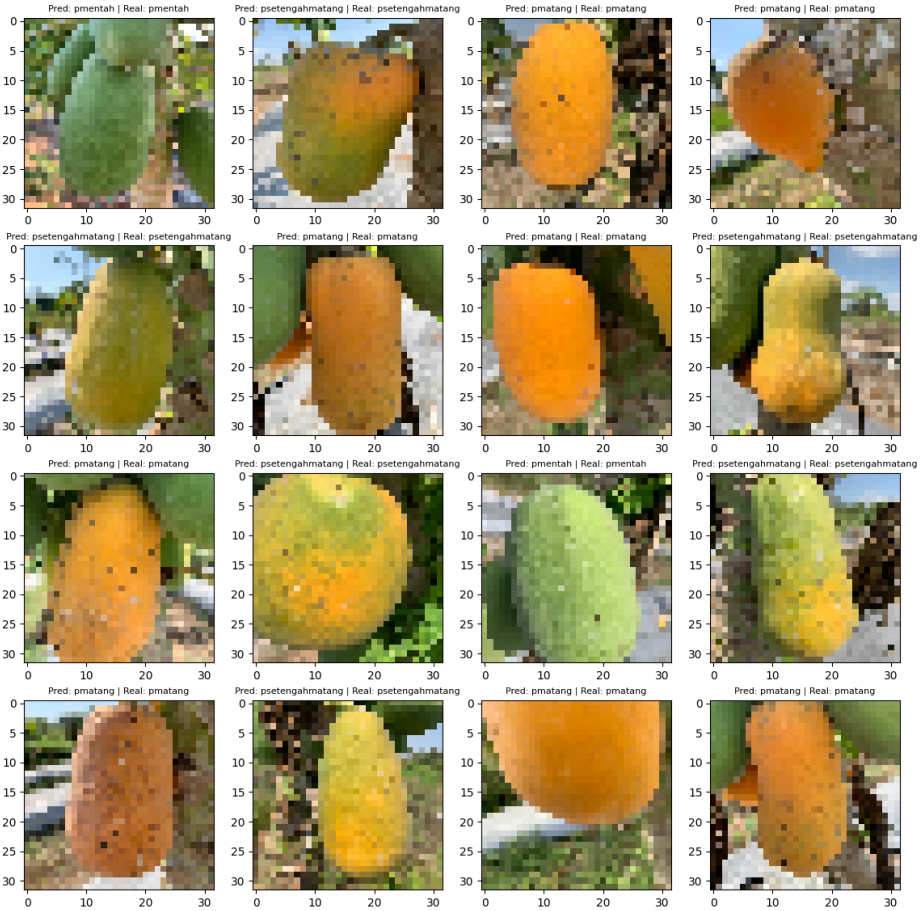
```
import numpy

plt.figure(figsize=(14,14))
for images, labels in test_ds.take(1):
  classifications = model(images)
  # print(classifications)

  for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    index = numpy.argmax(classifications[i])
```

```
    plt.title("Pred:  "  +  class_names[index]  +  "  |  Real:  "  +
class_names[labels[i]], fontsize=8)
```

Gambar 4. 13 Kode untuk menguji model kembali dengan menampilkan visualisasinya.

Kode pada Gambar 4. 13 bertujuan untuk memvisualisasikan hasil prediksi model pada beberapa contoh gambar dari dataset pengujian. Pertama, mengambil satu batch dari dataset pengujian menggunakan *test_ds.take*(1). Kemudian, model digunakan untuk melakukan prediksi pada *batch* tersebut dengan memanggil *model*(*images*). Hasil prediksi disimpan dalam *variabel classifications*.

Selanjutnya, melakukan iterasi melalui beberapa contoh gambar dalam *batch* tersebut (16 gambar dalam contoh ini). Untuk setiap gambar, ditampilkan gambar aslinya menggunakan *plt.imshow()*. Kemudian, kita menambahkan judul pada gambar yang menunjukkan prediksi yang dilakukan oleh model *(class_names[index])* dan label sebenarnya dari gambar tersebut *(class_names[labels[i]])*.

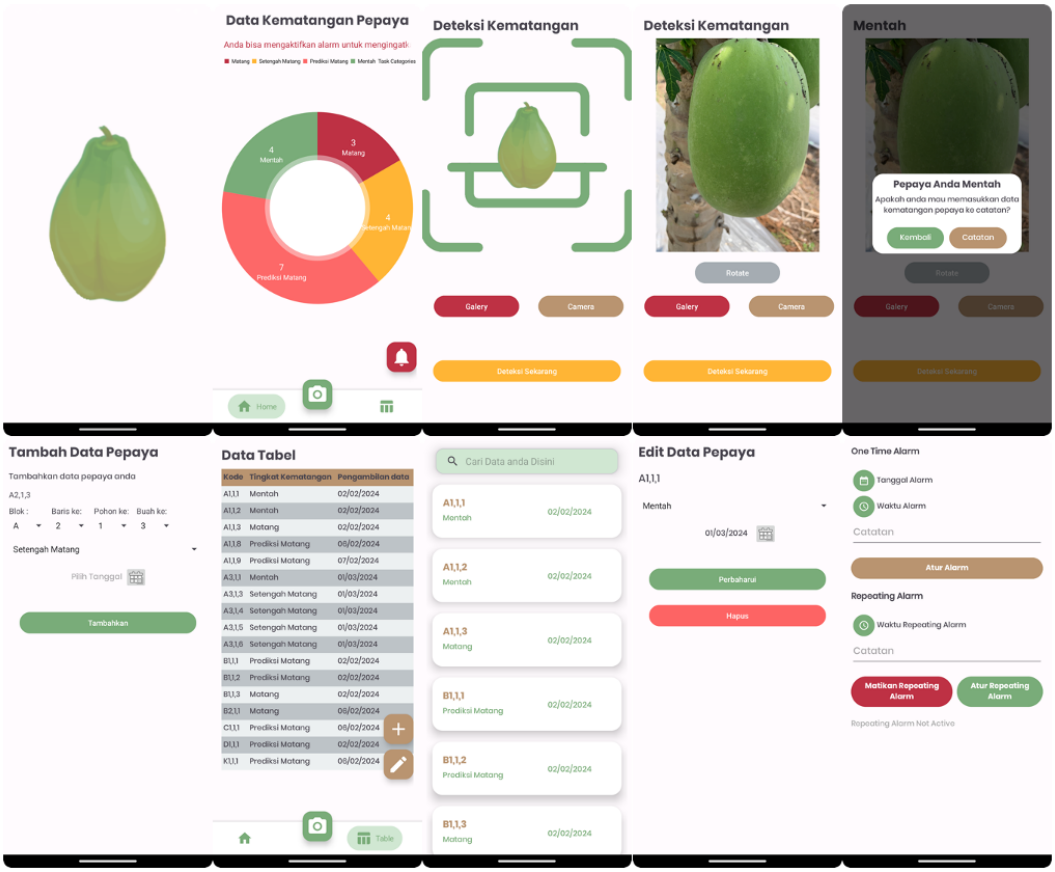


Gambar 4. 14 Hasil *testing* model dengan visualisasi.

Gambar 4. 14 menampilkan visualisasi hasil prediksi model terhadap data uji. Dalam gambar-gambar ini, setiap gambar dari data uji disajikan bersama dengan label prediksi yang diberikan oleh model serta label yang sebenarnya. Melalui visualisasi ini, kita dapat langsung melihat kemampuan model dalam mengklasifikasikan gambar-gambar pada data uji dengan akurasi. Dengan menggunakan 16 sampel data uji secara acak untuk pengujian, Gambar 4. 14 mengungkapkan hasil yang memuaskan, di mana semua prediksi yang dibuat oleh model terbukti benar.

4.6 Pembuatan Aplikasi

Setelah memastikan model telah dibuat dengan baik, langkah selanjutnya adalah memulai pembuatan aplikasi Android dengan mengimplementasikan rancangan antarmuka ke dalam aplikasi. Dengan mengacu pada Gambar 4. 15, telah diimplementasikan desain antarmuka ke dalam aplikasi Android.



Gambar 4. 15 Hasil Implementasi desain ke perangkat android.

Setelah menyelesaikan pembuatan antarmuka, langkah selanjutnya adalah memasuki tahap pengembangan fitur-fitur yang akan dimasukkan ke dalam aplikasi. Proses ini melibatkan

implementasi berbagai fungsionalitas yang telah direncanakan sebelumnya untuk memenuhi kebutuhan dan tujuan aplikasi.

4.6.1 Pemasangan Model

Tahap pertama dalam pemasangan model adalah melakukan konversi model ke dalam bentuk *TensorFlow Lite*. Proses ini penting karena *TensorFlow Lite* adalah *framework* yang dioptimalkan khusus untuk menjalankan model machine learning di perangkat *mobile*, seperti aplikasi Android.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open("modelbaru.tflite", 'wb') as f:
  f.write(tflite_model)
```

Gambar 4. 16 Kode untuk mengkonversi model.

Kode pada Gambar 4. 16 mengambil model yang telah dilatih (model) dalam format *Keras* dan mengonversinya ke dalam format *TensorFlow Lite* menggunakan *tf.lite.TFLiteConverter*. Hasil konversi disimpan ke dalam file "modelbaru.tflite". Hal ini memungkinkan untuk memuat model tersebut ke dalam aplikasi Android dan menjalankannya di perangkat *mobile* secara efisien.

Langkah selanjutnya adalah mengimpor model ke Android Studio. Setelah berhasil mengimpor model, langkah berikutnya adalah membuat fungsi agar model dapat mendeteksi suatu gambar yang dipilih dengan menekan tombol deteksi.

```
var tensorImage = TensorImage(DataType.FLOAT32)
            tensorImage.load(bitmap)

            tensorImage = imageProcessor.process(tensorImage)

            val model = Modelbaru.newInstance(this)


            val inputFeature0 = TensorBuffer.createFixedSize(intArrayOf(1,
32, 32, 3), DataType.FLOAT32)
            inputFeature0.loadBuffer(tensorImage.buffer)


            val outputs = model.process(inputFeature0)
            val                        outputFeature0                        =
outputs.outputFeature0AsTensorBuffer.floatArray

            var maxIdx = 0
```

```
outputFeature0.forEachIndexed{index, fl ->
    if (outputFeature0[maxIdx] < fl){
        maxIdx = index
    }
}
binding.judulDeteksi.text = maxIdx.toString()
when (maxIdx) {
    0 ->
    {
        binding.judulDeteksi.text = "Matang"
        showDialogBoxMentah("Pepaya Anda Matang")
    }
    1 ->
    {
        binding.judulDeteksi.text = "Mentah"
        showDialogBoxMentah("Pepaya Anda Mentah")
    }
    2 ->
    {
        binding.judulDeteksi.text = "SetengahMatang"
        showDialogBoxMentah("Pepaya Anda Setengah Matang")
    }
    else -> binding.judulDeteksi.text = "Unknown"
}

model.close()
```

Gambar 4. 17 Implementasi fungsi untuk mendeteksi gambar.

Kode pada Gambar 4. 17 merupakan implementasi fungsi untuk mendeteksi gambar menggunakan model yang telah diimpor ke dalam aplikasi Android. Pertama, gambar yang dipilih oleh pengguna dimuat ke dalam objek *tensorImage*, yang merupakan objek *TensorImage* dengan tipe data *FLOAT32*. Kemudian, gambar tersebut diproses menggunakan *imageProcessor* untuk persiapan *input* model.

Selanjutnya, model yang telah diimpor ke aplikasi Android diinisialisasi menggunakan Modelbaru.*newInstance(this). Input* yang telah diproses dimuat ke dalam objek *inputFeature0* dengan ukuran yang sesuai dengan kebutuhan model. Input dimasukkan ke dalam model dengan *model.process(inputFeature0),* dan hasilnya disimpan dalam objek *outputs*. *Output* model berupa *tensor buffer*, yang kemudian diakses dan diubah menjadi *array float*.

Setelah itu, dilakukan pengolahan pada *array float* untuk mendapatkan indeks nilai maksimum, yang menunjukkan hasil klasifikasi dari model. Nilai indeks ini digunakan untuk menentukan hasil deteksi berdasarkan kategori yang telah ditentukan, seperti "Matang", "Mentah", "SetengahMatang", atau "*Unknown*". Hasil deteksi tersebut kemudian ditampilkan pada elemen antarmuka pengguna dan ditampilkan dalam bentuk teks. Terakhir, setelah selesai digunakan, model ditutup dengan menggunakan *model.close()* untuk menghindari pemborosan sumber daya.

Tentunya, dalam menyediakan opsi bagi pengguna untuk memasukkan gambar ke dalam aplikasi, terdapat dua langkah utama yang harus dipertimbangkan. Pertama, pengguna dapat menggunakan kamera untuk mengambil gambar secara langsung. Kedua, mereka dapat memilih gambar dari galeri ponsel mereka.

```
Private fun startCamera() {
        currentImageUri = getImageUri(this)
        launcherIntentCamera.launch(currentImageUri)
    }

    private val launcherIntentCamera = registerForActivityResult(
        ActivityResultContracts.TakePicture()
    ) { isSuccess ->
        if (isSuccess) {
            showImage()
        }
    }

    private fun showImage() {
        currentImageUri?.let { uri ->
            Log.d("Image URI", "showImage: $uri")
            // Convert URI to Bitmap
            bitmap = getBitmapFromUri(uri, true) // Adjust isBackCamera
value based on your camera logic

            binding.imageViewDeteksi.setImageBitmap(bitmap)
            binding.buttonRotate.visibility = View.VISIBLE
        }
    }



binding.buttonGalery.setOnClickListener {
            var intent = Intent()
            intent.action = Intent.ACTION_GET_CONTENT
            intent.type = "image/*"
            startActivityForResult(intent,100)
        }

override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == 100) {
            var uri = data?.data
            bitmap =
MediaStore.Images.Media.getBitmap(this.contentResolver, uri)

            // Memeriksa rotasi gambar dan memutarnya jika diperlukan
            val rotation = getRotationFromExif(uri)
            if (rotation != 0) {
                val matrix = Matrix()
                matrix.postRotate(rotation.toFloat())
                bitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.width,
bitmap.height, matrix, true)
            }
```

```
            binding.imageViewDeteksi.setImageBitmap(bitmap)
            binding.buttonRotate.visibility = View.VISIBLE

        }
    }

    private fun getRotationFromExif(uri: Uri?): Int {
        val inputStream = uri?.let { contentResolver.openInputStream(it) }
        inputStream?.use {
            val exif = ExifInterface(it)
            return when
(exif.getAttributeInt(ExifInterface.TAG_ORIENTATION,
ExifInterface.ORIENTATION_NORMAL)) {
                ExifInterface.ORIENTATION_ROTATE_90 -> 90
                ExifInterface.ORIENTATION_ROTATE_180 -> 180
                ExifInterface.ORIENTATION_ROTATE_270 -> 270
                else -> 0
            }
        }
        return 0
    }
```

Gambar 4. 18 Kode untuk membuka kamera dan galeri.

Pada Gambar 4. 18 terlihat fungsi *startCamera*() digunakan untuk memulai kamera untuk pengambilan gambar. Di dalamnya, kita mendapatkan URI gambar saat ini dengan fungsi *getImageUri*() dan kemudian memulai aktivitas kamera menggunakan *launcherIntentCamera*. *LauncherIntentCamera* adalah objek yang mendaftarkan aktivitas untuk hasil pengambilan gambar. Ketika pengambilan gambar selesai, fungsi *showImage*() dipanggil untuk menampilkan gambar yang diambil. Fungsi *showImage*() digunakan untuk menampilkan gambar yang telah diambil dari kamera. Dalam kasus ini, gambar diambil dari URI yang diperoleh sebelumnya, diubah menjadi *bitmap*, dan ditampilkan di *imageViewDeteksi*. Tombol untuk memutar gambar juga ditampilkan.

Untuk memilih gambar dari galeri, pengguna dapat menekan tombol *buttonGalery*. Ini akan membuka galeri gambar menggunakan intent dengan aksi *ACTION_GET_CONTENT*. Setelah pengguna memilih gambar dari galeri, hasilnya diperoleh dalam *onActivityResult*(). Di dalamnya, gambar diambil dari URI, diperiksa rotasinya, dan kemudian ditampilkan di *imageViewDeteksi*. Fungsi *getRotationFromExif*() digunakan untuk mendapatkan informasi rotasi gambar dari metadata *Exif*. Ini berguna untuk memperbaiki rotasi gambar jika perlu sebelum menampilkannya.

4.6.2 Pembuatan Pencatatan Data Pepaya

Langkah selanjutnya adalah melakukan pengembangan fitur pencatatan pepaya dengan menggunakan *Room database*. Dalam konteks ini, Room database akan menjadi fondasi utama untuk menyimpan dan mengelola data terkait pepaya.

```
@Entity(tableName = "task")
data class Task(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="id")
    val id: Long = 0,

    @ColumnInfo(name="title")
    var title: String,

    @ColumnInfo(name="description")
    var description: String,

    @ColumnInfo(name="dueDate")
    var dueDate: Long,

    @ColumnInfo(name="creationTime")
    var creationTime: Long = System.currentTimeMillis()
)
```

Gambar 4. 19 Kode kelas Task.

Pada Gambar 4. 19 terlihat kelas *Task* adalah representasi data yang digunakan untuk menyimpan informasi tentang tugas dalam aplikasi. Dalam konteks database, kelas ini dikonfigurasi dengan anotasi yang mendefinisikan tabelnya sebagai "task". Properti id digunakan sebagai kunci utama dan akan secara otomatis di-*generate* saat *entitas* baru dibuat. Properti ini memungkinkan setiap tugas memiliki identitas unik. Properti *title* dan *description* adalah string yang mewakili judul dan deskripsi dari tugas tersebut. Properti *dueDate* adalah waktu saat tugas harus diselesaikan, disimpan dalam format waktu *long* untuk kemudahan pengolahan. Properti *creationTime* merepresentasikan waktu saat tugas dibuat, secara default diatur sebagai waktu sistem saat *entitas* dibuat, tetapi dapat diubah jika diperlukan. Dengan struktur ini, kelas *Task* menyediakan representasi yang jelas dan terstruktur untuk data tugas dalam aplikasi, memungkinkan manipulasi data dengan mudah dan efisien.

```
@Entity(tableName = "task")
data class Task(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="id")
    val id: Long = 0,

    @ColumnInfo(name="title")
```

```
    var title: String,

    @ColumnInfo(name="description")
    var description: String,

    @ColumnInfo(name="dueDate")
    var dueDate: Long,

    @ColumnInfo(name="creationTime")
    var creationTime: Long = System.currentTimeMillis()
)
```

Gambar 4. 20 Kode kelas DAO.

Setelah selesai membuat kelas untuk entitas dilanjutkan dengan membuat kelas *TaskDao*. Pada Gambar 4. 20 Kelas *TaskDao* adalah antarmuka yang digunakan untuk mengakses dan memanipulasi data tugas dalam aplikasi menggunakan metode akses data. Metode *getAllTasks*() digunakan untuk mengambil semua tugas yang ada dari database, diurutkan berdasarkan tanggal jatuh tempo secara menaik. Metode *insertTask*(*task: Task*) digunakan untuk menyisipkan tugas baru ke dalam database, dengan menangani konflik jika tugas dengan identitas yang sama sudah ada. Metode *getTaskById*(*taskId*: *Long*) digunakan untuk mengambil tugas berdasarkan ID yang diberikan. Metode *updateTask*(*task*: *Task*) digunakan untuk memperbarui tugas yang ada di dalam database. Metode *deleteTask*(*task*: *Task*) digunakan untuk menghapus tugas dari database. Metode *searchTasks*(*query*: *String*) digunakan untuk mencari tugas berdasarkan judul yang cocok dengan kueri yang diberikan. Terakhir, metode *updateDescriptionAfterFourDays*(*currentTime*: *Long*) digunakan untuk memperbarui deskripsi tugas yang masih dalam status "Setengah Matang" menjadi "Prediksi Matang" jika tugas tersebut telah ada selama empat hari atau lebih, sesuai dengan perhitungan waktu yang diberikan. Dengan demikian, kelas *TaskDao* menyediakan akses dan operasi *CRUD* (*Create*, Read*, Update*, *Delete*) yang diperlukan untuk manajemen data tugas dalam aplikasi.

```
@Database(entities = [Task::class], version = 2, exportSchema = false)
abstract class TaskDatabase : RoomDatabase() {
    abstract fun taskDao(): TaskDao

    companion object {
        @Volatile
        private var INSTANCE: TaskDatabase? = null

        @JvmStatic
        fun getDatabase(context: Context): TaskDatabase {
            if (INSTANCE == null) {
                synchronized(TaskDatabase::class.java) {
```

```
                    INSTANCE =
Room.databaseBuilder(context.applicationContext,
                        TaskDatabase::class.java, "task_database")
                        .fallbackToDestructiveMigration()
                        .build()

                    val taskDao = INSTANCE?.taskDao()
                    CoroutineScope(Dispatchers.IO).launch {

taskDao?.updateDescriptionAfterFourDays(System.currentTimeMillis())
                    }
                }
            }
            return INSTANCE as TaskDatabase
        }
    }
}
```

Gambar 4. 21 Kelas Database.

Selanjutnya pada Gambar 4. 21 dilakukan pembuatan kelas abstrak yang mewakili basis data *Room* untuk menyimpan dan mengelola tugas dalam aplikasi. Dengan menggunakan anotasi @Database, kelas ini dideklarasikan untuk mencakup entitas *Task* dan versi basis data saat ini. Metode abstrak taskDao() memberikan akses ke objek DAO yang diperlukan untuk berinteraksi dengan entitas *Task*.

Di dalam metode *getDatabase*(*context*: *Context*), sebuah instansi tunggal dari *TaskDatabase* dibuat dengan menerapkan pola *Singleton*. Jika instansi belum ada, instansi baru akan dibuat dengan menggunakan *Room.databaseBuilder*(), dengan mengonfigurasi basis data dengan nama "*task_database*" dan memberikan pembaruan ke versi yang lebih tinggi dengan *fallbackToDestructiveMigration*(), yang memungkinkan migrasi basis data tanpa menyebabkan kehilangan data. Selain itu, dalam blok sinkronisasi, sebuah tugas *CoroutineScope* diinisialisasi di mana *updateDescriptionAfterFourDays*() dari *TaskDao* dipanggil menggunakan *Dispatchers.IO* untuk memperbarui deskripsi tugas yang relevan setelah empat hari atau lebih. Dengan demikian, kelas *TaskDatabase* bertanggung jawab untuk inisialisasi dan pengelolaan koneksi ke basis data serta menyediakan akses ke objek DAO yang diperlukan untuk operasi database.

```
class TaskRepository(private val taskDao: TaskDao) {
    val allTasks: LiveData<List<Task>> = taskDao.getAllTasks()

    suspend fun insertTask(task: Task) {
        taskDao.insertTask(task)
    }

    fun getTaskById(taskId: Long): LiveData<Task> {
```

```
        return taskDao.getTaskById(taskId)
    }

    suspend fun updateTask(task: Task) {
        taskDao.updateTask(task)
    }

    suspend fun deleteTask(task: Task) {
        taskDao.deleteTask(task)
    }

    fun searchTasks(query: String): LiveData<List<Task>> {
        return taskDao.searchTasks("%$query%")
    }

}
```

Gambar 4. 22 Kode Repository.

Langkah selanjutnya yaitu membuat kelas manajemen data tugas yaitu *Repository*. Pada Gambar 4. 22 Kelas *TaskRepository* mempunyai tanggung jawab atas manajemen akses dan manipulasi data tugas dalam aplikasi. Dalam *repository* ini, terdapat beberapa metode yang memfasilitasi operasi dasar pada data tugas. Pertama, terdapat properti *allTasks* yang merupakan *LiveData* yang berisi daftar semua tugas. Metode *insertTask*(*task*: *Task*) dan *deleteTask*(*task*: *Task*) digunakan untuk menyisipkan dan menghapus tugas dari basis data secara asinkron, sementara *updateTask*(*task*: *Task*) bertugas memperbarui informasi tugas yang ada. Metode *getTaskById*(*taskId*: *Long*) digunakan untuk mendapatkan tugas berdasarkan ID yang diberikan, dan *searchTasks*(*query*: *String*) digunakan untuk mencari tugas berdasarkan kueri yang diberikan. Dengan demikian, kelas *TaskRepository* menyediakan abstraksi antarmuka yang bersih dan terorganisir untuk berinteraksi dengan basis data tugas, memisahkan logika akses data dari logika bisnis aplikasi.

```
class TaskViewModel(application: Application) :
AndroidViewModel(application) {
    private val repository: TaskRepository
    val allTasks: LiveData<List<Task>>

    init {
        val taskDao = TaskDatabase.getDatabase(application).taskDao()
        repository = TaskRepository(taskDao)
        allTasks = repository.allTasks
    }

    fun insertTask(task: Task) = viewModelScope.launch {
        repository.insertTask(task)
    }

    fun getTaskById(taskId: Long): LiveData<Task> {
```

```
        return repository.getTaskById(taskId)
    }

    fun updateTask(task: Task) = viewModelScope.launch {
        repository.updateTask(task)
    }

    fun deleteTask(task: Task) = viewModelScope.launch {
        repository.deleteTask(task)
    }

    fun searchTasks(query: String): LiveData<List<Task>> {
        return repository.searchTasks(query)
    }

}
```

Gambar 4. 23 Kode ViewModel.

Setelah membuat *Repository* dilanjutkan dengan membuat kelas penghubung antara UI dan *Repository* yang dapat dilihat pada Gambar 4. 23. Kelas *TaskViewModel* bertindak sebagai penghubung antara tampilan pengguna (UI) dan *TaskRepository*. Dalam kelas ini, properti *allTasks* bertindak sebagai *LiveData* yang mengamati perubahan pada daftar tugas. Melalui inisialisasi, kelas mengakses *instance TaskDao* dari *TaskDatabase* menggunakan *getDatabase*() dan *taskDao*(), kemudian meneruskannya ke *TaskRepository*. Metode-metode seperti *insertTask*(*task: Task*), *updateTask*(*task: Task*), dan *deleteTask*(*task:* Task) dipanggil dari *repository* dengan menggunakan *viewModelScope.launch* untuk menjalankan operasi secara asinkron. Metode *getTaskById*(*taskId*: *Long*) dan *searchTasks*(*query*: *String*) memfasilitasi pencarian dan pengambilan tugas berdasarkan ID atau kueri yang diberikan. Dengan demikian, kelas *TaskViewModel* membantu memisahkan logika tampilan dari logika akses data, memastikan bahwa tampilan dapat bereaksi secara dinamis terhadap perubahan data dengan efisien.

Langkah selanjutnya yaitu adalah mengintegrasikan *activity* yang telah di buat sebelumnya dengan *ViewModel*. Pada *activity* tersebut, dipanggul fungsi-fungsi yang sudah dibuat dalam *ViewModel* untuk berinteraksi Contoh pada Gambar 4. 24.

```
//Contoh pemanggilan fungsi untuk menambahkan catatan baru
val task = Task(title = title, description = description, dueDate = dueDate)
                        taskViewModel.insertTask(task)

//Contoh pemanggilan fungsi untuk menghapus tugas
taskViewModel.deleteTask(originalTask)
        finish()

//Contoh pemanggilan fungsi untuk memperbarui tugas
```

```
taskViewModel.updateTask(originalTask)
            finish()
```

Gambar 4. 24 Contoh kode pemanggilan fungsi ViewModel.

4.6.3 Pembuatan fitur Prediksi dan Pengingat

Langkah selanjutnya dalam pengembangan aplikasi adalah implementasi fitur pengingat (*reminder*) dan fitur prediksi kematangan pepaya. Implementasi fitur pengingat dimulai dengan membuat kelas *AlarmReceiver* berfungsi sebagai penerima alarm yang akan menangani pengiriman notifikasi pada waktu yang ditentukan.

```
override fun onReceive(context: Context, intent: Intent) {
        val type = intent.getStringExtra(EXTRA_TYPE)
        val message = intent.getStringExtra(EXTRA_MESSAGE)

        val title = if (type.equals(TYPE_ONE_TIME, ignoreCase = true))
TYPE_ONE_TIME else TYPE_REPEATING
        val notifId = if (type.equals(TYPE_ONE_TIME, ignoreCase = true))
ID_ONETIME else ID_REPEATING

        if (message != null) {
            showAlarmNotification(context, title, message, notifId)
        }

    }
```

Gambar 4. 25 Contoh *onReceive*.

Pada kelas *AlarmReceive* terdapat metode *onReceive*() pada Gambar 4. 25 yang akan dieksekusi saat alarm dipicu untuk menampilkan notifikasi sesuai dengan pesan yang telah diterima. Alarm dapat diatur sebagai alarm satu kali atau alarm berulang, tergantung pada jenis alarm yang diberikan. Dua jenis alarm ini akan diidentifikasi menggunakan konstanta yang telah ditentukan.

```
fun setOneTimeAlarm(
        context: Context,
        type: String,
        date: String,
        time: String,
        message: String
    ) {
        // Validasi inputan date dan time terlebih dahulu
        if (isDateInvalid(date, DATE_FORMAT) || isDateInvalid(time,
TIME_FORMAT)) return

        val alarmManager = context.getSystemService(Context.ALARM_SERVICE)
as AlarmManager
        val intent = Intent(context, AlarmReceiver::class.java)
```

```
        intent.putExtra(EXTRA_MESSAGE, message)
        intent.putExtra(EXTRA_TYPE, type)

        val requestCode = generateUniqueRequestCode() // Generate a unique
requestCode
        Log.e("ONE TIME", "$date $time - RequestCode: $requestCode")

        val dateArray = date.split("-").toTypedArray()
        val timeArray = time.split(":").toTypedArray()

        val calendar = Calendar.getInstance()
        calendar.set(Calendar.YEAR, Integer.parseInt(dateArray[0]))
        calendar.set(Calendar.MONTH, Integer.parseInt(dateArray[1]) - 1)
        calendar.set(Calendar.DAY_OF_MONTH, Integer.parseInt(dateArray[2]))
        calendar.set(Calendar.HOUR_OF_DAY, Integer.parseInt(timeArray[0]))
        calendar.set(Calendar.MINUTE, Integer.parseInt(timeArray[1]))
        calendar.set(Calendar.SECOND, 0)

        val pendingIntent = PendingIntent.getBroadcast(
            context,
            requestCode,
            intent,
            PendingIntent.FLAG_UPDATE_CURRENT                              or
PendingIntent.FLAG_IMMUTABLE
        )
        alarmManager.set(AlarmManager.RTC_WAKEUP,    calendar.timeInMillis,
pendingIntent)

        Toast.makeText(context,     "One     time     alarm     set     up",
Toast.LENGTH_SHORT).show()
    }
```

Gambar 4. 26 Kode setOneTimeAlarm.

Fungsi *setOneTimeAlarm* pada Gambar 4. 26 digunakan untuk menetapkan alarm satu kali di aplikasi Android. Pertama, inputan tanggal (*date*) dan waktu (*time*) divalidasi untuk memastikan format yang benar menggunakan fungsi *isDateInvalid*. Jika format tidak valid, fungsi berhenti dan alarm tidak ditetapkan. Kemudian, *alarmManager* diinisialisasi dengan layanan *AlarmManager* yang diperoleh dari *Context*. *Intent* disiapkan untuk memulai *AlarmReceiver*, dan pesan serta tipe *alarm* dimasukkan sebagai ekstra dalam *intent*. Sebuah kode permintaan unik dihasilkan dengan fungsi *generateUniqueRequestCode*, dan informasi ini dicatat untuk keperluan *debugging*.

Tanggal dan waktu yang diterima dipisahkan menjadi *array*, lalu sebuah objek *Calendar* dibuat dan diatur dengan nilai-nilai yang sesuai. *PendingIntent* kemudian dibuat menggunakan *PendingIntent.getBroadcast*, dengan kode permintaan yang unik, *intent*, dan bendera yang sesuai. Akhirnya, alarm diatur menggunakan *alarmManager.set*, yang akan memicu alarm pada waktu yang telah ditentukan. Sebuah pesan *toast* ditampilkan untuk memberi tahu pengguna bahwa alarm satu kali telah diatur.

```
fun setRepeatingAlarm(context: Context, type: String, time: String, message:
String) {

        // Validasi inputan waktu terlebih dahulu
        if (isDateInvalid(time, TIME_FORMAT)) return

        val alarmManager = context.getSystemService(Context.ALARM_SERVICE)
as AlarmManager
        val intent = Intent(context, AlarmReceiver::class.java)
        intent.putExtra(EXTRA_MESSAGE, message)
        intent.putExtra(EXTRA_TYPE, type)

        val timeArray = time.split(":".toRegex()).dropLastWhile {
it.isEmpty() }.toTypedArray()

        val calendar = Calendar.getInstance()
        calendar.set(Calendar.HOUR_OF_DAY, Integer.parseInt(timeArray[0]))
        calendar.set(Calendar.MINUTE, Integer.parseInt(timeArray[1]))
        calendar.set(Calendar.SECOND, 0)

        val pendingIntent = PendingIntent.getBroadcast(
            context,
            ID_REPEATING,
            intent,
            PendingIntent.FLAG_IMMUTABLE
        )
        alarmManager.setInexactRepeating(
            AlarmManager.RTC_WAKEUP,
            calendar.timeInMillis,
            4 * AlarmManager.INTERVAL_DAY,
            pendingIntent
        )

        Toast.makeText(context, "Repeating alarm set up",
Toast.LENGTH_SHORT).show()
    }
```

Gambar 4. 27 Kode fungsi *setRepeatingAlarm*.

Fungsi *setRepeatingAlarm* yang disajikan pada Gambar 4. 27 adalah bagian dari kelas *AlarmReceiver* yang bertujuan untuk menetapkan alarm berulang. Pertama, inputan waktu (*time*) divalidasi untuk memastikan format yang benar menggunakan fungsi *isDateInvalid*. Jika format tidak valid, fungsi berhenti dan alarm tidak ditetapkan. Selanjutnya, layanan *AlarmManager* diinisialisasi dari *Context*, dan sebuah intent dipersiapkan untuk memulai *AlarmReceiver*, dengan menyertakan pesan dan tipe alarm sebagai ekstra dalam intent.

Waktu yang diterima dipisahkan menjadi *array*, dan sebuah objek *Calendar* dibuat dan diatur dengan jam, menit, dan detik yang sesuai. *PendingIntent* kemudian dibuat menggunakan *PendingIntent.getBroadcast*, dengan kode permintaan yang telah ditentukan (*ID_REPEATING*), *intent*, dan bendera yang sesuai.

Alarm diatur sebagai ulang menggunakan *alarmManager.setInexactRepeating*, yang akan memicu alarm pada waktu yang telah ditentukan, dengan interval ulang sebesar 4 hari. Sebuah pesan toast ditampilkan untuk memberi tahu pengguna bahwa alarm berulang telah diatur. Metode *setInexactRepeating* dipilih untuk efisiensi daya dan pengelolaan sumber daya yang lebih baik pada perangkat.

```
fun cancelAlarm(context: Context, type: String) {
        val alarmManager = context.getSystemService(Context.ALARM_SERVICE)
as AlarmManager
        val intent = Intent(context, AlarmReceiver::class.java)
        val requestCode =
            if (type.equals(TYPE_ONE_TIME, ignoreCase = true)) ID_ONETIME
else ID_REPEATING
        val pendingIntent = PendingIntent.getBroadcast(
            context,
            requestCode,
            intent,
            PendingIntent.FLAG_NO_CREATE or PendingIntent.FLAG_IMMUTABLE
        )

        if (pendingIntent != null) {
            pendingIntent.cancel()
            alarmManager.cancel(pendingIntent)
            Toast.makeText(context,    "Repeating    alarm    dibatalkan",
Toast.LENGTH_SHORT).show()
        }
    }
```

Gambar 4. 28 Contoh metode *cancelAlarm*.

Kelas *AlarmReceiver* juga menyediakan metode *cancelAlarm*() pada Gambar 4. 28 untuk membatalkan alarm yang telah diatur sebelumnya. Metode ini membatalkan *PendingIntent* yang terkait dengan alarm yang akan dibatalkan.

```
private fun showAlarmNotification(context: Context, title: String, message:
String, notifId: Int) {
        val channelId = "Channel_1"
        val channelName = "AlarmManager channel"

        val                    notificationManagerCompat                    =
context.getSystemService(Context.NOTIFICATION_SERVICE)                    as
NotificationManager
        val                           alarmSound                            =
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION)
        val builder = NotificationCompat.Builder(context, channelId)
            .setSmallIcon(R.drawable.baseline_notifications_24)
            .setContentTitle(title)
            .setContentText(message)
            .setColor(ContextCompat.getColor(context,
android.R.color.transparent))
            .setVibrate(longArrayOf(1000, 1000, 1000, 1000, 1000))
```

```
            .setSound(alarmSound)

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(channelId,
                channelName,
                NotificationManager.IMPORTANCE_DEFAULT)

            channel.enableVibration(true)
            channel.vibrationPattern = longArrayOf(1000, 1000, 1000, 1000,
1000)

            builder.setChannelId(channelId)

            notificationManagerCompat.createNotificationChannel(channel)
        }
        val notification = builder.build()

        notificationManagerCompat.notify(notifId, notification)
    }
```

Gambar 4. 29 Contoh fungsi *showAlarmNotification*.

Terahir pada kelas *AlarmReceiver* terdapat fungsi *showAlarmNotification* digunakan untuk menampilkan notifikasi alarm dalam aplikasi Android. Pada Gambar 4. 29 sebuah *channel ID* dan nama channel ditentukan untuk notifikasi. Kemudian, layanan *NotificationManager* diinisialisasi dari *Context*, dan *default alarm sound* diambil menggunakan *RingtoneManager*. Sebuah objek *NotificationCompat.Builder* dibuat dengan menggunakan channel ID yang telah ditentukan, dan berbagai atribut notifikasi ditetapkan, seperti ikon kecil, judul, isi pesan, warna, getaran, dan suara. Jika versi Android yang digunakan adalah 8.0 atau yang lebih baru, sebuah channel notifikasi dibuat dengan pengaturan getaran yang sesuai. Kemudian, *builder* diberi *channel* ID yang sesuai dan *channel* notifikasi dibuat menggunakan *notificationManagerCompat.createNotificationChannel*(*channel*). Notifikasi akhirnya dibangun menggunakan konfigurasi dari builder, dan ditampilkan menggunakan *notificationManagerCompat.notify* dengan ID notifikasi yang diberikan (*notifId*). Ini akan menampilkan notifikasi alarm kepada pengguna dengan judul, pesan, getaran, dan suara yang sesuai.

Langkah selanjutnya adalah menerapkan fungsi yang telah dibuat ke dalam aktivitas dengan memanggilnya. Pada contoh kode yang terdapat dalam Gambar 4. 30 , terlihat penggunaan fungsi *setOneTimeAlarm*. Fungsi ini akan digunakan untuk mengatur alarm satu kali.

```
R.id.btn_set_once_alarm -> {
                val onceDate = binding?.tvOnceDate?.text.toString()
```

```
                val onceTime = binding?.tvOnceTime?.text.toString()
                val onceMessage = binding?.edtOnceMessage?.text.toString()

                alarmReceiver.setOneTimeAlarm(this,
AlarmReceiver.TYPE_ONE_TIME,
                    onceDate,
                    onceTime,
                    onceMessage)
            }
```

Gambar 4. 30 Contoh pemanggilan fungsi *setOneTimeAlarm*.

Selanjutnya masuk pada implementasi fitur prediksi kematangan pepaya yang dimulai dengan menerima input dari pengguna saat mereka menambahkan catatan pepaya. Saat pengguna menekan tombol untuk menambahkan catatan, aplikasi akan menerima data yang telah diisi sebelumnya oleh pengguna, seperti kode blok, klasifikasi, dan waktu. Jika pepaya diklasifikasikan sebagai "Setengah Matang", database akan otomatis mengubah status pepaya menjadi "Prediksi Matang" untuk periode selama 4 hari ke depan, sesuai dengan acuan dari jurnal yang dijadikan referensi. Berdasarkan hasil wawancara dengan petani dari kebun Ken-Ken *Farm*, pepaya biasanya dipetik saat mencapai kematangan setengah matang. Oleh karena itu, sistem ini hanya dapat memprediksi pepaya yang akan dikonsumsi langsung, sementara pepaya yang akan didistribusikan kepada pengepul akan dipetik saat mencapai kematangan setengah matang.

```
……
@Query("UPDATE task SET description = 'Prediksi Matang' WHERE description =
'Setengah Matang' AND :currentTime - creationTime >= 345600000")
    suspend fun updateDescriptionAfterFourDays(currentTime: Long)
……
```

Gambar 4. 31 Potongan kode untuk memperbarui database.

Implementasi fitur ini dimulai dengan menggunakan kode pada Gambar 4. 31 yang terletak pada kelas *TaskDao*. *Query* yang digunakan akan memperbarui deskripsi pepaya menjadi "Prediksi Matang" dari "Setengah Matang" setelah 4 hari berlalu sejak catatan pepaya ditambahkan. Selanjutnya, alarm akan diatur untuk mengingatkan pengguna memeriksa kebun, karena pepaya diperkirakan akan matang dalam 4 hari ke depan.

```
……
// Check if the description is "Setengah Matang" to set the alarm
                        if (description == "Setengah Matang") {
                            setAlarmForSetengahMatang(task)
                        }
```

```
……
……
val alarmTime = task.dueDate + (4 * 24 * 60 * 60 * 1000) // 4 days later
        alarmManager.setExact(
            AlarmManager.RTC_WAKEUP,
            alarmTime,
            pendingIntent
        )
……
```

Gambar 4. 32 Potongan kode untuk mengatur alarm prediksi.

Dalam potongan kode pada Gambar 4. 32, waktu alarm dihitung dengan menambahkan 4 hari (dalam milidetik) ke waktu saat ini. Dengan menggunakan AlarmManager, alarm akan diatur untuk memberikan pengguna pemberitahuan saat yang tepat untuk memeriksa kebun, sesuai dengan prediksi kematangan pepaya.

Dengan langkah-langkah ini, aplikasi diharapkan akan memberikan prediksi kematangan pepaya yang akurat dan pengingat yang sesuai, meningkatkan efisiensi pengelolaan kebun pepaya oleh pengguna.

4.7 Pengujian Akhir

Pada sub bab ini, hasil pengujian aplikasi pendeteksi tingkat kematangan buah pepaya menggunakan metode *Blackbox* Testing akan disajikan. Pengujian dilakukan dengan menginstal aplikasi pada beberapa merek HP Android yang berbeda dan mengevaluasi performa serta fungsi dari setiap fitur aplikasi.

Skenario pengujian dibuat untuk mencakup semua fitur utama yang ada dalam aplikasi, termasuk:

1. Pendeketeksi kematangan buah pepaya menggunakan kamera.
2. Pendeketeksi kematangan buah pepaya menggunakan galery.
3. Pencatatan data pepaya.
4. Prediksi waktu petik berdasarkan data yang dimasukkan.
5. Penyimpanan dan akses data pepaya yang telah dicatat sebelumnya.
6. Notifikasi aplikasi untuk pemberitahuan bagi pengguna.

Tabel 4. 2 Tabel pengujian aplikasi pada perangkat android

No.	Perangkat Uji	Fitur yang Diuji	Hasil Pengujian
1.	Infinix Smart 8	Deteksi kematangan (Kamera)	Berhasil (gambar ter-*rotate* 90 drajat)

		Deteksi kematangan (Galeri) Pencatatan data Pepaya Prediksi waktu petik Penyimpanan data pepaya Notifikasi	Berhasil Berhasil Berhasil Tersimpan Tampil
2.	Samsung A03s	Deteksi kematangan (Kamera) Deteksi kematangan (Galeri) Pencatatan data Pepaya Prediksi waktu petik Penyimpanan data pepaya Notifikasi	Berhasil Berhasil Berhasil Berhasil Tersimpan Tampil
3.	Xiaomi Poco X3 NFC	Deteksi kematangan (Kamera) Deteksi kematangan (Galeri) Pencatatan data Pepaya Prediksi waktu petik Penyimpanan data pepaya Notifikasi	*Force close* (Teratasi) *Force close* (Teratasi) Berhasil Berhasil Tersimpan Tampil
4.	Vivo Z1 pro	Deteksi kematangan (Kamera) Deteksi kematangan (Galeri) Pencatatan data Pepaya Prediksi waktu petik Penyimpanan data pepaya Notifikasi	Berhasil Berhasil Berhasil Berhasil Tersimpan Tampil
5.	Xiaomi Redmi Note 7	Deteksi kematangan (Kamera) Deteksi kematangan (Galeri) Pencatatan data Pepaya Prediksi waktu petik Penyimpanan data pepaya Notifikasi	*Force close* (Teratasi) *Force close* (Teratasi) Berhasil Berhasil Tersimpan Tampil
6.	Xiaomi Redmi 10	Deteksi kematangan (Kamera) Deteksi kematangan (Galeri) Pencatatan data Pepaya	*Force close* (Teratasi) *Force close* (Teratasi) Berhasil

		Prediksi waktu petik Penyimpanan data pepaya Notifikasi	Berhasil Tersimpan Tampil

Tabel Tabel 4. 2 menyajikan hasil pengujian aplikasi pendeteksi tingkat kematangan buah pepaya menggunakan metode *Blackbox Testing* pada beberapa merek HP Android yang berbeda. Pengujian dilakukan untuk mengevaluasi performa dan fungsi dari setiap fitur aplikasi, termasuk deteksi kematangan menggunakan kamera dan galeri, pencatatan data pepaya, prediksi waktu petik, penyimpanan dan akses data pepaya, serta notifikasi aplikasi. Hasil pengujian menunjukkan bahwa sebagian besar fitur berhasil dijalankan pada beberapa perangkat. Namun, pada 3 perangkat, saat membuka halaman deteksi, terjadi *force close* karena warna yang diterapkan sebagai *backgroundTint* tidak dapat dikonversi menjadi *ComplexColor* yang diperlukan oleh sistem. Sebagai solusi, dilakukan perbaikan pada halaman deteksi dengan mengganti warna pada *backgroundTint* agar dapat dikonversi oleh sistem.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada penelitian ini dapat diperoleh kesimpulan sebagai berikut:

a. Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa pembuatan model untuk mengklasifikasikan tingkat kematangan buah papaya menggunakan metode *Convolutional Neural Network* (CNN) merupakan suatu tugas yang dapat dilakukan dengan tingkat akurasi yang tinggi. Langkah-langkah yang dilakukan dalam penelitian ini meliputi pengumpulan data gambar buah papaya dengan tiga tingkat kematangan yang berbeda, yaitu setengah matang, matang, dan mentah. Data tersebut kemudian diproses melalui tahap *preprocessing*, di mana dilakukan pemisahan data menjadi data *training*, data validasi, dan data *testing*. Selanjutnya, dilakukan perubahan ukuran citra menjadi 32x32 piksel untuk mempercepat proses pembelajaran dan efisiensi aplikasi. Perancangan arsitektur CNN menjadi langkah penting berikutnya, di mana dilakukan eksplorasi *hyperparameter* untuk memperoleh model yang optimal. Beberapa percobaan dilakukan dengan variasi *batch size*, *epochs*, dan arsitektur model, yang akhirnya menghasilkan model dengan akurasi yang terbaik. Setelah proses perancangan, dilakukan pelatihan model. Model berhasil dilatih dengan baik, menunjukkan peningkatan kinerja dari *epoch* ke *epoch*. Pengujian model dilakukan menggunakan data testing yang belum pernah dilihat sebelumnya, dan hasilnya menunjukkan bahwa model mampu mengklasifikasikan tingkat kematangan buah papaya dengan tingkat akurasi yang memuaskan, yaitu sebesar 96.97%. Dari hasil evaluasi model, juga dapat dilihat bahwa *confusion matrix* menunjukkan bahwa model memiliki kemampuan yang baik dalam mengklasifikasikan ketiga kelas kematangan buah papaya dengan tingkat akurasi yang seimbang. Dengan demikian, dapat disimpulkan bahwa penggunaan metode CNN dalam mengklasifikasikan tingkat kematangan buah papaya telah berhasil dan memberikan hasil yang memuaskan.

b. Aplikasi Android yang dikembangkan dalam penelitian ini membantu petani menentukan tingkat kematangan pepaya dan mencatat data waktu panen melalui beberapa fitur utama. Pertama, aplikasi menggunakan model CNN (Convolutional Neural Network) yang telah dilatih dan dikonversi ke format TensorFlow Lite (TFLite). Model ini memungkinkan aplikasi mengklasifikasikan gambar pepaya

berdasarkan tingkat kematangannya. Petani dapat mengambil foto pepaya melalui kamera perangkat atau memilih gambar dari galeri, dan aplikasi akan menganalisis gambar tersebut untuk menentukan apakah pepaya belum matang, hampir matang, atau sudah matang. Kedua, aplikasi memanfaatkan library Room untuk mencatat dan menyimpan informasi mengenai pepaya yang telah dianalisis, termasuk lokasi, hasil klasifikasi kematangan, serta tanggal dan waktu analisis. Dengan demikian, petani dapat mengakses riwayat analisis pepaya, yang membantu mereka melacak perkembangan kematangan buah dan memutuskan waktu panen yang optimal. Ketiga, aplikasi ini dilengkapi dengan fitur notifikasi yang mengingatkan petani tentang waktu panen berdasarkan klasifikasi kematangan yang telah dilakukan. Fitur ini memungkinkan petani mengatur jadwal panen mereka dengan lebih efisien, sehingga mengurangi risiko pepaya menjadi terlalu matang atau rusak. Seluruh fitur ini diimplementasikan menggunakan bahasa pemrograman Kotlin, yang memastikan aplikasi berjalan dengan lancar dan responsif di berbagai perangkat Android. Dengan integrasi teknologi ini, petani dapat menentukan tingkat kematangan pepaya secara akurat, mencatat data penting untuk manajemen panen yang lebih baik, dan membuat keputusan yang lebih tepat terkait waktu panen, yang pada akhirnya dapat meningkatkan kualitas dan kuantitas hasil panen pepaya.

5.2 Saran

Berikut adalah beberapa saran untuk pengembangan selanjutnya:

1. Peningkatan Keakuratan Model: Meskipun tingkat akurasi saat ini mencapai 96.97%, masih ada ruang untuk peningkatan keakuratan model. Melakukan eksplorasi lebih lanjut terhadap berbagai arsitektur CNN atau teknik pengolahan data yang lebih canggih dapat membantu meningkatkan kinerja model dalam memprediksi kematangan buah pepaya.
2. Ketersediaan Data yang Lebih Luas: Untuk meningkatkan ketepatan prediksi, penting untuk memastikan ketersediaan data yang representatif dan luas. Melakukan pengumpulan data yang lebih komprehensif terkait faktor-faktor yang memengaruhi kematangan buah pepaya, seperti kondisi cuaca, jenis tanah, atau praktik budidaya, dapat membantu meningkatkan keakuratan model.
3. Uji Coba dan Evaluasi Lanjutan: Melakukan uji coba lapangan yang lebih luas dan evaluasi lanjutan terhadap aplikasi ini akan membantu dalam

mengidentifikasi potensi masalah atau kekurangan yang mungkin terjadi di lingkungan nyata. Hal ini dapat memberikan masukan berharga untuk perbaikan dan pengembangan selanjutnya.

4. Penyempurnaan Antarmuka Pengguna: Perhatikan pengalaman pengguna secara menyeluruh, termasuk desain antarmuka yang intuitif, panduan pengguna yang jelas, dan responsif terhadap umpan balik pengguna. Penyempurnaan antarmuka pengguna dapat meningkatkan tingkat adopsi dan kepuasan pengguna terhadap aplikasi.

Dengan menerapkan saran-saran di atas, diharapkan aplikasi ini dapat terus berkembang dan memberikan manfaat yang lebih besar bagi pengguna serta pemangku kepentingan terkait.

DAFTAR PUSTAKA

Agustin, Tuti, Suyudi Suyudi, dan Hendar Nuryaman. 2019. "Kinerja Kelembagaan Agribisnis Pepaya California." *Jurnal Agristan* 1(2).

Agustina, Feri, dan Muhammad Sukron. 2022. "Deteksi kematangan buah pepaya menggunakan algoritma YOLO berbasis Android."

Aminudin, Arif. 2019. *Klasifikasi tingkat Kematangan Buah Pepaya menggunakan Metode K-Nearest Neighbor berdasarkan Warna Kulit Buah (Doctoral dissertation, University of Technology Yogyakarta)*.

Arkadia, Arvi, Sekar Ayu Damayanti, dan Desta Sandya Prasvita. 2021. *Klasifikasi Buah Mangga Badami Untuk Menentukan Tingkat Kematangan dengan Metode CNN*.

Bili, Yanri, Eliezer Purba, Naikson F. Saragih, Arina Prima Silalahi, Surianto Sitepu, Asaziduhu Gea, Fakultas Ilmu Komputer, dan Histori Artikel. 2022. *Perancangan Alat Pendeteksi Kematangan Buah Nanas Dengan Menggunakan Mikrokontroler Dengan Metode Convolutional Neural Network (CNN)*. Vol. 2.

Cho, Wan, Sang Kim, Myung Na, dan In Na. 2021. "Fruit Ripeness Prediction Based on DNN Feature Induction from Sparse Dataset." *Computers, Materials & Continua* 69:4003–24. doi: 10.32604/cmc.2021.018758.

Dwiantara, M. Redo. 2020. *Preference Konsumen Terhadap Buah Pepaya California dan Thailand di Kecamatan Gamping Kabupaten Sleman*.

Ellif, Sampe Holtan Sitorus, dan Rahmi Hidayati. 2021. "Klasifikasi Kematangan Pepaya Menggunakan Ruang warna HSV dan Metode Naive Bayes Classifier." 09:66–75.

Fadlilah, Umi, Abd Kadir Mahamad, dan Bana Handaga. 2021. "The Development of Android for Indonesian Sign Language Using Tensorflow Lite and CNN: An Initial Study." dalam *Journal of Physics: Conference Series*. Vol. 1858. IOP Publishing Ltd.

Febjislami, Shalati, Ketty Suketi, dan Yunianti Rahmi. 2018. *Morphological Characterization of flowers, fruit and fruit quality three genotypes of hybrid papaya*. Vol. 6.

Al Ghifari, Moch Azhar, Agung Panji Sasmito, dan Deddy Rudhistiar. 2022. "Aplikasi Pendeteksian Kematangan Tomat Menggunakan Thresholding." *JATI (Jurnal Mahasiswa Teknik Informatika)* 6(1):294–300.

Hossen, Md Sagar, Imdadul Haque, Md Saif Islam, Md Tanvir Ahmed, Md Jannati Nime, dan Md Ashiqul Islam. 2020. "Deep learning based classification of papaya disease

recognition." Hlm. 945–51 dalam *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*. IEEE.

Kattenborn, Teja, Jens Leitloff, Felix Schiefer, dan Stefan Hinz. 2021. "Review on Convolutional Neural Networks (CNN) in vegetation remote sensing." *ISPRS Journal of Photogrammetry and Remote Sensing* 173:24–49.

Nashrullah, Faiz, Suryo Adhi Wibowo, dan Gelar Budiman. 2020. "Investigasi parameter epoch pada arsitektur resnet-50 untuk klasifikasi pornografi." *Journal of Computer, Electronic, and Telecommunication* 1(1):1–8.

Noviani, Dwi Arief Prambudi, dan Fadli Mulyadi. 2020. *Sistem Pakar Diagnosis Penyakit Pada Tanaman Pepaya Menggunakan Metode Backward Chaining Berbasis Web*. Vol. 21.

Nurhopipah, Ade, dan Nurriza Amalia Larasati. 2021. "CNN hyperparameter optimization using random grid coarse-to-fine search for face classification." *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control* 19–26.

Nurmalasari, Nurmalasari, Yusuf Arif Setiawan, Widi Astuti, M. Rangga Ramadhan Saelan, Siti Masturoh, dan Tuti Haryanti. 2023. "Classification for Papaya Fruit Maturity Level with Convolutional Neural Network." *Jurnal Riset Informatika* 5(3):331–38. doi: 10.34288/jri.v5i3.541.

Reda, Mariam, Rawan Suwwan, Seba Alkafri, Yara Rashed, dan Tamer Shanableh. 2022. "AgroAId: A Mobile App System for Visual Classification of Plant Species and Diseases Using Deep Learning and TensorFlow Lite." *Informatics* 9(3). doi: 10.3390/informatics9030055.

Suryana, Asep, dan Rochanda Wiradinata. 2013. "Pengaruh konsentrasi kitosan terhadap lama simpan dan mutu pada dua tingkat kematangan pepaya Callina (Carica papaya L.)."

Wardani, Lidia Ardhia, I. Gede Pasek Suta Wijaya, dan Fitri Bimantoro. 2022. "Klasifikasi Jenis Dan Tingkat Kematangan Buah Pepaya Berdasarkan Fitur Warna, Tekstur Dan Bentuk Menggunakan Support Vector Machine." *Jurnal Teknologi Informasi, Komputer, dan Aplikasinya (JTIKA)* 4(1):75–87.

Xiao, Bingjie, Minh Nguyen, dan Weiqi Yan. 2023. "Fruit ripeness identification using YOLOv8 model." *Multimedia Tools and Applications* 83:1–18. doi: 10.1007/s11042-023-16570-9.

Yanto, Budi, Luth Fimawahib, Asep Supriyanto, B. Herawan Hayadi, Rinanda Rizki Pratama, Universitas Pasir Pengaraian, Jl Tuanku Tambusai, Rambah Hilir, Rokan Hulu, Universitas Hazarin, dan Jl Ahmad Yani. 2021. "Klasifikasi Tekstur Kematangan Buah Jeruk Manis Berdasarkan Tingkat Kecerahan Warna dengan Metode Deep Learning Convolutional Neural Network." 6(2):256–68.

LAMPIRAN

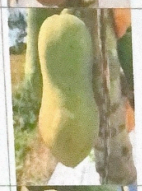
1. Lampiran penentuan tingkat kematangan pepaya oleh petani

No.	Foto	Keterangan	Benar/Salah
		Mentah	✓
		Mentah	✓
		Mentah	✓
		Mentah	✓

	Mentah	✓
	Mentah	✓
	Mentah	✓
	Mentah	✓

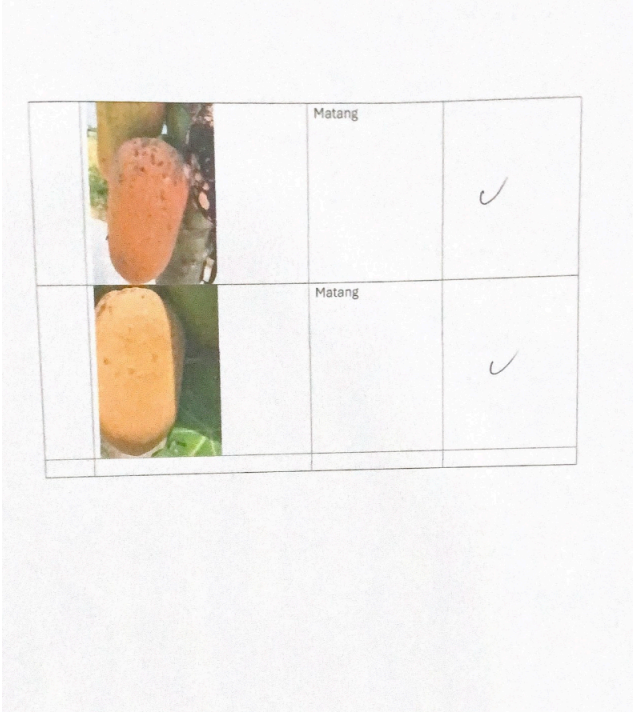
	Mentah	✓
	Mentah	✓
	Setengah matang	✓
	Setengah matang	✓

	Setengah matang	✓
	Setengah matang	✓
	Setengah matang	✓
	Setengah matang	✓

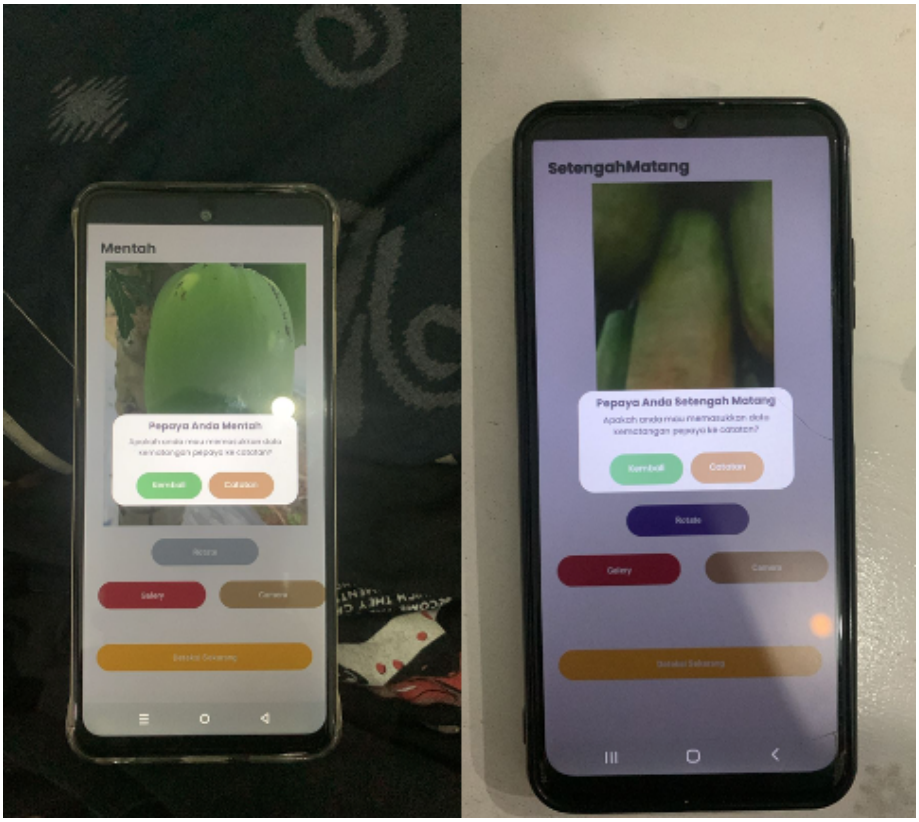
	Setengah matang	✓
	Setengah matang	✓
	Setengah matang	✓
	Setengah matang	✓

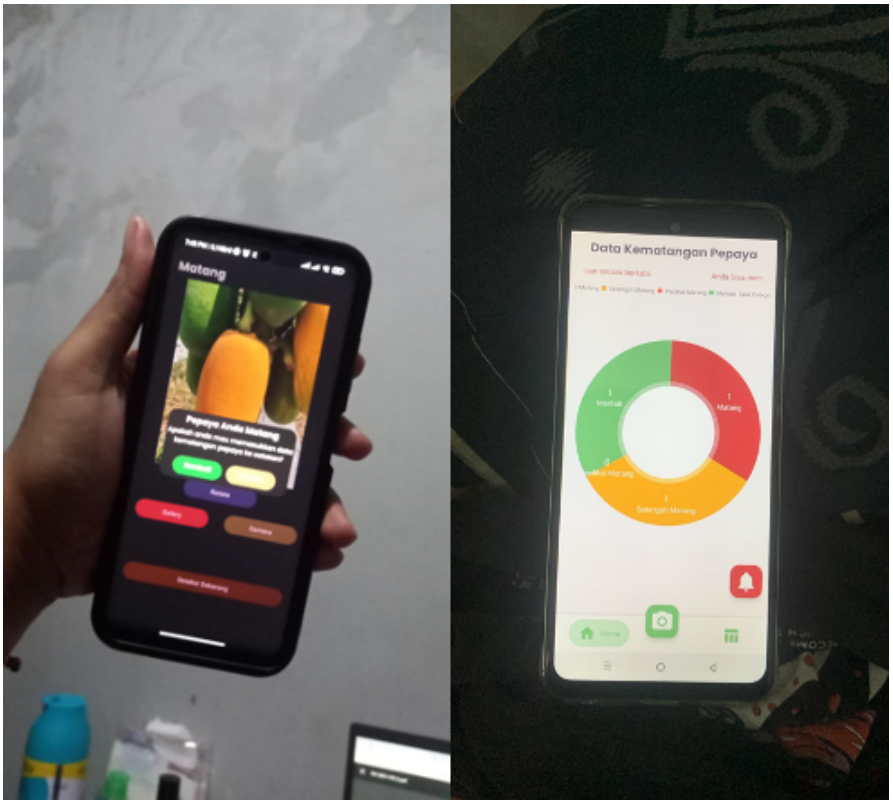
	Matang	✓
	Matang	✓
	Matang	✓
	Matang	✓

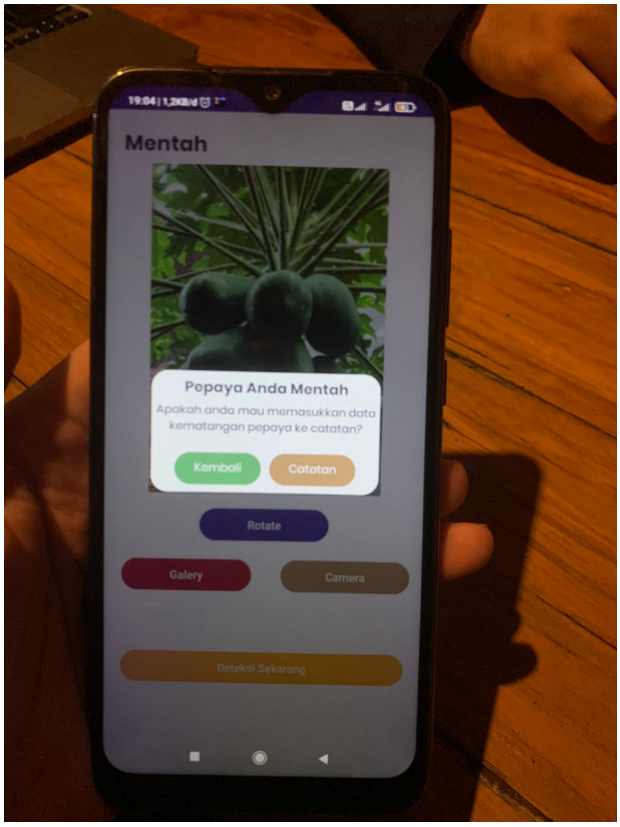
	Matang	✓
	Matang	✓
	Matang	✓
	Matang	✓



2. Lampiran pengujian aplikasi pada beberapa perangkat android




Matang
Data Kematangan Pepaya


Mentah
Pepaya Anda Mentah
Apakah anda mau memasukkan data kematangan pepaya ke catatan?
Kembali
Catatan
Rotate
Galery
Camera
Deteksi Sekarang