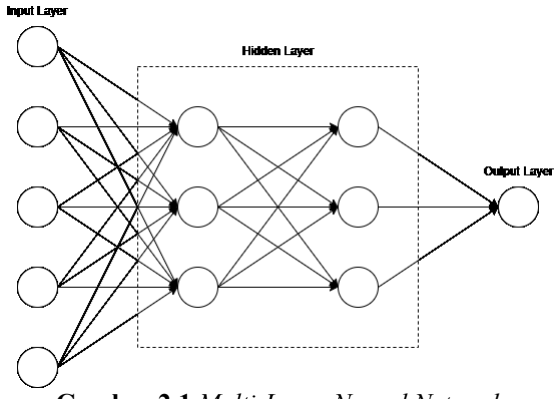
menangani berbagai jenis data, termasuk gambar, suara, dan teks, dengan tingkat akurasi yang sangat tinggi. Kemampuan ini juga mencakup adaptasi terhadap data yang berubah-ubah dan pemahaman konteks yang lebih baik, membuat *deep learning* sangat efektif untuk tugas-tugas seperti pengenalan gambar, pemrosesan bahasa alami, dan deteksi anomali. Perpaduan tersebut yang membuat *Deep Learning* dapat menghasilkan representasi fitur yang baik. *Deep Learning* terdiri atas:

1. Neural Network

Neural Network adalah representasi struktur neuron di otak suatu organisme, yang terdiri dari beberapa simpul. Jaringan simpul ini menghitung nilai bobot total dari masukan, memprosesnya di lapisan tersembunyi, dan mengeluarkan hasilnya dengan nilai bobot spesifik melalui fungsi aktivasi. *Neural Network* telah mengalami perkembangan dari arsitektur yang sederhana menjadi lebih kompleks.

Awalnya, jaringan saraf memiliki arsitektur yang sangat sederhana yang terdiri dari *input layer* dan *output layer*, yang disebut jaringan saraf lapisan tunggal (*single-layer*). Ketika *hidden layer* ditambahkan, terbentuklah jaringan saraf *multi-layer*, yang terdiri dari *input layer*, *hidden layer*, dan *output layer*. Jaringan saraf *multi-layer* ini memberikan gambaran bagaimana struktur yang lebih kompleks dapat meningkatkan kemampuan pemrosesan dan analisis dari *neural network*. Gambaran jaringan saraf *multi-layer* dapat dilihat pada gambar di bawah ini.



Gambar 2.1 *Multi-Layer Neural Network*

Untuk memperoleh nilai neuron tujuan (y), nilai pada neuron (x) dikalikan dengan bobot (w) dan ditambahkan dengan bias (b), kemudian diaktivasi

menggunakan fungsi (g). Hasil dari proses ini adalah nilai dari neuron tujuan (y). Persamaan ini dapat dirumuskan sebagai berikut.

$$\boldsymbol{y = g\ (\sum_{i=1}^{n} x_i w_i + b)} \quad \textbf{(2.1)}$$

2. *Activation Functions*

Fungsi aktivasi merupakan salah satu faktor yang menentukan apakah neuron pada jaringan saraf diaktifkan atau tidak.

1. *Softmax*

Fungsi *softmax* adalah fungsi aktivasi dalam *machine learning* yang digunakan untuk mengubah vektor angka menjadi vektor probabilitas yang menjumlahkan hingga satu. Ini biasanya digunakan pada lapisan output dari model klasifikasi, terutama dalam tugas klasifikasi multi-kelas. Fungsi *softmax* mengambil *input* dari *layer* sebelumnya, mengaplikasikan fungsi eksponensial pada setiap elemen, dan kemudian menormalisasi hasilnya dengan membagi setiap nilai eksponensial dengan jumlah semua nilai eksponensial tersebut.

Nilai probabilitas untuk setiap label ditentukan oleh fungsi *Softmax*. Rumus fungsi ini dapat dilihat pada persamaan di bawah ini, dimana nilai probabilitas (S) pada lapisan (y) diekstraksi dari neuron pada lapisan klasifikasi terakhir, yaitu eksponen (e) dibagi eksponen nilainya. nilai-nilai. Hasilnya mengubah label yang ada menjadi vektor dengan nilai antara 0 dan 1. Jika semua hasil dijumlahkan maka jumlahnya menjadi 1.

$$\boldsymbol{S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}} \quad \textbf{(2.2)}$$

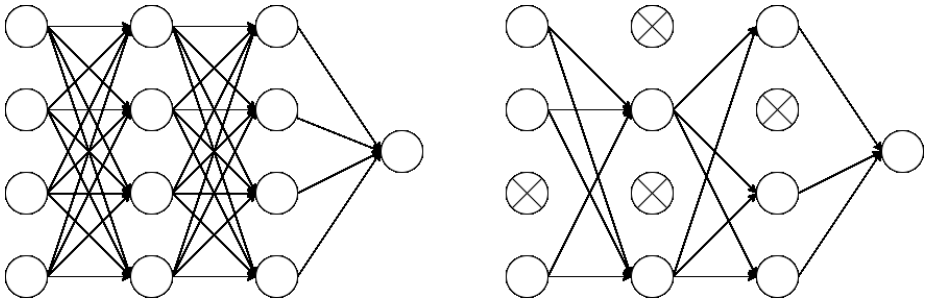
2. *Rectified Linear Unit* (ReLU)

Fungsi *Rectified Linear Unit* (ReLU) memiliki persamaan $f(z) = \max(0, z)$, yang berarti jika *output* bernilai positif, maka *output* tetap sama, tetapi jika output bernilai negatif, maka *output* menjadi 0. ReLU tidak hanya meningkatkan kinerja secara signifikan tetapi juga membantu mengurangi jumlah komputasi selama fase pelatihan. Hal ini dikarenakan nilai 0 pada *output* ketika nilai z negatif sehingga menyebabkan neuron tidak aktif.

3. *DropOut*

Overfitting pada suatu model juga dapat dicegah dengan penggunaan *DropOut.* Metode ini melibatkan pelatihan pemilihan neuron secara acak dalam jaringan, yang kemudian diaktifkan hingga mencapai nol. Neuron yang dipilih ini berubah setiap iterasi pelatihan, sehingga pembelajaran ditingkatkan dan *overfitting* dapat diminimalkan. Menonaktifkan sementara neuron-neuron yang dipilih secara acak dan mengatur aktivasi mereka menjadi nol, jaringan saraf dapat meningkatkan kualitas pembelajarannya.

Istilah "DropOut" digunakan untuk menggambarkan pemutusan sementara neuron (tersembunyi atau terlihat) dalam jaringan saraf. Dengan menonaktifkan sementara neuron dan menghapusnya dari jaringan beserta semua koneksi masuk dan keluarnya, jaringan saraf menjadi lebih kuat dalam menghadapi *overfitting*. Pemilihan neuron yang dikeluarkan dilakukan secara acak setiap iterasi pelatihan, yang membantu memastikan bahwa model tidak terlalu bergantung pada neuron tertentu dan meningkatkan generalisasi model.



Gambar 2.2 Perbandingan Proses *DropOut* Dalam *Neural Network*

4. Loss Function

Loss function adalah komponen penting dalam pembelajaran mesin dan jaringan saraf tiruan yang digunakan untuk mengukur seberapa baik model memprediksi atau mengestimasi hasil yang diinginkan. Fungsi ini menghitung selisih antara prediksi model dan nilai aktual yang diketahui dari data pelatihan. Semakin besar perbedaan ini, semakin tinggi nilai *loss function*, yang menunjukkan bahwa model tersebut memprediksi hasil yang diinginkan dengan buruk. Selama proses pelatihan, *loss function* digunakan untuk mengarahkan optimisasi model. Algoritma optimisasi, seperti *gradient descent*, menggunakan nilai *loss function* untuk

memperbarui bobot dan bias dalam jaringan dengan tujuan meminimalkan kesalahan prediksi.

5. Backpropagation

Backpropagation adalah algoritma yang digunakan untuk menemukan nilai minimum dari *loss function* dengan menyesuaikan bobot (*weight*) dalam jaringan saraf. Mengubah bobot dalam algoritma ini melalui aturan delta atau *gradient descent* dimaksudkan untuk mengurangi kesalahan atau *loss*. Proses ini melibatkan penghitungan gradien dari *loss function* terhadap setiap bobot dalam jaringan, kemudian memperbarui bobot tersebut dalam arah yang mengurangi nilai *loss function*.

Dengan melakukan iterasi melalui proses ini, bobot yang meminimalkan *loss function* akhirnya dianggap sebagai solusi untuk masalah pembelajaran. *Backpropagation* memungkinkan jaringan saraf belajar dari kesalahan sebelumnya dengan memperbaiki bobot secara berulang kali hingga model mencapai tingkat akurasi yang optimal. Algoritma ini sangat penting dalam pelatihan jaringan saraf dalam deep learning, karena efisiensinya dalam menangani jaringan yang sangat kompleks dengan banyak lapisan.

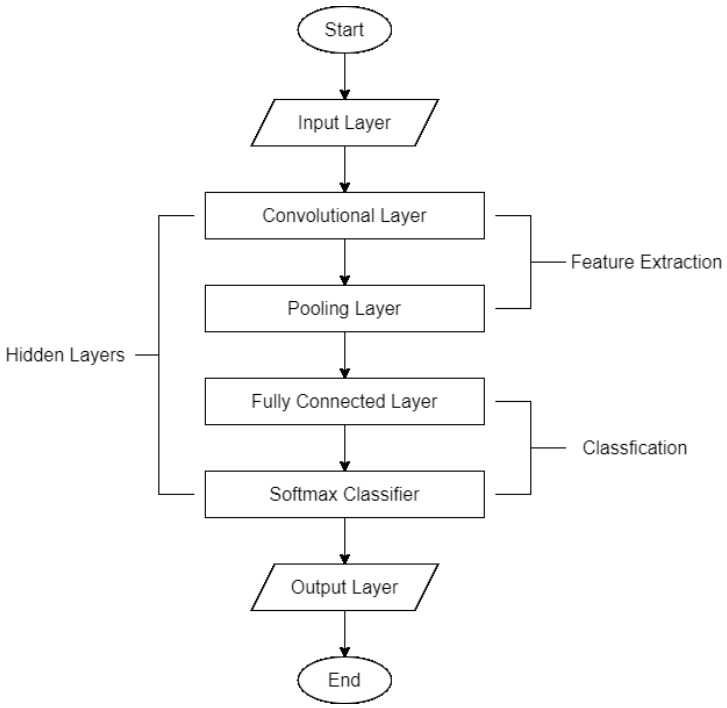
2.2.4 Klasifikasi

Klasifikasi adalah proses mengkategorikan data ke dalam kelompok atau kelas yang berbeda. Klasifikasi melibatkan pembuatan model yang dapat memprediksi label kelas untuk data baru berdasarkan pola yang ditemukan dalam data pelatihan. Klasifikasi sering digunakan dalam *machine learning* dan statistik untuk tugas-tugas seperti pengenalan gambar, deteksi spam, dan diagnosis medis. Kategori tersebut dilambangkan dengan nilai diskrit, tanpa signifikansi yang ditempatkan pada urutan nilainya. Ada dua tahap klasifikasi, dengan tahap pertama menjadi proses pembelajaran yang berpuncak pada pembuatan model yang pada dasarnya identik. Kedua adalah proses klasifikasi, dimana model yang dibangun digunakan untuk memprediksi label dari data yang disediakan.

2.2.5 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) merupakan jenis pembelajaran mendalam yang sering digunakan dalam pemrosesan dan analisis gambar. CNN

terdiri dari banyak lapisan representasi, yang masing-masing melakukan tugas spesifik dalam ekstraksi fitur. Struktur arsitektur yang dalam dan kompleks memungkinkan pengambilan karakteristik representasi data secara otomatis melalui serangkaian transformasi *non-linier* multi-layer dan perkiraan fungsi *non-linier*. Lapisan-lapisan ini memungkinkan CNN mengenali pola, tepi, tekstur, dan fitur lainnya dalam data, sehingga menciptakan model yang sangat kuat dan mampu melakukan klasifikasi, segmentasi, serta deteksi objek dengan akurasi tinggi.



Gambar 2.3 *Flowchart* dari CNN

Diagram tersebut menggambarkan alur kerja dari sebuah *Convolutional Neural Network* (CNN), yang terdiri dari beberapa lapisan untuk proses pengenalan dan klasifikasi fitur. Proses dimulai dari lapisan *input*, di mana data masukan berupa citra diterima. Setiap piksel dalam citra ini menjadi *input* bagi jaringan. Kemudian, pada lapisan konvolusi, fitur-fitur dari citra diekstraksi menggunakan filter yang menerapkan operasi konvolusi, membantu mendeteksi fitur seperti tekstur, tepi, dan pola. Dengan mengekstraksi fitur dan mengurangi ukuran spasial peta fitur lapisan konvolusional, lapisan penggabungan membantu mengendalikan *overfitting* dengan meminimalkan parameter dan kalkulasi. Semua neuron di lapisan sebelumnya terhubung ke setiap neuron di *fully connected layer*, dan fitur yang

diekstraksi digabungkan menjadi representasi yang kuat oleh lapisan yang terhubung sepenuhnya. Lapisan *softmax classifier* kemudian mengubah output dari lapisan *fully connected* menjadi probabilitas yang mewakili kelas-kelas yang mungkin, dengan setiap nilai output diubah menjadi nilai probabilitas dalam rentang [0, 1] yang jika dijumlahkan akan menjadi satu. Akhirnya, lapisan output menghasilkan prediksi akhir dari jaringan, berdasarkan probabilitas yang dihitung oleh softmax classifier. Proses ini dimulai dari titik awal dan berakhir pada titik akhir, di mana hasil klasifikasi akhir dihasilkan. Lapisan konvolusi dan pooling adalah bagian dari *hidden layers*, yang bekerja untuk mengekstraksi dan mengompres fitur dari citra masukan, sedangkan proses klasifikasi terjadi di lapisan *fully connected* dan *softmax classifier*, menggabungkan dan menggunakan fitur-fitur yang diekstraksi untuk membuat prediksi akhir tentang kelas citra. Secara umum, CNN terdiri dari beberapa layer.

2.2.5.1 Convolutional Layer

Convolutional Layer adalah proses pemrosesan gambar yang menggunakan *mask* eksternal atau *sub-windows* untuk membuat gambar baru dengan mengubah ukuran gambar melalui operasi konvolusi. Melalui langkah *encoding*, tujuan utamanya adalah untuk mengisolasi aspek-aspek penting dari suatu gambar seperti deteksi tepi dan fitur-fitur dalam kaitannya dengan warna/arah gradien (Radikto et al., 2022). Menggeser filter pada citra diperlukan untuk menyelesaikan proses ini. Proses ini memastikan bahwa semua filter memiliki nilai bobot dan bias yang sama di seluruh citra. Lapisan ini menghasilkan citra baru yang disebut peta fitur (*feature map*). *Feature map* menonjolkan fitur unik dari citra asli. *Convolutional layer* memiliki dua komponen penting:

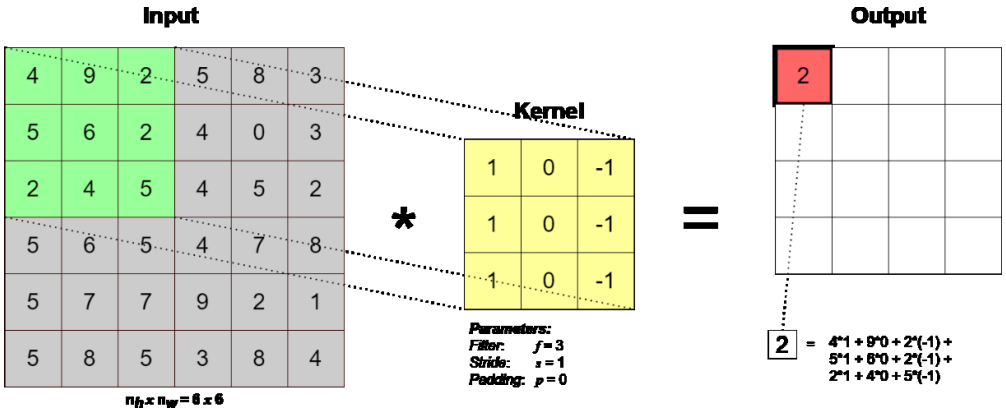
1. *Strides*

   Strides adalah parameter dalam operasi konvolusi yang menentukan seberapa jauh *filter* (*kernel*) bergeser di sepanjang citra input selama proses konvolusi. *Strides* mengontrol langkah-langkah horizontal dan vertikal dari filter saat memindai seluruh citra. Misalnya, jika *stride* diatur ke 1, filter akan bergeser satu piksel pada satu waktu baik secara horizontal maupun vertikal, menghasilkan *output feature map* dengan ukuran yang hampir sama dengan

input, dikurangi ukuran filter. Sebaliknya, jika *stride* diatur ke 2, filter akan bergeser dua piksel pada satu waktu, mengurangi ukuran *feature map output* menjadi setengah dari ukuran input secara spasial. Dengan mengatur *stride* yang lebih besar, ukuran *output feature map* akan berkurang lebih cepat, mengurangi jumlah parameter dan komputasi yang diperlukan, namun dapat menyebabkan hilangnya informasi detail dalam citra karena pengambilan sampel yang lebih jarang. Dengan demikian, semakin kecil *stride* yang digunakan akan menyebabkan semakin detail hasil konvolusi namun menyebabkan semakin tinggi beban komputasi yang harus dilakukan oleh komputer (Sediyono & Kartowisastro, 2021).

2. *Padding*

Padding adalah teknik yang digunakan dalam operasi konvolusi untuk menambah tepi tambahan di sekitar citra input sebelum menerapkan filter. Tujuan utama dari padding adalah untuk mengontrol ukuran *output feature map* dan mencegah pengurangan ukuran citra setelah setiap operasi konvolusi. *Zero padding*, mengisi tepi tambahan dengan nilai nol, menjaga dimensi spasial dari citra input sehingga ukuran *output feature map* tetap sama dengan ukuran input setelah konvolusi. Dalam *same padding*, jumlah padding diatur agar ukuran *output feature map* sama dengan ukuran *input*, memastikan bahwa informasi dari tepi citra tidak hilang selama proses konvolusi. *Valid padding*, di sisi lain, tidak menambahkan padding, sehingga konvolusi hanya dilakukan pada area yang sepenuhnya berada dalam citra asli, biasanya mengurangi ukuran *feature map output* dibandingkan dengan *input*. *Padding* sangat penting dalam desain jaringan konvolusional yang dalam dengan banyak lapisan konvolusi karena membantu mempertahankan ukuran citra dan memastikan bahwa semua fitur penting dari citra input dipertahankan selama proses konvolusi.



Gambar 2.4 Operasi Konvolusi menggunakan *Zero Padding*

Dari proses konvolusi pada gambar 2.4, dapat dihitung berdasarkan formula:

$$\frac{(\mathbf{N} - \mathbf{F} + \mathbf{2P})}{\mathbf{S}} + \mathbf{1} \qquad \textbf{(2.3)}$$

N = Ukuran spasial (H1 = W1)

F = *Kernel*/*Filter*

P = *Padding*

S = *Stride*

Secara umum, jika masukan ke lapisan konvolusional adalah gambar berukuran W1 x H1 x D1, maka keluaran lapisan tersebut akan berupa gambar baru berukuran W2 x H2 x D2, yang dapat dihitung menggunakan rumus berikut:

$$\boldsymbol{W2} = \frac{(\boldsymbol{W1} - \boldsymbol{F} + \boldsymbol{2P})}{\boldsymbol{S}} + \mathbf{1} = \boldsymbol{H2} \qquad \textbf{(2.4)}$$

F = Ukuran Spasial *Filter* (lebar/tinggi).

S = *Stride*

P = *Padding*.

Ada beberapa hal yang perlu diperhatikan pada *layer* ini:

1. Ketebalan filter selalu sesuai dengan ketebalan atau *volume* gambar masukan yang digunakan.
2. Ukuran *filter* (F) biasanya berupa angka ganjil. Secara intuitif, filter berukuran ganjil memberikan representasi yang lebih baik karena mencakup bagian kiri dan kanan secara merata.

3. Pada lapisan ini, semua filter yang digunakan memiliki ukuran yang sama untuk memudahkan proses perhitungan. Lapisan konvolusional biasanya dilengkapi dengan sejumlah filter (K) yang habis dibagi 2. Beberapa pustaka memiliki subrutin khusus untuk menghitung kernel atau kelipatan 2 dimensi yang dapat meningkatkan efisiensi.
4. Zero padding biasanya disesuaikan sehingga ukuran spasial keluaran tetap sama dengan ukuran spasial masukan.

$$(\boldsymbol{P} = \frac{\boldsymbol{F-1}}{\boldsymbol{2}}) \quad \textbf{(2.5)}$$

F = Ukuran Spasial *Filter* (lebar/tinggi).

P = *Padding*.

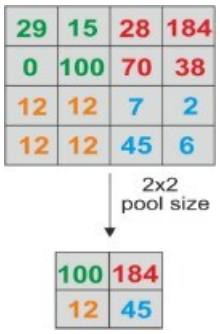
2.2.5.2 Pooling Layer

Pooling Layer adalah komponen arsitektur CNN yang digunakan untuk mengurangi dimensi spasial dari *input* peta fitur (*feature maps*). Pada lapisan ini dilakukan melalui operasi *down-sampling*, seperti *max pooling* atau *average pooling*, yang bertujuan untuk mengurangi jumlah parameter dan komputasi dalam jaringan. Dengan demikian, *pooling layer* membantu mencegah *overfitting* dan membuat model lebih efisien. *Pooling layer* mengambil hasil dari lapisan konvolusi sebagai masukan dan menghasilkan peta fitur yang lebih kecil, yang merupakan representasi ringkas dari peta fitur asli. Ada dua operator *pooling layer* yang sering digunakan yaitu:

1. Max Pooling

Di antara pendekatan yang populer adalah *Max Pooling*. Pemilihan neuron dengan nilai aktivasi tertinggi di setiap medan reseptif lokal (*grid cell*) dilakukan melalui metode ini. Metode ini hanya mengambil nilai tertinggi dari setiap *grid cell* dan mengirimkannya ke tahap pemrosesan berikutnya. (Firmansyah, 2021). Pendekatan ini sangat berguna dalam mengurangi dimensi data tanpa kehilangan fitur-fitur penting, yang pada akhirnya meningkatkan efisiensi komputasi dan performa model secara keseluruhan. Selain itu, *Max Pooling* membantu dalam mengatasi masalah *overfitting* dengan menyederhanakan representasi data dan membuat model lebih generalisasi terhadap data baru. Dalam konteks pengenalan

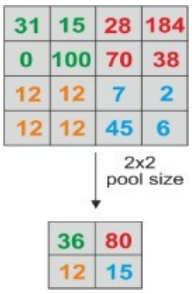
pola dan citra, *Max Pooling* memungkinkan model untuk lebih fokus pada fitur-fitur yang paling relevan, seperti tepi atau tekstur yang dominan, sehingga meningkatkan akurasi dalam klasifikasi dan deteksi objek.



Gambar 2.5 *Max Pooling* (Radikto et al., 2022)

2. Average Pooling

Average Pooling adalah metode *pooling* yang sering digunakan dalam CNN. Dalam metode ini, keluaran setiap bidang reseptif dihitung dengan mengambil rata-rata seluruh aktivasi di bidang tersebut (Firmansyah, 2021). Dengan kata lain, setiap nilai dalam bidang reseptif lokal dijumlahkan, kemudian dibagi dengan jumlah total nilai dalam bidang tersebut untuk mendapatkan nilai rata-rata. Metode ini memungkinkan informasi dari seluruh bidang reseptif untuk dipertimbangkan, bukan hanya nilai tertinggi seperti pada *Max Pooling*, sehingga dapat menghasilkan representasi yang lebih halus dan kurang sensitif terhadap variasi lokal.

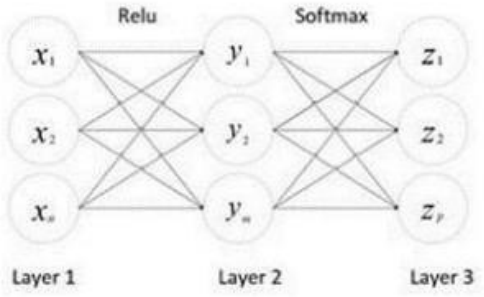


Gambar 2.6 *Average Pooling* (Radikto et al., 2022)

2.2.5.3 Fully Connected Layer

Peta fitur yang diperoleh dari proses ekstraksi fitur berbentuk *array* multidimensi, sehingga selalu diperlukan proses pembentukan kembali agar peta fitur dapat diubah menjadi vektor, yang kemudian digunakan sebagai *input* untuk

fully connected layer. Oleh karena itu, dibutuhkan *flatten* untuk dapat terhubung dengan *fully connected layer* (Magdalena et al., 2021).



Gambar 2.7 *Fully Connected Layer* (Radikto et al., 2022)

2.2.5.4 MobileNetV2

MobileNetV2 adalah arsitektur CNN yang dirancang untuk mengatasi kebutuhan sumber daya komputasi yang berlebihan. Arsitektur MobileNetV2 berbeda dari arsitektur CNN pada umumnya dalam hal penggunaan lapisan konvolusi. Struktur dasar MobileNetV2 adalah blok *bottleneck* konvolusi terpisah kedalaman dengan *residual* menggunakan *stride* $s = 1$ atau $s = 2$. Blok *bottleneck residual* ini terdiri dari tiga lapisan, yaitu lapisan konvolusi dengan ukuran kernel 1×1 dan fungsi aktivasi ReLU6, lapisan konvolusi kedalaman dengan ukuran kernel 3×3 dan fungsi aktivasi ReLU6, dan lapisan konvolusi dengan ukuran kernel 1×1 dan fungsi aktivasi linear. Fungsi ReLU6 adalah modifikasi dari fungsi ReLU dengan batas atas nilai aktivasi sebesar 6 (Sugiarto et al., 2023).

Tabel 2.2 Detail Arsitektur MobileNetV2

Layer	Input Size	*r*	*s*	*e*	*c*
Convolutional 3 x 3	224 x 224 x 3	1	2	-	32
Bottleneck	112 x 112 x 32	1	1	1	16
Bottleneck	112 x 112 x 16	2	2	6	24
Bottleneck	56 x 56 x 24	3	2	6	32
Bottleneck	28 x 28 x 32	4	2	6	64
Bottleneck	14 x 14 x 64	3	1	6	96
Bottleneck	14 x 14 x 96	3	2	6	160
Bottleneck	7 x 7 x 160	1	1	6	320

Convolutional 1 x 1	7 x 7 x 320	1	1	-	1280
Average Pooling 7 x 7	7 x 7 x 1280	1	-	-	-
Convolutional 1 x 1	1 x 1 x 1280	-	-	-	*k*

Sumber: (Sugiarto et al., 2023)

Tabel ini menggambarkan struktur jaringan saraf konvolusional (*Convolutional Neural Network*) yang terdiri dari berbagai jenis lapisan. Setiap lapisan memiliki peran khusus dalam memproses data input menjadi fitur yang lebih abstrak dan informatif. Variabel yang digunakan dalam tabel meliputi beberapa elemen penting. Pertama, *Layer* merujuk pada jenis lapisan seperti konvolusional atau *bottleneck*. *Input Size* menunjukkan ukuran input yang diterima oleh lapisan tersebut dalam format tinggi x lebar x jumlah saluran (*channels*). Variabel *r (Stride)* adalah jumlah langkah yang diambil oleh *filter* atau *kernel* saat melintasi *input*, yang menentukan seberapa banyak *filter* bergeser pada setiap langkah dan mempengaruhi ukuran output. Variabel *s (Kernel Size)* adalah ukuran filter atau kernel yang digunakan dalam operasi konvolusi, berfungsi sebagai matriks kecil yang digunakan untuk memindai *input* dan menerapkan operasi konvolusi. Variabel *e (Ekspansi)* adalah faktor ekspansi yang digunakan dalam beberapa lapisan, terutama *bottleneck*, untuk memperbesar dimensi saluran fitur sebelum menguranginya kembali, membantu menangkap lebih banyak informasi dari data. Variabel *c (Jumlah Filter/Channels)* merujuk pada jumlah *filter* yang digunakan dalam lapisan konvolusional, yang menentukan jumlah saluran fitur yang dihasilkan oleh lapisan tersebut.

Sebagai contoh, lapisan konvolusional 3x3 menggunakan filter 3x3 dengan *stride* 1 untuk memproses input berukuran 224 x 224 x 3 dan menghasilkan 32 saluran fitur dengan *kernel size* 2. Lapisan *bottleneck*, yang memiliki berbagai ukuran input dan konfigurasi, berfungsi untuk mengurangi dan kemudian memperluas dimensi fitur, menangkap lebih banyak informasi. Misalnya, bottleneck pertama memproses input 112 x 112 x 32 dengan *stride* 1 dan ekspansi 1, menghasilkan 16 saluran fitur, sedangkan *bottleneck* kedua memproses input dengan *stride* 2 dan *ekspansi* 6, menghasilkan 24 saluran fitur. Lapisan average pooling 7x7 berfungsi mengurangi dimensi data dengan mengambil rata-rata dari

semua nilai dalam bidang reseptif, memproses input 7 x 7 x 1280 dengan *stride* 1 dan menghasilkan *output* yang lebih kecil. Akhirnya, *convolutional layer* 1x1 menggunakan filter 1x1 untuk mengurangi dimensi saluran fitur, memproses input 1 x 1 x 1280 dan menghasilkan output dengan jumlah saluran yang sesuai dengan jumlah kelas (k). Struktur ini dirancang untuk mengoptimalkan ekstraksi fitur dan klasifikasi akhir, dengan pengaturan *stride*, ukuran *kernel*, dan jumlah *filter* yang disesuaikan pada setiap lapisan.

2.2.6 Tensorflow

TensorFlow adalah sebuah framework *open-source* yang dikembangkan oleh *Google* untuk memfasilitasi pembuatan, pelatihan, dan implementasi model *machine learning* dan *deep learning*. TensorFlow memanfaatkan konsep graf komputasi, di mana operasi matematika dan data yang mengalir direpresentasikan sebagai *node* dan *edges* dalam graf tersebut. Dengan kemampuannya yang dapat diterapkan di berbagai platform, termasuk *desktop*, *server*, hingga perangkat *mobile*, *framework* memungkinkan pengembangan model yang dapat berjalan secara efisien di berbagai lingkungan. Selain itu, TensorFlow mendukung berbagai jenis model pembelajaran mesin, mulai dari yang sederhana hingga kompleks seperti *Convolutional Neural Network* (CNN). TensorFlow memungkinkan pengembang untuk bereksperimen dengan optimasi baru dan algoritma untuk proses pelatihan (Firmansyah, 2021).

2.2.7 Firebase

Firebase adalah platform seluler yang membantu developer mengembangkan aplikasi berkualitas tinggi secara cepat, berbasis pengguna, dan dapat menghasilkan uang lebih banyak (Furqon et al., 2019). Dikembangkan oleh Google, Firebase menawarkan fitur seperti autentikasi pengguna, basis data *real-time*, penyimpanan *cloud*, *hosting*, analitik, dan notifikasi. Salah satu fitur utamanya adalah *Firebase Realtime Database*, yang memungkinkan data disimpan dan disinkronkan di seluruh klien dalam waktu nyata. Firebase juga menyediakan *Firebase Authentication*, yang mempermudah proses login pengguna melalui email, media sosial, atau metode lainnya. Selain itu, *Firebase Cloud Messaging* memungkinkan pengembang mengirim notifikasi ke pengguna aplikasi secara langsung. Dengan

menggunakan Firebase, pengembang dapat fokus pada pengembangan fitur-fitur inti aplikasi tanpa harus khawatir tentang pengelolaan server atau infrastruktur *backend* lainnya.

2.2.8 Kotlin

Kotlin merupakan bahasa pemrograman yang pragmatis untuk *android* yang mengkombinasikan *object oriented* (OO) dan bahasa fungsional (Febriandirza, 2020). Kotlin adalah bahasa pemrograman modern yang dikembangkan oleh JetBrains, dirancang untuk berjalan di atas *Java Virtual Machine* (JVM) dan sepenuhnya *interoperable* dengan Java. Kotlin menawarkan sintaks yang lebih ringkas, aman, dan lebih mudah dibaca dibandingkan Java, sehingga meningkatkan produktivitas pengembang. Bahasa ini sangat populer di kalangan pengembang Android, karena Google telah mengadopsi Kotlin sebagai bahasa utama untuk pengembangan aplikasi *Android* sejak 2017.

2.2.9 Confusion Matrix

Dalam masalah klasifikasi, *confusion matrix* digunakan untuk mengevaluasi prediktabilitas kelas yang ada dalam model yang mendasarinya. *Confusion Matrix* menampilkan hasil prediksi dalam format tabel yang menunjukkan empat kemungkinan hasil: *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN). Ini memungkinkan jumlah prediksi yang benar dan salah untuk setiap kelas, yang membantu mengidentifikasi kesalahan saat membuat kesalahan ini dan dengan demikian membantu dalam memahami kesalahan model saat membuat prediksi.

Dalam kinerja *Confusion Matrix*, digunakan untuk mengukur hasil dari algoritma yang telah diterapkan. Beberapa pengukuran kinerja dalam *confusion matrix* meliputi *Accuracy*, *Precision*, *Recall*, dan *F1-score* atau *F-Measure*. Berikut adalah rumus-rumus untuk mengukur kinerja algoritma menggunakan *confusion matrix*.

1. *Accuracy*

   Akurasi dihitung dengan menilai tingkat ketepatan prediksi yang benar terhadap keseluruhan data.

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TP + TN}}{\boldsymbol{TP + TN + FP + FN}} \quad \textbf{(2.6)}$$

2. *Precision*

Precision diukur untuk menilai keakuratan antara data yang diharapkan dengan hasil prediksi dari algoritma yang digunakan. Berdasarkan pernyataan ini, *Precision* adalah rasio prediksi yang dihitung menggunakan persamaan dari keseluruhan hasil yang diprediksi sebagai positif.

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FP}} \quad \textbf{(2.7)}$$

3. *Recall*

Recall diukur untuk menilai seberapa baik algoritma berhasil menemukan kembali informasi yang benar. Berdasarkan pernyataan ini, *Recall* adalah rasio yang menggambarkan keseluruhan data yang benar-benar positif.

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}} \quad \textbf{(2.8)}$$

4. *F1-Score*

Nilai *F1-Score*, atau *F-Measure*, dihitung dari kombinasi *Precision* dan *Recall* yang membandingkan kategori prediksi dengan kategori aktual.

$$\boldsymbol{F1 - Score} = \frac{\textbf{2 x } \boldsymbol{Recall} \textbf{ x Precision}}{\boldsymbol{Recall\ + Precission}} \quad \textbf{(2.9)}$$

Menurut (Firmansyah, 2021), akurasi memiliki penilaian sebagai berikut.

1. Kisaran nilai 0.90-1.00 dianggap sebagai klasifikasi yang sangat baik (*excellent classification*).
2. Kisaran nilai 0.80-0.90 dianggap sebagai klasifikasi yang baik (*good classification*).
3. Kisaran nilai 0.70-0.80 dianggap sebagai klasifikasi yang cukup (*fair classification*).
4. Kisaran nilai 0.60-0.70 dianggap sebagai klasifikasi yang kurang baik (*poor classification*).
5. Kisaran nilai 0.50-0.0 dianggap sebagai kegagalan (*failure*).

2.2.10 Black Box

Pengujian perangkat lunak dilakukan melalui *black box*, yang melibatkan pemeriksaan komponen internal perangkat lunak tanpa memberikan perhatian apa

pun (Dace et al., 2023). Tujuan utama dari pengujian *Black Box* adalah untuk memverifikasi fungsionalitas sistem sesuai dengan persyaratan yang telah ditentukan, tanpa memperhatikan bagaimana sistem tersebut diimplementasikan secara internal. Dalam pengujian ini, tester hanya berfokus pada interaksi antara input dan output, memastikan bahwa sistem menghasilkan output yang benar dan sesuai ketika diberikan input tertentu. Dengan memeriksa struktur kode internal dengan cara ini, adalah mungkin untuk mengidentifikasi cacat atau kesalahan yang mungkin tidak jelas selama pengujian *White Box*. Hasil pengujian *Black Box* dapat memberikan jaminan bahwa perangkat lunak bekerja sebagaimana mestinya dalam berbagai skenario penggunaan, meningkatkan keandalan dan kepuasan pengguna.

BAB III
ANALISIS DAN PERANCANGAN

3.1 Analisis

3.1.1 Identifikasi Masalah

Kesegaran daging sapi adalah aspek yang sangat penting. Metode penilaian kesegaran daging sapi yang digunakan saat ini sebagian besar masih berupa penilaian visual atau sensoris yang dilakukan oleh tenaga ahli. Metode ini memiliki beberapa tantangan. Pertama, penilaian secara sensoris cenderung sangat subjektif karena berdasarkan pengalaman dan penilaian individu. Kedua, konsistensi hasil penilaian bisa menjadi masalah, terutama jika penilaian dilakukan oleh lebih dari satu individu. Oleh karena itu, terdapat kebutuhan yang mendesak untuk solusi yang dapat memberikan penilaian kesegaran daging sapi secara lebih akurat, objektif, efisien, dan konsisten. Salah satu solusi yang tampak menjanjikan adalah melalui penerapan teknologi, khususnya penggunaan *Convolutional Neural Network* (CNN) dalam aplikasi pengklasifikasian kesegaran daging sapi.

Tabel 3.1 Tabel analisis SWOT

Strengths (Kekuatan)	**Weaknesses (Kelemahan)**
Penggunaan CNN dapat memberikan penilaian yang lebih objektif dan akurat tentang kesegaran daging sapi. Metode ini juga memiliki potensi untuk lebih efisien dibandingkan dengan metode manual.	Implementasi CNN memerlukan pengetahuan mendalam tentang neural networks dan image processing. Selain itu, dibutuhkan data yang cukup besar untuk melatih model dengan efektif.
Opportunities (Peluang)	**Threats (Ancaman)**
Dengan meningkatnya teknologi dan kemampuan komputasi, aplikasi seperti ini memiliki potensi besar untuk digunakan secara luas dalam industri makanan dan pengolahan daging sapi. Ini juga dapat membantu dalam memenuhi standar keamanan makanan yang semakin ketat	Ketersediaan data yang cukup dan relevan bisa menjadi tantangan. Selain itu, dapat ada resistensi dari beberapa sektor industri atau konsumen terhadap penggunaan AI dalam penilaian kesegaran daging sapi.

Sumber: (Sasoko & Mahrudi, 2023)

3.1.2 Pemecahan Masalah

Pemecahan masalah terhadap tantangan dalam penilaian kesegaran daging sapi dapat diatasi dengan pengembangan aplikasi pengklasifikasian kesegaran daging sapi yang memanfaatkan *Convolutional Neural Network* (CNN). Aplikasi ini akan memanfaatkan keunggulan teknologi *image processing* dan *machine learning* untuk melakukan penilaian secara akurat, objektif, dan konsisten. Terakhir, mengingat potensi resistensi dari konsumen terhadap penggunaan

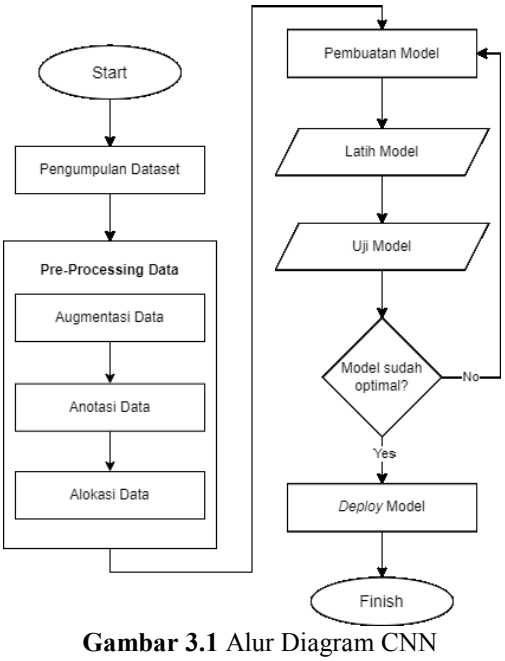
Artificial Intelligence (AI) dalam penilaian kesegaran makanan, komunikasi dan edukasi intensif kepada konsumen perlu dilakukan untuk membangun pemahaman dan kepercayaan terhadap aplikasi ini.

3.2 Perancangan

Perancangan sistem yang saya buat terdiri dari empat komponen utama, yaitu perancangan model CNN, perancangan sistem *android*, perancangan antarmuka pengguna, dan rancangan pengujian. Perancangan model CNN mencakup pengembangan arsitektur CNN untuk pemrosesan dan analisis data. Perancangan sistem *android* melibatkan pengembangan aplikasi yang berfungsi di platform *android* untuk memastikan aksesibilitas dan kegunaan yang optimal. Perancangan antarmuka pengguna fokus pada menciptakan pengalaman pengguna yang intuitif dan efisien. Terakhir, rancangan pengujian dirancang untuk memastikan bahwa semua komponen sistem bekerja dan memenuhi kriteria kinerja yang diharapkan melalui serangkaian uji coba dan evaluasi menyeluruh.

3.2.1 Perancangan Sistem CNN

Proses dalam sistem klasifikasi menggunakan *Convolutional Neural Network* (CNN) sebagai berikut:

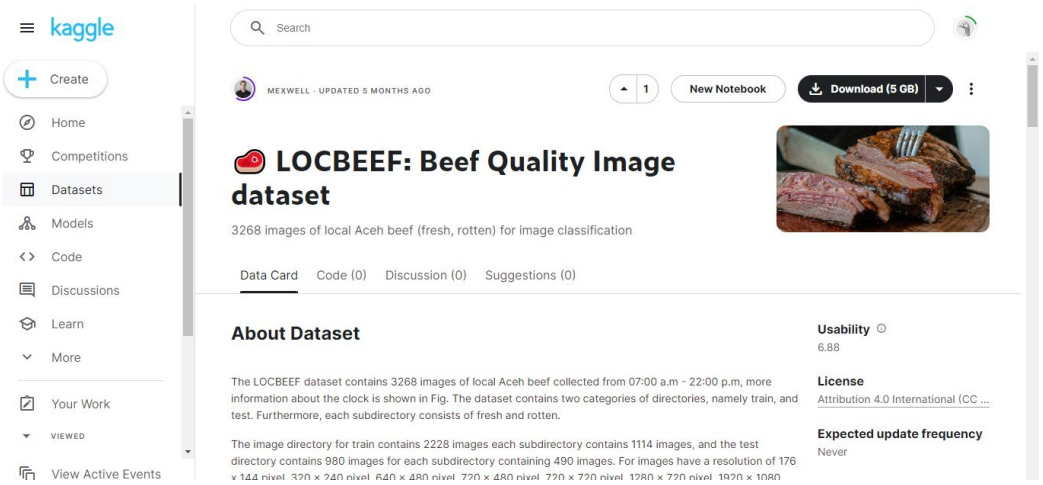


Gambar 3.1 Alur Diagram CNN

Pada perancangan sistem CNN pada penelitian ini terdapat beberapa proses yaitu pengumpulan *dataset, pre-processing data,* pembuatan model, pelatihan dan pengujian model, dan yang terakhir adalah *deploy* model. Alur perancangan dimulai dengan pengumpulan dataset, kemudian data akan diaugmentasi agar dataset yang digunakan pada proses latih (*training)* lebih bervariasi dan juga disesuaikan ukurannya sesuai dengan *pre-trained model* yang akan digunakan. Setelah itu dilakukan pelabelan pada data yang sudah dikumpulkan pada tahap sebelumnya. Berikutnya dilakukan pembagian data latih (*training*) dan validasi (validation). Tahap selanjutnya adalah modeling, yaitu merancang arsitektur yang akan digunakan untuk model yang akan digunakan pada saat proses *training* mulai dari memuat model dari library *tensorflow* dan *keras*, menambahkan beberapa *hidden layer*, dan mengatur *hyperparameter* yang digunakan pada penelitian ini. Dilanjutkan dengan proses *training* menggunakan model yang sudah dirancang di tahap sebelumnya. Data *training* dilatih menggunakan *pre-trained model* yaitu *MobileNetV2* hingga mendapatkan akurasi yang maksimal. Setelah itu dilakukan *testing model* yaitu menguji model menggunakan data uji (*test*). Apabila hasil belum maksimal maka akan dilakukan pemodelan ulang. Namun bila model sudah maksimal maka akan di *convert* dengan format tflite agar bisa disimpan di *firebase*.

3.2.1.1 Pengumpulan Dataset

Pengumpulan Dataset (*Dataset Collection*) adalah proses mengumpulkan data yang akan digunakan dalam penelitian untuk mengidentifikasi daging sapi segar dan tidak segar. Dalam penelitian ini, dataset yang digunakan berasal dari situs www.kaggle.com dengan nama "LOCBEEF: *Beef Quality Image dataset*" dari pengguna bernama Mexwell. Dataset ini terdiri dari 3268 citra, yang meliputi 2228 data latih dan 1114 data uji. Namun, dalam penelitian ini hanya digunakan 2080 data latih dan 624 data uji.



Gambar 3.2 Pengumpulan *Dataset* pada Kaggle

3.2.1.2 *Augmentasi* Data

Augmentasi data adalah proses transformasi citra asli. Pada penelitian ini melakukan proses augmentasi data menggunakan pengubahan ukuran citra (*resize*), pembalikan (*flipping*), penyesuaian kecerahan dan kontras, dan rotasi (*rotation*). Proses *resize* dilakukan dengan mengubah ukuran gambar menjadi lebar 224 *pixel* dan tinggi 224 *pixel* untuk menyesuaikan *input layer* dari arsitektur CNN *MobileNetV2*.

3.2.1.3 Anotasi Data

Pada proses anotasi data, dilakukan pelabelan objek daging sapi pada gambar untuk mendeteksi kesegaran daging sapi. Anotasi dilakukan untuk mempermudah sistem melakukan training dalam mengenali daging sapi segar dan daging sapi tidak segar.

3.2.1.4 Alokasi Data

Tahap ini melibatkan persiapan untuk pengolahan dan klasifikasi data. Setiap data tentang kesegaran daging sapi dibagi menjadi data pelatihan dan data pengujian. Data pelatihan digunakan untuk mencari hasil terbaik, sedangkan data pengujian digunakan untuk menguji model yang telah dihasilkan selama proses pelatihan. Dari 2080 citra yang tersedia, data dibagi dengan proporsi 70% untuk data pelatihan dan 30% untuk data pengujian.

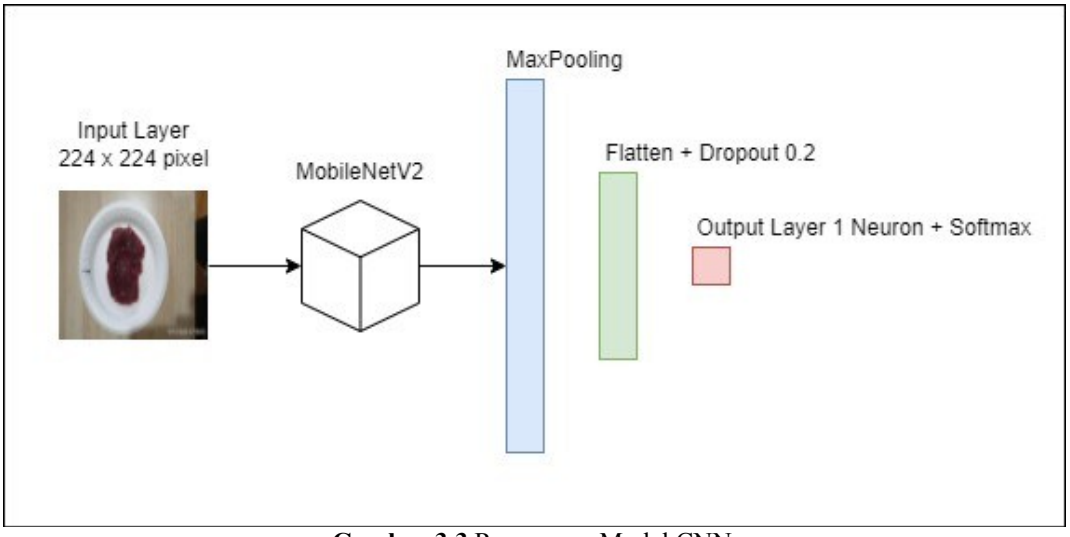
Tabel 3.2 Pembagian Data

Jenis Data	Data *Training* (70%)	Data *Testing* (30%)
Segar	728	312
Tidak Segar	728	312
Total	1456	624

Sumber: (Tri Wahyu Qur'ana, 2023)

3.2.1.5 Pembuatan Model

Pembuatan model adalah salah satu proses penting dalam pembelajaran mesin dan kecerdasan buatan yang melibatkan pemodelan, pelatihan, dan evaluasi model untuk memprediksi atau mengklasifikasikan data. Proses ini sangat penting untuk penerapan model dalam aplikasi produksi, di mana model dapat digunakan untuk melakukan prediksi secara *real-time*. Pada penelitian ini, model yang akan digunakan adalah model *MobileNetV2*, salah satu model yang sudah dilatih dan disimpan pada *library Tensorflow* dan *Keras*. Model tersebut dipilih karena ukurannya yang terbilang cukup kecil yaitu 14 MB, dan memiliki akurasi yang cukup tinggi dan komputasi yang cepat untuk diaplikasikan ke dalam sebuah perangkat bergerak. Untuk memastikan model bekerja secara optimal, dilakukan berbagai langkah penyesuaian dan peningkatan selama proses pelatihan. Pada penelitian ini, model dibuat berdasarkan referensi dari model yang sudah dibuat pada penelitian pendeteksi kesegaran ikan yang dilakukan oleh (Hanifa et al., 2023) dengan sedikit perubahan pada layer akhir. Berikut adalah gambaran dari model yang dibuat pada penelitian ini.



Gambar 3.3 Rancangan Model CNN

Dalam upaya mengoptimalkan kinerja model dalam penelitian ini dilakukan *tuning hyperparameter*. *Tuning hyperparameter* adalah bagian penting dari proses ini, yang melibatkan penyesuaian berbagai parameter seperti fungsi *loss*, *optimizer*, *learning rate*, dan *jumlah epoch*. Dengan melakukan *tuning hyperparameter* ini, kinerja model dapat dioptimalkan sehingga mampu memberikan hasil yang lebih akurat dan efisien saat digunakan untuk prediksi pada data baru. *Tuning* ini memungkinkan model untuk menyesuaikan diri dengan lebih baik terhadap dataset spesifik yang digunakan, sehingga meningkatkan kemampuan generalisasi model.

3.2.1.6 Pelatihan Model

Pelatihan model merupakan langkah penting dalam pengembangan pembelajaran mesin dan sistem kecerdasan buatan. Proses ini melibatkan pemberian data pelatihan ke model. Pembelajaran untuk mengidentifikasi pola dan hubungan dalam data dicapai dengan menyesuaikan parameter internal model. Untuk meningkatkan stabilitas pelatihan dan efisiensi komputasi, data pelatihan biasanya diurutkan menjadi beberapa kelompok kecil selama fase awal. Model ini melakukan iterasi pada setiap kumpulan data dan memperbarui bobotnya. Selama proses ini, fungsi *loss* dibuat yang mengukur perbedaan antara kesalahan dalam model dan nilai *real* melalui penyesuaian laju pembelajaran, dengan optimasi tambahan oleh pengoptimal yang bertanggung jawab untuk mengurangi ukuran fungsi ini.

Dalam penelitian ini, hyperparameter seperti *learning rate* dan jumlah *epoch* diatur untuk meningkatkan kinerja model. Fungsi loss yang digunakan adalah *sparse_categorical_crossentropy*, yang cocok untuk klasifikasi multi kelas. Optimizer yang dipilih adalah Adam, dikenal karena efisiensinya dalam mengoptimalkan model, dengan learning rate diatur pada 0,0001. Selain itu, model dilatih selama 20 *epoch*, yang berarti seluruh dataset pelatihan diproses sebanyak 20 kali selama pelatihan.

Selama pelatihan, performa model terus dipantau menggunakan data validasi. Data validasi adalah bagian dari kumpulan data yang tidak digunakan untuk pelatihan, namun digunakan untuk mengukur performa model pada setiap *epoch* secara independen. Dengan memantau metrik seperti kehilangan dan

keakuratan data validasi, akan dapat menilai apakah kinerja model meningkat atau memburuk dan melakukan penyesuaian yang diperlukan. Penilaian berkelanjutan membantu mengidentifikasi masalah seperti *overfitting* di mana model terlalu terspesialisasi untuk data pelatihan dan tidak beradaptasi dengan input baru. Model yang dilatih dapat memprediksi data yang sebelumnya tidak terbayangkan setelah fase pelatihan, dan dapat diterapkan pada data baru yang belum dikenali sebelumnya.

3.2.1.7 Pengujian Model

Pengujian model (*Testing model*) adalah tahapan yang sangat penting dalam pengembangan dan penerapan sistem pembelajaran mesin. Setelah model dilatih menggunakan data pelatihan, perlu dilakukan evaluasi kinerja model terhadap data yang belum pernah dilihat oleh model sebelumnya, yaitu data uji (*data test*). Hal ini dilakukan untuk menilai generalisasi model terhadap data baru. Proses *testing* ini membantu dalam mengidentifikasi model bekerja dengan baik dalam kondisi nyata dan mampu membuat prediksi yang akurat pada data yang tidak terlihat selama pelatihan.

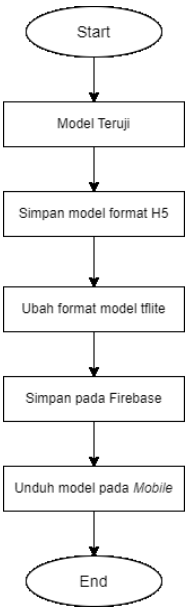
Selama tahap ini, model diuji dengan serangkaian input baru, dan hasil prediksi dibandingkan dengan label sebenarnya untuk menghitung berbagai metrik evaluasi. Metrik-metrik ini memberikan wawasan tentang kemampuan model dalam mengenali pola dan membuat prediksi. Misalnya, akurasi mengukur persentase prediksi yang benar, sedangkan presisi dan *recall* memberikan informasi lebih rinci tentang kinerja model pada kelas tertentu, yang sangat penting dalam kasus ketidakseimbangan kelas. Evaluasi yang cermat pada tahap ini sangat penting untuk memastikan model siap digunakan.

Selain itu, hasil dari tahap pengujian juga dapat mengungkap masalah yang mungkin tidak terlihat selama pelatihan, seperti *overfitting* atau *underfitting*. Model yang terlalu spesifik untuk data baru dan tidak memiliki generalisasi yang memadai dapat menjadi *overfitting*. Di sisi lain, kegagalan model untuk memasukkan pola dari data pelatihan menyebabkan *underfitting.* Dengan melakukan *testing* secara menyeluruh, dapat mengambil keputusan yang lebih baik mengenai penyesuaian model atau penggunaan teknik tambahan seperti regularisasi atau data

augmentation untuk meningkatkan kinerja model. Pengujian model yang efektif memastikan bahwa model yang dikembangkan tidak hanya berfungsi dengan baik pada data pelatihan, tetapi juga dapat diandalkan saat menangkap data baru yang tidak pernah dikenali sebelumnya.

3.2.1.8 Deployment

Deployment adalah tahap di mana model yang telah dilatih dan diuji akan diintegrasikan ke dalam sebuah aplikasi android. Pada penelitian ini, model yang telah dilatih akan disimpan ke dalam *Firebase*, sebuah platform *cloud* yang menawarkan solusi penyimpanan dan hosting yang kuat dan terintegrasi. Firebase memungkinkan penyimpanan model secara aman dan akses cepat oleh aplikasi, yang berarti model dapat diakses dan digunakan untuk prediksi secara real-time. Dengan menggunakan *Firebase*, dapat memanfaatkan keunggulan seperti sinkronisasi data secara *real-time* untuk pemindaian kesegaran daging sapi. Berikut ini adalah perancangan untuk *deploy model* pada *Firebase* dan integrasi pada *Mobile*.



Gambar 3.4 Alur Diagram *Deployment*

Proses *deployment* dimulai saat model sudah dilatih dan diuji, model tersebut akan disimpan dengan format h5. Setelah itu, format *file* tersebut diubah terlebih dahulu menjadi format *tflite* karena pada *firebase* hanya dapat menerima model dengan format tersebut. Setelah model sudah tersimpan pada firebase, nantinya

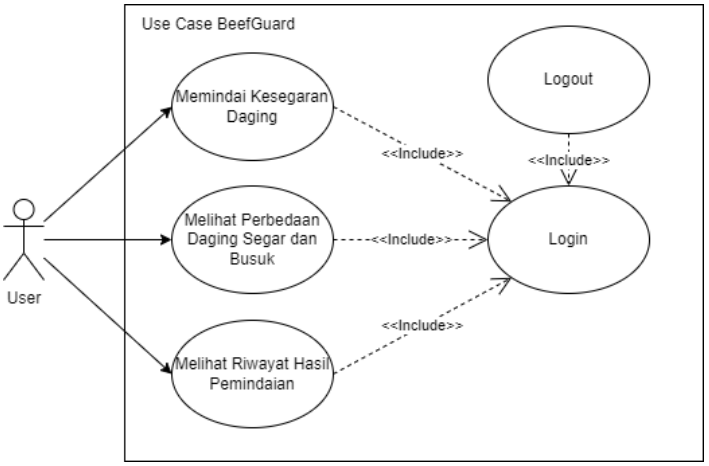
aplikasi perangkat bergerak yang sudah dirancang akan mengunduh model tersebut pada penyimpanan lokal sebelum melakukan klasifikasi.

3.2.2 Perancangan Sistem *Android*

Pada perancangan sistem *android* ini mencakup penggunaan *Unified Modeling Language* (UML) dengan fokus pada pembuatan *Use case diagram*. *Use case diagram* ini akan membantu dalam menggambarkan berbagai interaksi antara pengguna dan sistem, mengidentifikasi kebutuhan fungsional, serta memastikan bahwa semua skenario penggunaan utama terdefinisi dengan jelas. Pendekatan ini bertujuan untuk merancang sistem Android yang intuitif, efisien, dan mampu memenuhi kebutuhan pengguna secara optimal.

3.2.2.1 Perancangan *Use Case*

Interaksi antara satu atau beberapa aktor dan sistem yang sedang dirancang diuraikan dalam *Usecase diagram*. (Sintaro, 2022). Gambar dibawah ini merupakan *usecase diagram* yang berfungsi untuk mendeskripsikan interaksi *user* dengan sistem yang akan dibuat pada penelitian ini.

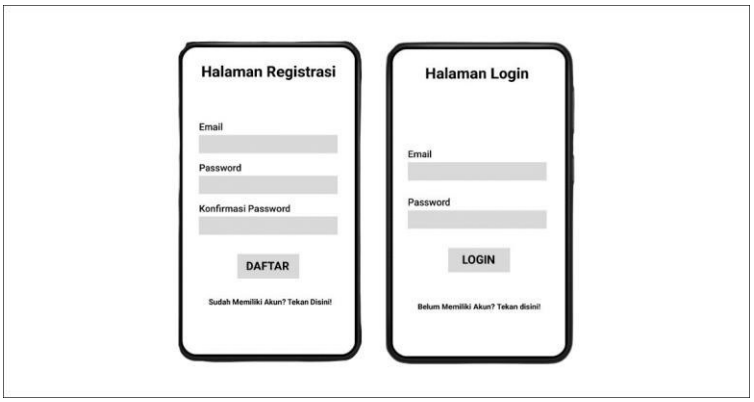


Gambar 3.5 *Usecase Diagram*

Pada *Usecase diagram* tersebut terdapat satu aktor yaitu *User* yang dapat melakukan pemindaian kesegaran daging sapi, mengetahui perbedaan daging sapi segar dan busuk dari literatur yang tersedia pada aplikasi, dan melihat riwayat dari hasil pemindaian yang sudah dilakukan sebelumnya. Namun, aktor harus melakukan *login* terlebih dahulu. Apabila sudah melakukan *login*, maka user juga dapat melakukan *logout* pada aplikasi.

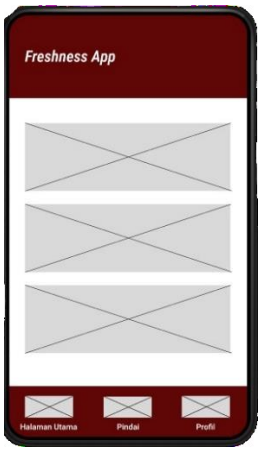
3.2.3 Perancangan *User Interface*

Pada perancangan *user interface*, alur aplikasi akan diawali dengan autentikasi, yang merupakan halaman untuk pengguna melakukan login atau registrasi terlebih dahulu. Dibawah ini adalah gambaran detail rancangan halaman autentikasi.



Gambar 3.6 Perancangan Halaman Autentikasi

Setelah melakukan proses dan berhasil melakukan autentikasi, pengguna diarahkan ke halaman utama. Halaman utama memuat beberapa informasi mengenai fitur yang ada pada aplikasi ini. Mulai dari ciri-ciri daging sapi, riwayat pemindaian daging sapi, dan halaman profil pengembang. Dibawah ini adalah gambaran detail dari rancangan halaman utama.

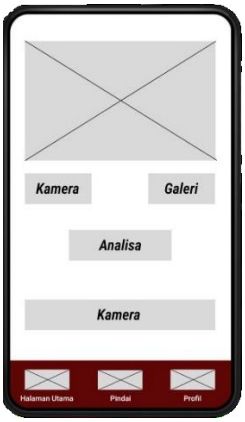


Gambar 3.7 Perancangan Halaman Utama

Pada halaman utama, apabila pengguna memilih tombol pertama, maka pengguna akan masuk ke dalam halaman ciri-ciri daging sapi segar. Halaman ini memuat informasi tentang ciri-ciri daging sapi yang masih segar. Dibawah ini adalah gambaran detail dari rancangan halaman ciri daging sapi segar.

Gambar 3.8 Perancangan Halaman Ciri Daging Sapi Segar

Terdapat halaman pindai (*scan*) yang memuat tombol untuk mengambil gambar dari kamera atau galeri. Setelah memilih gambar, pengguna dapat menggunakan tombol analisa untuk menjalankan model pengklasifikasi kesegaran daging sapi. Dibawah ini adalah gambaran detail dari halaman *scan*.

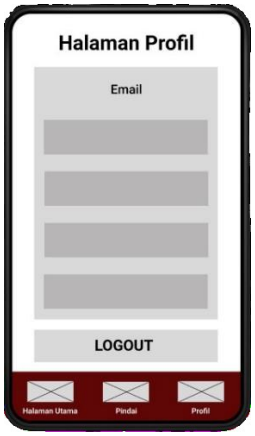


Gambar 3.9 Perancangan Halaman *Scan*

Setelah melakukan pemindaian, pengguna mendapatkan hasil analisa dari klasifikasi kesegaran daging sapi. Hasil yang didapat berupa teks yang apabila gambar daging sapi dinyatakan segar, maka sistem akan menyatakan bahwa "Daging sapi ini merupakan Daging Segar". Namun, apabila gambar daging sapi dinyatakan busuk, maka sistem akan menyatakan bahwa "Daging sapi ini merupakan Daging Tidak Segar". Dibawah ini adalah gambaran detail rancangan dari halaman hasil pindai.

Gambar 3.10 Perancangan Halaman Hasil Pindai

Pada halaman profil, pengguna dapat melihat data diri yang disimpan pada saat melakukan registrasi. Terdapat juga tombol *logout* untuk menghapus *session* yang tersimpan. Dibawah ini adalah gambaran detail dari rancangan halaman profil.



Gambar 3.11 Perancangan Halaman Profil

3.2.4 Rancangan Pengujian

Tujuan dari perancangan pengujian sistem adalah untuk memastikan bahwa sistem yang dikembangkan dapat berfungsi secara efektif dan memenuhi spesifikasi tertentu. Penelitian ini menguji kinerja model menggunakan metode pengujian *confusion matrix* dan *black box*. *Confusion Matrix* digunakan untuk menilai kinerja model klasifikasi. Dengan memberikan wawasan tentang jumlah prediksi yang benar dan salah untuk setiap jenis, hal ini dapat membantu dalam mengidentifikasi dan menganalisis kesalahan prediksi. Di sisi lain, pengujian *black box* mengevaluasi sistem berdasarkan masukan dan keluarannya tanpa melihat struktur

internalnya untuk memastikan bahwa semua fungsionalitas berfungsi seperti yang diharapkan.

3.2.4.1 Pengujian Confusion Matrix

Confusion matrix adalah teknik yang digunakan untuk mengukur tingkat akurasi atau keberhasilan model dalam mendeteksi kesegaran daging sapi. *Confusion matrix* adalah tabel yang menampilkan jumlah baris data uji yang diprediksi dengan benar atau salah oleh model klasifikasi.

3.2.4.2 Pengujian *Black Box*

Pengujian *black box* adalah pendekatan desain sistem yang memanfaatkan spesifikasi perangkat lunak. Fokusnya adalah pada hasil yang dihasilkan ketika masukan yang dipilih dan kondisi kinerja terpenuhi. Tujuan dari pengujian ini adalah untuk menilai kelayakan sistem yang telah dibangun.