# BAB IV HASIL DAN ANALISIS

## 4.1 Data

Data yang digunakan adalah data yang diperoleh dari dari laman <https://www.kaggle.com/datasets/omkargurav/face-mask-dataset>. Data tersebut memiliki dua kelas yaitu *with\_mask* dan *without mask* untuk melatih *machine learning.*

### 4.1.1 *Preprocessing*

Dalam tahap ini peneliti melakukan tahap mengelola data citra agar data tersebut dimasukkan ke pelatihan mesin dalam *Tensorflow* untuk membuat model dari CNN.

### 4.1.2 Membuat Variabel Parameter

Pada tahap ini akan mendefinisikan variabel konstanta sebagai berikut :

A black screen with white numbers

Description automatically generated

**Gambar 4.1** Inisialisasi Parameter

**Tabel 4.1** Parameter Skenario Pengujian

|  |  |  |
| --- | --- | --- |
| No. | Parameter | Value |
| 1 | Epoch | 100 |
| 2 | Target Size | 224 x 224 |
| 3 | Batch Size | 32 |
| 4 | Learning Rate | 0,0001 |

*INIT\_LR* adalah *learning rate* awal yang akan digunakan dalam algoritma pelatihan. *Learning rate* mengontrol seberapa besar langkah yang diambil saat mengoptimasi model selama pelatihan.

*EPOCHS* adalah jumlah *epoch* (iterasi) yang akan digunakan dalam pelatihan model. Setiap epoch mewakili satu kali siklus melalui seluruh dataset pelatihan.

BS (*Batch Size*) adalah ukuran *batch* yang akan digunakan selama pelatihan. Pelatihan *deep learning* sering dilakukan dalam *batch*, di mana model diperbarui setelah melihat sejumlah data.

### 4.1.3 Implementasi Memuat Data Citra

Proses selanjutnya yaitu memuat seluruh data citra yang akan dilakukan untuk pelatihan model yang diinginkan

A screen shot of a computer program

Description automatically generated

**Gambar 4.2** *Code Blok* untuk memuat data citra.

Pada gambar 4.2 seluruh blok kode ini bertanggung jawab untuk memproses gambar dari direktori "dataset", mengambil label kelas dari jalur gambar, melakukan *preprocessing* gambar sesuai dengan model yang akan digunakan, dan mengubah label menjadi format yang sesuai untuk pelatihan jaringan saraf. Data gambar yang telah diolah dan label yang telah di-*one-hot-encoded* siap digunakan untuk melatih model.

### 4.1.4 *Data Augmentation*

(Progress)

Dengan menggunakan objek aug ini, dapat menghasilkan variasi dari data pelatihan saat model dilatih, meningkatkan kemampuan umum model dalam mengenali objek yang berbeda-beda dalam berbagai kondisi. Dalam penelitian ini peneliti melakukan 3 skenario pembagian dataset sebagai berikut

(Progress Revisi)

## 4.2 Membuat Model Jaringan CNN Dengan arsitektur VGG16Net

**Sebuah gambar berisi teks, cuplikan layar, Font

Deskripsi dibuat secara otomatis**

**Gambar 4.5** Membangun *base model* VGG16Net

Pada gambar 4.5 bertujuan untuk membangun model dasar (*base model*) dengan menggunakan arsitektur *VGG16* yang telah dilatih sebelumnya pada dataset *ImageNet*. *tf.keras.applications.VGG16* merupakan fungsi yang digunakan untuk memanggil model *VGG16* dari *TensorFlow.* weights="imagenet" Ini mengindikasikan bahwa kita ingin menggunakan bobot yang telah dilatih pada *dataset ImageNet*. *include\_top=False* Ini menghilangkan lapisan *fully connected* *(top layers)* yang terhubung langsung dengan *output*, sehingga model dapat digunakan untuk ekstraksi fitur.

*input\_tensor=Input(shape=(224, 224, 3))* merupakan mendefinisikan input untuk model, yaitu gambar dengan ukuran 224x224 piksel dan 3 channel warna (RGB).

## 4.3 Implementasi Tahap Pembuatan Model

Dalam tahap ini peneliti langsung membuat proses pembuatan model CNN untuk melakukan pelatihan

Sebuah gambar berisi teks, cuplikan layar, software, Font

Deskripsi dibuat secara otomatis

**Gambar 4.6** *Code Blok* Pembuatan Model

Pada Gambar 4.6 *Code Blok* tersebut digunakan untuk membangun model *deep learning* dengan dua lapisan *output*. Model ini terdiri dari dua bagian utama, yaitu:

1. *Base model*, yang merupakan model yang sudah ada sebelumnya. Dalam kasus ini, base model adalah model VGG16, yang merupakan model convolutional neural network yang telah terbukti efektif untuk berbagai tugas pengenalan objek.
2. *Head model*, yang merupakan model tambahan yang ditambahkan di atas base model untuk menghasilkan output akhir. Head model ini terdiri dari beberapa lapisan *neural network*, yang berfungsi untuk menganalisis fitur-fitur yang dihasilkan oleh *base model* dan menghasilkan prediksi kelas.

Kode tersebut pertama-tama mengambil output dari base model, yaitu lapisan pool5. Kemudian, output tersebut diproses oleh beberapa lapisan neural network pada head model, yaitu:

1. *AveragePooling2D*, yang berfungsi untuk mengurangi ukuran output.
2. *Flatten*, yang berfungsi untuk mengubah output menjadi vektor.
3. *Dense*, yang berfungsi untuk menerapkan fungsi aktivasi relu pada vektor tersebut.
4. *Dropout*, yang berfungsi untuk mencegah *overfitting*.
5. *Dense*, yang berfungsi untuk menerapkan fungsi aktivasi *softmax* pada *vektor* tersebut.

Fungsi aktivasi *relu* berfungsi untuk memotong nilai-nilai negatif dan menjaga nilai-nilai positif. Fungsi aktivasi *softmax* berfungsi untuk mendistribusikan probabilitas pada kelas-kelas yang berbeda.

Pada akhir kode, model yang sudah dibangun tersebut diringkas menggunakan fungsi *“summary()”.* Fungsi ini akan menampilkan informasi tentang model, seperti jumlah parameter, ukuran model, dan arsitektur model.

(Progress)

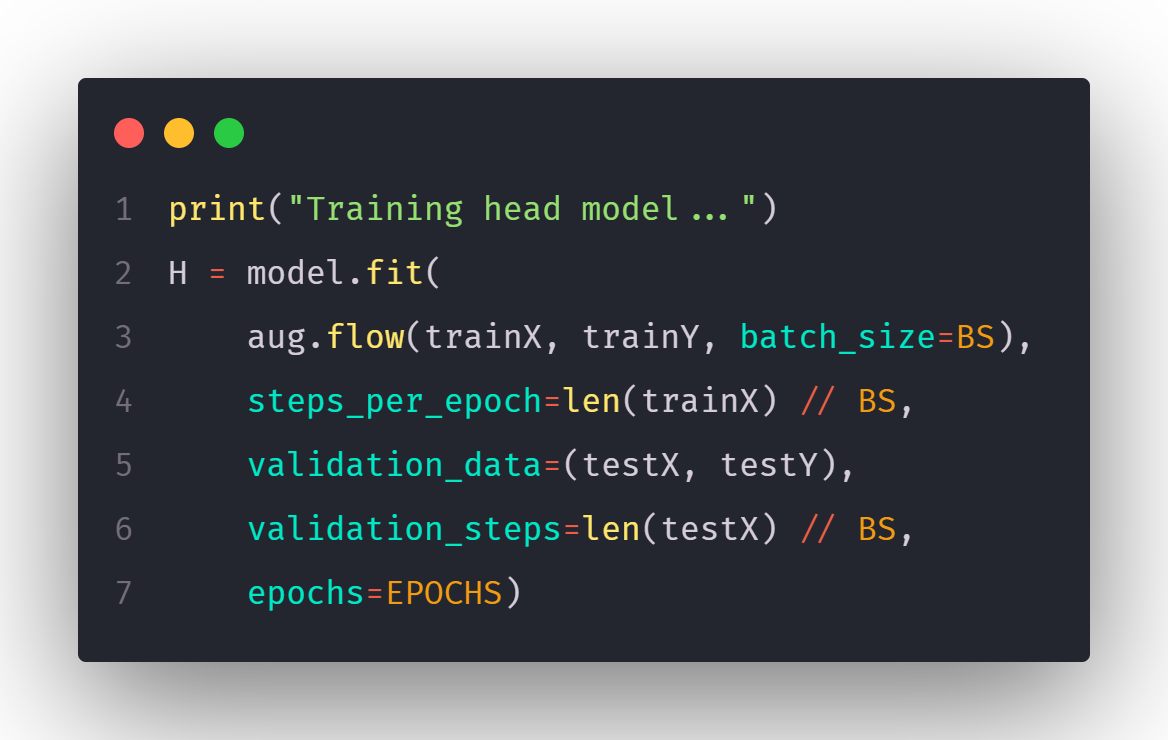
**Gambar 4.7** *Code Blok* Tahap Iterasi

Pada gambar 4.7 dalam tahap ini, pertama melakukan iterasi menggunakan *loop*, setiap lapisan dalam model dasar (VGG16 dalam kasus ini) dinyatakan sebagai tidak dapat diubah atau "*non-trainable*". Ini dilakukan dengan tujuan mematikan pelatihan pada lapisan-lapisan ini sehingga bobot dan bias mereka tidak akan diperbarui selama proses pelatihan berikutnya. Langkah ini mendukung penggunaan model dasar sebagai *ekstraktor* fitur yang stabil. Selanjutnya, kompilasi model dilakukan dengan konfigurasi yang relevan. Sebuah pengoptimalan *Adam* diatur dengan tingkat pembelajaran awal yang telah ditentukan dan tingkat pembelajaran yang berkurang seiring berjalannya *epoch (decay).* Pengoptimalan ini akan mengatur penyesuaian bobot model selama pelatihan. Model tersebut dikompilasi dengan fungsi kerugian *binary cross-entropy* (cocok untuk tugas klasifikasi biner), pengoptimalan yang telah diatur sebelumnya, serta metrik akurasi untuk evaluasi kinerja model. Akhirnya, model yang telah dikompilasi ditampilkan dalam bentuk ringkasan untuk memberikan wawasan tentang struktur keseluruhan model, termasuk lapisan-lapisan yang diikutsertakan, ukuran output, dan jumlah parameter yang dapat diatur.

Semua langkah ini membangun dasar untuk proses pelatihan model dalam penelitian ini, yang kemudian dapat diterapkan pada data pelatihan untuk menghasilkan hasil yang diharapkan sesuai dengan tujuan penelitian.

## 4.4 Implementasi Tahap Pelatihan Model

Dalam tahap ini model akan dilakukan proses *training* untuk mengetahui seberapa akurasi dalam penelitian ini



**Gambar 4.8** *Code Blok* Pelatihan Model

Pada gambar 4.8 bagian ini, dilakukan pelatihan *head model*. Fungsi *model.fit()* digunakan untuk melatih model yang telah dibuat sebelumnya dengan data pelatihan dan validasi. Data pelatihan (*trainX dan trainY*) diberikan ke dalam generator augmentasi data (*aug.flow()*) dengan ukuran *batch* yang ditentukan (*batch\_size=BS*). Jumlah langkah per *epoch* ditentukan dengan *steps\_per\_epoch=len(trainX)* // *BS*. Data validasi *(testX dan testY)* diberikan langsung ke model untuk evaluasi dan validasi, dan jumlah langkah validasi per *epoch* ditentukan dengan *validation\_steps=len(testX) // BS*.

## 4.5 Implementasi *Confusion Matrix*

Pada tahap ini, dilakukan visualisasi *confusion matrix*, suatu metode evaluasi performa model klasifikasi. *Confusion matrix* digunakan untuk memahami sejauh mana model dapat mengklasifikasikan data dengan benar. Struktur confusion matrix terdiri dari empat elemen utama, yaitu *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN).

A computer screen with many colorful text

Description automatically generated with medium confidence

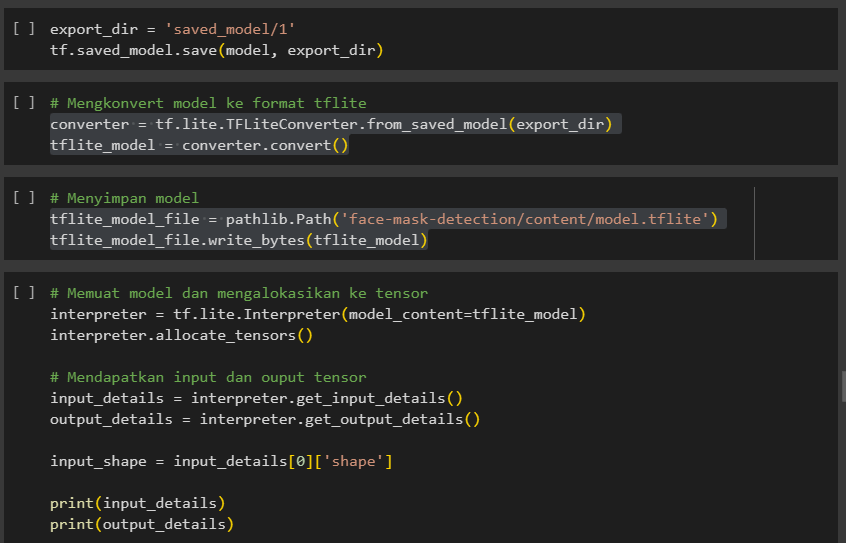
**Gambar 4.9** Implementasi *Confusion Matrix.*

Pada gambar 4.9 Berikut dalam konteks analisis model klasifikasi, konsep-konsep berikut menjadi krusial untuk memahami visualisasi *confusion matrix*. Pertama, istilah "*classes*" merujuk pada daftar kelas atau label yang digunakan dalam model klasifikasi. Selanjutnya, fungsi "*plt.imshow(...)"* dari modul *matplotlib* digunakan untuk menampilkan matriks sebagai gambar dengan skala warna hijau, menciptakan visualisasi yang memperlihatkan distribusi nilai di dalamnya. Untuk memberikan konteks lebih lanjut terhadap visualisasi ini, "plt.title(...)" digunakan untuk menambahkan judul "*Confusion Matrix*" pada plot. Pentingnya interpretasi terhadap nilai-nilai dalam matriks diperkuat dengan penambahan "*plt.colorbar()"* yang memberikan referensi visual berupa bar warna. Sementara itu, definisi label sumbu X dan Y diperoleh melalui "*tick\_marks, plt.xticks(...),* dan *plt.yticks(...)*", memberikan informasi representatif mengenai kelas-kelas pada sumbu X dan Y.

Demi penekanan visual pada nilai-nilai yang signifikan, parameter "fmt dan thresh" digunakan untuk menentukan format dan threshold yang mempengaruhi warna teks pada matriks. Untuk memperkaya informasi, "*plt.text(...)*" menambahkan teks pada setiap sel matriks, menyajikan nilai-nilai konkret di lokasi masing-masing elemen dalam matriks. Agar tata letak visual tetap rapi, "*plt.tight\_layout()*" diaplikasikan, meningkatkan kejelasan visualisasi secara keseluruhan. Akhirnya, label pada sumbu Y dan X diberikan oleh "*plt.ylabel(...)*" dan "*plt.xlabel(...)*", menyediakan panduan yang mempermudah interpretasi terhadap sumbu-sumbu tersebut. Dengan demikian, keseluruhan konsep ini memberikan fondasi yang kuat untuk memahami dan mengevaluasi performa model klasifikasi melalui visualisasi *confusion matrix.*

## 4.6 Implementasi menyimpan dan konversi model

Pada bagian implementasi menyimpan dan konversi model, dilakukan ekspor (*export*) model ke format *TensorFlow SavedModel* dan konversi ke format *TensorFlow Lite* untuk implementasi di lingkungan perangkat seluler atau sumber daya terbatas lainnya.



**Gambar 4.10** Implementasi menyimpan dan konversi model.

Tahap implementasi menyimpan dan mengonversi model dimulai dengan menentukan direktori di mana model akan disimpan dalam format *TensorFlow SavedModel*. Model tersebut kemudian disimpan di direktori yang telah ditentukan. Setelah itu, dilakukan konversi model tersebut ke dalam format *TensorFlow Lite* menggunakan objek *konverter TensorFlow Lite*. Model yang telah dikonversi disimpan sebagai file .tflite pada lokasi yang telah ditentukan sebelumnya. Selanjutnya, sebuah *interpreter TensorFlow Lite* dibuat untuk model yang baru saja dikonversi, dengan alokasi memori yang sesuai.

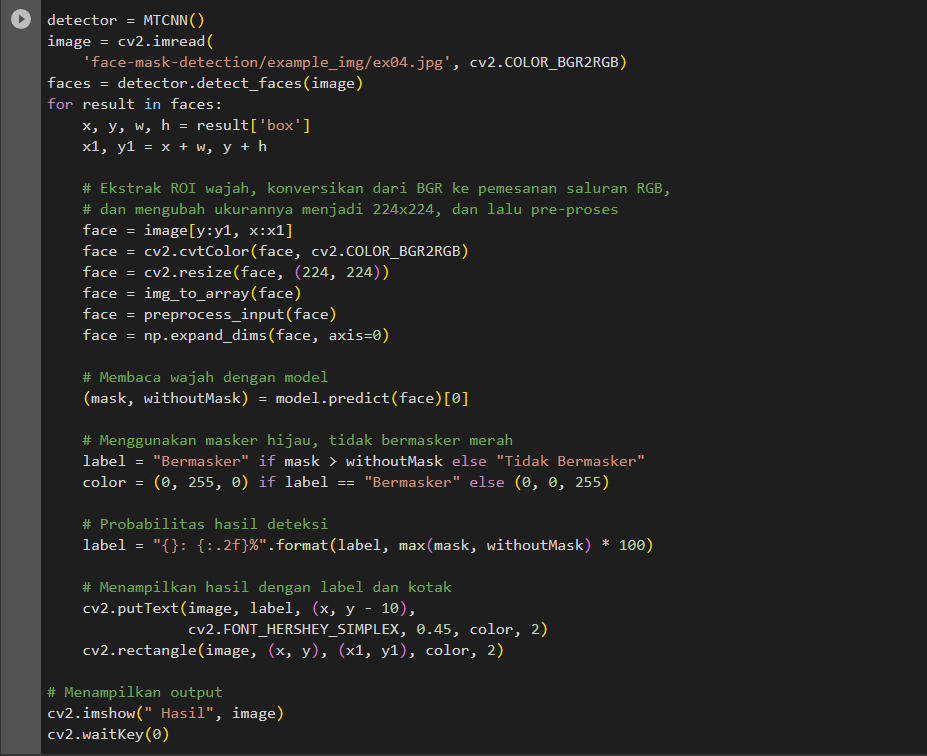
Informasi mengenai detail *input* dan *output* dari model *TensorFlow Lite* diperoleh untuk keperluan analisis lebih lanjut. Terakhir, bentuk (shape) dari input model *TensorFlow Lite* diekstraksi untuk memahami karakteristik input yang diharapkan. Semua langkah ini disajikan secara terstruktur untuk memfasilitasi pemahaman proses implementasi ini pada tahapan penyimpanan dan konversi model.

## 4.7 Implementasi Pengujian Model dengan MTCNN

Dalam tahap ini, dilakukan implementasi pengujian model deteksi masker menggunakan MTCNN (*Multi-task Cascaded Convolutional Networks*). Pertama, dilakukan inisialisasi detektor wajah menggunakan MTCNN. Selanjutnya, citra yang akan diuji dibaca menggunakan *OpenCV*, dengan konversi warna dari BGR ke RGB. Detektor MTCNN kemudian digunakan untuk mendeteksi wajah pada citra, dan hasil deteksi disimpan dalam variabel *faces*

Melalui iterasi pada hasil deteksi, koordinat dan dimensi kotak pembatas wajah diambil untuk mengekstrak *region of interest* (*ROI*) wajah. *ROI* wajah kemudian diubah ukurannya menjadi 224x224, diubah menjadi format array, dan diproses sesuai dengan preprocessing yang diterapkan pada model. Setelah itu, model digunakan untuk memprediksi apakah wajah tersebut menggunakan masker atau tidak.

Hasil prediksi kemudian digunakan untuk menentukan label ("Bermasker" atau "Tidak Bermasker") dan warna yang sesuai (hijau untuk bermasker dan merah untuk tidak bermasker). Probabilitas hasil deteksi juga ditampilkan sebagai bagian dari label. Hasil akhir ditampilkan pada citra asli dengan penambahan label dan kotak pembatas berwarna.



**Gambar 4.11** Implementasi Pengujian Model dengan MTCNN

Dengan implementasi pada gambar 4.11 ini, model deteksi masker diuji pada citra yang memiliki wajah, dan hasilnya ditampilkan secara visual dengan penambahan label dan kotak pembatas pada setiap wajah yang terdeteksi.

## 4.8 Hasil Pengujian Skenario Training

Pada semua hasil penelitian pengujian ini akan menggunakan konfigurasi sebagai berikut:

**Tabel 4.2** Parameter Untuk Pengujian Skenario.

|  |  |  |
| --- | --- | --- |
| No. | Parameter | Value |
| 1 | Epoch | 30, 50, dan 100 |
| 2 | Target Size | 224 x 224 RGB |
| 3 | Batch Size | 32 |
| 4 | Learning Rate | 0,0001 |

Tabel 4.5 berisi parameter-parameter yang digunakan dalam pengujian skenario. Parameter pertama adalah "*Epoch*" dengan nilai 100, yang mengacu pada jumlah iterasi saat melatih model. Parameter kedua adalah "*Target Size*" dengan nilai 224 x 224 RGB, yang menunjukkan dimensi target gambar yang digunakan dalam proses pelatihan. Parameter ketiga adalah "*Batch Size*" dengan nilai 32, yang menunjukkan jumlah sampel yang diproses dalam satu iterasi pelatihan. Parameter terakhir adalah "*Learning Rate*" dengan nilai 0,0001, yang mengindikasikan seberapa besar perubahan bobot model terjadi pada setiap langkah pembelajaran.

**Tabel 4.3** Parameter Data Augmentasi

|  |  |
| --- | --- |
| Parameter | Penjelasan |
| rotation\_range=20 | Rentang rotasi dalam derajat untuk memutar gambar. |
| zoom\_range=0.15 | Rentang level zoom-in dan zoom-out pada gambar. |
| width\_shift\_range=0.2 | Rentang pergeseran horizontal gambar. |
| height\_shift\_range=0.2 | Rentang pergeseran vertikal gambar. |
| shear\_range=0.15 | Rentang pergeseran sudut pemotongan. |
| horizontal\_flip=True | Kemungkinan untuk memutar gambar secara horizontal. |
| fill\_mode="nearest" | Cara mengisi piksel yang kosong setelah augmentasi (dalam hal ini, menggunakan piksel terdekat). |

**Tabel 4.4** Arsitektur Jaringan VGG16net

| Layer (type) | Output Shape | Param |
| --- | --- | --- |
| input\_1 (InputLayer) | (None, 224, 224, 3) | 0 |
| block1\_conv1 (Conv2D) | (None, 224, 224, 64) | 1792 |
| block1\_conv2 (Conv2D) | (None, 224, 224, 64) | 36928 |
| block1\_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2\_conv1 (Conv2D) | (None, 112, 112, 128) | 73856 |
| block2\_conv2 (Conv2D) | (None, 112, 112, 128) | 147584 |
| block2\_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3\_conv1 (Conv2D) | (None, 56, 56, 256) | 295168 |
| block3\_conv2 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3\_conv3 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3\_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4\_conv1 (Conv2D) | (None, 28, 28, 512) | 1180160 |
| block4\_conv2 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4\_conv3 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4\_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5\_conv1 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5\_conv2 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5\_conv3 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5\_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| Total params: |  | 14,714,688 |
| Trainable params: |  | 0 |
| Non-trainable params: |  | 147,44,688 |

Tabel 4.4 adalah tabel yang menggambarkan arsitektur jaringan *VGG16net.* Arsitektur ini terdiri dari beberapa layer yang dirinci dalam tabel. Setiap layer memiliki jenis (*type*), bentuk *output* (*Output Shape*), dan jumlah parameter yang digunakan (*Param*).

Jaringan *VGG16net* terdiri dari beberapa *layer Conv2D* yang digunakan untuk melakukan konvolusi pada *input* gambar. Setiap layer *Conv2D* memiliki output dengan dimensi yang berbeda-beda. Selain itu, terdapat juga layer *MaxPooling2D* yang berfungsi untuk mengurangi dimensi *spatialis* dari *output* *Conv2D*.

Arsitektur jaringan *VGG16ne*t terdiri dari 5 blok. Setiap blok terdiri dari beberapa layer *Conv2D* dan diakhiri dengan *layer MaxPooling2D*. Dimensi *output* dari setiap blok secara bertahap mengecil seiring dengan meningkatnya tingkat kompleksitas fitur yang dihasilkan.

Total parameter dalam jaringan ini adalah 14.714.688. Parameter ini merupakan jumlah total bobot yang dapat diubah oleh algoritma pelatihan. Dalam kasus ini, semua parameter dalam jaringan ini ditetapkan sebagai *non-trainable*, yang berarti nilai bobotnya tetap tidak berubah selama pelatihan.

**Tabel 4.5** Susunan Pengujian Model *Convolutional Neural Network .*

| Layer (type) | Output Shape | Param |
| --- | --- | --- |
| input\_1 (InputLayer) | (None, 224, 224, 3) | 0 |
| block1\_conv1 (Conv2D) | (None, 224, 224, 64) | 1792 |
| block1\_conv2 (Conv2D) | (None, 224, 224, 64) | 36928 |
| block1\_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2\_conv1 (Conv2D) | (None, 112, 112, 128) | 73856 |
| block2\_conv2 (Conv2D) | (None, 112, 112, 128) | 147584 |
| block2\_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3\_conv1 (Conv2D) | (None, 56, 56, 256) | 295168 |
| block3\_conv2 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3\_conv3 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3\_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4\_conv1 (Conv2D) | (None, 28, 28, 512) | 1180160 |
| block4\_conv2 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4\_conv3 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4\_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5\_conv1 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5\_conv2 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5\_conv3 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5\_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| average\_pooling2d (AveragePooling2D) | (None, 1, 1, 512) | 0 |
| flatten (Flatten) | (None, 512) | 0 |
| dense (Dense) | (None, 128) | 65664 |
| dropout (Dropout) | (None, 128) | 0 |
| dense\_1 (Dense) | (None, 2) | 258 |
| Total params: |  | 14,780,610 |
| Trainable params |  | 65,922 |
| Non-trainable params: |  | 14,714,688 |

Tabel 4.7 menampilkan susunan serta parameter-parameter dari Model *Convolutional Neural Network* yang diuji. Model ini terdiri dari beberapa jenis layer yang terhubung secara berurutan. Layer input awal (input\_1) memiliki dimensi None x 224 x 224 x 3, dengan total parameter sebesar 0.

Selanjutnya, terdapat beberapa layer konvolusi (Conv2D) dan *layer pooling* (*MaxPooling2D*) yang membentuk bagian-bagian blok (*block*) dari model ini. Setiap blok terdiri dari beberapa layer *conv2D* yang memiliki dimensi yang berbeda, dengan parameter yang berbeda pula. Blok-blok ini secara berturut-turut meningkatkan kompleksitas pemrosesan dan mengekstraksi fitur-fitur dari input gambar.

Pada akhirnya, terdapat beberapa layer lain seperti *average\_pooling2d* yang melakukan proses *pooling* rata-rata serta layer *flatten* yang mengubah *output* menjadi dimensi 1D. Terakhir, terdapat dua layer *dense* (*Dense*) yang berfungsi sebagai layer terhubung penuh (*fully connected*) dengan *output layer* terakhir berdimensi 2 untuk melakukan klasifikasi.

Total parameter dari model ini sebesar 14,780,610, dengan 65,922 parameter yang dapat diubah (*trainable*) dan sisanya 14,714,688 parameter tidak dapat diubah (*non-trainable*). Informasi ini relevan untuk memahami struktur dan kompleksitas model *Convolutional Neural Network* yang diuji dalam konteks penelitian skripsi.

Dalam konfigurasi tersebut melibatkan dua skenario *dataset* yang terbagi menjadi dua yaitu *full dataset* dan *half dataset* dengan menggunakan metode *K-fold Cross Validation* yang terdiri dari 3 fold, 5 fold, dan 10 fold dengan menggunakan 3 *optimizer* yaitu *Adam*, *SGD*, dan *RMSprop.* Lalu Ketika mendapatkan model yang terbaik maka akan diujikan dengan uji data tunggal yang melibatkan data baru citra masker untuk melihat seberapa akurat model tersebut untuk mendeteksi masker.

## 4.9 Hasil Pengujian Data Tunggal

## 