# 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 :



**Gambar 4.1** Inisialisasi Parameter

**Tabel 4.1** Parameter Skenario Pengujian

|  |  |  |
| --- | --- | --- |
| No. | Parameter | Value |
| 1 | Epoch | 100 |
| 2 | Target Size | 224 x 224 RGB |
| 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 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

Sebuah gambar berisi teks, cuplikan layar, software, Sistem operasi

Deskripsi dibuat secara otomatis

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

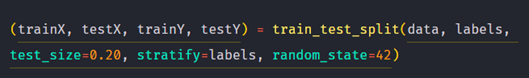
Pada *Code Blok* Gambar 4.2 ini adalah bagian dari proses persiapan data yang dibutuhkan sebelum melatih model jaringan saraf. Ini dilakukan dengan langkah-langkah sebagai berikut:

1. Pada baris pertama (print("Menginput gambar...")), pesan ini hanya mencetak ke konsol untuk memberikan informasi kepada pengguna bahwa proses input gambar sedang dimulai.
2. Baris kedua (*imagePaths* = *list(paths.list\_images("dataset")))* menghasilkan daftar jalur file gambar di direktori "dataset". Ini diasumsikan bahwa semua gambar yang akan digunakan dalam penelitian disimpan di dalam direktori ini. Fungsi *paths.list\_images()* dari modul imutils digunakan untuk mengambil semua jalur file gambar dalam direktori tersebut.
3. Dua variabel, data dan labels, diinisialisasi sebagai list kosong pada baris keempat dan kelima.
4. Blok berikutnya (*for imagePath in imagePaths:*) merupakan *loop* yang akan mengiterasi melalui setiap jalur *file* gambar dalam *imagePaths.*
5. Baris pertama dalam *loop* (*label =* *imagePath.split(os.path.sep)[-2])* mengambil label kelas dari jalur file. Ini dilakukan dengan membagi jalur menggunakan pemisah jalur sistem operasi *(os.path.sep)* dan mengambil elemen kedua dari belakang (-2).
6. Baris berikutnya *(image = load\_img(imagePath, target\_size=(224, 224)))* memuat gambar dari jalur yang ditentukan dan mengubahnya menjadi ukuran 224x224 piksel. Kemudian, gambar diubah menjadi bentuk array menggunakan *img\_to\_array().* Selanjutnya, gambar diproses menggunakan fungsi *preprocess\_input()* yang berkaitan dengan model yang digunakan (VGG16Net, dalam hal ini).
7. Baris terakhir dalam *loop (data.append(image)* dan *labels.append(label))* menambahkan gambar yang telah dimuat dan label yang telah diekstrak ke dalam *list* data dan *labels*.
8. Setelah *loop* selesai, data dan label dalam bentuk *list* diubah menjadi *array* NumPy untuk digunakan dalam pelatihan model. Baris pertama (data = np.array(data, dtype="float32")) mengubah list data menjadi array NumPy dengan tipe data float32.
9. Baris selanjutnya (lb = LabelBinarizer()) membuat objek LabelBinarizer untuk melakukan one-hot encoding pada label. One-hot encoding mengubah label kelas menjadi bentuk biner di mana setiap kelas direpresentasikan oleh vektor biner dengan nilai 1 di indeks yang sesuai dengan kelasnya.
10. labels = lb.fit\_transform(labels) melakukan one-hot encoding pada label. Fungsi fit\_transform() dari LabelBinarizer akan mengonversi label ke dalam format one-hot encoding.
11. labels = to\_categorical(labels) mengonversi label yang telah di-one-hot-encoded menjadi format kategorikal menggunakan fungsi to\_categorical() dari TensorFlow.
12. Pada baris terakhir *(print("Input gambar berhasil")),* pesan ini dicetak untuk memberitahu pengguna bahwa proses pengolahan gambar telah selesai dan data gambar telah berhasil diinput.

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 Membuat objek *Data Generator* dan *Data Augmentation*

Setelah data citra dimuat maka langsung membuat *data generator* dan *data augmentation*



**Gambar 4.3** Mempartisi Data

Tahap ini melakukan pembagian data menjadi dua bagian yaitu data pelatihan dan pengujian. Dalam hal ini, data gambar yang telah diolah sebelumnya (*data*) dan label yang di-*one-hot-encoded* (labels) digunakan. Parameter test\_size=0.20 mengalokasikan 20% data sebagai pengujian dan 80% sebagai pelatihan. Stratifikasi dengan *stratify=labels* mempertahankan distribusi kelas yang seimbang antara kedua *subset*, sementara *random\_state=42* memberikan konsistensi dalam hasil pembagian setiap kali kode dijalankan.

Sebuah gambar berisi teks, cuplikan layar, tampilan, software

Deskripsi dibuat secara otomatis

**Gambar 4.4** Membuat *Data Augmentation*

*ImageDataGenerator* adalah kelas yang disediakan oleh *TensorFlow* untuk melakukan augmentasi data gambar secara otomatis selama pelatihan model. Augmentasi data adalah teknik untuk meningkatkan variasi data pelatihan dengan membuat variasi baru dari gambar yang ada, seperti rotasi, *zoom*, pergeseran, pemotongan, dll. Ini membantu model belajar dengan lebih baik dan mengurangi *overfitting*.

Dalam *Code Blok* ini, aug adalah objek *ImageDataGenerator* yang telah dibentuk dengan sejumlah parameter untuk mengatur jenis augmentasi yang akan diterapkan pada gambar-gambar pelatihan

**Tabel 4.2** Parameter *Data Augmentation*

|  |  |
| --- | --- |
| 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). |

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

**Tabel 4.3** Pembagian *Full Dataset*

|  |  |  |
| --- | --- | --- |
| Rasio | Data Training | Data Testing |
| 90/10 | 3449 | 384 |
| 80/20 | 3066 | 767 |
| 70/30 | 2683 | 1150 |

**Tabel 4.4** Pembagian *Half Dataset*

|  |  |  |
| --- | --- | --- |
| Rasio | Data Training | Data Testing |
| 90/10 | 1440 | 160 |
| 80/20 | 1280 | 320 |
| 70/30 | 1120 | 480 |

## 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

Potongan kode 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.

**Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Deskripsi dibuat secara otomatis**

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

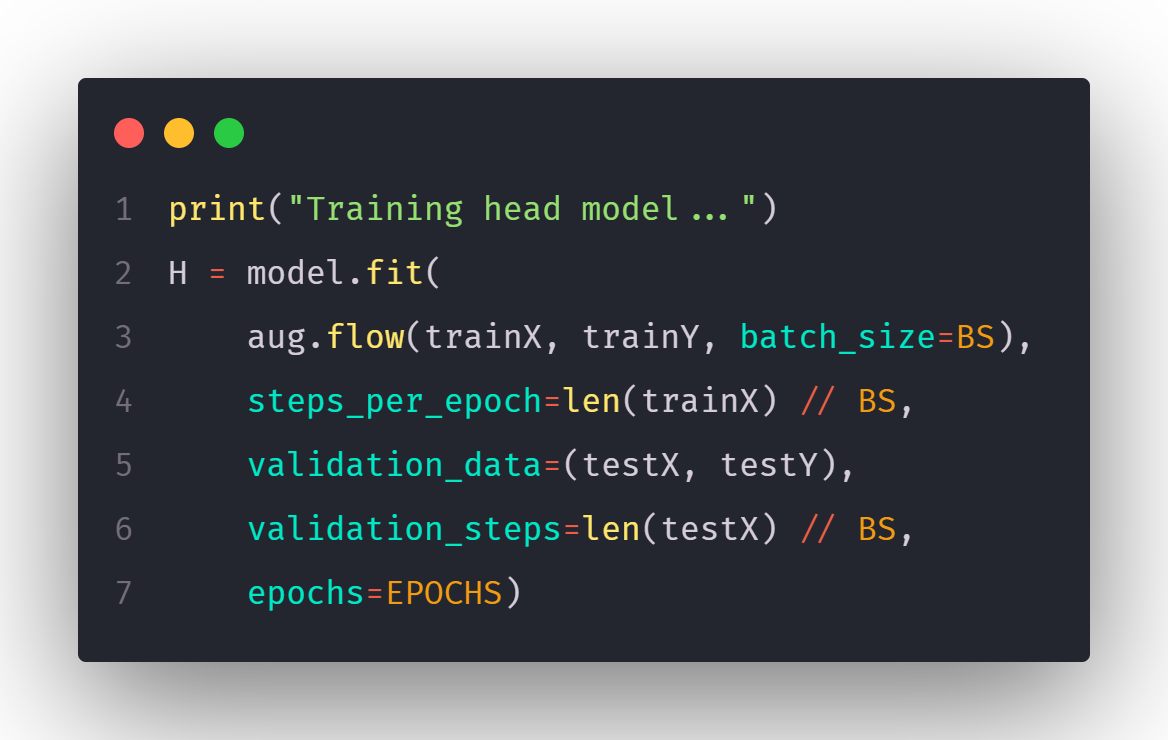
Pada gambar 4.7 dalam tahap ini, pertama melakukan iterasi menggunakan *loop*, setiap lapisan dalam model dasar (VGG16 dalam kasus ini) dinyatakan sebagai tidak dapat diubah atau "*non-trainable*". Ini dilakukan dengan tujuan mematikan pelatihan pada lapisan-lapisan ini sehingga bobot dan bias mereka 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), pengoptimal yang telah diatur sebelumnya, serta metrik akurasi untuk evaluasi kinerja model. Akhirnya, model yang telah diompilasi 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.

|  |  |  |
| --- | --- | --- |
| 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 |

## 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*. Proses pelatihan dijalankan selama jumlah epoch yang telah ditentukan *(epochs=EPOCHS)*. Hasil pelatihan, seperti loss dan akurasi, akan disimpan dalam variabel H untuk analisis lebih lanjut.

## 4.5 Hasil Pengujian Skenario

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

**Tabel 4.5** Parameter Untuk Pengujian Skenario.

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

**Tabel 4.6** Tabel 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 4.7** 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 |

Dalam konfigurasi tersebut melibatkan dua skenario dataset yang terbagi menjadi dua yaitu *full dataset* dan *half dataset* dengan menggunakan metode split data dan *K-fold Cross Validation* yang terdiri dari 3 fold, 5 fold, dan 7 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.

### 4.5.1 Skenario Full Dataset Adam

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.9** Adam Full Dataset 90/10

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

A green squares with white numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

**Gambar 4.10** Adam Full Dataset 80/20

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

A green squares with white numbers

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Gambar 4.11** Adam Full Dataset 70/30

### 4.5.2 Skenario Full Dataset RMSProp

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.12** RMSProp Full Dataset 90/10

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screen shot of a black screen

Description automatically generated

**Gambar 4.13** RMSProp Full Dataset 80/20

A graph of a graph

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Gambar 4.14** RMSProp Full Dataset 70/30

### 4.5.3 Skenario Full Dataset SGD

A graph of different colored lines

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

**Gambar 4.15** SGD Full Dataset 90/10

A graph of different colored lines

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.16** SGD Full Dataset 80/20

A graph of different colored lines

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.17** SGD Full Dataset 70/30

### 4.5.4 Skenario Half Dataset Adam

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.18** Adam Half Dataset 90/10

A graph of a function

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.19** Adam Half Dataset 80/20

A graph of a graph of a function

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

**Gambar 4.20** Adam Half Dataset 70/30

### 4.5.5 Skenario Half Dataset RMSProp

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.21** RMSProp Half Dataset 90/10

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screen shot of a black screen

Description automatically generated

**Gambar 4.22** RMSProp Half Dataset 80/20

A graph of a function

Description automatically generated with medium confidence

A green squares with white numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Gambar 4.23** RMSProp Half Dataset 70/30

### 4.5.6 Skenario Half Dataset SGD

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Gambar 4.24** SGD Half Dataset 90/10

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

A green squares with white numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

**Gambar 4.25** SGD Half Dataset 80/20

A graph of a graph of a graph

Description automatically generated with medium confidence

A green squares with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Gambar 4.26** SGD Half Dataset 70/30

### 4.5.7 Hasil Akurasi Tanpa Data Augmentasi Full Dataset

**Tabel 4.8** Hasil Akurasi tanpa Data Augmentasi Full Dataset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Optimizer | Split Data | | |
| 70/30 | 80/20 | 90/10 |
| 1 | Adam | 0.71 | 0.64 | 0.57 |
| 2 | RMSProp | 0.56 | 0.57 | 0.50 |
| 3 | SGD | 0.47 | 0.48 | 0.53 |

### 4.5.8 Hasil Akurasi Dengan Data Augmentasi Full Dataset

**Tabel 4.9** Hasil Akurasi Dengan Data Augmentasi Full Dataset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Optimizer | Split Data | | |
| 70/30 | 80/20 | 90/10 |
| 1 | Adam | 0.98 | 0.99 | 0.98 |
| 2 | RMSProp | 0.99 | 0.99 | 0.99 |
| 3 | SGD | 0.90 | 0.89 | 0.86 |