LAPORAN PEMBELAJARAN MESIN

JARINGAN KONVOLUSIONAL

Gregorius Gama Abi Surya [1)],Puti Windrahmatullah [2)], Mario Desendi Manurung[3)], Nabilah Wahyu Hafizhah [4)], Very Andreas [5)],

Program Studi Sains Data, Jurusan Sains, Institut Teknologi Sumatera
Email : gregorius.120450018@student.itera.ac.id [1)], puti.120450070@student.itera.ac.id [2)], mario.120450082@student.itera.ac.id [3)], nabilah.120450106@student.itera.ac.id [4)], very.120450110@student.itera.ac.id [5)]

1. PENDAHULUAN

Convolutional Neural Network (CNN) adalah salah satu jenis algoritma Deep Learning yang banyak digunakan dalam pemrosesan data gambar. Konsep dasar dari CNN didasarkan pada operasi matematika yang disebut konvolusi, yang memungkinkan pengolahan data citra dan identifikasi fitur-fitur yang penting untuk tugas spesifik seperti pengenalan wajah atau deteksi objek. CNN memiliki kemampuan untuk mempelajari pola-pola kompleks dalam gambar melalui lapisan-lapisan penghubung yang terdiri dari filter-filter konvolusi, yang menerapkan operasi konvolusi pada gambar input untuk menghasilkan representasi-fitur yang lebih tinggi. Kemudian, representasi-fitur ini diolah lebih lanjut melalui lapisan-lapisan lain seperti lapisan pooling dan lapisan fully connected, sebelum akhirnya menghasilkan output yang mencerminkan hasil prediksi atau klasifikasi yang diinginkan. Dengan demikian, CNN telah menjadi algoritma yang sangat efektif dalam bidang pengolahan gambar dan telah diterapkan secara luas dalam berbagai aplikasi seperti pengenalan objek, deteksi wajah, pengolahan citra medis, dan lainnya.

2. ALGORITMA

Beberapa konsep dasar yang menjadi landasan teori CNN antara lain:

2.1 Konvolusi

Konvolusi adalah suatu operasi matematika yang digunakan dalam Convolutional Neural Network (CNN) untuk memproses data input dan menghasilkan representasi-fitur yang relevan. Dalam konteks CNN, konvolusi dilakukan dengan menerapkan filter

atau kernel pada citra input. Filter ini, berupa matriks dengan bobot yang dapat dipelajari, bergerak secara bergeser melintasi seluruh citra dengan tujuan untuk mengekstraksi fitur-fitur penting pada setiap posisi.

Dalam proses konvolusi, filter tersebut secara berulang diterapkan pada setiap bagian citra dengan melakukan perkalian titik antara elemen-elemen filter dan piksel-piksel yang sesuai pada citra. Hasil perkalian titik ini kemudian dijumlahkan untuk menghasilkan nilai yang merepresentasikan fitur pada posisi tersebut. Dengan melakukan konvolusi, CNN dapat menangkap pola-pola lokal yang signifikan dalam citra, seperti tepi, sudut, atau tekstur.

Proses konvolusi ini penting dalam CNN karena memungkinkan model untuk mempelajari fitur-fitur visual yang penting dan relevan untuk tugas tertentu, seperti pengenalan wajah atau deteksi objek. Melalui pembelajaran yang berulang, CNN dapat mengoptimalkan bobot filter agar dapat menangkap fitur-fitur yang spesifik dari data input. Dengan demikian, konvolusi menjadi landasan teori yang penting dalam pengolahan data gambar menggunakan CNN.

2.2 Pooling

Pooling adalah suatu operasi yang penting dalam Convolutional Neural Network (CNN) yang bertujuan untuk mengurangi ukuran data input dan menghilangkan redundansi informasi yang tidak perlu. Dalam CNN, pooling dilakukan dengan membagi citra input menjadi daerah-daerah kecil dan mengambil nilai rata-rata atau nilai maksimum dari setiap daerah tersebut.

Proses pooling dapat dilakukan dengan berbagai metode, seperti max pooling dan average pooling. Pada max pooling, nilai maksimum diambil dari setiap daerah kecil pada citra input, sementara pada average pooling, nilai rata-rata diambil. Dengan mengambil nilai ekstrim atau rata-rata dari setiap daerah, pooling membantu mengurangi kompleksitas data dan mempertahankan informasi penting.

Tujuan dari operasi pooling adalah mengurangi dimensi data dan menghilangkan redundansi informasi, sehingga memudahkan pemrosesan pada layer-layer selanjutnya dalam CNN. Selain itu, pooling juga membantu dalam menciptakan invarian spasial terhadap translasi pada fitur-fitur yang diidentifikasi oleh konvolusi. Dengan mengurangi ukuran data, pooling juga membantu dalam mengontrol overfitting dan mengurangi beban komputasi yang diperlukan dalam proses pelatihan.

Dalam keseluruhan arsitektur CNN, konvolusi dan pooling bekerja bersama-sama untuk memproses data input, mengekstraksi fitur-fitur yang penting, dan menghasilkan representasi-fitur yang lebih sederhana dan terfokus. Dengan adanya konsep konvolusi dan pooling ini, CNN dapat menjadi algoritma yang sangat efektif dalam pengolahan data gambar dan telah sukses diterapkan dalam berbagai tugas pengenalan objek, deteksi wajah, dan analisis citra lainnya.

2.3 Aktivasi

Dalam Convolutional Neural Network (CNN), aktivasi merupakan salah satu komponen penting yang digunakan untuk memperkenalkan non-linearitas dalam model. Fungsi aktivasi, seperti ReLU (Rectified Linear Unit), digunakan untuk memberikan respons non-linear terhadap input yang diberikan kepada setiap neuron dalam jaringan CNN.

Dalam CNN, setelah proses konvolusi dan pooling, hasilnya akan melewati fungsi aktivasi yang ditentukan, seperti ReLU. Fungsi ReLU mengubah semua nilai negatif menjadi nol, sementara nilai non-negatif tetap tidak berubah. Hal ini memungkinkan CNN untuk menangkap hubungan non-linear antara fitur-fitur dalam citra, sehingga memperluas kapasitas model untuk mempelajari representasi-fitur yang lebih kompleks.

Fungsi aktivasi ReLU sangat populer dalam CNN karena sederhana dalam perhitungan dan membantu dalam mengatasi masalah gradien yang menghilang (vanishing gradient problem) yang terjadi dalam jaringan neural yang lebih dalam. Dengan mengaktifkan neuron-neuron dalam jaringan CNN, fungsi aktivasi ReLU membantu meningkatkan

kemampuan model untuk mempelajari representasi-fitur yang lebih baik dan menghasilkan prediksi yang akurat.

Selain ReLU, terdapat juga berbagai fungsi aktivasi lain yang digunakan dalam CNN, seperti fungsi sigmoid, tangen hiperbolik (tanh), dan fungsi aktivasi luaran softmax untuk klasifikasi multi-kelas. Pemilihan fungsi aktivasi yang tepat sangat penting dalam merancang arsitektur CNN yang efektif dan sesuai dengan tugas yang dihadapi.

2.4 Arsitektur CNN

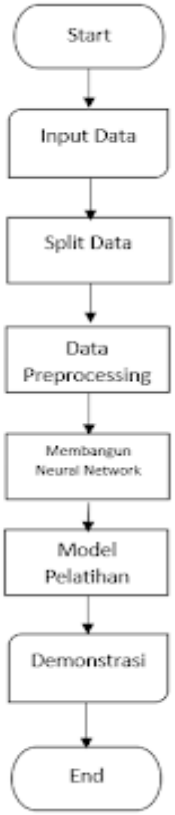
Convolutional Neural Network (CNN) memiliki arsitektur yang terstruktur dan kompleks, yang terdiri dari berbagai lapisan atau layer yang berperan dalam memproses data gambar secara hierarkis. Arsitektur ini dirancang untuk memungkinkan model CNN untuk secara efektif mempelajari representasi-fitur pada input citra dengan tingkat abstraksi yang semakin kompleks.

Pertama, terdapat lapisan konvolusi dalam arsitektur CNN, dimana filter atau kernel diterapkan pada citra input untuk mengekstraksi fitur-fitur visual yang penting. Lapisan konvolusi ini memungkinkan model untuk menangkap pola-pola lokal pada citra.Selanjutnya, terdapat lapisan pooling, yang bertujuan untuk mengurangi dimensi data input dan menghilangkan redundansi informasi dengan mengambil nilai rata-rata atau maksimum dari daerah-daerah kecil pada citra. Lapisan pooling membantu menyederhanakan representasi-fitur yang dihasilkan dari lapisan konvolusi sebelumnya.Setelah itu, terdapat lapisan fully-connected (sepenuhnya terhubung), yang berfungsi untuk menghubungkan setiap neuron dalam lapisan sebelumnya dengan setiap neuron dalam lapisan ini. Lapisan fully-connected ini memungkinkan model CNN untuk menggabungkan dan memproses fitur-fitur yang telah diekstraksi secara global, sehingga dapat membuat keputusan atau prediksi berdasarkan fitur-fitur tersebut.

Kombinasi dari lapisan konvolusi, pooling, dan fully-connected dalam arsitektur CNN memungkinkan model untuk mempelajari representasi-fitur secara bertahap

dan hierarkis, di mana fitur-fitur yang lebih kompleks dibangun dari fitur-fitur yang lebih sederhana. Dengan cara ini, model CNN dapat mengenali pola-pola yang semakin kompleks pada citra dan melakukan tugas-tugas seperti pengenalan objek atau klasifikasi dengan tingkat akurasi yang tinggi.

3. FLOWCHART



4. PEMROGRAMAN

4.1 Data Sawi

4.1.1 Import Library

```
[ ]  import os
     import shutil
     import random
     import pandas as pd
     import matplotlib.pyplot as plt
     import numpy as np
     import tensorflow as tf
     keras = tf.keras
```

Pada kode di atas, terdapat beberapa baris yang mengimpor modul dan library yang diperlukan untuk melakukan berbagai operasi pada data, visualisasi, dan pembangunan model neural network. Modul os, shutil, dan random digunakan untuk operasi file dan direktori, seperti membaca daftar file, membuat direktori, memindahkan atau menghapus file atau direktori, serta melakukan pengacakan angka atau elemen. Modul pandas akan digunakan untuk membaca dan memanipulasi data dalam format tabel, sementara modul matplotlib.pyplot akan digunakan untuk membuat plot atau grafik visualisasi data. Modul numpy akan digunakan untuk melakukan operasi matematika pada array atau matriks multidimensi, dan modul tensorflow akan digunakan untuk pengembangan dan pelatihan model machine learning. Dalam hal ini, modul keras dari TensorFlow digunakan dengan alias keras untuk mempermudah penggunaan API keras dalam kode.

Dengan mengimpor modul-modul ini, kita dapat memanfaatkan berbagai fungsi dan fitur yang disediakan oleh masing-masing modul untuk melakukan manipulasi data, visualisasi, serta membangun dan melatih model neural network dengan lebih efisien. Modul dan library yang diimpor dalam kode tersebut membantu dalam menjalankan operasi yang dibutuhkan untuk tujuan pengembangan dan analisis data, serta mempermudah proses pengembangan model machine learning dengan menggunakan TensorFlow dan Keras.

4.1.2 Split Data

```
[ ]  def loading_data():
         dirname = '/content/drive/MyDrive/SAWI/'
         dirtype = ['train/', 'valid/']
         dirclass = ['Data Sawi Ada Hama/', 'Data Sawi Tanpa Hama/']

         x_train = []
         y_train = []
         x_test = []
         y_test = []

         for typ in dirtype :
             for cls in dirclass :
                 for i in os.listdir(dirname+typ+cls):
                     if typ == 'train/':
                         x_train.append(dirname+'train/'+cls+i)
                         y_train.append(cls[:-1])
                     else:
                         x_test.append(dirname+'valid/'+cls+i)
                         y_test.append(cls[:-1])
         return np.array(x_train), np.array(y_train), np.array(x_test), np.array(y_test)

     #Run loading data() function
     x_train, y_train, x_test, y_test = loading_data()
```

Fungsi loading_data() di atas merupakan implementasi yang digunakan untuk memuat data citra dalam tugas pengenalan hama pada tanaman sawi. Pada awalnya, direktori utama yang berisi data citra ditentukan dengan menggunakan variabel dirname yang menunjukkan jalur file di Google Drive. Selanjutnya, terdapat variabel dirtype yang merupakan daftar yang berisi dua tipe direktori, yaitu "train/" dan "valid/", yang digunakan untuk data latih dan validasi/pengujian.

Selanjutnya, variabel dirclass merupakan daftar yang berisi dua kelas data, yaitu "Data Sawi Ada Hama/" dan "Data Sawi Tanpa Hama/", yang mewakili citra-citra sawi dengan atau tanpa hama. Kemudian, empat variabel x_train, y_train, x_test, dan y_test diinisialisasi sebagai array kosong yang akan digunakan untuk menyimpan path file dan label kelas citra latih dan pengujian.

Pada baris ke-7 hingga 19, terdapat tiga perulangan yang menggunakan nested loops untuk mengiterasi melalui tipe direktori (train/ dan valid/) dan kelas direktori (Data Sawi Ada Hama/ dan Data Sawi Tanpa Hama/). Setiap file citra dalam direktori tersebut ditambahkan ke x_train dan y_train jika berada dalam direktori train/, atau ditambahkan ke

x_test dan y_test jika berada dalam direktori valid/.Dalam setiap iterasi, jalur lengkap ke file citra ditambahkan ke x_train atau x_test sesuai dengan tipe direktori yang sedang diperiksa. Sementara itu, label kelas citra yang diperoleh dengan menghapus karakter '/' dari nama direktori ditambahkan ke y_train atau y_test. Terakhir, fungsi loading_data() mengembalikan empat array NumPy: x_train, y_train, x_test, dan y_test, yang berisi path file dan label kelas citra untuk data latih dan pengujian.

Fungsi ini memainkan peran penting dalam memuat dan mempersiapkan data citra yang akan digunakan dalam pelatihan dan evaluasi model pengenalan hama pada tanaman sawi.

4.1.3 Data Preprocessing

```
#DATA Preprocessing
#Processing image files into numpy array
def process_label(label):
    label = [i == unique_label for i in label]
    label = np.array(label).astype(int)
    return label

def processImage(path):
    image = tf.io.read_file(path)
    image = tf.image.decode_jpeg(image,channels =3)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, size=[224,224])
    return image

#Create Batch Data of NumPy Array
def pairData(image, label):
    return processImage(image), label

def batchData(image, label = None, for_valid= False, for_test = False):
    if for_test:
        data = tf.data.Dataset.from_tensor_slices((image))
        batch = data.map(processImage).batch(32)
        return batch
    elif for_valid:
        data = tf.data.Dataset.from_tensor_slices((tf.constant(image),tf.constant(label)
        batch = data.map(pairData).batch(32)
        return batch
    else:
        data = tf.data.Dataset.from_tensor_slices((tf.constant(image),tf.constant(label)
        data = data.shuffle(buffer_size=len(image))
        batch = data.map(pairData).batch(32)
        return batch

unique_label = np.unique(y_test)
y_test = process_label(y_test)
y_train = process_label(y_train)

train_data = batchData(x_train, y_train)
valid_data = batchData(x_test, y_test, for_valid = True)

x_train, y_train, x_test, y_test = loading_data()
x_train, x_test
y_train, y_test
```

Fungsi-fungsi yang ada dalam kode tersebut bertujuan untuk memproses dan memuat data citra. Proses dimulai dengan mengubah label kelas menjadi representasi numerik menggunakan fungsi `process_label`. Selanjutnya, citra-citra dibaca dan diproses

menggunakan fungsi `processImage` agar sesuai dengan format yang diinginkan. Fungsi `pairData` digunakan untuk menggabungkan citra-citra yang telah diproses dengan labelnya.

Setelah proses awal selesai, dilakukan pembuatan batch data menggunakan fungsi `batchData` dengan memasukkan array citra dan label sebagai input. Dalam proses ini, label kelas juga diubah menjadi representasi numerik menggunakan fungsi `process_label`. Kemudian, dilakukan pembuatan data latih dan validasi menggunakan fungsi `batchData`, dengan data latih diacak agar tidak terjadi bias dalam proses pembelajaran.

Terakhir, data citra dan label dimuat menggunakan fungsi `loading_data`. Hasilnya, terdapat array `x_train` dan `x_test` yang berisi path file citra.Output yang ditampilkan adalah array yang berisi label kelas untuk setiap citra dalam data. Dalam kasus ini, semua label kelas pada array tersebut memiliki nilai "Data Sawi Ada Hama" dan "Data Sawi Tanpa Hama".

4.1.4 Membangun Neural Network

```
[ ]  #Create model structure
     model = keras.Sequential([
         #Input Layer
         keras.layers.Conv2D(input_shape = (224,224,3), filters=32, kernel_size=(3,3), activation = 'relu'),
         keras.layers.MaxPooling2D(),

         #Hidden Layer
         keras.layers.Conv2D(input_shape = (224,224,3), filters = 64, kernel_size=(3,3), activation = 'relu'),
         keras.layers.MaxPooling2D(),
         keras.layers.Conv2D(input_shape = (224,224,3), filters = 128, kernel_size=(3,3), activation = 'relu'),
         keras.layers.MaxPooling2D(),
         keras.layers.Flatten(),
         keras.layers.Dense(128),
         keras.layers.Activation('relu'),

         #Output Layer
         keras.layers.Dense(2),
         keras.layers.Activation('softmax')
     ])

     # Compile model
     model.compile(loss = 'categorical_crossentropy', optimizer = keras.optimizers.Adam(), metrics = ['acc'])

     model.summary()
```

Dalam kode tersebut, kita menggunakan TensorFlow's Keras API untuk membuat struktur model. Model ini terdiri dari beberapa jenis layer seperti konvolusi, max pooling, flatten, dan fully connected. Kita mulai dengan layer konvolusi yang menggunakan 32 filter

dengan ukuran kernel 3x3 dan menggunakan fungsi aktivasi ReLU. Kemudian, kita melakukan max pooling untuk mengurangi dimensi output sebelumnya. Proses ini diulang dengan jumlah filter yang berbeda pada layer konvolusi berikutnya. Setelah itu, kita menggunakan layer flatten untuk mengubah tensor 3D menjadi vektor 1D agar dapat dihubungkan dengan layer fully connected. Layer fully connected terdiri dari 128 neuron dengan fungsi aktivasi ReLU. Selanjutnya, kita memiliki layer output dengan 2 neuron yang mewakili jumlah kelas output yang diinginkan, dan menggunakan fungsi aktivasi softmax untuk menghasilkan probabilitas kelas. Model ini dikompilasi dengan menggunakan fungsi loss categorical cross-entropy, optimizer Adam, dan metrik evaluasi akurasi. Terakhir, kita menampilkan ringkasan dari struktur model yang mencakup jumlah parameter yang dapat diubah dan jumlah output dari setiap layer. Model ini biasanya digunakan dalam pengenalan citra dengan menggunakan arsitektur Convolutional Neural Network (CNN).

Hasil output tersebut adalah representasi dari struktur model neural network "sequential". Model ini terdiri dari beberapa layer, termasuk layer konvolusi, max pooling, flatten, dan fully connected. Jumlah filter pada setiap layer konvolusi secara berurutan adalah 32, 64, dan 128. Setelah setiap layer konvolusi, dilakukan max pooling untuk mengurangi dimensi citra. Terdapat dua layer fully connected, masing-masing dengan 128 dan 2 unit. Fungsi aktivasi diaplikasikan setelah setiap layer fully connected. Total parameter dalam model ini adalah 11,169,218, yang akan diubah selama proses pelatihan. Tidak ada parameter yang tidak dapat diubah dalam model ini.

4.1.5 Model Pelatihan

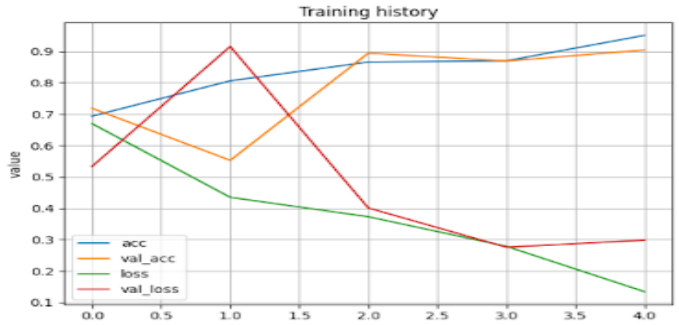
```
import keras
import os
import matplotlib.pyplot as plt
if not os.path.exists('output'):
    os.makedirs('output')
#Model Pelatihan
#Buat panggilan balik penghentian awal
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)

#Simpan riwayat model menjadi variabel
history = model.fit(train_data, validation_data = valid_data, validation_freq=1, epochs = 5,callbacks = [early_stopping],verbose = 1,)

#Plot model training history
def plot_history():
    plt.plot(history.history['acc'], label='acc')
    plt.plot(history.history['val_acc'], label = 'val_acc')
    plt.plot(history.history['loss'], label = 'loss')
    plt.plot(history.history['val_loss'], label= 'val_loss')
    plt.legend()
    plt.title('Training history')
    plt.xlabel('epoch')
    plt.ylabel('value')
    plt.tight_layout()
    plt.grid(True)
    plt.savefig('output/training_history.jpg')
    plt.show()

plot_history()
```

```
Epoch 1/5
25/25 [==============================] - 158s 6s/step - loss: 0.6696 - acc: 0.6938 - val_loss: 0.5335 - val_acc: 0.7186
Epoch 2/5
25/25 [==============================] - 121s 5s/step - loss: 0.4349 - acc: 0.8062 - val_loss: 0.9157 - val_acc: 0.5528
Epoch 3/5
25/25 [==============================] - 124s 5s/step - loss: 0.3730 - acc: 0.8662 - val_loss: 0.4089 - val_acc: 0.8945
Epoch 4/5
25/25 [==============================] - 138s 5s/step - loss: 0.2787 - acc: 0.8700 - val_loss: 0.2758 - val_acc: 0.8693
Epoch 5/5
25/25 [==============================] - 131s 5s/step - loss: 0.1341 - acc: 0.9513 - val_loss: 0.2983 - val_acc: 0.9045
```



Dalam kode di atas, kita mengimpor modul yang diperlukan dan melakukan pelatihan model menggunakan metode fit(). Objek EarlyStopping digunakan sebagai callback untuk menghentikan pelatihan jika tidak ada peningkatan dalam val_loss. Riwayat pelatihan disimpan dalam variabel history. Selanjutnya, kita mendefinisikan fungsi plot_history() untuk membuat grafik yang menampilkan perubahan akurasi dan loss dari setiap epoch. Grafik tersebut memberikan informasi tentang performa model dan tren pembelajaran selama pelatihan.

Hasil output menampilkan nilai loss dan akurasi pada setiap epoch, serta waktu yang dibutuhkan untuk melatih satu epoch. Pada grafik, garis biru menunjukkan akurasi pelatihan, garis jingga menunjukkan akurasi validasi, garis hijau menunjukkan loss pelatihan, dan garis merah menunjukkan loss validasi. Peningkatan akurasi pelatihan dan penurunan loss pelatihan menunjukkan bahwa model sedang belajar dengan baik. Pada akhir pelatihan (epoch ke-5), diperoleh loss validasi sebesar 0.2983 dan akurasi validasi sebesar 0.9045. Grafik training history memberikan gambaran tentang performa model dalam mempelajari pola-pola yang ada dalam data latih dan validasi.

Dengan demikian, kode ini menggambarkan pelatihan model neural network menggunakan Keras API. Model ini dirancang untuk pengenalan citra dengan menggunakan arsitektur Convolutional Neural Network (CNN). Pelatihan model ini mencapai hasil yang baik dengan akurasi validasi sebesar 0.9045 pada epoch terakhir. Grafik training history memberikan visualisasi yang membantu dalam mengevaluasi performa model selama pelatihan.

4.1.6 Model Lain

```
[ ]  # Create mode structure
     import keras
     import keras.utils
     from keras.models import Sequential
     from keras.layers import Dense, Dropout, Activation
     from tensorflow.keras import optimizers
     model2 = keras.Sequential([
         # Input layer
         keras.layers.Conv2D(input_shape=(224,224,3),filters=32,kernel_size=(3,3),activation='relu'),
         keras.layers.MaxPooling2D(),

         # Hidden Layer
         keras.layers.Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),activation='relu'),
         keras.layers.MaxPooling2D(),
         keras.layers.Conv2D(input_shape=(224,224,3),filters=128,kernel_size=(3,3),activation='relu'),
         keras.layers.MaxPooling2D(),
         keras.layers.Flatten(),
         keras.layers.Dense(128),
         keras.layers.Activation('relu'),

         # Output layer
         keras.layers.Dense(2),
         keras.layers.Activation('sigmoid')
     ])

     # Compile model
     model2.compile(loss='categorical_crossentropy',optimizer=keras.optimizers.SGD(),metrics=['acc'])
```

Kode di atas menggunakan modul dan kelas dari Keras dan TensorFlow untuk membangun sebuah model neural network dengan arsitektur Convolutional Neural Network (CNN) untuk pengenalan citra. Model ini terdiri dari beberapa layer yang ditambahkan secara sekuensial menggunakan objek Sequential. Layer-layer tersebut meliputi layer konvolusi, max pooling, flatten, dan fully-connected. Model ini memiliki fungsi aktivasi ReLU dan sigmoid, serta menggunakan categorical_crossentropy sebagai loss function dan SGD sebagai optimizer. Ringkasan struktur model menampilkan informasi tentang jumlah parameter yang dapat diubah dan output dari setiap layer. Model ini memiliki jumlah parameter sebesar 11,169,218 yang dapat diubah. Dalam pelatihan model, ditemukan overfitting dan perbedaan kinerja antara data training dan data validasi pada epoch ke-2.

```
[ ]  # buat panggilan bail penghentian awal
     early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss',patience=3)

     # simpan riwayat model menjadi variabel
     history2 = model2.fit(train_data,validation_data = valid_data,validation_freq=1,epochs=15,callbacks=[early_stopping],verbose = 1)

     Epoch 1/15
     25/25 [==============================] - 115s 5s/step - loss: 0.6652 - acc: 0.5962 - val_loss: 0.7730 - val_acc: 0.5075
     Epoch 2/15
     25/25 [==============================] - 116s 5s/step - loss: 0.6196 - acc: 0.6388 - val_loss: 0.7270 - val_acc: 0.5075
     Epoch 3/15
     25/25 [==============================] - 113s 5s/step - loss: 0.5589 - acc: 0.7113 - val_loss: 0.7430 - val_acc: 0.5176
     Epoch 4/15
     25/25 [==============================] - 115s 5s/step - loss: 0.5065 - acc: 0.7513 - val_loss: 0.7772 - val_acc: 0.5126
     Epoch 5/15
     25/25 [==============================] - 112s 4s/step - loss: 0.5090 - acc: 0.7550 - val_loss: 0.6810 - val_acc: 0.5477
     Epoch 6/15
     25/25 [==============================] - 113s 5s/step - loss: 0.4592 - acc: 0.7925 - val_loss: 0.7480 - val_acc: 0.5779
     Epoch 7/15
     25/25 [==============================] - 112s 4s/step - loss: 0.4388 - acc: 0.7975 - val_loss: 0.8044 - val_acc: 0.5025
     Epoch 8/15
     25/25 [==============================] - 123s 5s/step - loss: 0.4735 - acc: 0.7713 - val_loss: 0.7648 - val_acc: 0.5427
```

Pada baris kode `early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)`, kita membuat objek EarlyStopping yang akan digunakan untuk mengimplementasikan early stopping. Early stopping adalah teknik yang memungkinkan model berhenti melakukan pelatihan jika tidak ada perbaikan yang signifikan dalam loss pada data validasi. Dalam kasus ini, kita menggunakan 'val_loss' sebagai monitor untuk melihat perubahan loss pada data validasi. Patience yang ditentukan adalah 3, yang berarti model akan berhenti jika tidak ada perbaikan dalam loss pada data validasi selama 3 epoch berturut-turut.

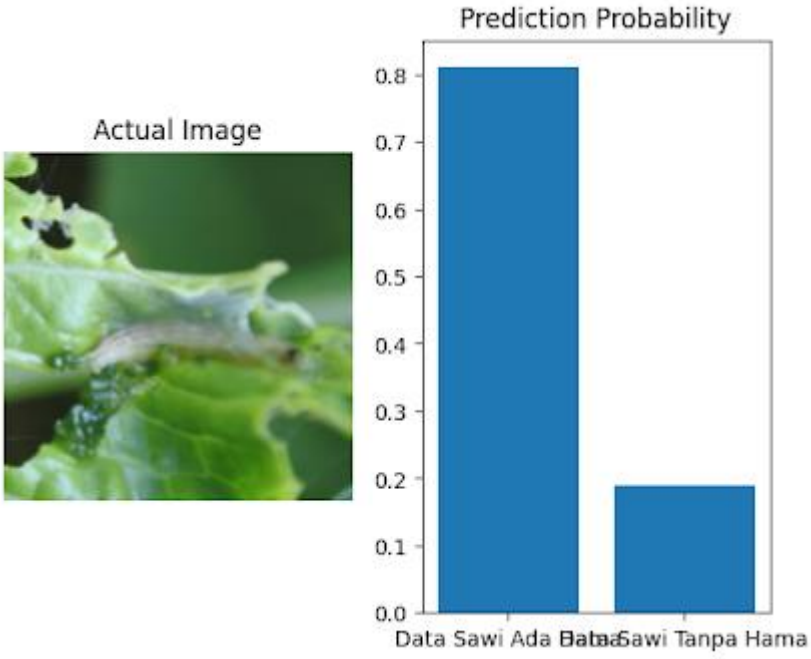
Pada baris kode `history2 = model2.fit(train_data, validation_data=valid_data, validation_freq=1, epochs=15, callbacks=[early_stopping], verbose=1)`, kita melakukan pelatihan model dengan menggunakan metode fit() pada objek model2. train_data adalah data pelatihan yang digunakan, valid_data adalah data validasi. Dengan menggunakan validation_freq=1, validasi akan dilakukan setiap epoch. Jumlah epoch yang ditentukan adalah 15. Callbacks [early_stopping] digunakan untuk menerapkan early stopping yang telah kita definisikan sebelumnya. Dengan verbose=1, kita akan melihat informasi detail mengenai proses pelatihan model.

Hasil output yang diberikan adalah informasi mengenai proses pelatihan model. Setiap epoch menampilkan nilai loss (loss) dan akurasi (acc) pada data pelatihan (25/25) dan data validasi (val_loss, val_acc). Output ini memberikan gambaran tentang kinerja model selama pelatihan, dengan melihat perubahan loss dan akurasi pada setiap epoch.

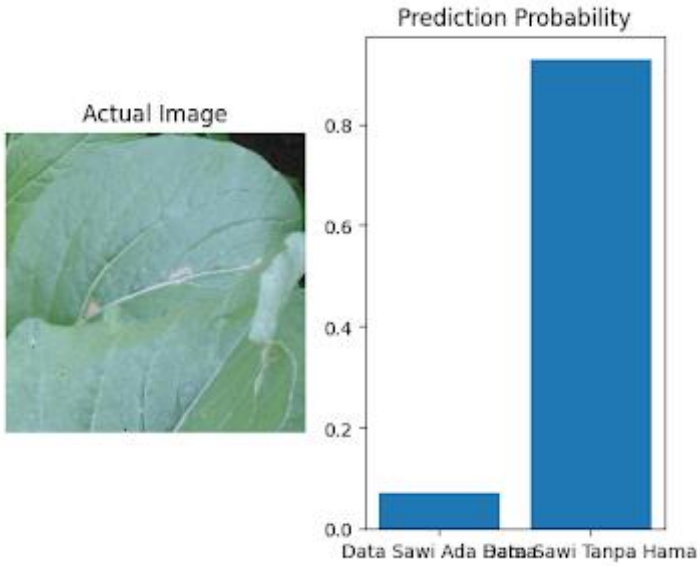
Grafik yang dihasilkan menunjukkan perubahan akurasi dan loss pada setiap epoch selama proses pelatihan model. Garis biru merepresentasikan akurasi pada data pelatihan (acc), garis jingga merepresentasikan akurasi pada data validasi (val_acc), garis hijau merepresentasikan loss pada data pelatihan (loss), dan garis merah merepresentasikan loss pada data validasi (val_loss). Melalui grafik ini, kita dapat menganalisis kinerja model dan mengidentifikasi adanya overfitting atau underfitting.

Namun, berdasarkan hasil pelatihan model dengan 8 epoch, terlihat bahwa akurasi validasi masih rendah, berkisar antara 50-60%, dan tidak menunjukkan perbaikan yang signifikan seiring dengan penambahan epoch. Hal ini mengindikasikan bahwa model yang digunakan mungkin terlalu sederhana untuk mempelajari pola yang kompleks dalam data. Untuk meningkatkan kinerja model, perlu dilakukan penyesuaian model seperti menambahkan lebih banyak layer atau mengubah arsitektur model, atau dapat juga menggunakan teknik lain seperti transfer learning.

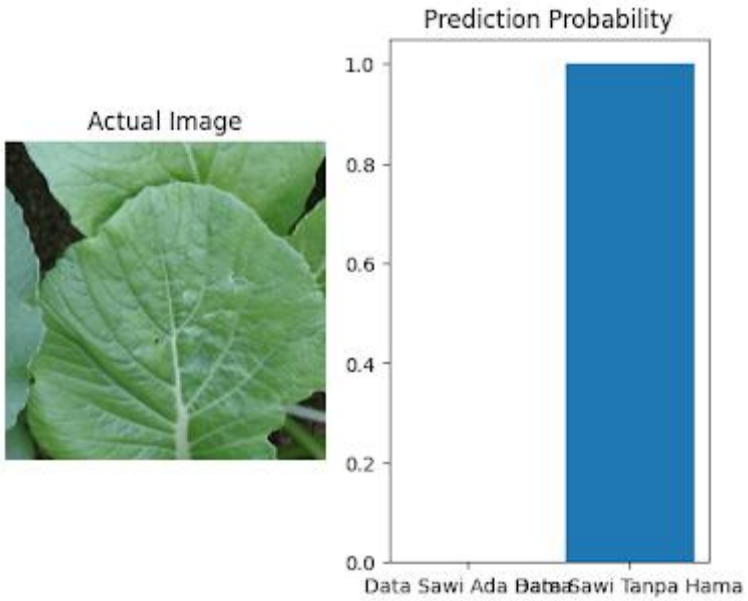
4.1.7 Demonstrasi



Pada Gambar Diatas Dilihat Bahwa Sawi Ada Hama dengan Prediksi Probabilitas 0.8 dan tanpa hama 0.2



Gambar Diatas dlihat bahwa Sawi ada hama dengan prediksi probabilitas 0.1 dengan sawi tanpa hama > 0.8



Gambar Diatas dilihat bahwa Sawi Ada hama dengan prediksi probabilitas 0 dan sawi tanpa sebesar 1.0

4.2) Data Masker

4.2.1 Import Data

```
[ ]  import os
     import shutil

     # mengakses dataset awal
     data_dir = '/content/drive/MyDrive/data/'

     # membuat informasi class
     all_class = {'train':{},'valid':{}}
     for cl in os.listdir(data_dir):
         items = os.listdir(f'{data_dir}/{cl}')
         test_count = int(0.2*(len(items)))
         all_class['train'][cl] = items[test_count:]
         all_class['valid'][cl] = items[:test_count]
```

Kode tersebut akan membuat sebuah dictionary bernama all_class yang terdiri dari dua bagian yaitu data latih dan data validasi. Setiap bagian terdiri dari dictionary lagi yang berisi nama class sebagai key dan list item sebagai value. List item tersebut berisi nama file gambar yang termasuk dalam class tersebut dan telah terpisah menjadi data latih atau data validasi.

Pada baris pertama, import os dan import shutil adalah library Python yang digunakan untuk mengakses file dan direktori.Selanjutnya, data_dir = '/content/drive/MyDrive/data/' adalah path dari direktori dataset yang ingin diakses. Kemudian dilakukan iterasi pada setiap class di dalam dataset menggunakan os.listdir() dan disimpan dalam variabel items.Selanjutnya, pada bagian test_count = int(0.2*(len(items))), variabel test_count akan berisi 20% dari jumlah total item pada class tersebut, yang nantinya akan dijadikan data validasi.

Kemudian, pada bagian all_class['train'][cl] = items[test_count:], items[test_count:] akan berisi data latih dari class tersebut yang dimulai dari urutan item ke-20%+1 sampai dengan item terakhir. Sedangkan pada all_class['valid'][cl] = items[:test_count], items[:test_count] akan berisi data validasi dari class tersebut yang dimulai dari item pertama sampai dengan urutan item ke-20%. Dengan menggunakan kode tersebut, dataset dapat dipisahkan menjadi data latih dan data validasi dengan pembagian 80:20 untuk setiap class-nya.

4.2.2 Split Data

```
[ ]  for category,cl in all_class.items():
         for sub_class,item in cl.items():
             for i in item:
                 print(f'{category}/{sub_class}/{i}')
                 des_path = f'dataset/{category}/{sub_class}/'
                 os.makedirs(des_path, exist_ok = True)
                 shutil.move(f'{data_dir}/{sub_class}/{i}',des_path)
```

```
[ ]  def loading_data():
         dirname = '/content/dataset/'
         dirtype = ['train/', 'valid/']
         dirclass = ['with_mask/', 'without_mask/']

         x_train = []
         y_train = []
         x_test = []
         y_test = []

         for typ in dirtype:
             for cls in dirtype:
                 for cls in dirclass:
                     for i in os.listdir(dirname+typ+cls):
                         if typ == 'train/':
                           x_train.append(dirname+'train/'+cls+i)
                           y_train.append(cls[:-1])
                         else:
                             x_test.append(dirname+'valid/'+cls+i)
                             y_test.append(cls[:-1])
         return np.array(x_train), np.array(y_train), np.array(x_test), np.array(y_test)

     # Run loading_data() function
     x_train, y_train, x_test, y_test = loading_data()
```

Pada bagian awal, didefinisikan dirname = '/content/dataset/' sebagai path dari direktori dataset. Kemudian, didefinisikan juga dirtype = ['train/', 'valid/'] yang berisi jenis dataset yang akan dimuat, yaitu data latih dan data validasi. Selanjutnya, didefinisikan dirclass = ['with_mask/', 'without_mask/'] yang berisi class pada dataset.

Kemudian, pada bagian x_train = [], y_train = [], x_test = [], dan y_test = [] akan digunakan untuk menyimpan gambar dan label dari data latih dan data validasi.

Pada bagian selanjutnya, menggunakan nested loop untuk iterasi pada setiap class pada setiap jenis dataset.

Pada bagian for i in os.listdir(dirname+typ+cls):, setiap file gambar pada class tersebut akan dimuat menggunakan os.listdir() dan diiterasi menggunakan for. Jika jenis dataset adalah data latih, maka file gambar akan disimpan di variabel x_train dan label classnya disimpan di variabel y_train. Jika jenis dataset adalah data validasi, maka file gambar akan disimpan di variabel x_test dan label classnya disimpan di variabel y_test.

Variabel cls[:-1] digunakan untuk membuang tanda '/' pada nama class, sehingga hanya tersisa 'with_mask' atau 'without_mask' saja.

Pada bagian terakhir, data latih dan data validasi akan dikembalikan dalam bentuk array numpy menggunakan np.array().

4.2.3 Data Preprocessing

```
[ ]  # Processing image
     def process_label(label):
         label = [i == unique_label for i in label]
         label = np.array(label).astype(int)
         return label

     def processImage(path):
         image = tf.io.read_file(path)
         image = tf.image.decode_jpeg(image,channels=3)
         image = tf.image.convert_image_dtype(image,tf.float32)
         image = tf.image.resize(image,size=[224,224])
         return image
```

```
# Create Batch data of numpy array return processImage(image), label
def pairData(image, label):
    return processImage(image), label

def batchData(image,label=None, for_valid=False, for_test=False):
    if for_test:
        data = tf.data.Dataset.from_tensor_slices((image))
        batch = data.map(processImage).batch(32)
        return batch
    elif for_valid:
        data = tf.data.Dataset.from_tensor_slices((tf.constant(image), tf.constant(label)))
        batch = data.map(pairData).batch(32)
        return batch
    else:
        data = tf.data.Dataset.from_tensor_slices ((tf.constant(image), tf.constant(label)))
        data = data.shuffle(buffer_size=len(image))
        batch = data.map(pairData).batch(32)
        return batch

unique_label = np.unique(y_test)
y_test = process_label(y_test)
y_train = process_label(y_train)

train_data = batchData(x_train,y_train)
valid_data = batchData(x_test,y_test, for_valid=True)

x_train,y_train,x_test,y_test = loading_data()
x_train, x_test
y_train, y_test
```

Pada bagian awal, terdapat fungsi process_label(label) yang digunakan untuk mengubah label menjadi array numpy yang berisi 0 dan 1. Fungsi ini akan dipanggil untuk mengubah label pada data latih dan data validasi.

Selanjutnya, terdapat fungsi processImage(path) yang digunakan untuk memproses gambar pada dataset. Gambar akan dibaca menggunakan tf.io.read_file(path) dan didecode menjadi tensor menggunakan tf.image.decode_jpeg(image,channels=3). Kemudian, gambar akan diubah menjadi tipe data float32 menggunakan tf.image.convert_image_dtype(image,tf.float32) dan diresize menjadi ukuran [224,224] menggunakan tf.image.resize(image,size=[224,224]). Fungsi ini akan dipanggil pada saat membuat batch data.

Kemudian, terdapat fungsi pairData(image, label) yang digunakan untuk membuat pasangan gambar dan label yang akan diproses menjadi batch data.

Selanjutnya, terdapat fungsi batchData(image,label=None, for_valid=False, for_test=False) yang digunakan untuk membuat batch data dari array numpy gambar dan label. Fungsi ini menerima empat parameter yaitu image dan label yang merupakan array numpy gambar dan label, for_valid yang digunakan untuk menentukan apakah fungsi ini akan digunakan untuk membuat batch data data validasi, dan for_test yang digunakan untuk menentukan apakah fungsi ini akan digunakan untuk membuat batch data data test.

Jika for_test=True, maka fungsi akan membuat batch data dari array numpy gambar tanpa label menggunakan tf.data.Dataset.from_tensor_slices((image)) dan memproses setiap gambar menggunakan processImage. Batch data akan dibuat dengan ukuran 32 menggunakan batch = data.map(processImage).batch(32) dan dikembalikan pada akhir fungsi.

Jika for_valid=True, maka fungsi akan membuat batch data dari array numpy gambar dan label menggunakan tf.data.Dataset.from_tensor_slices((tf.constant(image), tf.constant(label))) dan memanggil fungsi pairData untuk setiap pasangan gambar dan label menggunakan data.map(pairData). Batch data akan dibuat dengan ukuran 32 menggunakan .batch(32) dan dikembalikan pada akhir fungsi.

Jika kedua parameter tersebut bernilai False, maka fungsi akan membuat batch data dari array numpy gambar dan label menggunakan tf.data.Dataset.from_tensor_slices((tf.constant(image), tf.constant(label))), mengacak data menggunakan data.shuffle(buffer_size=len(image)), dan memanggil fungsi pairData untuk setiap pasangan gambar dan label menggunakan data.map(pairData). Batch data akan dibuat dengan ukuran 32 menggunakan .batch(32) dan dikembalikan pada akhir fungsi.

Pada bagian terakhir, dilakukan pemanggilan fungsi loading_data() untuk memuat data, kemudian hasil pemrosesan gambar dan label akan disimpan pada variabel x_train, y_train, x_test, dan y_test.

4.2.4 Membangun Neural Network

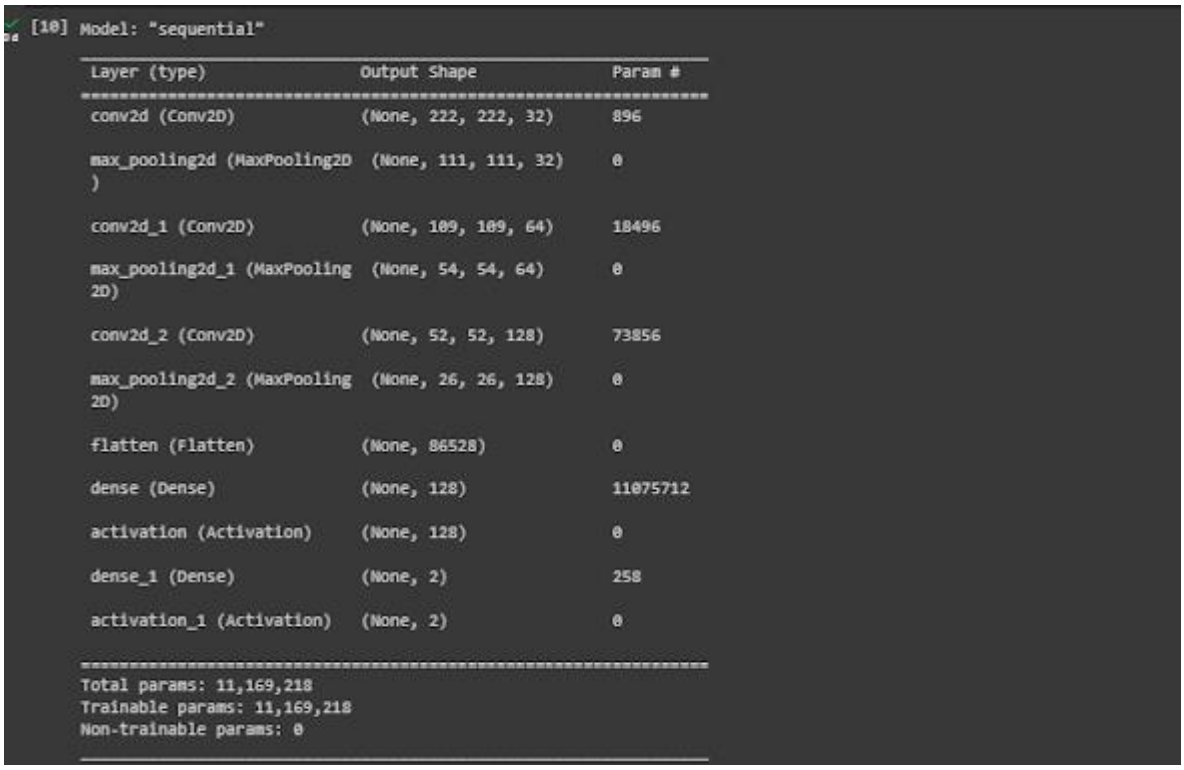
```
[ ]  # Create mode structure
     import keras
     import keras.utils
     from keras.models import Sequential
     from keras.layers import Dense, Dropout, Activation
     from tensorflow.keras import optimizers
     model3 = keras.Sequential([
         # Input layer
         keras.layers.Conv2D(input_shape=(224,224,3),filters=32,kernel_size=(3,3),activation='relu'),
         keras.layers.MaxPooling2D(),

         # Hidden Layer
         keras.layers.Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),activation='relu'),
         keras.layers.MaxPooling2D(),
         keras.layers.Conv2D(input_shape=(224,224,3),filters=128,kernel_size=(3,3),activation='relu'),
         keras.layers.MaxPooling2D(),
         keras.layers.Flatten(),
         keras.layers.Dense(128),
         keras.layers.Activation('relu'),

         # Output layer
         keras.layers.Dense(2),
         keras.layers.Activation('softmax')
     ])

     # Compile model
     model3.compile(loss='categorical_crossentropy',optimizer=keras.optimizers.Adam(),metrics=['acc'])
```

```
[ ]  model3.summary()
```



```
[10] Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 flatten (Flatten)           (None, 86528)             0

 dense (Dense)               (None, 128)               11075712

 activation (Activation)     (None, 128)               0

 dense_1 (Dense)             (None, 2)                 258

 activation_1 (Activation)   (None, 2)                 0

=================================================================
Total params: 11,169,218
Trainable params: 11,169,218
Non-trainable params: 0
_________________________________________________________________
```

`Model yang diberikan adalah sebuah Sequential model yang terdiri dari beberapa layer. Layer-layer tersebut adalah :

- Conv2D adalah layer konvolusi 2 dimensi yang digunakan untuk memproses input gambar pada level fitur.
- MaxPooling2D adalah layer pooling 2 dimensi yang digunakan untuk mengurangi dimensi spasial dari representasi fitur sehingga mengurangi jumlah parameter dan mengontrol overfitting.
- Flatten adalah layer untuk melakukan penggabungan dari output layer sebelumnya sehingga menjadi input untuk layer Dense selanjutnya.
- Dense adalah layer fully connected dengan setiap neuron dihubungkan ke semua neuron pada layer sebelumnya dan setelahnya.
- Activation adalah layer aktivasi yang menambahkan non-linearitas ke dalam model dengan menggunakan fungsi aktivasi.

`Total parameter model ini sebesar 11,169,218 yang semuanya dapat dipelajari selama proses training. Model ini terdiri dari 2 output neuron yang masing-masing merepresentasikan kemungkinan gambar yang diinputkan memiliki masker atau tidak.

4.2.5 Model Pelatihan

```
[ ]  early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss',patience=3)
```

`Pada bagian pertama, dilakukan pemanggilan keras.callbacks.EarlyStopping untuk menghentikan proses training secara otomatis jika model sudah mencapai kondisi yang diinginkan.

```
# simpan riwayat model menjadi variabel
history3 = model3.fit(train_data,validation_data = valid_data,validation_freq=1,epochs=5,callbacks=[early_stopping],verbose = 1)
```

`Kode tersebut menjalankan model3 yang telah dibuat dengan data train dan data validasi menggunakan metode fit pada Keras. Proses pelatihan model dilakukan selama 5 epoch, dengan penggunaan EarlyStopping callback untuk menghentikan pelatihan lebih awal jika tidak ada peningkatan pada performa model setelah 3 epoch berturut-turut pada data validasi. Seluruh proses pelatihan akan dicetak pada output karena verbose=1. Variabel history3 menyimpan riwayat pelatihan model yang dapat digunakan untuk analisis lebih lanjut.

Sehingga Mendapat Output Epoch 1/5 - 5/5 :

```
Epoch 1/5
19/19 [==============================] - 89s 4s/step - loss: 0.9986 - acc: 0.5845 - val_loss: 0.6726 - val_acc: 0.6027
Epoch 2/5
19/19 [==============================] - 70s 4s/step - loss: 0.5506 - acc: 0.7416 - val_loss: 0.5759 - val_acc: 0.7534
Epoch 3/5
19/19 [==============================] - 71s 4s/step - loss: 0.4457 - acc: 0.7939 - val_loss: 0.4465 - val_acc: 0.8493
Epoch 4/5
19/19 [==============================] - 73s 4s/step - loss: 0.2746 - acc: 0.8834 - val_loss: 0.5575 - val_acc: 0.8356
Epoch 5/5
19/19 [==============================] - 69s 4s/step - loss: 0.2011 - acc: 0.9307 - val_loss: 0.5892 - val_acc: 0.8493
```

`Dari hasil training model yang dilakukan dengan menggunakan arsitektur Convolutional Neural Network (CNN) dengan 3 layer konvolusi dan 1 layer dense pada output, serta menggunakan optimiser Adam dan loss function categorical crossentropy, diperoleh hasil akurasi yang cukup baik dengan akurasi validasi mencapai 84,93% pada epoch ke-5. Hasil ini menunjukkan model berhasil mempelajari karakteristik pada dataset dan mampu mengklasifikasikan gambar dengan cukup baik. Selain itu, penggunaan early stopping membantu mencegah overfitting dan menghentikan training ketika performa model tidak meningkat dalam beberapa epoch terakhir.

4.2.6 Plot Model Training History

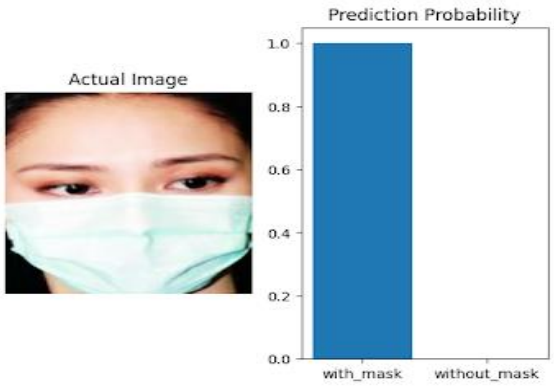
```
[ ]  def plot_history():
         plt.plot(history3.history['acc'],label='acc')
         plt.plot(history3.history['val_acc'],label='val_acc')
         plt.plot(history3.history['loss'],label='loss')
         plt.plot(history3.history['val_loss'],label='val_loss')
         plt.legend()
         plt.title('Training History')
         plt.xlabel('epoch')
         plt.ylabel('value')
         plt.tight_layout()
         plt.grid(True)
         plt.show()

     plot_history()
```

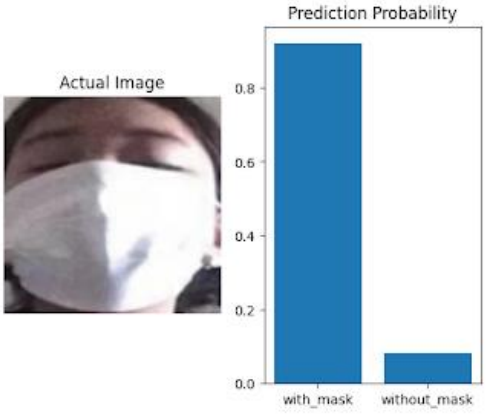
Sehingga Menghasilkan Output Grafik Seperti di Bawah :

`Dari hasil output grafik tersebut, dapat disimpulkan bahwa model yang dibangun cukup baik karena akurasi validasi dan akurasi pada data training meningkat seiring dengan peningkatan jumlah epoch. Namun, terlihat ada overfitting pada model tersebut, dimana akurasi pada data training meningkat namun akurasi pada data validasi tidak naik seiringnya dan terjadi peningkatan loss pada data validasi setelah epoch ke-2. Oleh karena itu, perlu dilakukan penyesuaian pada model untuk menghindari overfitting seperti dengan menambahkan regularisasi atau dropout layer pada model.

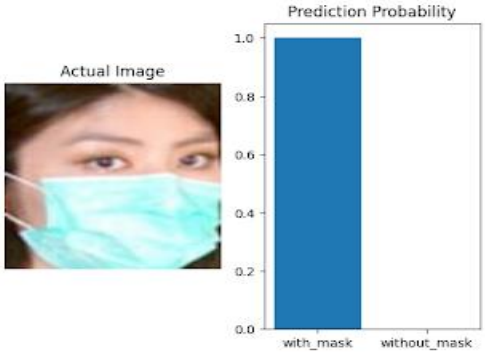
4.2.7 Demonstrasi



Gambar diatas dilihat bahwa prediksi probabiltas dengan masker senilai 1.0 dan tanpa masker senilai 0



Gambar diatas dilihat bahwa prediksi probabiltas dengan masker senilai >0.8 dan tanpa masker senilai 0.1



Gambar diatas dilihat bahwa prediksi probabiltas dengan masker senilai 1.0 dan tanpa masker senilai 0

5. KESIMPULAN

Pada data Sawi dapat dilihat Perbedaan dari kedua model tersebut terletak pada penekanan pada hasil pelatihan model dan interpretasi dari grafik training history. Paragraf pertama lebih fokus pada hasil akhir pelatihan model, di mana akurasi validasi mencapai 0.9045 pada epoch terakhir. Sementara itu, model lainnya lebih menekankan bahwa akurasi validasi masih rendah dan tidak mengalami peningkatan yang signifikan seiring dengan bertambahnya epoch. Hal ini menunjukkan bahwa model yang digunakan mungkin terlalu sederhana dan perlu dilakukan penyesuaian untuk meningkatkan kinerjanya, model yang dilatih dengan 8 epoch belum mencapai akurasi validasi yang memuaskan. Meskipun grafik training history menunjukkan adanya peningkatan dalam akurasi pelatihan dan penurunan dalam loss pelatihan, tetapi akurasi validasi masih rendah dan tidak ada perbaikan yang signifikan seiring dengan penambahan epoch. Oleh karena itu, diperlukan penyesuaian model lebih lanjut untuk meningkatkan kinerja model dalam mempelajari pola-pola yang ada dalam data. Sedangkan pada data Masker model yang dibangun cukup baik karena akurasi validasi dan akurasi pada data training meningkat seiring dengan peningkatan jumlah epoch. Namun, terlihat ada overfitting pada model tersebut, dimana akurasi pada data training meningkat namun akurasi pada data validasi tidak naik seiringnya dan terjadi peningkatan loss pada data validasi

6. DAFTAR PUSTAKA

Fajri, M., Adiwijaya, A., & Wahyono, S. (2019). Pengenalan Emosi Wajah Menggunakan Metode Convolutional Neural Network (CNN) pada Citra Grayscale. Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK), 6(2), 187-196.

Santoso, M. B., & Lestari, D. E. (2019). Klasifikasi Emosi Ekspresi Wajah dengan Metode Convolutional Neural Network (CNN) dan Gray Level Co-Occurrence Matrix (GLCM). Jurnal Sistem dan Informatika (JSI), 7(3), 149-156.

Aditya, P. I., Pratiwi, R., & Laksmono, H. (2018). Klasifikasi Sentimen Berita Menggunakan Metode Convolutional Neural Network (CNN). Jurnal Ilmu Komputer dan Informasi, 11(1), 11-20.

Arifin, F., & Widyawan, A. (2018). Identifikasi Kondisi Jalan Berlubang dengan Metode Convolutional Neural Network (CNN). Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI), 7(4), 342-347.

Raharja, S. J., & Ramadani, R. (2017). Penerapan Metode Convolutional Neural Network (CNN) pada Pengenalan Bahasa Isyarat. Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI), 6(4), 287-292.

LAMPIRAN

Link Code :

https://colab.research.google.com/drive/1VA08D_JhkNBLbfcwl0PL9HjzpS1RF_cc?usp=sharing

Dataset Sawi : https://www.kaggle.com/datasets/cendekialuthfietanz/caisim

Dataset Facemask : https://www.kaggle.com/datasets/omkargurav/face-mask-dataset