***P-ISSN : 2338-2899***
***E-ISSN: 2807-1271***

# Identifikasi Dan Klasifikasi Citra Penyakit Daun Tomat Menggunakan Arsitektur Inception V4

*Muhammad Islahfari Wahid[1], Syamsul Akbar Mustamin[2], dan Armin Lawi[3]*

*[123] Program Studi Sistem Informasi, Fakultas MIPA, Universitas Hasanuddin*

*Email: [1]wahidmi18h@student.unhas.ac.id, [2]mustaminsa18h@student.unhas.ac.id, [3]armin@unhas.ac.id*

***Abstract* — Tomato is one of the vegetables commonly consumed in the world. Tomato (*Lycopersicum esculentum Mill.*) belongs to the *Solanaceae family*, which is one of the most potential vegetable commodities to be developed. Indonesia is one of the countries that produce tomatoes. Tomato production in Indonesia itself is extraordinary, in 2019 tomato production reached 1,020,333 (tons). This means that tomatoes are one of the fruits that are needed in Indonesia. However, the quality and quantity of tomato plants in Indonesia in the last 5 years have decreased due to various types of diseases. Therefore we need a method that can identify tomato plant diseases with optimal results. The method used is Deep Learning using the Convolutional Neural Network algorithm along with the InceptionV4 architecture. Deep Learning is a machine learning method that works by imitating the working system of the human brain, this system is called a Neural Network. CNN is one type of neural network, whose main function is used for image data. The InceptionV4 architecture is a development of the previous generation, InceptionV3, which can produce very good performance with relatively low computing. The research was conducted with 2000 image data of tomato leaf disease which was divided into 10 classes with each class having 200 image data. In this study, the distribution of data with a ratio of 7:3, training data as much as 70%, and testing data as much as 30%. The results of this study obtained an accuracy of 90.00% to identify the type of tomato plant disease.**

***Keyword* — Convolution Neural Network (CNN), Deep Learning, image processing, InceptionV4, tomato leaf disease.**

*Abstrak* — Tomat merupakan salah satu sayuran yang umum dikonsumsi di dunia. Tanaman tomat (*Lycopersicum esculentum Mill*.) termasuk *famili Solanaceae* merupakan salah satu komoditas sayuran yang sangat potensial untuk dikembangkan. Indonesia merupakan salah satu negara yang memproduksi tomat. Produksi tomat di Indonesia sendiri sangat luar biasa, pada tahun 2019 produksi tomat mencapai 1.020.333 (ton). Artinya tomat menjadi salah satu buah yang sangat dibutuhkan di Indonesia. Namun kualitas dan kuantitas tanaman tomat di Indonesia dalam kurung waktu 5 tahun terakhir mengalami penurunan karena berbagai jenis penyakit. Oleh karena itu dibutuhkan sebuah metode yang dapat mengidentifikasi penyakit tanaman tomat dengan hasil yang optimal. Metode yang digunakan adalah Deep Learning dengan menggunakan algoritma *Convolutional Neural Network (CNN)* beserta arsitektur *InceptionV4*. Deep Learning adalah sebuah metode machine learning yang bekerja dengan cara meniru system kerja otak manusia, system ini disebut *Neural Network*. *CNN* merupakan salah satu jenis dari neural network, yang fungsi utamanya digunakan untuk data citra. Arsitektur *InceptionV4* merupakan pengembangan dari generasi sebelumnya yaitu *InceptionV3* yang mampu menghasilkan performa sangat baik dengan komputasi yang relatif rendah. Penelitian dilakukan dengan 2000 data citra penyakit daun tomat yang terbagi menjadi 10 kelas dengan masing-masing kelas memiliki 200 data citra. Dalam penelitian ini dilakukan pembagian data dengan rasio 7:3, data training sebanyak 70% dan data testing sebanyak 30%. Dari hasil penelitian ini didapatkan akurasi sebesar 90.00% untuk mengidentifikasi jenis penyakit tanaman tomat.

*Kata kunci — Convolutional Neural Network (CNN)*, Deep Learning, pengolahan citra, *InceptionV4*, penyakit daun tomat.

## I. PENDAHULUAN

Tomat (*Lycopersicum esculentum Mill.*) merupakan salah satu komoditas hortikultura yang bernilai ekonomi tinggi. Tomat adalah salah satu komoditas sayuran yang sangat potensial untuk dikembangkan. Tanaman ini dapat ditanam secara luas di dataran rendah maupun dataran tinggi [3]. Produksi tomat di Indonesia sangat tinggi, buktinya pada tahun 2019 produksi tomat mencapai angka 1,020,333 tons (BPS, 2019).

Buah tomat masih memerlukan penanganan yang serius, terutama dalam kualitas dan kuantitas. Rata-rata produksi tomat di Indonesia masih terbilang rendah, yaitu 6,3 ton/ha. Jika dibandingkan dengan Negara-negara seperti Taiwan dan India yang memiliki rata-rata produksi tomat sebanyak 21 ton/ha, dan 9,5 ton/ha [4]. Salah satu penyebab rendahnya kualitas dan kuantitas tomat adalah penyakit pada daun tomat. Maka dari itu, identifikasi penyakit daun tomat dapat meningkatkan kualitas dan kuantitas tomat.

Penelitian ini sudah pernah dilakukan sebelumnya dengan menggunakan beberapa algoritma berbeda yaitu *RandomForest*, *SVM*, dan *Convolutional Neural Network (CNN)*. Hasil penelitian tersebut mendapat tingkat akurasi yang lumayan baik yaitu di atas 95%.

Saat ini metode yang paling baik dalam pengenalan citra yaitu *CNN*. Metode ini memliki beragam arsitektur yaitu *ResNet50*, *MobileNet*, *Xception*, *InceptionV4*, dll. Karena itu penelitian ini berfokus tentang identifikasi penyakit daun tomat dengan menggunakan arsitektur *InceptionV4*.

## II. PENELITIAN TERKAIT

Penelitian dengan menggunakan data citra telah banyak digunakan dalam berbagai bidang. Salah satunya pada bidang pertanian, dalam bidang pertanian sudah banyak penelitian yang berhubungan dengan data citra seperti sistem untuk mendeteksi gangguan serangga, menentukan masa panen, dan sebagainya. Berikut beberapa penelitian mengenai *image processing* pada bidang pertanian.

Penelitian yang dilakukan oleh Felix, et al (2019), dengan judul *Implementasi CNN dan SVM untuk Identifikasi Penyakit Tomat via Daun*. Penelitian ini menggunakan 200

data citra yang terbagi dalam empat kelas. Peneliti melakukan transformasi data citra dari warna *Red Green Blue (RGB)* menjadi *HSV*, kemudian diubah menjadi *Grayscale*. Hasil dari penelitian ini didapatkan akurasi menggunakan algoritma *CNN* sebesar 97,5% dan *SVM* sebesar 95% [5].

Penelitian yang dilakukan oleh Margi Cahyanti, et al (2020), dengan judul *Klasififkasi Penyakit Tanaman Apel dari Citra Daun dengan Convolutional Neural Network*. Penelitian ini diimplementasikan menggunakan aplikasi android. Peneliti menggunakan data citra sebanyak 7700 untuk data training dan 1943 untuk data testing. Dilakukan perubahan ukuran data citra menjadi 256x256 dan menggunakan warna *RGB*. Hasilnya didapatkan akurasi sebesar 97,1% dengan menggunakan algoritma *CNN* [6].

Penelitian yang dilakukan oleh Dwi Fitriana Sari dan Daniel Swanjaya (2020), dengan judul *Implementasi Convolutional Neural Network untuk Identifikasi Penyakit Daun Gambus*. Dataset pada penelitian ini terdiri dari empat kelas. Peneliti menggunakan arsitektur *MobileNet* untuk mengklasifikasi data citra penyakit daun gambus. Hasil dari penelitian ini didapatkan akurasi sebesar 90% [7].

Penelitian yang dilakukan oleh Umi Khultsum dan Agus Subekti (2021), dengan judul *Penerapan Algoritma Random Forest dengan Kombinasi Ekstraksi Fitur untuk Klasifikasi Penyakit Daun Tomat*. Dataset yang digunakan berjumlah 10.000 data citra yang terdiri dari 10 kelas. Peneliti mengkombinasikan beberapa teknik ekstraksi fitur yaitu fitur *histogram warna*, *fitur haralick*, dan *fitur hu-moment*. Algoritma yang digunakan dalam penelitian ini adalah *Random Forest*. Hasilnya diperoleh akurasi sebesar 96% [8].

Penelitian yang dilakukan oleh Mohit Agarwal, et al (2019), dengan judul *ToLeD : Tomato Leaf Disease Detection Using Convolutional Neural Network*. Dataset pada penelitian ini terdiri dari 10 kelas dengan masing-masing kelas sebanyak 1000 data citra. Peneliti menggunakan tiga arsitektur berbeda untuk mengklasifikasi penyakit daun tomat yaitu *VGG16*, *InceptionV3*, dan *MobileNet*. Hasil eksperimen ini diperoleh masing-masing akurasi dari tiap arsitektur adalah 77,2%, 63,75%, dan 63,4% [9].

## III. BAHAN DAN METODE

### *A. Dataset*

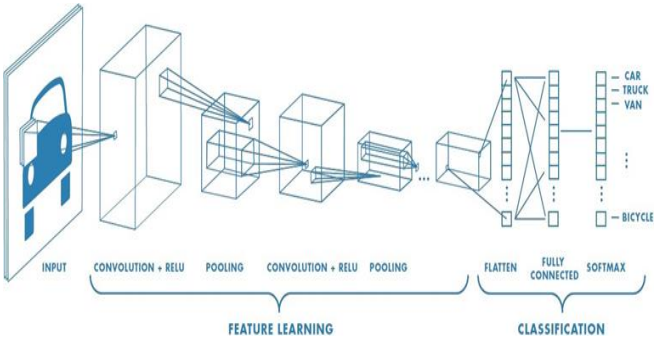
Data yang digunakan dalam penelitian ini diambil dari Repository Kaggle yaitu dataset Tomato leaf disease detection. Dataset ini berisi data citra penyakit daun tomat, yang terdiri dari 10 kelas dengan masing-masing kelas memiliki 200 data citra.

TABEL 1. INFORMASI DATASET PENYAKIT DAUN TOMAT

| No | Label | Jumlah Gambar |
|---|---|---|
| 1 | Tomato Late blight | 200 |
| 2 | Tomato Mosaic Virus | 200 |
| 3 | Tomato Spider Mites | 200 |
| 4 | Tomato Septoria Leaf Spot | 200 |
| 5 | Tomato Leaf Mold | 200 |
| 6 | Tomato Target Spot | 200 |
| 7 | Tomato Yellow Leaf Curl Virus | 200 |
| 8 | Tomato Healthy | 200 |
| 9 | Tomato Early Blight | 200 |
| 10 | Tomato Bacterial Spot | 200 |

### *B. Convolutional Neural Network*

*Convolutional Neural Network (CNN)* merupakan pengembangan dari *Multi-Layer Perceptron (MPL)* yang dirancang untuk mengolah data dalam bentuk grid, salah satunya adalah citra dua dimensi. *CNN* digunakan untuk mengklasifikasikan data yang terlabel dengan menggunakan metode *Supervised Learning*. *CNN* sering digunakan untuk mengenali benda atau mendetaksi suatu objek.




Gambar 1. Arsitektur *CNN*

i. Convolution Layer

*Convolution Layer* menggunakan filter untuk mengekstraksi sbeuah obejk. Filter ini berisi bobot yang digunakan untuk mendeteksi objek seperti tepi, kurva, atau warna. Output yang dihasilkan konvolusi adalah transformasi linear.

ii. Pooling Layer

*Pooling* merupakan metode yang bertujuan mengurangi ukuran matriks. Dalam pooling layer terdapat dua macam yaitu *average pooling* dan *max pooling*. *Average pooling* menghasilkan nilai rata-rata sedangkan *max pooling* menghasilkan nilai maksimal.

iii. Activation Function

*Rectification Linear Unit (ReLU)* merupakan sebuah fungsi yang bertujuan mengenalkan nonlinearitas dan meningkatkan representasi dari

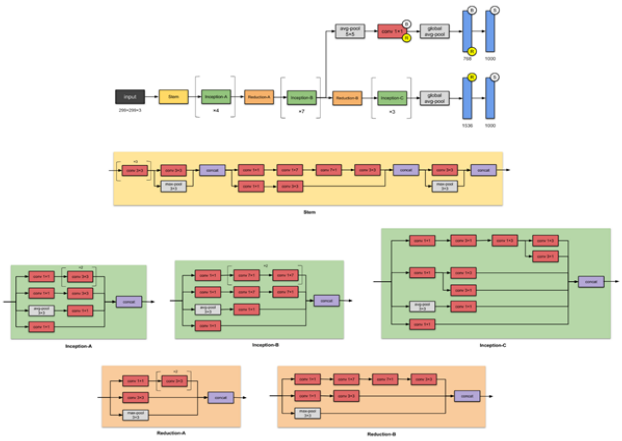
model. Jika nilai input neuron adalah negatif maka dapat dinyatakan outputnya adalah 0. Namun nilai output adalah nilai aktivasi itu sendiri, maka input dari neuron adalah nilai positif.

iv. Fully Connected Layer

*Fully Connected Layer* merupakan kumpulan neuron dari tiap layer yang saling terhubung. Layer ini menerima input vektor, sehingga sebelum masuk output dari *convolution layer* perlu ditransformasi menjadi satu dimensi.

### C. InceptionV4

*InceptionV4* merupakan pengembangan dari versi sebelumnya, yaitu *InceptionV3*. *InceptionV4* memiliki beberapa blok yaitu stem, A, B, dan C, sama seperti versi sebelumnya *InceptionV4* juga memiliki *reduction block* yang digunakan untuk mengubah lebar da tinggi grid. Perbedaan dari versi sebelumnya terletak pada blok stem dan beberapa perubahan kecil pada blok C. *InceptionV4* menerima input 299x299x3.



Gambar 3. Struktur arsitektur *InceptionV4*

### D. Confusion Matrix

*Confusion matrix* merupakan sebuah metode untuk mengukur peforma pada masalah klasifikasi dengan keluaran dua kelas maupun lebih. *Confusion matrix* biasanya digunakan untuk menghitung akurasi dari sebuah model machine learning. *Confusion matrix* memiliki tabel dengan empat kombinasi berbeda dari nilai prediksi dan nilai aktual yaitu *True Positive (TP)*, *True Negative (TN)*, *False Positive (FP)*, dan *False Negative (FN)*.

TABEL 2. *CONFUSION MATRIX*

| | | True Values | |
|---|---|---|---|
| | | True | False |
| Prediction | True | TP<br>Correct result | FP<br>Unexpected result |
| | False | FN<br>Missing result | TN<br>Correct absence of result |

i. Akurasi

Akurasi merupakan salah satu cara untuk mengukur kinerja dari model klasifikasi. Akurasi merupakan rasio prediksi benar dengan keseluruhan data yang biasanya dihitung dalam bentuk persentase. Berikut perhitungan akurasi.

$$Akurasi = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

ii. Loss Function

*Loss function* digunakan untuk mengoptimalkan model pada machine learning. *Loss function* dihitung dengan cara seberapa baik kinerja model dalam traning data dan validasi data. Nilai loss yang dihasilkan menunjukan seberapa baik atau buruknya sebuah model. Berikut perhitungan *loss function*.

$$J = \frac{1}{N}\sum_{i=1}^{N} t(\hat{y}_i, y_i) \quad (2)$$

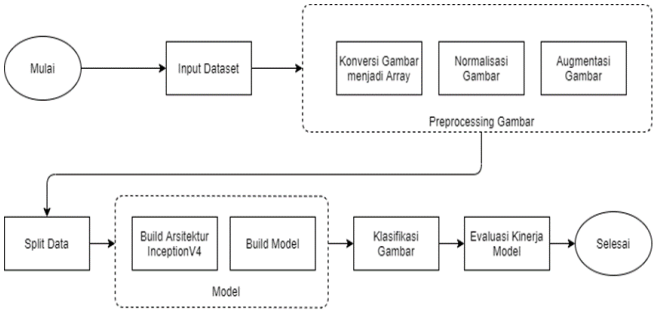
## IV. IMPLEMENTASI

### A. Instrumen Penelitian

Pada penelitian ini digunakan perangkat keras dan perangkat lunak. Perangkat keras yang digunakan adalah laptop Asus dengan spesifikasi yaitu *Processor Intel Core i7- Gen 8th*, *RAM 16GB*, dengan system operasi *Windows 10*. Perangkat ini digunakan untuk mencari dataset, menginisialisasi kerja *Google Collab*, dan membuat paper. Software yang digunakan adalah *Google Collab* dengan spesifikasi *RAM 12GB* dan *GPU Tesla*. *Google Collab* digunakan untuk memproses dataset, dan membangun dan menguji model.

### B. Alur Kerja

Berikut merupakan alur kerja penelitian.




Gambar 4. Alur kerja penelitian

Alur pertama dalam penelitian ini adalah melakukan input data dari *Google Drive* ke *Google Collab*. Langkah kedua dilakukan preprocessing yang bertujuan agar data siap diproses oleh model. Pada proses preprocessing terdapat tiga tahapan yang dilakukan, yaitu mengkonversi gambar menjadi array, normalisasi gambar, dan augmentasi gambar. Selanjutnya langkah ketiga dilakukan pembagian data

dengan perbandingan 70% data training dan 30% data testing. Langkah keempat adalah proses membangun model menggunakan arsitektur *InceptionV4*. Kemudian langkah kelima dilakukan klasifikasi gambar menggunakan model yang sudah dibangun. Langkah terakhir adalah evaluasi kinerja model sebagai bentuk pengukuran seberapa baik model yang sudah dibangun.

*C. Implementasi*

i. Import Dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive;
```

Gambar 5. Import dataset

Langkah awal berikan akses Google Collab ke Google Drive, agar dapat mengakses dataset yang tersimpan dalam Google Drive.

ii. Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
from os import listdir
```

```
import keras
from keras.preprocessing import image
from keras import backend as K
from keras.layers import Input
from keras.optimizers import Adam
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
```

```
import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Input, concatenate, BatchNormalization, Activation, GlobalAveragePooling2D
from tensorflow.keras.layers import Conv2D, MaxPooling2D, SeparableConv2D, DepthwiseConv2D, AveragePooling2D
```

Gambar 6. Import library yang dibutuhkan

Pada langkah ini import library-library yang dibutuhkan dalam preprocessing dan proses membangun model.

iii. Mengkonversi Citra menjadi Array

```
def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None :
            image = cv2.resize(image, default_image_size)
            return img_to_array(image)
        else :
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

Gambar 7. Fungsi konversi data citra menjadi array

Fungsi ini bertujuan mengubah data citra menjadi bentuk array.

iv. Memuat Data Citra

```
default_image_size = tuple((299, 299))

image_list, label_list = [], []
try:
    print("[INFO] Loading images ...")
    root_dir = listdir(dir_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)

    for plant_folder in root_dir :
        plant_disease_folder_list = listdir(f"{dir_root}/{plant_folder}")

        for disease_folder in plant_disease_folder_list :
            # remove .DS_Store from list
            if disease_folder == ".DS_Store" :
                plant_disease_folder_list.remove(disease_folder)

        for plant_disease_folder in plant_disease_folder_list:
            print(f"[INFO] Processing {plant_disease_folder} ...")
            plant_disease_image_list = listdir(f"{dir_root}/{plant_folder}/{plant_disease_folder}/")

            for single_plant_disease_image in plant_disease_image_list :
                if single_plant_disease_image == ".DS_Store" :
                    plant_disease_image_list.remove(single_plant_disease_image)

            for image in plant_disease_image_list[:200]:
                image_directory = f"{dir_root}/{plant_folder}/{plant_disease_folder}/{image}"
                if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True:
                    image_list.append(convert_image_to_array(image_directory))
                    label_list.append(plant_disease_folder)
    %time print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error : {e}")
```

Gambar 8. Proses pengolahan data citra menjadi array

Pada proses ini data citra akan dimuat dan dikonversi menjadi array dengan ukuran *default* 299x299. Hasil dari proses ini berjumlah 2000 data citra yang dikonversi menjadi array.

v. Memberi Label pada Data Citra

```
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
n_classes = len(label_binarizer.classes_)
```

Gambar 9. Proses memberikan label

Setelah data citra diubah menjadi array, proses selanjutnya diberikan label menggunakan fungsi LabelBinarizer.

vi. Normalisasi Data Citra

```
np_image_list = np.array(image_list, dtype=np.float16) / 225.0
```

Gambar 10. Normalisasi data citra

Tujuan dari normalisasi adalah membuat data citra berada antara nilai 0 sampai dengan 225.

vii. Split Data

```
x_train, x_test, y_train, y_test = train_test_split(
    np_image_list,
    image_labels,
    test_size=0.3,
    random_state = 42)
```

Gambar 11. Split data

Proses ini membagi data dengan rasio 7:3, untuk data training sebanyak 70% dan data testing sebanyak 30%. Proses pembagian data dilakukan secara acak. Hasil proses ini data training

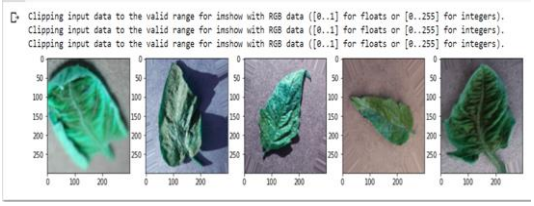
sebanyak 1400 data citra dan data testing sebanyak 600 data citra.

viii. Augmentasi Data Citra

```
aug = ImageDataGenerator(
    rotation_range=40,
    horizontal_flip=True,
    shear_range = 0.2,
    fill_mode = 'nearest',
    zoom_range = 0.2)
```

Gambar 12. Proses augmentasi data citra

Augmentasi data citra bertujuan untuk meningkatkan jumlah data citra. Augmentasi memanipulasi data citra tanpa kehilangan inti dari data tersebut. Hasil proses augmentasi sebagai berikut.




Gambar 13. Hasil proses augmentasi

ix. Membangun Model InceptionV4

a. Fungsi Convolutional Block

```
def conv2d_bn(inputs ,filters, kernel_size, padding, strides):
  inputs = Conv2D(filters, kernel_size, strides=strides, padding=padding)(inputs)
  inputs = BatchNormalization(axis = 3)(inputs)
  outputs = Activation('relu')(inputs)
  return outputs
```

Gambar 14. Fungsi *convolutional block*

Dalam fungsi ini terdapat beberapa fungsi. Layer pada fungsi ini yaitu *Conv2D* dan *BatchNormalization*. Terdapat juga fungsi *Activation ReLu*.

b. Block Stem

```
x = conv2d_bn(inputs, kernel_size=(3,3), filters=32, strides=2, padding="valid")
x = conv2d_bn(x, kernel_size=(3,3), filters=32, strides=1, padding="valid")
x = conv2d_bn(x, kernel_size=(3,3), filters=64, strides=1, padding="same")
x_1 = MaxPooling2D((3,3), strides = 2)(x)
x_2 = conv2d_bn(x, kernel_size=(3,3), filters=96, strides=2, padding="valid")
x = concatenate([x_1, x_2], axis=-1)
x_1 = conv2d_bn(x, kernel_size=(1,1), filters=64, strides=1, padding="same")
x_1 = conv2d_bn(x_1, kernel_size=(3,3), filters=96, strides=1, padding="valid")
x_2 = conv2d_bn(x, kernel_size=(1,1), filters=64, strides=1, padding="same")
x_2 = conv2d_bn(x_2, kernel_size=(7,1), filters=64, strides=1, padding="same")
x_2 = conv2d_bn(x_2, kernel_size=(1,7), filters=64, strides=1, padding="same")
x_2 = conv2d_bn(x_2, kernel_size=(3,3), filters=96, strides=1, padding="valid")
x = concatenate([x_1, x_2], axis=-1)
x_1 = conv2d_bn(x, kernel_size=(3,3), filters=192, strides=2, padding="valid")
x_2 = MaxPooling2D((3,3), strides = 2)(x)
x = concatenate([x_1, x_2], axis=-1)
```

Gambar 15. Block Stem

Block Steam merupakan inisiasi awal sebelum measuk ke block selanjutnya.

c. Block A

```
def inception_block_A(inputs):
  branch_1 = conv2d_bn(inputs, kernel_size=(1,1), filters=64, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1,kernel_size=(3,3), filters=96, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(3,3), filters=96, strides = 1, padding="same")

  branch_2 = conv2d_bn(inputs, kernel_size=(1,1), filters=64, strides = 1, padding="same")
  branch_2 = conv2d_bn(branch_2, kernel_size=(3,3), filters=96, strides = 1, padding="same")

  branch_3 = conv2d_bn(inputs, kernel_size=(1,1), filters=96, strides = 1, padding="same")

  branch_4 = AveragePooling2D((3,3), strides = 1, padding="same")(inputs)
  branch_4 = conv2d_bn(branch_4, kernel_size=(1,1), filters=96, strides = 1, padding="same")
  output   = concatenate([branch_1, branch_2, branch_3, branch_4], axis=-1)

  return output
```

Gambar 16. Block A

Pada blok A terdapat empat cabang. Output dari blok A merupakan hasil *concat* dari empat cabang tersebut.

d. Block B

```
def inception_block_B(inputs):
  branch_1 = conv2d_bn(inputs, kernel_size=(1,1), filters=192, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(1,7), filters=192, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(7,1), filters=224, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(1,7), filters=224, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(7,1), filters=256, strides = 1, padding="same")


  branch_2 = conv2d_bn(inputs, kernel_size=(1,1), filters=192, strides = 1, padding="same")
  branch_2 = conv2d_bn(branch_2, kernel_size=(1,7), filters=224, strides = 1, padding="same")
  branch_2 = conv2d_bn(branch_2, kernel_size=(1,7), filters=256, strides = 1, padding="same")


  branch_3 = conv2d_bn(inputs, kernel_size=(1,1), filters=384, strides = 1, padding="same")

  branch_4 = AveragePooling2D((3,3), strides = 1, padding="same")(inputs)
  branch_4 = conv2d_bn(branch_4, kernel_size=(1,1), filters=128, strides = 1, padding="same")
  output =  concatenate([branch_1, branch_2, branch_3, branch_4], axis=-1)

  return output
```

Gambar 17. Block B

Blok B kurang lebih sama dengan blok A yaitu terdapat empat cabang, yang membedakan blok B memiliki cabang lebih panjang dan ukuran layer yang beragam.

e. Block C

```
def inception_block_C(inputs):
  branch_1 = conv2d_bn(inputs, kernel_size=(1,1), filters=384, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(1,3), filters=448, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(3,1), filters=512, strides = 1, padding="same")
  branch_1_1 = conv2d_bn(branch_1, kernel_size=(1,3), filters=256, strides = 1, padding="same")
  branch_1_2 = conv2d_bn(branch_1, kernel_size=(3,1), filters=256, strides = 1, padding="same")

  branch_2 = conv2d_bn(inputs, kernel_size=(1,1), filters=384, strides = 1, padding="same")
  branch_2_1 = conv2d_bn(branch_2, kernel_size=(3,1), filters=256, strides = 1, padding="same")
  branch_2_2 = conv2d_bn(branch_2, kernel_size=(1,3), filters=256, strides = 1, padding="same")

  branch_3 = conv2d_bn(inputs, kernel_size=(1,1), filters=256, strides = 1, padding="same")

  branch_4 = AveragePooling2D((3,3), strides = 1, padding="same")(inputs)
  branch_4 = conv2d_bn(branch_4, kernel_size=(1,1), filters=256, strides = 1, padding="same")
  output =  concatenate([branch_1_1, branch_1_2, branch_2_1, branch_2_2 , branch_3, branch_4], axis=-1)

  return output
```

Gambar 18. Block C

Pada Blok C terdapat empat cabang dan memiliki cabang terpanjang dari blok lainnya.

f. Reduction Block A

```
def reduction_block_A(inputs):
  branch_1 = conv2d_bn(inputs, kernel_size=(1,1), filters=192, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(3,3), filters=224, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(3,3), filters=256, strides = 2, padding="valid")

  branch_2 = conv2d_bn(inputs, kernel_size=(3,3), filters=384, strides = 2, padding="valid")

  branch_3 = AveragePooling2D((3,3), strides = 2)(inputs)
  output =  concatenate([branch_1, branch_2, branch_3], axis=-1)

  return output
```

Gambar 19. Reduction Block A

Pada reduction blok A terdapat tiga cabang, cabang ini terdiri dari beberapa layer yaitu layer konvolusi dan max pooling.

g. Reduction Block B

```
def reduction_block_B(inputs):
  branch_1 = conv2d_bn(inputs, kernel_size=(1,1), filters=256, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(1,7), filters=256, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(7,1), filters=320, strides = 1, padding="same")
  branch_1 = conv2d_bn(branch_1, kernel_size=(3,3), filters=320, strides = 2, padding="valid")

  branch_2 = conv2d_bn(inputs, kernel_size=(1,1), filters=192, strides = 1, padding="same")
  branch_2 = conv2d_bn(branch_2, kernel_size=(3,3), filters=192, strides = 2, padding="valid")

  branch_3 = AveragePooling2D((3,3), strides = 2)(inputs)

  output =  concatenate([branch_1, branch_2, branch_3], axis=-1)

  return output
```

Gambar 20. Reduction Block B

Reduction blok B kurang lebih sama dangan reduction blok A, yang membedakan terletak pada panjang cabang dan ukuran layer konvolusi.

h. Model InceptionV4

```
inception_A  = inception_block("A", x)
for i in range(3): inception_A  = inception_block("A", inception_A)

reduction_A  = reduction_block_A(inception_A)

inception_B  = inception_block("B", reduction_A)
for i in range(6):inception_B  = inception_block("B", inception_B)

reduction_B  = reduction_block_B(inception_B)

inception_C  = inception_block("C", reduction_B)
inception_C = inception_block("C", inception_C)
inception_C = inception_block("C", inception_C)

global_pool = AveragePooling2D((8,8), padding='valid')(inception_C)
dense = Dropout(0.8)(global_pool)
dense = Flatten()(dense)
output = Dense(10, activation='softmax')(dense)
```

Gambar 21. Model *InceptionV4*

Setelah semua blok dibuat selanjutnya dilakukan pembuatan model menggunakan arsitektur InceptionV4. Arsitektur InceptionV4 menerima input 299x299x3. Selanjutnya masuk ke Blok stem, setelah itu dilakukan blok A sebanyak empat kali. Setelah dari blok A masuk ke reduction blok A. Kemudian masuk ke blok B sebanyak tujuh kali, setelah itu masuk ke reduction blok B. Blok terakhir ada blok C sebanyak tiga kali. Setelah semua proses selesai, outputnya akan diubah menjadi satu dimensi.

x. Summary Model

```
concatenate_18 (Concatenate)    (None, 8, 8, 1536)   0          activation_142[0][0]
                                                                activation_143[0][0]
                                                                activation_145[0][0]
                                                                activation_146[0][0]
                                                                activation_147[0][0]
                                                                activation_148[0][0]
__________________________________________________________________________________________________
average_pooling2d_16 (AveragePo (None, 1, 1, 1536)   0          concatenate_18[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 1, 1, 1536)   0          average_pooling2d_16[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1536)         0          dropout[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 10)           15370      flatten[0][0]
==================================================================================================
Total params: 41,252,938
Trainable params: 41,189,770
Non-trainable params: 63,168
__________________________________________________________________________________________________
```

Gambar 22. Summary model

xi. Method Callback

```
class Callback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_accuracy') >= 0.90):
      self.model.stop_training = True

stopProcces = Callback()
```

Gambar 23. Method callback

Method ini berfungsi menghentikan train model jika akurasi yang di inginkan sudah terpenuhi.

xii. Compile Model

```
model.compile(loss='categorical_crossentropy',
              optimizer=tf.optimizers.Adam(lr=1e-3),
              metrics=['accuracy'])
```

Gambar 24. Compile model

Model di compile dengan menggunakan optimizer Adam. Loss function yang digunakan adalah categorical crossentropy karena kelas model lebih dari dua.

xiii. Fitting Model

```
procces = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=32),
    validation_data=(x_test,y_test),
    steps_per_epoch=t_steps,
    epochs=150,
    verbose=1,
    callbacks=[stopProcces])
```

Gambar 24. Fitting model

Pada proses fitting model digunakan data training sebagai train model, kemudian data test sebagai validasi data. Inisiasi epoch sebanyak 150, namun proses akan berakhir jika akurasi sudah terpenuhi. Berikut hasil training model

```
43/43 [==============================] - 47s 1s/step - loss: 0.2980 - accuracy: 0.9027 - val_loss: 1.0236 - val_accuracy: 0.7467
Epoch 97/150
43/43 [==============================] - 47s 1s/step - loss: 0.2214 - accuracy: 0.9183 - val_loss: 2.1193 - val_accuracy: 0.6083
Epoch 98/150
43/43 [==============================] - 47s 1s/step - loss: 0.2432 - accuracy: 0.9163 - val_loss: 0.6886 - val_accuracy: 0.8133
Epoch 99/150
43/43 [==============================] - 47s 1s/step - loss: 0.2726 - accuracy: 0.9159 - val_loss: 0.5903 - val_accuracy: 0.8333
Epoch 100/150
43/43 [==============================] - 47s 1s/step - loss: 0.2611 - accuracy: 0.9147 - val_loss: 0.3500 - val_accuracy: 0.9000
```

Gambar 25. Hasil training model

Dari hasil tersebut dapat dilihat proses berhenti pada epoch ke-100, karena akurasi yang diinginkan sudah terpenuhi yaitu 91,47% dan validasi akurasi sebesar 90,00%.

xiv. Visualisasi Akurasi Train Model

```
import matplotlib.pyplot as plt

accuracy = procces.history['accuracy']
val_accuracy = procces.history['val_accuracy']

plt.plot(accuracy)
plt.plot(val_accuracy)
plt.title('Accuracy Function')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(['train', 'val'])
plt.show()
```

Gambar 26. Visualisasi akurasi train model

xv. Visualisasi Loss Train Model

```
loss = procces.history['loss']
val_loss = procces.history['val_loss']

plt.plot(loss)
plt.plot(val_loss)
plt.title('Loss Function')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'val'])
plt.show()
```

Gambar 27. Visualisasi loss train model

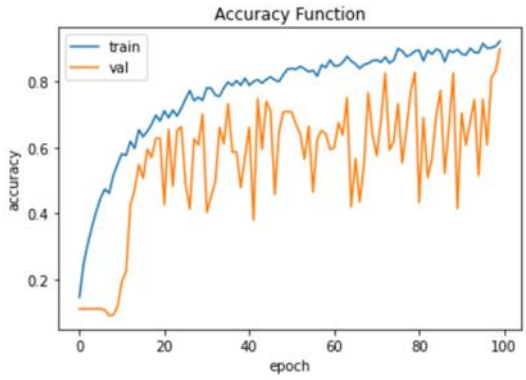
xvi. Confusion Matrix Model

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

pred = model.predict(x_test)
pred = np.argmax(pred,axis=1)
y_test2 = np.argmax(y_test,axis=1)

cm = confusion_matrix(y_test2,pred)

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(label_binarizer.classes_,rotation=90); ax.yaxis.set_ticklabels(label_binarizer.classes_,rotation=0);
```

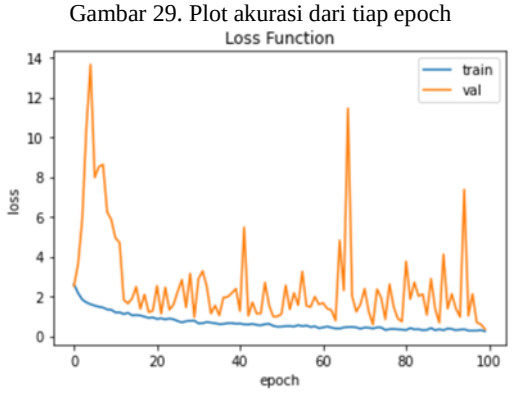
Gambar 28. Confusion matrix

## V. HASIL DAN PEMBAHASAN

Berdasarkan proses train model menggunakan data training dan testing sebanyak 100 epoch, didapatkan plot akurasi dan loss.

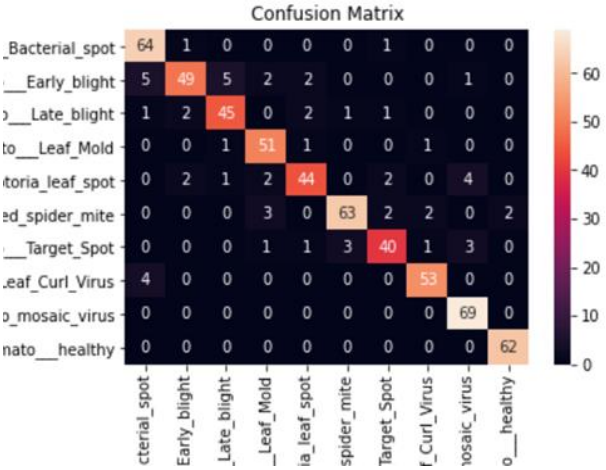



Gambar 29. Plot akurasi dari tiap epoch




Gambar 30. Plot loss dari tiap epoch

Dapat dilihat pada grafik. Validasi akurasi sangat tidak konsisten, terjadi beberapa kali penurunan yang sangat signifikan. Selanjutnya dilakukan visualisasi *confusion matrix*.




Gambar 31. Visualisasi confusion matrix

Pada *confusion matrix* terjadi beberapa kesalahan prediksi. Kesalahan prediksi terbanyak terjadi pada kelas *Tomato_Target_Spot*. Terakhir lakukan evaluasi model menggunakan data testing.

```
16/16 [==============================] - 4s 266ms/step - loss: 0.3695 - accuracy: 0.9000
Test Accuracy: 89.99999761581421
```

Gambar 32. Evaluasi model

Hasil evaluasi model diperoleh akurasi sebesar 90,00%.

## VI. KESIMPULAN

Berdasarkan dari alur kerja yang dilakukan, maka diperoleh hasil klasifikasi menggunakan arsitektur InceptionV4 sebagai berikut.

i. Hasil penelitian ini diperoleh akurasi sebesar 90,00% dalam mengidentifikasi jenis penyakit daun tomat.
ii. Dalam penelitian ini validasi akurasi dari train model masih belum konsisten, dapat dilihat pada gambar 29.

## DAFTAR ACUAN


[1] Agrios, G. N. 2005. *Plant Pathology*. Fifth Edition. USA : Academic Press.

[2] Sanoubar, R. and Barbanti, L. (2017) 'Fungal diseases on tomato plant under greenhouse condition', *European Journal of Biological Research*, 7(4), pp. 299–308. doi: 10.5281/zenodo.1011161.

[3] Setiawati, W., B.K. Udiarto, dan N. Gunaeni. "Preferensi Beberapa Varietas Tomat dan Pola Infestasi Hama Kutu Kebul serta Pengaruhnya terhadap Intensitas Serangan Virus Kuning," *J.Hort*. 17(4):374-386, 2007.

[4] Kartapradja, R. dan D. Djuariah. "Pengaruh tingkat kematangan buah tomat terhadap daya kecambah, pertumbuhan dan hasil tomat," Buletin Penelitian Hortikultura Vol XXIV/2, 1992.

[5] Felix. et al. "Implementasi CNN dan SVM untuk Identifikasi Penyakit Tomat via Daun," Vol 20, No 2, Oktober 2019.

[6] Cahyanti, Margi. et al. "Klasifikasi Penyakit Tanaman Apel dari Citra Daun dengan Convolutional Neural Network," *SEBATIK 1410-3737*.

[7] Sari, Dwi Fitriana. dan Daniel Swanjaya. "Implementasi Convolutional Neural Network Untuk Identifikasi Penyakit Daun Gambas," *Seminar Nasional Inovasi Teknologi*, Juli 2020

[8] Khultsum, Umi. Dan Agus Subekti. "Penerapan Algoritma Random Forest dengan Kombinasi Ekstraksi Fitur Untuk Klasifikasi Penyakit Daun Tomat," JURNAL MEDIA INFORMATIKA INDONESIA. Vol 5, No 1, Januari 2021.

[9] Agarhal, Mohit. et al. "ToLeD : Tomato Leaf Disease Detection using Convolutional Neural Network," *International Conference on Computational Intelligence and Data Science (ICCIDS)*, 2020