

# Lembar Jawaban UAS

## Deep Learning

---

**Kelompok : 16**

**Anggota : 5180411382 - MUHAMMAD FAUZAN PARANDITHA**  
**5180411267 – CHOLIF ARDERY HARISON**  
**5180411382 – ALFIAN HIDAYATULLOH**

**Pernyataan:**

Dengan mengerjakan ujian ini, maka kelompok kami menyatakan bahwa laporan ini **KAMI KERJAKAN SENDIRI** dengan tidak asal meng-*copy paste* penelitian orang lain. Seluruh acuan maupun pedoman yang kelompok kami gunakan telah tercantum pada bagian Referensi. Kami bersedia diberi **nilai E** jika pernyataan ini terbukti salah.

---

### Klasifikasi Penyakit Pada Tanaman Apel Berdasarkan Citra Daun Menggunakan Algoritma CNN

#### I. Deskripsi Dataset

Adapun *dataset* yang digunakan dalam penelitian kami bersumber dari situs Github dengan alamat URL <https://github.com/spMohanty/PlantVillage-Dataset>. *Dataset* tersebut berupa kumpulan citra daun dengan 38 kelas yang terdiri atas beberapa jenis tanaman sayur dan buah. Namun pada penelitian kami hanya menggunakan 4 kelas, antara lain: daun apel *Scab*, daun apel *Black Rot*, daun apel *Cedar Rust*, dan daun apel *Healthy*. Alasan pengurangan jumlah kelas agar kami dapat mereduksi lamanya proses pelatihan yang disebabkan oleh banyaknya jumlah citra daun. Kemudian dari 4 kelas gambar tersebut, kami satukan menjadi dua *folder* yakni *train* (untuk proses pelatihan) dan *val* (untuk proses prediksi). Tujuan *dataset* ini ialah untuk memprediksi penyakit pada tanaman apel berdasarkan citra daun. Berikut kerangka dan isi *folder* yang ada dalam *dataset* kami.

→ dataset (*Folder*)

→ train (*Folder*)

→ daun\_apel\_black\_rot (*Folder*, berisi 496 gambar)

→ daun\_apel\_cedar\_rust (*Folder*, berisi 220 gambar)

→ daun\_apel\_healthy (*Folder*, berisi 1.316 gambar)

→ daun\_apel\_scab (*Folder*, berisi 504 gambar)

→ val (*Folder*)





→ daun\_apel\_black\_rot (*Folder*, berisi 125 gambar)

→ daun\_apel\_cedar\_rust (*Folder*, berisi 55 gambar)

→ daun\_apel\_healthy (*Folder*, berisi 329 gambar)

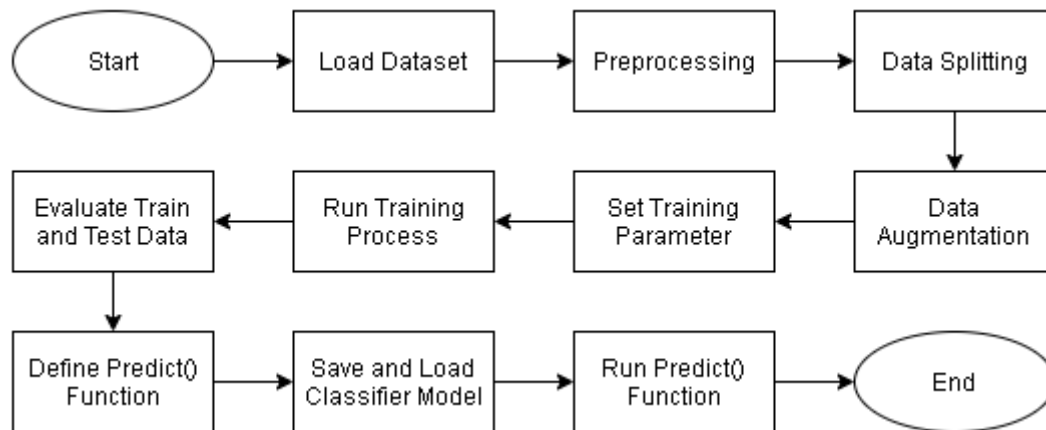
→ daun\_apel\_scab (*Folder*, berisi 126 gambar)

Penjelasan terkait masing-masing kelas citra daun, dapat dilihat pada tabel di bawah ini.

No.	Citra Daun	Kelas Citra	Deskripsi / Uraian
1.		Black Rot	Gejala sering terjadi pada awal musim semi, ketika daun membentang. Penyakit ini muncul sebagai bintik-bintik kecil berwarna coklat di permukaan atas daun dan membesar dengan diameter 1/8 inci hingga 1/4 inci.
2.		Cedar Rust	Ketika musim semi mandi dimulai, <i>galls</i> tumbuh dan melepaskan spora, yang dibawa oleh angin untuk menginfeksi pohon apel. Spora tersebut kemudian menjadi penyakit yang menyebabkan bintik-bintik merah coklat pada daun dan dapat merusak buah. Gejala yang terjadi pada daun sangat mempengaruhi kemampuan pohon apel untuk mengumpulkan sinar matahari dan nutrisi dari udara, merusak produksi buah, dan dapat menyebabkan kematian tanaman.
3.		Scab	Penyakit <i>Scab</i> dapat menyerang daun dan buah. Penyakit ini membentuk bintik-bintik hitam di permukaan atas daun. Bintik-bintik gelap juga dapat muncul di permukaan bawah. Daun yang terinfeksi parah menjadi layu dan gugur saat musim panas. Gejala pada buah mirip dengan yang ditemukan pada daun. Bintik-bintik coklat dan mungkin memiliki spora di tengah. Buah yang sangat terpengaruh oleh penyakit ini dapat jatuh, bahkan ketika masih berusia muda.
4.		Healthy	Daun dengan kelas <i>Healthy</i> memiliki pola dan karakteristik pertumbuhan yang berbeda. Daun sehat biasanya memiliki pertumbuhan yang kuat (lebat), warna hijau seragam, dan daun terbuka bukan melengkung (membentang tegak).

## II. Langkah-langkah dalam Machine Learning

Adapun tahap-tahap *machine learning* yang dilakukan dalam penelitian ini seperti diilustrasikan pada gambar di bawah ini.



### 1) Load Dataset

Pada tahap ini, dilakukan proses memanggil *dataset.zip* yang telah disimpan di Google Drive. Setelah itu, *dataset* tersebut akan diekstraksi ke dalam *folder* yang telah ditentukan.

### 2) Preprocessing

Pada tahap ini, dilakukan proses mengatur *folder* data untuk pelatihan dan normalisasi (*resize*) gambar ke dimensi 256x256 piksel. Setelah itu, *list* gambar dikonversi menjadi *array* gambar. Kemudian dilakukan transformasi kelas citra daun dari bentuk *list* menjadi biner dan menyimpan hasilnya agar mesin komputer lebih mudah dalam melakukan klasifikasi. Fitur yang digunakan dalam penelitian ini ialah ekstraksi bentuk dan warna pada citra daun.

### 3) Data Splitting

Pada tahap ini, dilakukan proses membagi *dataset* menjadi data latih dan data uji dengan perbandingan 80:20. Hal ini perlu dilakukan agar mesin komputer dapat melakukan pelatihan dan menguji data berdasarkan hasil pelatihan, kemudian mengevaluasi hasil pengujian.

### 4) Data Augmentation

Pada tahap ini, dilakukan proses peningkatan (penggandaan) jumlah data yang digunakan dengan memanfaatkan *library preprocessing* dari Keras. Selain itu, penggandaan dilakukan dengan menerapkan beberapa kriteria *preprocessing* gambar seperti *rotate*, *zoom*, *flip*, dan sebagainya. Hal ini perlu dilakukan agar mesin komputer dapat melakukan klasifikasi dengan lebih efektif karena telah mengenali berbagai macam fitur pada citra daun.

### 5) Set Training Parameter

Pada tahap ini, dilakukan proses mengatur parameter pelatihan dalam membangun model *classifier* citra daun. Adapun beberapa parameter yang didefinisikan antara lain: *epochs*,

*steps*, *LR*, *batch\_size*, *width*, *height*, *depth*. Uraian lebih lengkap dan penentuan nilai parameter ini akan dijelaskan pada bagian Skema Neural Network.

#### 6) Run Training Process

Pada tahap ini, dilakukan proses pelatihan dengan algoritma *Convolutional Neural Network* (CNN) terhadap data latih dan data uji yang telah dibagi sebelumnya. Citra daun yang digunakan dalam tahap pelatihan berasal dari folder *train*. Uraian lebih lengkap akan dijelaskan pada bagian Skema Neural Network.

#### 7) Evaluate Train and Test Data

Pada tahap ini, dilakukan proses evaluasi terhadap hasil pelatihan data latih dan pengujian data uji. Hasil evaluasi kemudian akan diilustrasikan ke dalam bentuk grafik *plot*.

#### 8) Define Predict() Function

Pada tahap ini, dilakukan proses mendefinisikan fungsi untuk memprediksi citra daun menggunakan model *classifier* hasil pelatihan.

#### 9) Save and Load Classifier Model

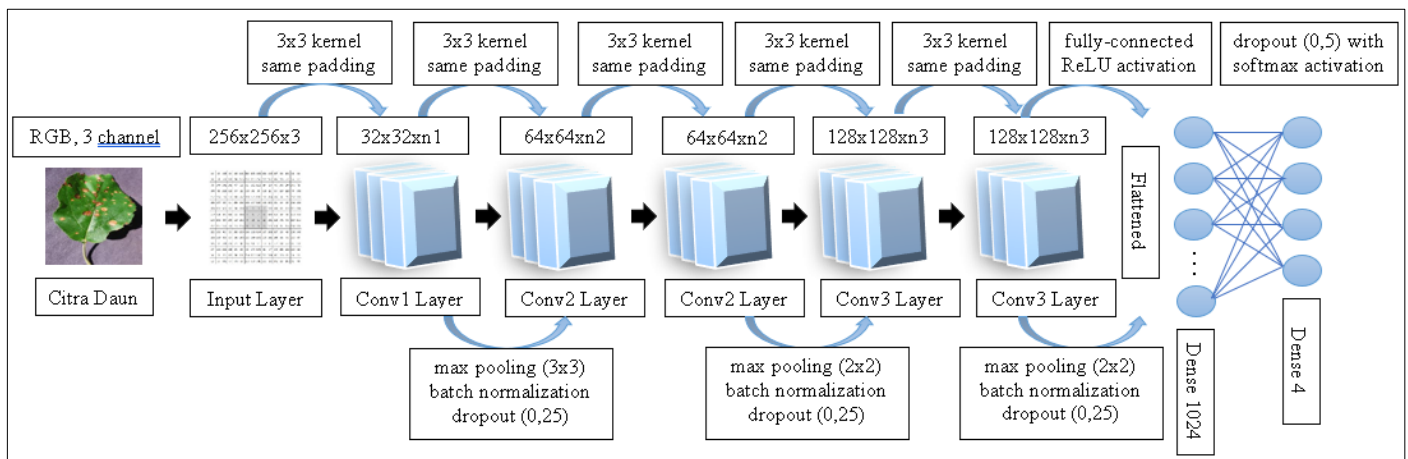
Pada tahap ini, dilakukan proses penyimpanan model *classifier* hasil pelatihan dan melakukan *load* (menggunakan kembali) model tersebut untuk digunakan dalam tahap prediksi.

#### 10) Run Predict() Function

Pada tahap ini, dilakukan proses prediksi dengan referensi model *classifier* yang telah disimpan sebelumnya. Citra daun yang digunakan dalam tahap prediksi berasal dari folder *val*. Sebenarnya, pada tahap inilah kita bisa benar-benar menilai akurasi model dengan akurat karena menggunakan gambar baru yang belum pernah dipelajari oleh mesin komputer.

### III. Skema Neural Network

Adapun skema *neural network* yang diterapkan dalam penelitian ini seperti diilustrasikan pada gambar di bawah ini.



- Citra yang digunakan pada penelitian ini ialah citra warna (*true color 8-bit*) yang mana mempunyai nilai spesifik dari kombinasi tiga *channel* warna RGB (Red, Green, Blue). Nilai intensitas warna RGB berada di antara 0 – 255. Citra tersebut kemudian dinormalisasi dengan dikecilkan dimensinya (*resize*) menjadi 256 x 256 piksel agar memiliki dimensi yang sama sehingga mudah untuk diolah.
- Adapun parameter-parameter yang digunakan dalam proses pelatihan sebagai berikut.
  - Jumlah *layer* = 28 (yang terdiri atas 6 *Layer BatchNormalization*, 5 *Layer Convolution*, 3 *Layer Pooling*, 7 *Layer Activation*, 1 *Layer Flatten*, 4 *Layer Dropout*, dan 2 *Layer Dense*)
  - Jumlah *neuron* = 1028 (yang terdiri atas 1024 *node input* dan 4 *node output*)
  - *Epochs* = 10
  - *Steps* = 100
  - *Learning rate* = 1e-3 (0,001)
  - *Batch size* = 32
  - *Width* = 256
  - *Height* = 256
  - *Depth* = 3
- Penjelasan atau deskripsi dari parameter-parameter di atas adalah sebagai berikut.
  - *Layer BatchNormalization*. *Layer* ini dirancang untuk secara otomatis menormalisasi *input* yang mana akan diproses pada *layer* selanjutnya. Dengan menggunakan *layer* ini, dapat mempercepat proses pelatihan cukup drastis, dan dalam beberapa kasus meningkatkan kinerja model melalui efek regularisasi yang sederhana.
  - *Layer Convolution*. Merupakan *layer* utama dalam Keras yang berfungsi menerapkan *filter* (aturan pergeseran matriks) terhadap *input* untuk membuat *features map* yang meringkas seluruh fitur yang ada pada sebuah *dataset*. Pada penelitian ini, digunakan *layer* jenis Conv2D karena *input* yang digunakan dalam bentuk data spasial (gambar).
  - *Layer Pooling*. *Layer* ini mirip dengan *layer convolution*, tapi memiliki fungsi tertentu seperti *max pooling*, yang mengambil nilai maksimum di bagian *filter* tertentu. Fungsi ini biasanya digunakan untuk mereduksi *input* secara spasial (mengurangi jumlah parameter) dengan operasi *down-sampling*.
  - *Layer Activation*. *Layer* ini menerapkan fungsi aktivasi terhadap *input* pada *layer-layer* sebelumnya. Adapun penelitian ini menggunakan fungsi aktivasi ReLU pada *input* karena dapat mempercepat pelatihan serta memperoleh performa yang lebih baik, dan *softmax* pada *output* karena mengimplementasikan klasifikasi multi-kelas.

- *Layer Flatten*. Secara sederhana, *layer* ini bertugas mengkonversi *features map* multi-dimensi menjadi satu dimensi tanpa seleksi fitur apa pun. *Layer* ini sangat dibutuhkan jika kita ingin menggunakan *Layer Dense (fully-connected)*.
- *Layer Dropout*. Adalah *layer* yang memperkirakan proses pelatihan pada sejumlah besar *neuron* dengan arsitektur yang berbeda secara paralel. *Layer* ini juga berfungsi untuk mencegah masalah *overfitting* yang mana menghasilkan performa buruk ketika memprediksi data yang belum dikenali sebelumnya.
- *Layer Dense*. Merupakan *layer* yang terhubung secara mendalam, yang berarti setiap *neuron* di *layer* ini menerima *input* dari semua *neuron* dari *layer* sebelumnya. Di belakang layar, *layer* ini melakukan perkalian matriks-vektor. Nilai yang digunakan dalam matriks sebenarnya adalah parameter yang dapat dilatih dan diperbarui dengan metode *backpropagation*. Output yang dihasilkan oleh *layer* ini adalah vektor multi-dimensi. Dengan demikian, *layer* ini pada dasarnya digunakan untuk mengubah dimensi vektor.
- *Epoch* adalah sebuah *hyperparameter* dari *gradient descent* yang mengontrol jumlah iterasi yang harus dilalui oleh *dataset* saat proses pelatihan berlangsung.
- *Step per epoch* adalah jumlah langkah yang harus dilalui setiap *epoch* berlangsung.
- *Learning rate* adalah nilai yang mengatur seberapa besar *update* yang dilakukan terhadap nilai bobot. Jika *learning rate* diatur cukup kecil, maka nilai *error rate* juga dijamin turun setiap kali *update*.
- *Batch size* adalah jumlah sampel data yang disebarkan ke tiap *neuron*. Contoh: jika kita mempunyai 100 baris data dan *batch size* kita adalah 5, maka *neural network* akan menggunakan 5 sampel data pertama dari 100 baris data yang kita miliki, lalu disebarkan ke tiap *neuron* untuk dilakukan proses pelatihan.
- *Width*, *height*, dan *depth* adalah dimensi atau ukuran filter pada *layer convolution*. Ukuran ketebalan (*depth*) dari sebuah filter selalu mengikuti ketebalan/volume dari *input* yang digunakan. Tinggi dan lebar (*height* dan *width*) pada umumnya berukuran ganjil. Secara intuisi, filter berukuran ganjil memberikan representasi yang lebih baik.
- *Padding* adalah parameter yang menentukan jumlah *pixel* (berisi nilai 0) yang akan ditambahkan di setiap sisi dari *input*. Hal ini digunakan dengan tujuan untuk memanipulasi dimensi *output* dari *layer convolution*. Dengan menggunakan *padding*, kita dapat mengukur dimensi *output* agar tetap sama seperti dimensi *input*, atau setidaknya tidak berkurang secara drastis. Sehingga kita bisa menggunakan *layer convolution* yang lebih dalam dan lebih banyak fitur yang berhasil diekstrak.

- *Kernel* atau *filter* adalah *Kernel* adalah matriks yang menggerakkan data *input*, melakukan fungsi perkalian *dot product* terhadap data *input*, dan mendapatkan *output* sebagai matriks *dot product*. *Kernel* bergerak pada data *input* berdasarkan nilai *stride*.
- *Stride* adalah parameter yang menentukan berapa jumlah pergeseran *filter*. Jika nilai *stride* adalah 1, maka *filter* pada *layer convolution* akan bergeser sebanyak 1 piksel secara horizontal lalu vertikal.

### Cara Kerja atau Alur CNN:

- 1) Memecah gambar menjadi gambar dengan bagian yang lebih kecil. Dari gambar sebuah daun, hasil dari proses konvolusi dapat diilustrasikan seperti sebuah matriks yang berisi pecahan bagian gambar yang lebih kecil. Dengan ini, gambar asli dari sebuah daun dipecah menjadi 256 gambar yang lebih kecil dengan dimensi yang telah ditentukan di atas.
- 2) Memasukkan setiap gambar yang lebih kecil ke setiap *node neural network*. Setiap gambar kecil dari hasil konvolusi tersebut kemudian dijadikan *input* untuk menghasilkan sebuah representasi fitur. Hal inilah yang kemudian memberikan algoritma CNN kemampuan mengenali sebuah objek, di manapun posisi objek tersebut muncul pada sebuah gambar. Proses ini dilakukan untuk semua bagian dari masing-masing gambar kecilnya, dengan menggunakan *filter* yang sama. Dengan kata lain, setiap bagian gambar akan memiliki faktor pengali yang sama.
- 3) Menyimpan hasil dari masing-masing gambar kecil ke dalam *array*. Ukuran *array* nantinya tergantung pada jumlah masing-masing gambar kecil tersebut.
- 4) *Down-sampling*. Pada langkah 3, *array* masih terlalu besar karena memuat gambar yang cukup banyak. Maka untuk mengecilkan ukuran *array* itu digunakan *down-sampling* yang dalam *neural network* dinamakan *max-pooling* (mengambil nilai *pixel* terbesar di setiap *pooling kernel*). Dengan begitu, sekalipun mengurangi jumlah parameter, informasi terpenting dari bagian gambar tersebut tetap diambil.
- 5) Membuat prediksi. Sebelumnya kita telah merubah dari gambar yang berukuran besar menjadi *array* yang cukup kecil. *Array* merupakan sekelompok angka, jadi dengan menggunakan *array* kecil itu, kita bisa masukkan ke dalam *node neural network* lain. *Node neural network* yang paling terakhir akan memutuskan apakah gambarnya cocok atau tidak. Untuk memberikan perbedaan dari langkah konvolusi, maka proses ini bisa kita sebut dengan *fully-connected network*.

#### IV. Penjelasan Source Code

```
1 import os
2 from os import listdir
3 import zipfile
4 local_zip = "/content/drive/My Drive/dataset.zip"
5 zip_ref = zipfile.ZipFile(local_zip, 'r')
6 zip_ref.extractall("/content/drive/My Drive/workspace")
7 zip_ref.close()
8 !ls "/content/drive/My Drive/workspace/dataset"
9 import numpy as np
10 import pickle
11 import cv2
12 import matplotlib.pyplot as plt
13 from sklearn.preprocessing import LabelBinarizer
14 from keras.models import Sequential
15 from keras.layers.normalization import BatchNormalization
16 from keras.layers.convolutional import Conv2D
17 from keras.layers.convolutional import MaxPooling2D
18 from keras.layers.core import Activation, Flatten, Dropout, Dense
19 from keras import backend as K
20 from keras.preprocessing.image import ImageDataGenerator
21 from keras.optimizers import Adam
22 from keras.preprocessing import image
23 from keras.preprocessing.image import img_to_array
24 from sklearn.preprocessing import MultiLabelBinarizer
25 from sklearn.model_selection import train_test_split
26 DEFAULT_IMAGE_SIZE = tuple((256, 256))
27 N_IMAGES = 1000
28 root_dir = '/content/drive/My Drive/workspace/dataset'
29 train_dir = os.path.join(root_dir, 'train')
30 val_dir = os.path.join(root_dir, 'val')
```

Baris ke-1 dan 2 adalah *script* untuk memanggil *library* OS yang berfungsi menjembatani interaksi antara Google Colab dengan sistem operasi kita. Baris ke-3 adalah *script* untuk memanggil *library* ZIPFILE yang berfungsi mengekstraksi *file dataset.zip* yang telah diunggah ke Google Drive. Baris ke-4 sampai 7 adalah *script* yang berfungsi untuk melakukan proses ekstraksi *file dataset.zip* ke dalam *folder* yang telah ditentukan. Baris ke-8 adalah *script* yang berfungsi untuk menampilkan isi *folder* yang telah ditentukan. Baris ke-9 sampai 25 adalah *script* untuk memanggil *library-library* yang dibutuhkan untuk membangun sistem pengklasifikasi citra, antara lain: numpy, pickle, cv2, matplotlib, scikit-learn, dan keras. Baris ke-26 adalah *script* yang berfungsi untuk mendefinisikan dimensi gambar yang akan dijadikan *input* ke sistem pengklasifikasi citra. Baris ke-27 adalah *script* yang berfungsi untuk mendefinisikan jumlah gambar yang akan digunakan dalam sistem pengklasifikasi citra. Baris ke-28 sampai 30 adalah *script* yang berfungsi untuk mendefinisikan *folder* tempat *dataset* dan data prediksi berada.



```

32 def convert_image_to_array(image_dir):
33     try:
34         image = cv2.imread(image_dir)
35         if image is not None:
36             image = cv2.resize(image, DEFAULT_IMAGE_SIZE)
37             return img_to_array(image)
38         else:
39             return np.array([])
40     except Exception as e:
41         print(f"Error : {e}")
42         return None
43 image_list, label_list = [], []
44 try:
45     print("[INFO] Sedang memproses gambar...")
46     plant_disease_folder_list = listdir(train_dir)
47     for plant_disease_folder in plant_disease_folder_list:
48         print(f"[INFO] Memproses folder {plant_disease_folder}...")
49         plant_disease_image_list = listdir(f"{train_dir}/{plant_disease_folder}")
50         for image in plant_disease_image_list[:N_IMAGES]:
51             image_directory = f"{train_dir}/{plant_disease_folder}/{image}"
52             if image_directory.endswith(".jpg")==True or image_directory.endswith(".JPG")==True:
53                 image_list.append(convert_image_to_array(image_directory))
54                 label_list.append(plant_disease_folder)
55         print("[INFO] Berhasil memproses seluruh gambar.")
56 except Exception as e:
57     print(f"Error : {e}")
58 np_image_list = np.array(image_list, dtype=np.float16) / 225.0
59 print()
60 image_len = len(image_list)
61 print(f"Jumlah seluruh gambar: {image_len}")

```

Baris ke-32 sampai 42 adalah *script* yang berfungsi mendefinisikan *function* untuk mengkonversi gambar menjadi bentuk *array* dengan memanfaatkan *library* cv2. Pada baris tersebut juga terjadi proses menormalisasi atau mereduksi (*resize*) dimensi gambar menjadi ukuran 256 x 256 piksel. Baris ke-43 adalah *script* yang berfungsi untuk mendefinisikan tipe data *list* dari variabel *image\_list* dan *label\_list*. Baris ke-44 sampai 57 adalah *script* untuk memproses atau membaca gambar yang ada pada *folder dataset* dengan melakukan iterasi pada tiap *folder*. Setelah semua gambar berhasil terbaca, selanjutnya akan disatukan dan dimasukkan ke variabel yang telah ditentukan sebelumnya dengan perintah *append*. Baris ke-58 dan 59 adalah *script* yang berfungsi untuk mengkonversi variabel *image\_list* yang bertipe data *list* menjadi bentuk *array*. Baris ke-60 dan 61 adalah *script* yang berfungsi untuk menghitung jumlah gambar yang berhasil terbaca.

```

62 label_binarizer = LabelBinarizer()
63 image_labels = label_binarizer.fit_transform(label_list)
64 pickle.dump(label_binarizer, open('plant_disease_label_transform.pkl', 'wb'))
65 n_classes = len(label_binarizer.classes_)
66 print("Jumlah kelas gambar: ", n_classes)
67 augment = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
68                               height_shift_range=0.1, shear_range=0.2,
69                               zoom_range=0.2, horizontal_flip=True,
70                               fill_mode="nearest")
71 x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state=45)
72 EPOCHS = 10
73 STEPS = 100
74 LR = 1e-3
75 BATCH_SIZE = 32
76 WIDTH = 256
77 HEIGHT = 256
78 DEPTH = 3
79 model = Sequential()
80 inputShape = (HEIGHT, WIDTH, DEPTH)
81 chanDim = -1
82 if K.image_data_format() == "channels_first":
83     inputShape = (DEPTH, HEIGHT, WIDTH)
84     chanDim = 1

```

Baris ke-62 dan 63 adalah *script* yang berfungsi untuk mentransformasi kelas atau label citra daun menjadi *feature map* yang mana akan digunakan untuk proses pelatihan. Baris ke-64 adalah *script* yang berfungsi untuk menyimpan data hasil konversi label citra yang mana akan digunakan untuk proses prediksi atau validasi. Baris ke-65 dan 66 adalah *script* yang berfungsi untuk menghitung jumlah kelas citra daun. Baris ke-67 sampai 70 adalah *script* yang berfungsi untuk melakukan proses augmentasi data (memperbanyak sampel gambar dengan kriteria tertentu) agar sistem pengklasifikasi dapat mengenali citra daun dengan berbagai kondisi, sehingga akan meningkatkan performa saat pelatihan berlangsung. Baris ke-71 adalah *script* yang berfungsi untuk melakukan proses *splitting dataset* menjadi data latih dan data uji (x artinya *input* citra dan y artinya *output* atau label citra) dengan perbandingan 80:20 (*Pareto Principle*). Baris ke-72 sampai 78 adalah *script* yang berfungsi untuk mendefinisikan parameter-parameter yang akan digunakan dalam proses pelatihan. Baris ke-79 adalah *script* yang berfungsi untuk membuat model sekuensial sehingga mudah dikonfigurasi per *layer*. Baris ke-80 adalah *script* yang berfungsi untuk mendefinisikan dimensi *input* yang akan dimasukkan ke dalam *layer* (256x256 adalah dimensi citra yang telah *resize*, 3 adalah jumlah *channel* pada citra RGB). Sehingga nantinya bentuk *array input* (citra) kita menjadi kubus karena memiliki parameter *depth*. Baris ke-81 adalah *script* yang berfungsi untuk mengaktifkan fitur peningkatan performa pelatihan dengan mengatur *channel ordering format* menjadi *channel\_last*. Parameter -1 artinya *channel* citra akan menjadi dimensi *input* terakhir ([*height*][*width*][*channel*]) Baris ke-82 sampai 84 adalah *script* yang berfungsi mengaktifkan juga *channel ordering format* menjadi *channel\_first*. Kita perlu mengaktifkan keduanya agar bisa memperoleh performa pelatihan yang lebih baik.

```
85 model.add(Conv2D(32, (3, 3), padding="same", input_shape=input_shape))
86 model.add(Activation("relu"))
87 model.add(BatchNormalization(axis=chanDim))
88 model.add(MaxPooling2D(pool_size=(3, 3)))
89 model.add(Dropout(0.25))
90 model.add(Conv2D(64, (3, 3), padding="same"))
91 model.add(Activation("relu"))
92 model.add(BatchNormalization(axis=chanDim))
93 model.add(Conv2D(64, (3, 3), padding="same"))
94 model.add(Activation("relu"))
95 model.add(BatchNormalization(axis=chanDim))
96 model.add(MaxPooling2D(pool_size=(2, 2)))
97 model.add(Dropout(0.25))
98 model.add(Conv2D(128, (3, 3), padding="same"))
99 model.add(Activation("relu"))
100 model.add(BatchNormalization(axis=chanDim))
101 model.add(Conv2D(128, (3, 3), padding="same"))
102 model.add(Activation("relu"))
103 model.add(BatchNormalization(axis=chanDim))
104 model.add(MaxPooling2D(pool_size=(2, 2)))
105 model.add(Dropout(0.25))
106 model.add(Flatten())
107 model.add(Dense(1024))
108 model.add(Activation("relu"))
109 model.add(BatchNormalization())
110 model.add(Dropout(0.5))
111 model.add(Dense(n_classes))
112 model.add(Activation("softmax"))
113 model.summary()
```

Baris ke-85 sampai 89 adalah *script* yang berfungsi untuk membuat *layer convolution* pertama yaitu dengan bentuk 2D (data spasial), *filter* berukuran 32, *filter* berdimensi 3 x 3, *same padding* aktif (dimensi *output* akan sama dengan dimensi *input*), *input\_shape* sesuai dengan dimensi yang telah didefinisikan sebelumnya. Kemudian pada *layer convolution* pertama ini juga diterapkan fungsi aktivasi ReLU, fungsi BatchNormalization, fungsi MaxPooling dengan ukuran *pool* 3 x 3, dan fungsi *dropout* dengan nilai 0,25. Baris ke-90 sampai 92 adalah *script* yang berfungsi untuk membuat *layer convolution* kedua yaitu dengan bentuk 2D (data spasial), *filter* berukuran 64, *filter* berdimensi 3 x 3, dan *same padding* aktif (dimensi *output* akan sama dengan dimensi *input*). Kemudian pada *layer convolution* kedua ini juga diterapkan fungsi aktivasi ReLU dan fungsi BatchNormalization. Baris ke-93 sampai 97 adalah *script* yang berfungsi untuk membuat *layer convolution* ketiga yaitu dengan bentuk 2D (data spasial), *filter* berukuran 64, *filter* berdimensi 3 x 3, dan *same padding* aktif (dimensi *output* akan sama dengan dimensi *input*). Kemudian pada *layer convolution* ketiga ini juga diterapkan fungsi aktivasi ReLU, fungsi BatchNormalization, fungsi MaxPooling dengan ukuran *pool* 2 x 2, dan fungsi *dropout* dengan nilai 0,25. Baris ke-98 sampai 100 adalah *script* yang berfungsi untuk membuat *layer convolution* keempat yaitu dengan bentuk 2D (data spasial), *filter* berukuran 128, *filter* berdimensi 3 x 3, dan *same padding* aktif (dimensi *output* akan sama dengan dimensi *input*). Kemudian pada *layer convolution* keempat ini juga diterapkan fungsi aktivasi ReLU dan fungsi BatchNormalization. Baris ke-101 sampai 105 adalah *script* yang berfungsi untuk membuat *layer convolution* kelima yaitu dengan bentuk 2D (data spasial), *filter* berukuran 128, *filter* berdimensi 3 x 3, dan *same padding* aktif (dimensi *output* akan sama dengan dimensi *input*). Kemudian pada *layer convolution* kelima ini juga diterapkan fungsi aktivasi ReLU, fungsi BatchNormalization, fungsi MaxPooling dengan ukuran *pool* 2 x 2, dan fungsi *dropout* dengan nilai 0,25. Selanjutnya akan dilakukan proses *flattening* (mengkonversi vektor data dua dimensi menjadi vektor data satu dimensi). Berikutnya adalah proses *dense* yang mana berfungsi untuk menambahkan *layer fully connected* (*neuron* di *layer* awal saling terhubung dengan *layer* berikutnya). 1024 adalah jumlah *node* atau *neuron input* dalam *layer fully connected* pertama. Kemudian pada *layer fully connected* pertama ini juga diterapkan fungsi aktivasi ReLU, fungsi BatchNormalization, dan fungsi *dropout* dengan nilai 0,5. Lalu ditambahkan lagi *layer fully connected* kedua (*neuron* di *layer* awal saling terhubung dengan *layer* berikutnya). 4 adalah jumlah *node* atau *neuron output* (kelas atau label citra) dalam *layer fully connected* kedua. Kemudian pada *layer fully connected* kedua ini juga diterapkan fungsi aktivasi *softmax*. Baris ke-113 adalah *script* yang berfungsi untuk menampilkan ringkasan konfigurasi yang telah diterapkan pada model pengklasifikasi citra daun.

```

114 opt = Adam(learning_rate=LR, decay=LR/EPOCHS)
115 model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
116 print("[INFO] Sedang melatih model...")
117 history = model.fit(augment.flow(x_train, y_train, batch_size=BATCH_SIZE),
118                     validation_data=(x_test, y_test),
119                     steps_per_epoch=len(x_train) // BATCH_SIZE,
120                     epochs=EPOCHS, verbose=1)
121 acc = history.history['accuracy']
122 val_acc = history.history['val_accuracy']
123 loss = history.history['loss']
124 val_loss = history.history['val_loss']
125 epochs = range(1, len(acc) + 1)
126 plt.plot(epochs, acc, 'b', label='Training accuracy')
127 plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
128 plt.title('Training and Validation accuracy')
129 plt.legend()
130 plt.figure()
131 plt.plot(epochs, loss, 'b', label='Training loss')
132 plt.plot(epochs, val_loss, 'r', label='Validation loss')
133 plt.title('Training and Validation loss')
134 plt.legend()
135 plt.show()
136 print("[INFO] Menghitung akurasi model...")
137 scores = model.evaluate(x_test, y_test)
138 print(f"Test Accuracy: {scores[1]*100}")

```

Baris ke-114 adalah *script* yang berfungsi untuk mengkonfigurasi *optimizer* yang akan digunakan pada proses pelatihan yang mana dengan parameter *decay*, nilai *learning rate* akan diperbarui setiap *epoch* tertentu berhasil dilalui. Baris ke-115 adalah *script* yang berfungsi untuk mengkonfigurasi model dengan menentukan perhitungan *loss rate* menggunakan *binary cross-entropy*, *Adam optimizer*, dan perhitungan akurasi pelatihan. Baris ke-116 sampai 120 adalah *script* untuk menjalankan proses pelatihan dan menampilkan hasil performa tiap *epoch*. Baris ke-121 sampai 135 adalah *script* yang berfungsi untuk menampilkan visualisasi graf *line plot* yang berisi nilai *loss rate* dan akurasi tiap *epoch* dari hasil proses pelatihan. Baris ke-136 sampai 138 adalah *script* yang berfungsi untuk mengevaluasi akurasi pada data uji dan menampilkan dalam bentuk persentase.

```

139 def make_keras_picklable():
140     def __reduce__(self):
141         model_metadata = saving_utils.model_metadata(self)
142         training_config = model_metadata.get("training_config", None)
143         model = serialize(self)
144         weights = self.get_weights()
145         return (unpack, (model, training_config, weights))
146
147     cls = Model
148     cls.__reduce__ = __reduce__
149     make_keras_picklable()
150     with open('model.pkl', 'wb') as f:
151         pickle.dump(model, f)
152     filename = 'plant_disease_label_transform.pkl'
153     image_labels = pickle.load(open(filename, 'rb'))
154     # Membuat fungsi untuk prediksi gambar
155     def predict_disease(image_path):
156         image_array = convert_image_to_array(image_path)
157         np_image = np.array(image_array, dtype=np.float16) / 225.0
158         np_image = np.expand_dims(np_image, 0)
159         plt.imshow(plt.imread(image_path))
160         result = model.predict_classes(np_image)
161         print((image_labels.classes_[result][0]))
162     predict_disease('/content/drive/My Drive/workspace/dataset/val/daun_apel_black_rot/e4d473a3-83ad-496d-a97f-5aee9d713d69__JR_FrgE.S 2782.JPG')
163     predict_disease('/content/drive/My Drive/workspace/dataset/val/daun_apel_cedar_rust/1e099d8e-b40c-4cf3-bb52-4e77b5a2dc69__FREC_C.Rust 0155.JPG')
164     predict_disease('/content/drive/My Drive/workspace/dataset/val/daun_apel_scab/d8b8a834-08df-43d9-8a1a-4bc8e508a1e3__FREC_Scab 3280.JPG')

```

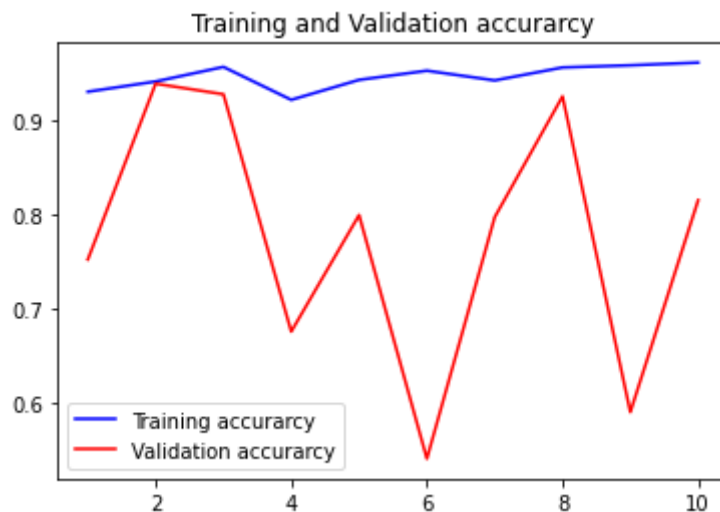
Baris ke-139 sampai 148 adalah *script* yang berfungsi untuk mendefinisikan fungsi agar model pengklasifikasi yang telah dilatih dapat disimpan (*pickable*). Baris ke-149 sampai 151 adalah *script* yang berfungsi untuk menyimpan model untuk digunakan pada proses prediksi nantinya. Baris ke-

152 dan 153 adalah *script* yang berfungsi untuk membuka atau mengaktifkan model data hasil konversi label citra yang telah disimpan sebelumnya. Baris ke-154 sampai 161 adalah *script* yang berfungsi untuk mendefinisikan fungsi prediksi citra yang mana telah menerapkan model pengklasifikasi dan model label. Baris ke-162 sampai 164 adalah *script* yang berfungsi untuk melakukan proses prediksi terhadap citra daun yang mana telah ditentukan lokasi *file path*-nya.

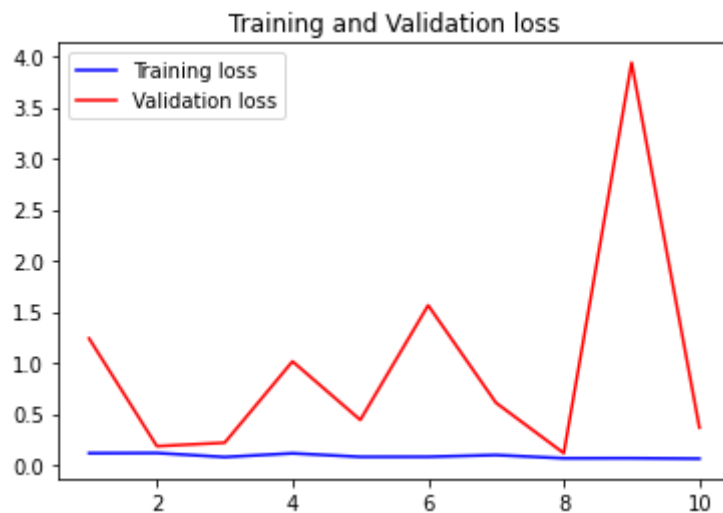
## V. Hasil dan Simpulan

<i>Epoch ke-</i>	<i>Akurasi Train</i>	<i>Loss Train</i>	<i>Akurasi Test</i>	<i>Loss Test</i>
1	0,9374	0,0980	0,7523	1,2378
2	0,9346	0,1340	0,9392	0,1838
3	0,9601	0,0710	0,9279	0,2177
4	0,9290	0,0959	0,6757	1,0133
5	0,9431	0,0778	0,7995	0,4406
6	0,9559	0,0744	0,5405	1,5622
7	0,9272	0,1137	0,7973	0,6084
8	0,9608	0,0556	0,9257	0,1157
9	0,9557	0,0692	0,5901	3,9368
10	0,9529	0,0808	0,8153	0,3657
<b>Rata-rata</b>	0,9456	0,0870	0,7763	0,9682

**Tabel** Akurasi dan Loss pada Proses Pelatihan



**Grafik** Akurasi pada Data Latih dan Data Uji



**Grafik** Loss Rate pada Data Latih dan Data Uji

Berdasarkan hasil perancangan dan implementasi sistem pengklasifikasi penyakit pada tanaman apel berdasarkan citra daun, maka kami memperoleh simpulan sebagai berikut:

- 1) Model ini sudah berhasil melakukan prediksi penyakit pada tanaman apel berdasarkan citra daun menggunakan algoritma *Convolutional Neural Network* (CNN).
- 2) *Dataset* citra daun terdiri atas 2220 data pelatihan dan 635 data validasi. Dari data pelatihan tersebut kemudian dilakukan proses *data splitting* dengan perbandingan 80:20.
- 3) Proses *preprocessing* yang dilakukan pada citra adalah melakukan reduksi dimensi citra agar berukuran sama, yaitu menjadi 256x256 piksel dan menggunakan tipe gambar RGB.
- 4) Model dapat mengklasifikasikan citra daun yang diujikan dengan baik, yaitu penyakit *apple scab*, *apple black rot*, *apple cedar rust*, dan tanaman apel sehat. Hasil akurasi final yang diperoleh dari pengujian terhadap data uji adalah sebesar 81,53%.

## VI. Saran

Nilai akurasi yang diperoleh pada data uji masih belum cukup baik. Hal ini masih dapat ditingkatkan pada penelitian selanjutnya. Maka dari itu, kami memberikan saran sebagai berikut:

- 1) Gunakan jumlah gambar yang mendekati pada masing-masing kelas agar dapat menghindari masalah *imbalanced classification*. Atau alternatifnya, gunakan metode *oversampling* untuk memperbanyak sampel gambar pada kelas dengan jumlah minoritas.
- 2) Gunakan ekstraksi fitur lain pada citra seperti tekstur, *grayscale*, dan sebagainya. Hal ini dapat membantu agar pengetahuan di bidang *image processing* terus berkembang.
- 3) Ubah parameter-parameter yang ada seperti dimensi citra, *epoch*, *learning rate*, dan sebagainya. Dengan semakin banyak percobaan yang dilakukan, bukan tidak mungkin untuk bisa memperoleh nilai akurasi yang maksimal.

- 4) Sistem yang diterapkan pada penelitian ini hanya sebatas analisis dan belum diimplementasikan dalam sebuah aplikasi *web* atau *mobile*. Dengan memanfaatkan model yang telah siap, tentunya akan lebih menghemat waktu dan berharap banyak masyarakat (khususnya petani apel) merasa terbantu sehingga hasil panen meningkat.

## VI. Referensi

- <https://keras.io/guides>
- <https://machinelearningmastery.com/category/deep-learning>
- <https://medium.com/@16611110/apa-itu-convolutional-neural-network-836f70b193a4>
- <https://medium.com/data-folks-indonesia/pemahaman-dasar-convolutional-neural-networks-bfa1bf0b06e1>
- <https://medium.com/@kevalnagda/plant-disease-detector-ddd914687349>
- <https://medium.com/@samuelsena/pengenalan-deep-learning-part-7-convolutional-neural-network-cnn-b003b477dc94>
- <https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers>
- <https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac>