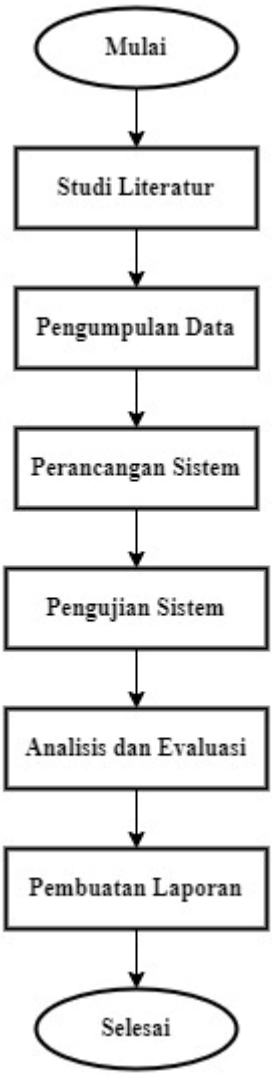
BAB 3
METODE PENELITIAN

3.1 TAHAPAN PERANCANGAN

3.2.1 Alur Penelitian

Langkah-langkah yang dijalankan dalam penelitian ini mengenai klasifikasi penyakit daun jagung menggunakan metode CNN dengan arsitektur VGG16 dan VGG19. Adapun alur penelitian terdapat pada Gambar 3.1.



Gambar 3. 1. Alur Penelitian

Pada Gambar 3.1 adalah tahapan dalam pengerjaan skripsi melibatkan langkah-langka, yaitu:

1. Studi Literatur

   Tahap ini, dilakukan penelitian studi literatur dengan mengumpulkan referensi dari berbagai sumber seperti buku, jurnal, artikel dan penelitian sebelumnya untuk memperoleh informasi dan memperluas pemahaman materi.
2. Pengumpulan Data

   Proses ini melibatkan pengumpulan data yang tersedia, dengan mengggunakan dataset penyakit daun jagung yang di ambil dari website *www.kaggle.com*.
3. Perancangan Sistem

   Sistem dirancang dengan metode CNN dengan menerapkan 2 arsitektur yaitu VGG16 dan VGG19.
4. Pengujian Sistem

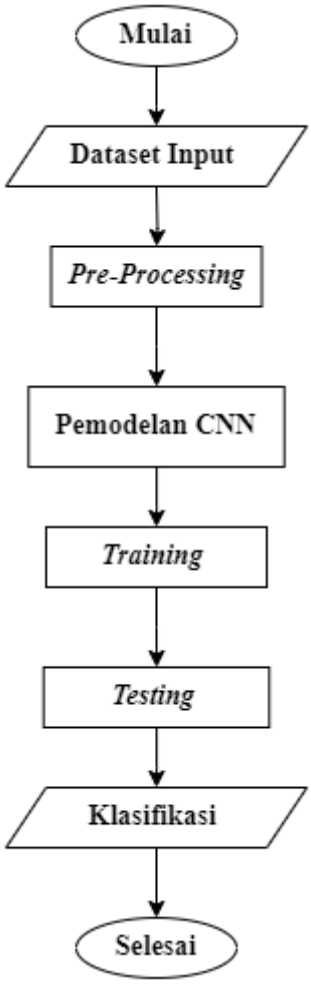
   Pada tahap ini, sistem diuji untuk mengklasifikasi penyakit daun jagung berdasarkan citra daun menggunakan metode CNN dengan arsitektur VGG16 dan VGG19 menggunakan optimasi SGD.
5. Analisis dan Evaluasi

   Tahap ini, melibatkan analisa kinerja sistem telah dirancang untuk klasifikasi penyakit daun jagung.
6. Pembuatan Laporan

   Terakhir, dilakukan pembuatan laporan yang berisi kesimpulan akhir dari analisa dan pengujian sistem dalam bentuk laporan skripsi.

3.2.2 Tahapan Perancangan Sistem

Dalam penelitian ini, dilakukan perancangan mengenai sistem untuk menerapkan metode CNN dengan 2 arsitektur VGG16 dan VGG19 dalam mengklasifikasi penyakit daun jagung yaitu 3 kelas berpenyakit dan 1 kelas sehat yaitu *Blight, Common Rust, Gray Leaf Spot*, dan *Healthy*. Adapun tahapan peracangan sistem terdapat pada Gambar 3.2.



Gambar 3. 2. Perancangan Sistem

Berdasarkan Gambar 3.2 mengenai sistem yang diusulkan terdiri dari beberapa tahapan, yaitu:

1. Dataset

   Perancangan sistem dimulai dengan tahap pengumpulan dataset. Pada tahap ini, dilakukan proses *input* dataset yang melibatkan pengumpulan data untuk keperluan sistem. Dataset yang di kumpulkan berupa data citra dan diambil dari website *kaggle.com* yang mencakup dari 3 kelas dari penyakit pada daun jagung dan 1 kelas sehat yaitu *Blight, Common Rust, Gray Leaf Spot*, dan *Healthy.*

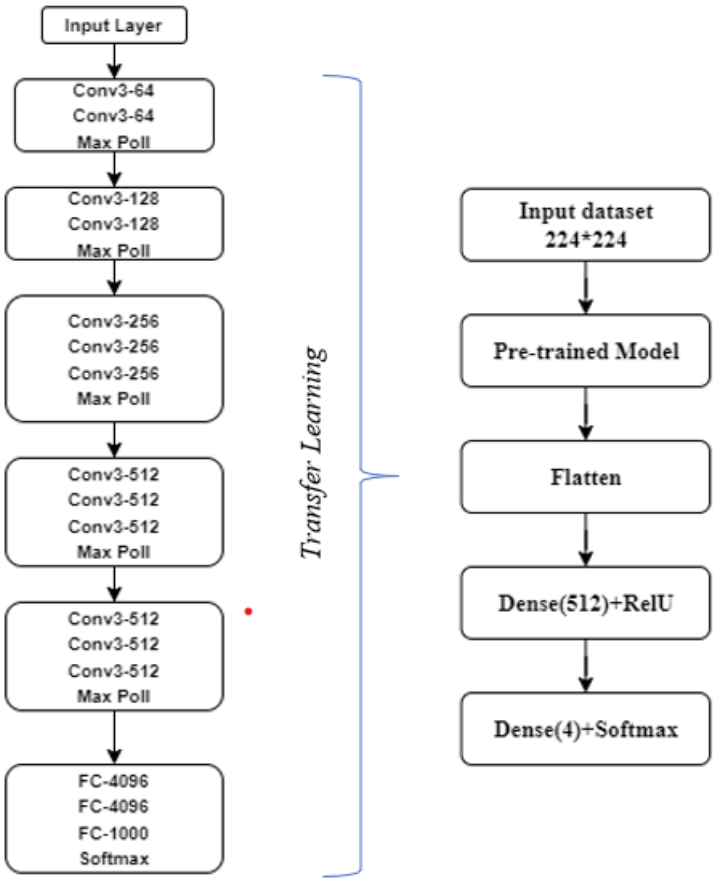
2. *Pre-processing*

   Setelah proses *input* citra, dilakukan *pre-processing* untuk mengoptimalkan kualitas citra dan meningkatkan kemampuan identifikasi objek dari sistem. Tahap

pra-pemrosesan hanya melibatkan perubahan dimensi citra. *Pre-processing* data ekspansi dilakukan dengan mengubah ukuran dan memperluas data. Data penelitian ini untuk pembagiannya yaitu 80% data *training* dan 20% data *testing.*

3. Pemodelan CNN

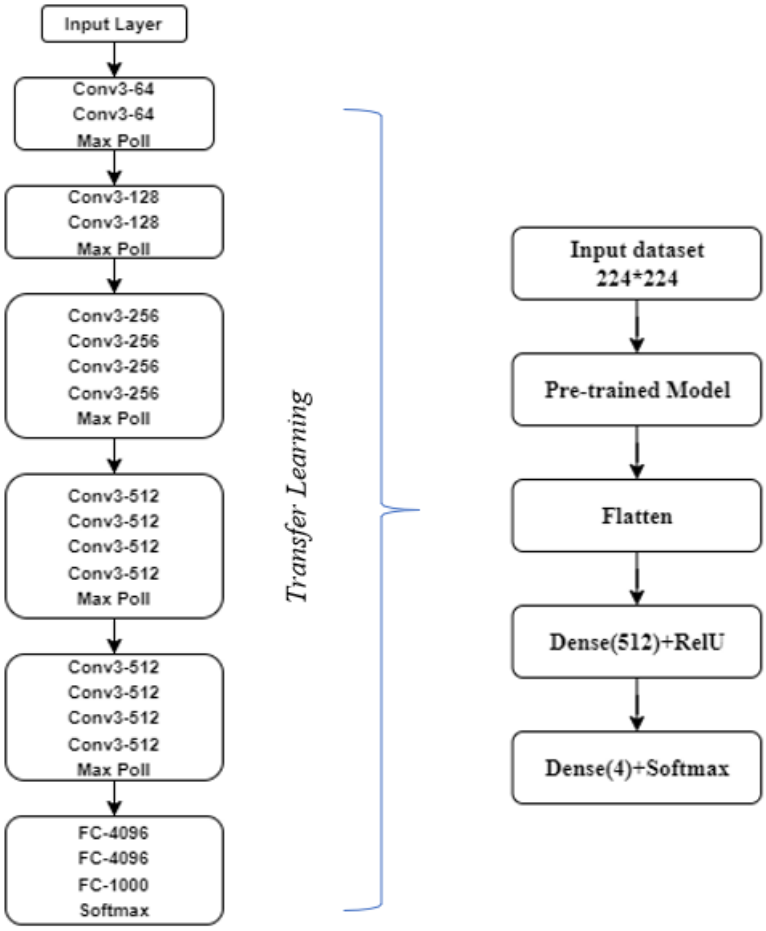
Perancangan model CNN pada penelitian ini digunakan arsitektur VGG16 dan VGG19 sebagai model *pre-trained*. Metode *transfer learning* diterapkan pada kedua arsitektur tersebut untuk melatih model dengan parameter atau data baru. Kedua arsitektur tersebut mengklasifikasikan gambar setelah melalui tahap *pre-processing* menjadi 3 kelas berpenyakit dan 1 kelas sehat, yaitu *Blight, Common Rust, Gray Leaf Spot,* dan *Healthy.* Setelah pembuatan *pre-trained model*, langkah selanjutnya adalah merancang model CNN untuk proses pelatihan.



Gambar 3. 3. Perancangan CNN VGG16 Sistem *Transfer Learning*

Berdasarkan Gambar 3.3 menjelaskan bahwa VGG16 merupakan salah satu metode CNN dengan total 16 lapisan, terdiri dari 13 lapisan konvolusional dan 3 lapisan *fully connected*, arsitektur ini memaparkan struktur konvolusional yang terdiri dari tiga set lapisan konvolusional. Setiap set dimulai dengan lapisan konvolusional menggunakan *filter* 3x3 dan *stride* 1, dengan jumlah *filter* berturut-turut sebanyak 64, 128, dan 256. Selanjutnya, masing-masing set konvolusional diikuti oleh lapisan *max pooling* dengan *filter* 2x2 dan *stride* 2. *Max pooling* digunakan untuk mereduksi dimensi *spasial* dan mengecilkan *resolusi* gambar. Proses berlanjut ke lapisan *fully connected* setelah lapisan konvolusional, dengan tiga lapisan yang masing-masing memiliki 4096 neuron. Ini diakhiri oleh lapisan *output* berisi 1000 neuron, sesuai dengan jumlah kelas dalam dataset *ImageNet* pada saat pengembangan model. Kemudian fungsi yang digunakan adalah aktivasi *ReLU* digunakan pada seluruh lapisan konvolusional untuk memperkenalkan elemen *non-linear* ke dalam model. Untuk mengatasi masalah *overfitting*, *dropout* diterapkan pada lapisan *fully connected* dengan tingkat *dropout* sebesar 0.5. Arsitektur ini memiliki lapisan *output* yang terdiri dari jumlah kelas yang diinginkan, dengan fungsi aktivasi yang sesuai dengan tugas yang dihadapi, seperti *softmax* untuk klasifikasi multi-kelas. Dengan jumlah parameter yang besar, terutama karena penggunaan banyak lapisan konvolusi dan *fully connected*, VGG16 memerlukan sumber daya komputasi yang signifikan. Keunggulan VGG16 terletak pada kesederhanaan strukturnya dan kontribusinya dalam memahami fitur gambar pada tugas klasifikasi.

Dalam perancangan model CNN ini, tipe arsitektur yang digunakan adalah *sequential*, dibuat dengan menggunakan *library* bawaan dari *TensorFlow* dan Keras. *Layer pre-trained* model dari proses *transfer learning* dimasukkan ke dalam model. Citra *input* memiliki ukuran 224x224 *piksel*. Dalam struktur model ini, lapisan terhubung sepenuhnya *(fully connected layer)* dari peta fitur yang dihasilkan oleh lapisan konvolusi dan lapisan *pooling* masih dalam bentuk *array multidimensional*, sehingga perlu dilakukan proses *flatten* untuk mengubahnya menjadi *input* bagi lapisan *fully connected*. Proses selanjutnya melibatkan *layer dense* (512) dengan aktivasi ReLU. *Output layer* menggunakan *dense* (4) dengan fungsi aktivasi *softmax*.



Gambar 3. 4. Perancangan CNN VGG19 Sistem *Transfer Learning*

Berdasarkan Gambar 3.4 menjelaskan bahwa model arsitektur VGG19 memiliki 19 layer dengan komposisi 16 *layer convolusional*, 5 *layer Max Pooling*, dan 3 *layer Fully Connected.* VGG19 melibatkan lima proses konvolusi, di mana setiap proses tersebut menggunakan jumlah filter yang berbeda. Sebagai contoh, *Conv*-1 memanfaatkan 64 *filter*, *Conv*-2 menggunakan 128 *filter*, *Conv*-3 menggunakan 256 *filter*, dan *Conv*-4 serta *Conv*-5 masing-masing menggunakan 512 *filter*. VGG19 digunakan sebagai model pra-pemrosesan dengan struktur yang menggabungkan lapisan konvolusional dan lapisan *non-linear* secara bergantian. Pendekatan ini terbukti lebih efektif daripada menggunakan hanya satu lapisan konvolusional. Struktur ini memungkinkan ekstraksi fitur gambar yang lebih baik dengan memanfaatkan teknik *Maxpooling* untuk *downsampling* dan mengadopsi unit *linier* modifikasi (*ReLU*) sebagai fungsi aktivasi.

Dalam perancangan model CNN ini, tipe arsitektur yang digunakan adalah *sequential,* dibuat dengan menggunakan *library* bawaan dari *TensorFlow* dan Keras. *Layer pre-trained* model dari proses *transfer learning* dimasukkan ke dalam model. Citra *input* memiliki ukuran 224x224 *piksel.* Dalam struktur model ini, lapisan terhubung sepenuhnya *(fully connected layer)* dari peta fitur yang dihasilkan oleh lapisan konvolusi dan lapisan *pooling* masih dalam bentuk *array multidimensional*, sehingga perlu dilakukan proses *flatten* untuk mengubahnya menjadi *input* bagi lapisan *fully connected.* Proses selanjutnya melibatkan *layer dense* (512) dengan aktivasi ReLU. *Output layer* menggunakan *dense* (4) dengan fungsi aktivasi *softmax.*

4. Tahap Pelatihan dan pengujian

   Pada tahap ini, terjadi pembagian data menjadi dua bagian, yakni data pelatihan dan data pengujian. Kedua set data tersebut akan digunakan oleh algoritma klasifikasi yang akan dibuat untuk membentuk model klasifikasi yang diinginkan.

5. Klasifikasi

   Langkah terakhir melibatkan proses klasifikasi data citra menjadi 3 kelas penyakit daun jagung dan 1 kelas sehat yaitu *Blight, Common Rust, Gray Leaf Spot,* dan *Healthy.*

3.2 IMPLEMENTASI CNN

Bahasa pemrograman yang digunakan adalah python dan implementasi dilakukan melalui *Google Colab* dengan menggunakan bahasa pemograman *python.* Berikut langkah-langkah implementasi yang dijalankan:

a. Tahap awal melibatkan koneksi antara *Google Colab* sebagai platform yang digunakan untuk melakukan pembuatan sistem ini dan dataset yang disimpan di *Google Drive*. *Google Drive* ini digunakan sebagai lokasi penyimpanan dataset. Adapun untuk potongan *code connect* ke *google drive* yang terdapat pada Gambar 3.5.

```
#Connect google drive

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

```
Mounted at /content/drive
```

Gambar 3. 5. Potongan *Code Connect* ke Google Drive

b. Tahap dua yaitu memasukan *library* agar pemrograman dapat berjalan. Adapun potongan *code import library* terdapat pada Gambar 3.6.

```
[2] import matplotlib.pyplot as plt
    import numpy as np
    from numpy import expand_dims
    import pandas as pd
    import seaborn as sns
    import cv2
    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tqdm import tqdm
    import os
    from sklearn.utils import shuffle
    from sklearn.model_selection import train_test_split
    from tensorflow.keras.applications import VGG16
    from tensorflow.keras.applications.vgg16 import preprocess_input
    from tensorflow.keras.preprocessing.image import load_img
    from tensorflow.keras.preprocessing.image import img_to_array
    from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, TensorBoard, ModelCheckpoint
    from sklearn.metrics import classification_report,confusion_matrix
    import ipywidgets as widgets
    import io
    from PIL import Image
    from IPython.display import display,clear_output
    from warnings import filterwarnings
```

Gambar 3. 6. Potongan *Code Import Library*

c. Tahap tiga yaitu membuat pelabelan kelas *dataset* citra yang akan dilakukan proses *training* dan *test*. Tahap pemrograman ini bertujuan untuk membaca file yang berapa di dalam folder *dataset* yang ada di *google drive*. Adapun potongan *code* pelabelan dataset terdapat pada Gambar 3.7.

```
[4] labels = ['Healthy', 'Blight', 'Common Rust', 'Gray Leaf Spot']
```

```
[5] X = []
    y = []

    image_size = 224

    for i in labels:
        folderPath = os.path.join('/content/drive/MyDrive/dataset corn leaf disease','4 kelas dataset',i)
        for j in tqdm(os.listdir(folderPath)):
            img = cv2.imread(os.path.join(folderPath,j))
            img = cv2.resize(img,(image_size,image_size))
            X.append(img)
            y.append(i)

    X = np.array(X)
    y = np.array(y)
```

```
100%|██████████| 1172/1172 [00:29<00:00, 40.36it/s]
100%|██████████| 1146/1146 [00:38<00:00, 30.09it/s]
100%|██████████| 1306/1306 [00:31<00:00, 41.92it/s]
100%|██████████| 574/574 [00:10<00:00, 52.56it/s]
```

Gambar 3. 7. Potongan *Code* Pelabelan Dataset

d. Tahap keempat, terlibat dalam proses pra-pemrosesan di mana dataset diacak dan dipisahkan menjadi data latih dan data uji. Pembagian dataset ini menggunakan fungsi *train_test_split.* Dalam tahap ini, terdapat empat variabel, yakni *X_train* yang berisi data pelatihan, *y_train* yang berisi label pelatihan, *X_test* yang berisi data uji, dan *y_test* yang berisi label pelatihan. Data pada variabel *X_train* dan *X_test* diubah tipe menjadi *float32*, dan nilai-nilai sebelumnya yang memiliki rentang antara 0 hingga 255 diubah menjadi rentang antara 0 hingga 1. Adapun potongan *code* pembagian dataset terdapat pada Gambar 3.8.

```
[9] import tensorflow as tf
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report
    from sklearn.metrics import confusion_matrix

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    X_train = X_train.astype('float32') #set x_train data type as float32
    X_test = X_test.astype('float32') #set x_test data type as float32
    X_train /= 255 #change x_train value between 0 - 1
    X_test /= 255 #change x_test value between 0 - 1
```

Gambar 3. 8. Potongan *Code* Pembagian Dataset

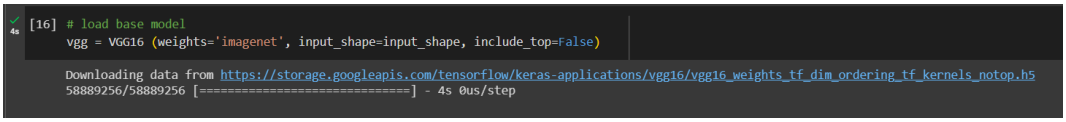
e. Tahap lima melibatkan *y_train* dan *y_test* dalam bentuk *categorical* yang kemudian disimpan dalam variabel baru bernama *y_train_new* dan *y_test_new*. Adapun potongan *code categorical* terdapat pada Gambar 3.9.

```
y_train_new = []
for i in y_train:
    y_train_new.append(labels.index(i))
y_train = y_train_new
y_train = tf.keras.utils.to_categorical(y_train)


y_test_new = []
for i in y_test:
    y_test_new.append(labels.index(i))
y_test = y_test_new
y_test = tf.keras.utils.to_categorical(y_test)
```

Gambar 3. 9. Potongan *Code Categorical*

f. Tahap keenam melibatkan pembuatan model CNN dengan menggunakan arsitektur VGG16 dan VGG19 menggunakan metode *transfer learning.* Pada potongan kode di bawah ini, hanya diperlihatkan penggunaan arsitektur VGG16. Adapun Gambar 3.10. mengenai *code* menggunakan *transfer learning* dan Gambar 3.11 mengenai code pemodelan.



Gambar 3. 10. Potongan *Code* Menggunakan *Transfer Learning*

```
#PEMODELAN
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, GlobalAveragePooling2D
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False

vgg16 = Sequential()
vgg16.add(vgg)
vgg16.add(tf.keras.layers.Flatten())
#Fully-connected
vgg16.add(tf.keras.layers.Dense(512, activation = 'relu'))
#output mengguunakan softmax
vgg16.add(tf.keras.layers.Dense(4, activation='softmax'))
```

Gambar 3. 11. Potongan *Code* Pemodelan

g. Tahap ketujuh melibatkan penentuan *hyperparameter* yang telah disiapkan sebelumnya. Dalam tahap ini, digunakan parameter *optimizer* SGD karena sesuai dengan pemodelan VGG16 menggunakan sistem *transfer learning*. *Learning rate* diatur sebesar 0,001 dengan *decay rate* 0,9, dan *decay step* 900. Adapun *code hyperparameter* terdapat pada Gambar 3.12.

```
[19] # Learning Rate
initial_learning_rate = 1e-3 # 1 x 10^-3

# Learning Rate Scheduler
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=900,
    decay_rate=0.9,
    staircase=True)


METRICS = ['accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]


vgg16.compile(optimizer=tf.keras.optimizers.SGD(lr_schedule),  # Very slow learning rate
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=METRICS)

#callback = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=4)
```

Gambar 3. 12. Potongan *Code Hyperparameter*

h. Tahap kedelapan mencakup pembuatan skrip untuk melatih data yang telah disiapkan. Dalam tahap ini, dilakukan pelatihan dengan menggunakan 30 *epoch* dan *batch size* 32. Adapun *code* proses *training* terdapat pada Gambar 3.13

```
vgg16_history= vgg16.fit(X_train, y_train,
          validation_data=(X_test, y_test),
          epochs=30,
          batch_size=32)
scores = vgg16.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 1/30
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5575: UserWarning: "`categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a
  output, from_logits = _get_logits(
105/105 [==============================] - 35s 253ms/step - loss: 0.8784 - accuracy: 0.6709 - precision: 0.8654 - recall: 0.4595 - val_loss: 0.6002 - val_accuracy: 0.8036 - val_prec
Epoch 2/30
105/105 [==============================] - 19s 182ms/step - loss: 0.5761 - accuracy: 0.8121 - precision: 0.8965 - recall: 0.6939 - val_loss: 0.4719 - val_accuracy: 0.8429 - val_prec
Epoch 3/30
105/105 [==============================] - 18s 174ms/step - loss: 0.4642 - accuracy: 0.8505 - precision: 0.9156 - recall: 0.7752 - val_loss: 0.4304 - val_accuracy: 0.8536 - val_prec
Epoch 4/30
105/105 [==============================] - 20s 191ms/step - loss: 0.4143 - accuracy: 0.8672 - precision: 0.9164 - recall: 0.8061 - val_loss: 0.3754 - val_accuracy: 0.8857 - val_prec
Epoch 5/30
105/105 [==============================] - 19s 186ms/step - loss: 0.3876 - accuracy: 0.8740 - precision: 0.9155 - recall: 0.8225 - val_loss: 0.3689 - val_accuracy: 0.8738 - val_prec
Epoch 6/30
105/105 [==============================] - 18s 168ms/step - loss: 0.3529 - accuracy: 0.8839 - precision: 0.9234 - recall: 0.8431 - val_loss: 0.3293 - val_accuracy: 0.9000 - val_prec
Epoch 7/30
105/105 [==============================] - 19s 186ms/step - loss: 0.3277 - accuracy: 0.8958 - precision: 0.9315 - recall: 0.8624 - val_loss: 0.3196 - val_accuracy: 0.8976 - val_prec
Epoch 8/30
105/105 [==============================] - 20s 188ms/step - loss: 0.3206 - accuracy: 0.8931 - precision: 0.9241 - recall: 0.8630 - val_loss: 0.3012 - val_accuracy: 0.9060 - val_prec
```

Gambar 3. 13. Potongan *Code Proses Training*

i. Tahap terakhir yaitu melakukan evaluasi prediksi. Adapun *code* evaluasi pengujian terdapat pada Gambar 3.14.

```
[23] #mencocokan data hasil training dengan data test
     pred = vgg16.predict(X_test)
     pred = np.argmax(pred,axis=1)
     y_test_new = np.argmax(y_test,axis=1)

     27/27 [==============================] - 4s 130ms/step
```

Gambar 3. 14. Potongan *Code* Evaluasi

j. Potongan *code* pemrograman untuk menampilkan kurva. Adapun *code* untuk menampilkan kurva terdapat pada Gambar 3.15.

```
filterwarnings('ignore')

epochs = [i for i in range (30)]
fig, ax = plt.subplots(1,2,figsize=(14,7))
train_acc = vgg16_history.history['accuracy']
train_loss = vgg16_history.history['loss']
val_acc = vgg16_history.history['val_accuracy']
val_loss = vgg16_history.history['val_loss']

#menampilkan judul
fig.text(s='Epochs vs. Training and Validation Accuracy/Loss',size=18,fontweight='bold',
         fontname='monospace',color=colors_dark[1],y=1,x=0.28,alpha=0.8)
#membuat kurva
sns.despine()
ax[0].plot(epochs, train_acc, marker='o',markerfacecolor=colors_green[2],color=colors_green[3],
           label = 'Training Accuracy')
ax[0].plot(epochs, val_acc, marker='o',markerfacecolor=colors_red[2],color=colors_red[3],
           label = 'Validation Accuracy')
ax[0].legend(frameon=False)
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')

sns.despine()
ax[1].plot(epochs, train_loss, marker='o',markerfacecolor=colors_green[2],color=colors_green[3],
           label ='Training Loss')
ax[1].plot(epochs, val_loss, marker='o',markerfacecolor=colors_red[2],color=colors_red[3],
           label = 'Validation Loss')
ax[1].legend(frameon=False)
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Training & Validation Loss')

fig.show()
```

Gambar 3. 15. Potongan *Code* Menampilkan Kurva

k. Potongan *code* pemrograman untuk menampilkan *confusion matrix*. Adapun *code* untuk menampilkan *confusion matrix* terdapat pada Gambar 3.16 dan *code* untuk *heatmap confusion matrix* terdapat Gambar 3.17.

```
#menampilkan skor akurasi dan confusion matriks
from sklearn.metrics import accuracy_score
print("Accuracy")
print(accuracy_score(y_test_new,pred))
print("")
print("Confusion matrix")
confusion_matrix(y_test_new,pred)
```

Gambar 3. 16. Potongan *Code* Menampilkan *Confusion Matrix*

```
[26] fig, ax = plt.subplots(figsize=(15,10))
     sns.heatmap(confusion_matrix(y_test_new,pred), linewidths=1, xticklabels=labels, yticklabels=labels, annot=True, ax=ax, fmt='g')
     fig.text (s='Heatmap of the Confusion Matrix',size=18,fontweight='bold',
               fontname='monospace', y=0.92,x=0.28,alpha=0.8)

     plt.show()
```

Gambar 3. 17. Potongan *Code Heatmap Confusion Matrix*

l. Potongan *code* pemrograman untuk menampilkan hasil prediksi. Adapun *code* menampilkan hasil prediksi terdapat pada Gambar 3.18.

```
[27] y_hat = vgg16.predict(X_test)
     # Plot a random sample of 54 test images, their predicted labels and ground truth
     figure = plt.figure(figsize= (25, 40))
     for i, index in enumerate(np.random.choice(X_test.shape[0], size=50, replace=False)):
         ax = figure.add_subplot(20, 4, i + 1, xticks=[], yticks=[]) #ukuran 10 X 7 untuk visualisasi gambar pada heatmap confusion matrix
         # Display each image
         ax.imshow(np.squeeze(X_test[index]))
         predict_index = np.argmax(y_hat[index])
         true_index = np.argmax(y_test[index])
         # Set the title for each image
         ax.set_title("{} ({})".format(labels[predict_index],
                                       labels[true_index]),
                                       color=("green" if predict_index == true_index else "red"))
     plt.show()
```

Gambar 3. 18. Potongan *Code* Menampilkan Hasil Prediksi