BAB 3
METODOLOGI PENELITIAN

Dalam melakukan penelitian ini, terdapat beberapa tahap yang dilakukan seperti mencari teori-teori yang dibutuhkan hingga mengolah data yang telah didapatkan. Berikut adalah penjelasan dari tahap-tahap pengerjaan penelitian ini.

3.1 Studi literatur

Pada tahap ini adalah proses pengumpulan, dan penelaahan informasi atau teori yang akan mendukung penelitian ini merupakan hal yang pertama yang dilakukan. Studi literatur membantu untuk memahami penelitian sebelumnya yang telah dilakukan dalam bidang yang sama. Teori yang didapatkan dari telaah literatur digunakan untuk memahami lebih lanjut terhadap penelitian yang akan dilakukan dengan cara mencari artikel, jurnal, dan buku yang tersedia pada situs yang tepercaya.

3.2 Pengolahan data

Tahap pengolahan data adalah proses awal sebelum menggunakan himpunan data dalam model yang dapat memengaruhi hasil kinerja model. Dalam pengolahan data, himpunan data akan dianalisis terlebih dahulu, kemudian menyamakan tipe data, menghapus data yang tidak perlu, identifikasi dan menangani nilai yang hilang atau nilai *null*. Setelah itu, himpunan data akan dibagi menjadi tiga bagian yaitu data *training*, data *validation*, dan data *testing*.

Terdapat dua metode yang akan digunakan yaitu *preprocessing* gambar dan augmentasi data. Teknik *preprocessing* gambar akan menggunakan *unsharp mask filtering*, teknik tersebut meningkatkan kontras pada tepi objek dalam foto yang mengidentifikasi kan nilai piksel yang berbeda dari piksel tetangganya dengan jumlah tertentu. Kemudian akan memilih fitur dari gambar, fitur yang akan dipilih adalah fitur bentuk dari gambar terutama bentuk dari jenis bakteri.

Dalam augmentasi data gambar akan menghasilkan himpunan data gambar yang telah dirotasi, per besar atau per kecil, pergeseran horizontal atau vertikal secara acak. Tujuan dari augmentasi data adalah mengatasi atau mencegah model *overfitting*.

Pada Kode 3.1 adalah potongan kode mengimpor *library*, menampilkan versi daftar *library* yang digunakan dalam penelitian ini dan daftar perangkat yang digunakan beserta detail perangkat yang terdeteksi. Hasil *output* dapat dilihat pada Gambar 3.1.

```
import os
import os.path
import glob
import pathlib
import numpy as np
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt
from scipy.ndimage import median_filter
from skimage import io
from PIL import Image
from tensorflow import keras
from tensorflow.keras import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import
    ImageDataGenerator, array_to_img, img_to_array, load_img
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from platform import python_version
from tensorflow.python.client import device_lib
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

print("Python Version", python_version())
print("numpy Version",np.__version__)
print("opencv python Version",cv2.__version__)
print("Tensorflow Version",tf.__version__)
print(device_lib.list_local_devices())
physical_devices_gpu = tf.config.list_physical_devices("GPU")
if physical_devices_gpu:
    try:
        logical_gpu = tf.config.experimental.list_logical_devices(
    'GPU')
        print(len(physical_devices_gpu), "Physical GPUs,", len(
    logical_gpu), "Logical GPUs")
    except RuntimeError as e:
        print(e)
```

Kode 3.1: Potongan kode mengimpor *library* menampilkan versi dan menampilkan daftar perangkat.

```
Python Version 3.9.16
numpy Version 1.23.5
opencv python Version 4.7.0
Tensorflow Version 2.6.0

[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 18145677694496396515
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 5711593472
locality {
  bus_id: 1
  links {
  }
}
incarnation: 1849130182407497849
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 3070 Ti, pci bus id: 0000:01:00.0, compute capability: 8.6"
]
1 Physical GPUs, 1 Logical GPUs
```

Gambar 3.1. Hasil *output* dari potongan kode 3.1.

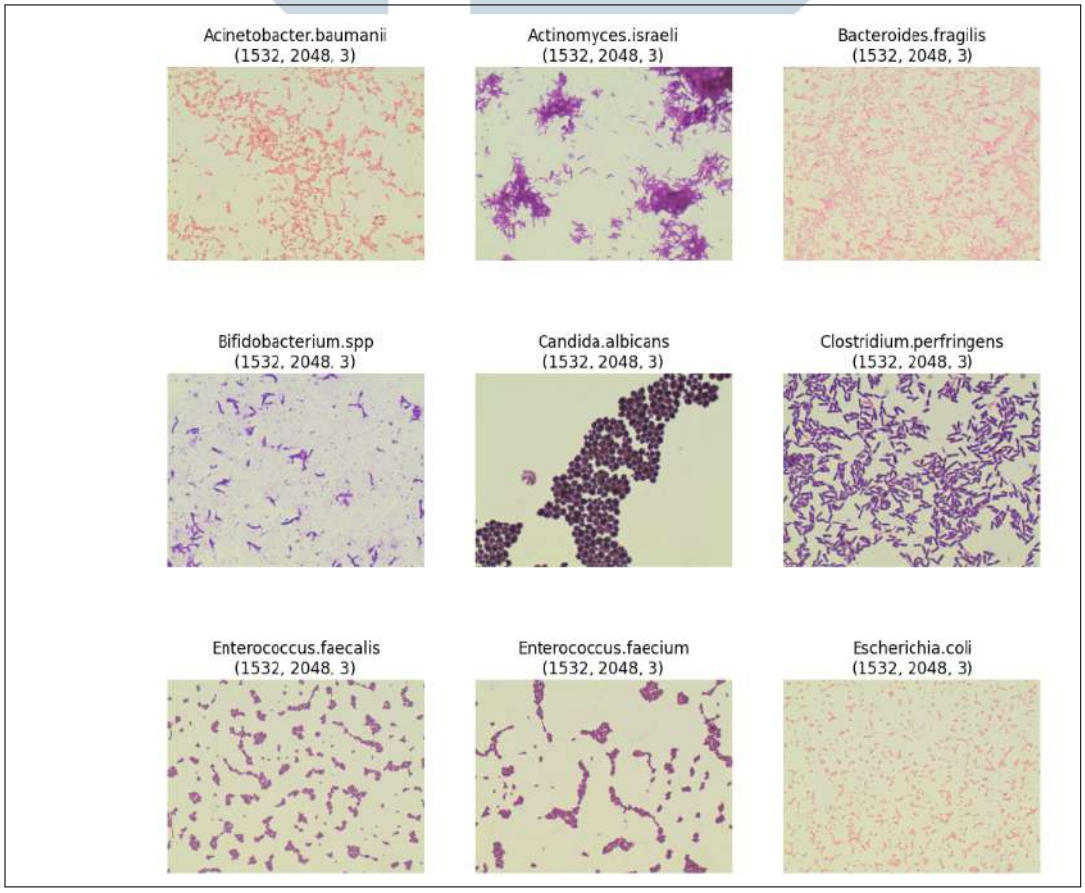
Pada Kode 3.2 adalah potongan kode yang mengatur *path directory* proyek dan menampilkan detail himpunan data DIBaS untuk dianalisis terlebih dahulu. Pada Gambar 3.2, diketahui himpunan data tersebut terdapat 692 total gambar dan memiliki jumlah 33 jenis bakteri (*classes*). Setiap gambar memiliki bentuk data dengan tinggi 1532 pixels, lebar 2048 pixels, dan tiga saluran (1532, 2048, 3). Selanjutnya adalah menampilkan sembilan gambar jenis bakteri yang berbeda yang dapat dilihat pada Gambar 3.3.

```
#set the base path of the project and datasets
root = r'D:\university\Projects\Python\Bacteria_Classification'
dataset_path = root + '/Bacterial_Species/'
raw_data_path = dataset_path + 'Raw'
raw_data_dir = pathlib.Path(raw_data_path)

image_count_raw = len(list(raw_data_dir.glob('*/*.tif')))
print('Total Raw Images: '+str(image_count_raw))

class_list_name = os.listdir(raw_data_path)
print('Class Count: '+str(len(class_list_name)))
print('Class list: ')
for i in range(len(class_list_name)):
    name = class_list_name[i]
    print(str(i+1)+'. '+name)
```

Kode 3.2: Potongan kode untuk mengatur *path directory* proyek dan himpunan data.

```
Total Raw Images: 692
Class Count: 33
Class list:
1. Acinetobacter.baumanii
2. Actinomyces.israeli
3. Bacteroides.fragilis
4. Bifidobacterium.spp
5. Candida.albicans
6. Clostridium.perfringens
7. Enterococcus.faecalis
8. Enterococcus.faecium
9. Escherichia.coli
10. Fusobacterium
11. Lactobacillus.casei
12. Lactobacillus.crispatus
13. Lactobacillus.delbrueckii
14. Lactobacillus.gasseri
15. Lactobacillus.jehnsenii
16. Lactobacillus.johnsonii
17. Lactobacillus.paracasei
18. Lactobacillus.plantarum
19. Lactobacillus.reuteri
20. Lactobacillus.rhamnosus
21. Lactobacillus.salivarius
22. Listeria.monocytogenes
23. Micrococcus.spp
24. Neisseria.gonorrhoeae
25. Porfyromonas.gingivalis
26. Propionibacterium.acnes
27. Proteus
28. Pseudomonas.aeruginosa
29. Staphylococcus.aureus
30. Staphylococcus.epidermidis
31. Staphylococcus.saprophiticus
32. Streptococcus.agalactiae
33. Veionella
```

Gambar 3.2. Hasil *output* dari potongan kode 3.2.



Gambar 3.3. Sembilan gambar jenis bakteri dari *raw data*.

Pada Kode 3.3 dibuat fungsi bernama *unsharp_img* dan *processImages*. Fungsi dari *unsharp_img* adalah menerima gambar lalu dilakukan *Unsharp Filter* gambar tersebut. *Input parameter size* dalam *Unsharp Filter* digunakan untuk mengatur *Median Filter* dalam *Unsharp Filter*, kemudian *input parameter strength* digunakan untuk pertajam gambar.

Sementara itu, fungsi bernama *processImages* berfungsi untuk mengubah nama format label kemudian melakukan pengecekan setiap dari gambar apakah gambar tersebut *corrupted*. Jika gambar tersebut *corrupted* maka akan tidak menggunakan gambar *corrupted* tersebut. Jika tidak maka akan lanjut ke proses selanjutnya yaitu melakukan potongan pada gambar dari horizontal kiri gambar ke kanan kemudian vertikal kebawah dengan dimensi potongan gambar 256x256 dan memanggil fungsi bernama *unsharp_img*.

```
# unsharp filter function
def unsharp_img(image, size, strength):
    # Median Filter
    img_mf = median_filter(image, size)
    # calculate laplician
    laplacian = cv2.Laplacian(img_mf, cv2.CV_64F)
    # sharpened image
    img_sharp = image - strength * laplacian
    # Saturate the pixels
    img_sharp[img_sharp > 255] = 255
    img_sharp[img_sharp < 0] = 0

    return img_sharp

#this function is to get and crop image of raw image into image dimension of 256x256
def ProcessImages(img_path, name, idx):
    idx += 1
    name = name.replace(".","_")
    img = cv2.imread(os.path.join(img_path))

    #check if image is corrupted
    if img is None:
        print(str(img_path) + '(discarded) \n Image is corrupted or None')
        return

    save_original_path = dataset_path + 'Original/' + str(name) + '/'
```

```
    save_cropped_path = dataset_path + 'Cropped/'+ str(name) + '/'
    if not os.path.exists(save_original_path):
        os.makedirs(save_original_path)
    if not os.path.exists(save_cropped_path):
        os.makedirs(save_cropped_path)

    filename = name+str(idx)
    fileformat = '.jpg'

    #original image
    cv2.imwrite(save_original_path+filename+fileformat, img)
    #cropped to 256 x 256
    img_height, img_width = img.shape[:2]
    crop_height, crop_width = 256,256
    for rows in np.arange(img_height - crop_height + 1, step =
crop_height):
        for cols in np.arange(img_width - crop_width + 1, step =
crop_width):
            img_crop = img[rows:rows+crop_height, cols:cols+
crop_width, :]
            #unsharp image
            img_unsharp = np.zeros_like(img_crop)
            for i in range(3):
                img_unsharp[:,:,i] = unsharp_img(img_crop[:,:,i],
2, 5)

            cv2.imwrite(save_cropped_path+filename+str(rows)+str(
cols)+fileformat, img_unsharp)

    return
```

Kode 3.3: Potongan kode fungsi *unsharp_img* dan *ProcessImages*.

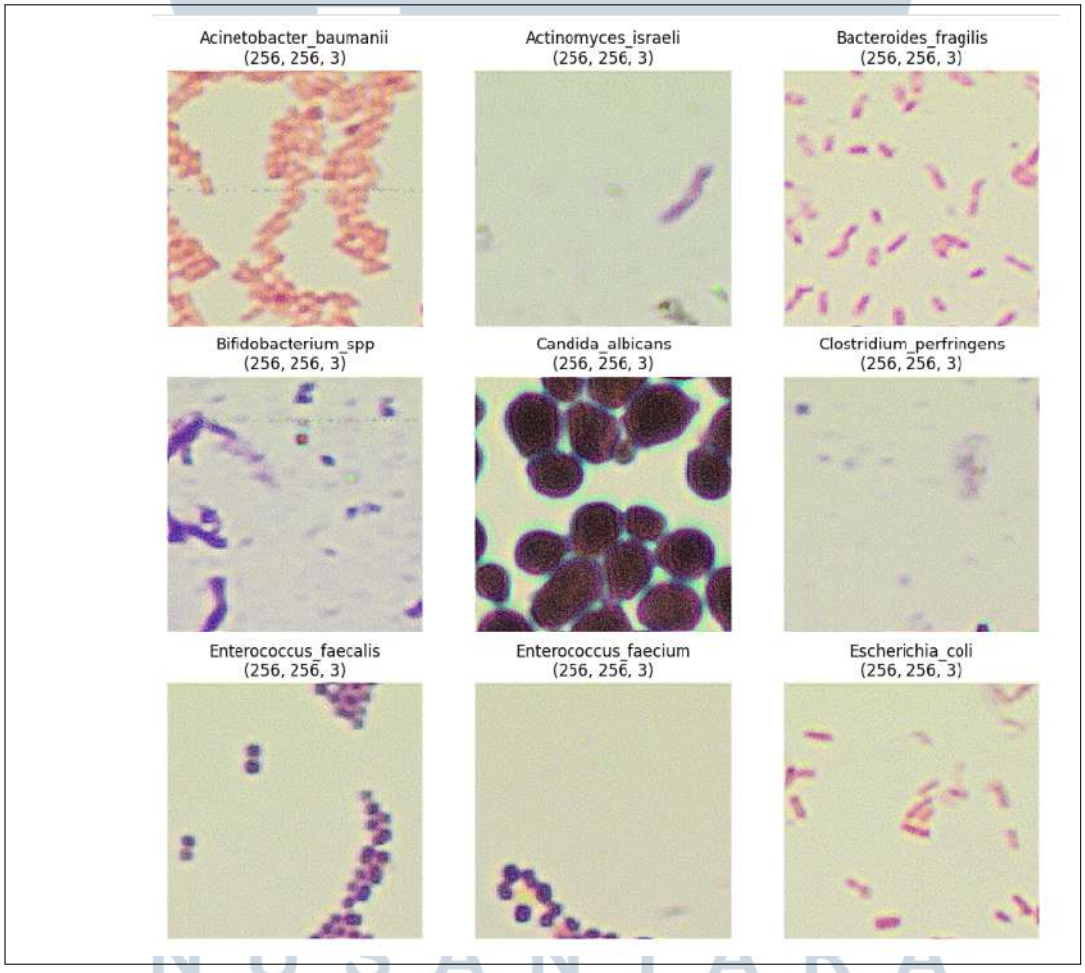
3.2.1 *Pre-processing* Gambar

Pada Kode 3.4 adalah proses selanjutnya untuk memanggil fungsi *processImages* untuk memproses *raw data* DIBaS dalam *looping* sebanyak dari total himpunan data yaitu sebanyak 692 kali. Dalam DIBaS terdapat tiga gambar yang tidak dapat digunakan dan total gambar yang telah dipotong sebanyak 27560. Hasil potongan gambar dapat dilihat pada Gambar 3.4.

```
for name in class_list_name:
    i = 0
```

```
    img_path = list(raw_data_dir.glob(name+'/*'))
    for path in img_path:
        ProcessImages(path, name, i)
        i += 1
crop_data_path = dataset_path + 'Cropped'
crop_data_dir = pathlib.Path(crop_data_path)
crop_data_dir_list = list(crop_data_dir.glob('*'))

crop_image_count = len(list(crop_data_dir.glob('*/*.jpg')))
print("Total Cropped images:", crop_image_count)
class_names = np.array(sorted([item.name for item in crop_data_dir
    .glob('*')]))
```

Kode 3.4: Potongan kode memanggil fungsi *processImages*.



Gambar 3.4. Sembilan gambar jenis bakteri yang telah dipotong.

Pada Kode 3.5 adalah potongan kode untuk membagi himpunan data gambar yang telah dipotong menjadi tiga bagian yaitu data *training*, data *validation*, dan

data *testing*. Setiap *class* akan menggunakan sebanyak 80% untuk data *training*, 10% untuk data *validation*, dan 10% untuk data *testing*.

```
for name in class_names:
    idx = 0
    img_crop = list(crop_data_dir.glob(name+'/*'))
    img_count = len(img_crop)
    train_count = img_count * 0.8
    validation_count = img_count * 0.1
    test_count = img_count * 0.1
    save_train_path = dataset_path + 'train/' + str(name) + '/'
    save_validation_path = dataset_path + 'validation/'+ str(name)
    + '/'
    save_test_path = dataset_path + 'test/'+ str(name) + '/'

    if not os.path.exists(save_train_path):
        os.makedirs(save_train_path)
    if not os.path.exists(save_validation_path):
        os.makedirs(save_validation_path)
    if not os.path.exists(save_test_path):
        os.makedirs(save_test_path)

    fileformat = '.jpg'

    for i in range(int(train_count)):
        img = cv2.imread(os.path.join(img_crop[idx]))
        cv2.imwrite(save_train_path+name+str(idx)+fileformat, img)
        idx += 1

    for i in range(int(validation_count)):
        img = cv2.imread(os.path.join(img_crop[idx]))
        cv2.imwrite(save_validation_path+name+str(idx)+fileformat,
    img)
        idx += 1

    for i in range(int(test_count)):
        img = cv2.imread(os.path.join(img_crop[idx]))
        cv2.imwrite(save_test_path+name+str(idx)+fileformat, img)
        idx += 1
```

Kode 3.5: Potongan kode untuk membagi himpunan data.

3.2.2 Augmentasi Data

Pada Kode 3.6 adalah proses untuk menambah jumlah himpunan data *training* dengan menggunakan metode augmentasi. pada kode berikut, augmentasi akan menghasilkan lebih dari 5000 gambar dari setiap *class*. Hasil gambar dari augmentasi adalah rotasi secara acak, pergeseran lebar, pergeseran tinggi, per besar atau per kecil, dan membalik secara horizontal dengan *fill_mode* konstan. Dimensi gambar augmentasi akan sama dengan dimensi gambar yang dipotong.

```
datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 35,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.25,
    zoom_range = 0.25,
    horizontal_flip=True,
    fill_mode='constant'
)

IMG_SIZE = 256

train_data_path = dataset_path + 'train'
train_data_dir = pathlib.Path(train_data_path)
train_data_dir_list = list(train_data_dir.glob('*'))

for name in class_names:
    img_path = list(train_data_dir.glob(name+'/*'))
    raw_train_dataset = []


    for img_name in img_path:
        img = io.imread(img_name)
        img = Image.fromarray(img)
        img = img.resize((IMG_SIZE,IMG_SIZE))
        raw_train_dataset.append(np.array(img))

    x = np.array(raw_train_dataset)

    Total_To_Generate = 6000 - len(x)

    save_train_aug_path = dataset_path + 'train_augmented/' + str(
    name) + '/'
```

```
34
35      if not os.path.exists(save_train_aug_path):
36          os.makedirs(save_train_aug_path)
37      i = 0
38
39      for batch in datagen.flow(x, batch_size=1,
40                                save_to_dir = save_train_aug_path,
41                                save_prefix = name,
42                                save_format = 'jpg'):
43          i += 1
44          if i > Total_To_Generate:
45              break
```

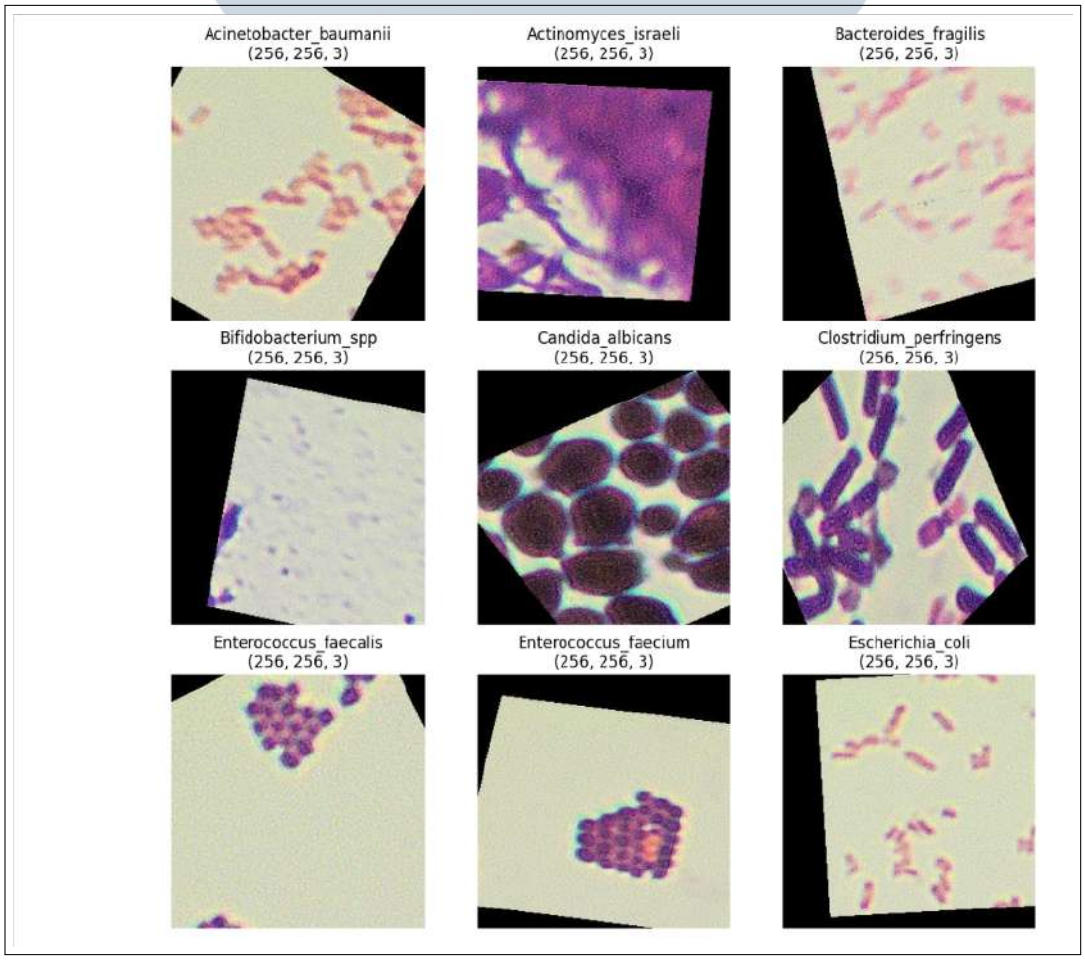
Kode 3.6: Potongan kode proses augmentasi.

Pada Kode 3.7 adalah potongan kode untuk menyatakan *path directory train*, *validation*, *test*, dan augmentasi. Kemudian menggabungkan data gambar *train* dan data gambar augmentasi menjadi satu. Total gambar yang telah dipotong untuk data *training* sebanyak 22.048 gambar, data *validation* sebanyak 2.756 gambar, data *testing* sebanyak 2.756 gambar, dan data yang telah di augmentasi sebanyak 175.916 gambar. Total keseluruhan adalah sebanyak 203.476 gambar. Hasil augmentasi gambar dapat dilihat pada Gambar 3.5.

```
1  train_aug_data_path = dataset_path + 'train_augmented'
2  train_aug_data_dir = pathlib.Path(train_aug_data_path)
3  train_aug_data_dir_list = list(train_aug_data_dir.glob('*'))
4
5  train_data_path = dataset_path + 'train'
6  train_data_dir = pathlib.Path(train_data_path)
7  train_data_dir_list = list(train_data_dir.glob('*'))
8
9  train_crop_dataset = None
10 train_crop_dataset = tf.data.Dataset.list_files(str(train_data_dir
       /'*/*'))
11
12 train_aug_dataset = None
13 train_aug_dataset = tf.data.Dataset.list_files(str(
       train_aug_data_dir/'*/*'))
14
15 train_dataset = None
16 train_dataset = train_crop_dataset.concatenate(train_aug_dataset)
17
18 validation_data_path = dataset_path + 'validation'
19 validation_data_dir = pathlib.Path(validation_data_path)
```

```
validation_data_dir_list = list(validation_data_dir.glob('*'))

validation_dataset = None
validation_dataset = tf.data.Dataset.list_files(str(
    validation_data_dir/'*/*'), shuffle = False)

train_image_count = len(list(train_data_dir.glob('*/*.jpg')))
aug_image_count = len(list(train_aug_data_dir.glob('*/*.jpg')))
validation_image_count = len(list(validation_data_dir.glob('*/*.
    jpg')))

test_data_path = dataset_path + 'test'
test_data_dir = pathlib.Path(test_data_path)
test_data_dir_list = list(test_data_dir.glob('*'))
```

Kode 3.7: Potongan kode untuk menyatakan *path directory train*, *validation*, *test*, dan augmentasi.



Gambar 3.5. Sembilan gambar jenis bakteri yang telah diaugmentasi.

Himpunan data masih dalam berbentuk daftar *path directory*, maka dari itu dibuat fungsi *process_path* untuk mengubah daftar tersebut menjadi *array* gambar yang dinormalisasi dan *label* pada gambar yang sesuai. Potongan kode fungsi *process_path* dapat dilihat pada Kode 3.8. Selanjutnya adalah membuat fungsi bernama *extractFeature* untuk ekstraksi fitur bentuk dari gambar terutama bentuk dari jenis bakteri.

```
#convert file path to label
def get_label(file_path):
    #convert the path to a list of path components
    parts = tf.strings.split(file_path, os.path.sep)
    # the second to last is the class-directory
    one_hot = parts[-2] == class_names
    #Integer encode the label
    return tf.argmax(one_hot)

#decode image
def decode_img(img):
    #convert the compressed string to a 3D uint8 tensor
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.cast(img, tf.float32)
    img = tf.image.resize(img, [img_height, img_width])
    img = (img / 255)
    return img

def process_path(file_path):
    label = get_label(file_path)
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label
```

Kode 3.8: Potongan kode fungsi *get_label*, *decode_img*, *process_path*.

Fungsi bernama *extractFeature* menggunakan Opencv Python *library*. Proses dalam *extractFeature* dimulai dari gambar diubah menjadi hitam putih dan diberikan *filter* Gaussian, lalu *noise* pada gambar dapat dihilangkan menggunakan *fastN1MeansDenoising*. Proses selanjutnya adalah membuat *mask* dari hasil titik *contour* yang terdeteksi dari gambar. *Mask* tersebut digunakan untuk menghilangkan *background* pada gambar dalam *bitwise_and*. Hasil gambar yang dikembalikan adalah gambar penuh warna dengan *background* yang dihapus dan dimensi (128, 128, 3). Potongan kode fungsi *extractFeature* dapat dilihat pada Kode 3.9.

```
img_height = img_width = 128

def extractFeature(img, isTest=False):
    if isTest:
        mat_img = img
    else:
        mat_img = img.numpy() * 255

    mat_img = np.uint8(mat_img)
    mat_gray = cv2.cvtColor(mat_img, cv2.COLOR_BGR2GRAY)
    mat_blur = cv2.GaussianBlur(mat_gray, (3,3), 0)
    mat_denoise = cv2.fastNlMeansDenoising(mat_blur)
    thresh = cv2.adaptiveThreshold(mat_denoise, 255, cv2.
    ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
    thresh_invert = cv2.bitwise_not(thresh)

    contours, hierarchy = cv2.findContours(thresh_invert, cv2.
    RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    mask = np.zeros([128,128],np.uint8)

    for contour in contours:
        cv2.drawContours(mask, [contour], 0, (255, 255, 255), -1)

    removed_bg = cv2.bitwise_and(mat_img,mat_img, mask=mask)
    result = np.float32(removed_bg)
    result = result / 255

    feature_img = tf.cast(result, tf.float32)

    return feature_img

# tf py function to call extract feature function
def tf_extractFeature(img, label):
    [img,] = tf.py_function(extractFeature, [img], [tf.float32])
    img.set_shape((img_height, img_width, 3))

    return img, label
```

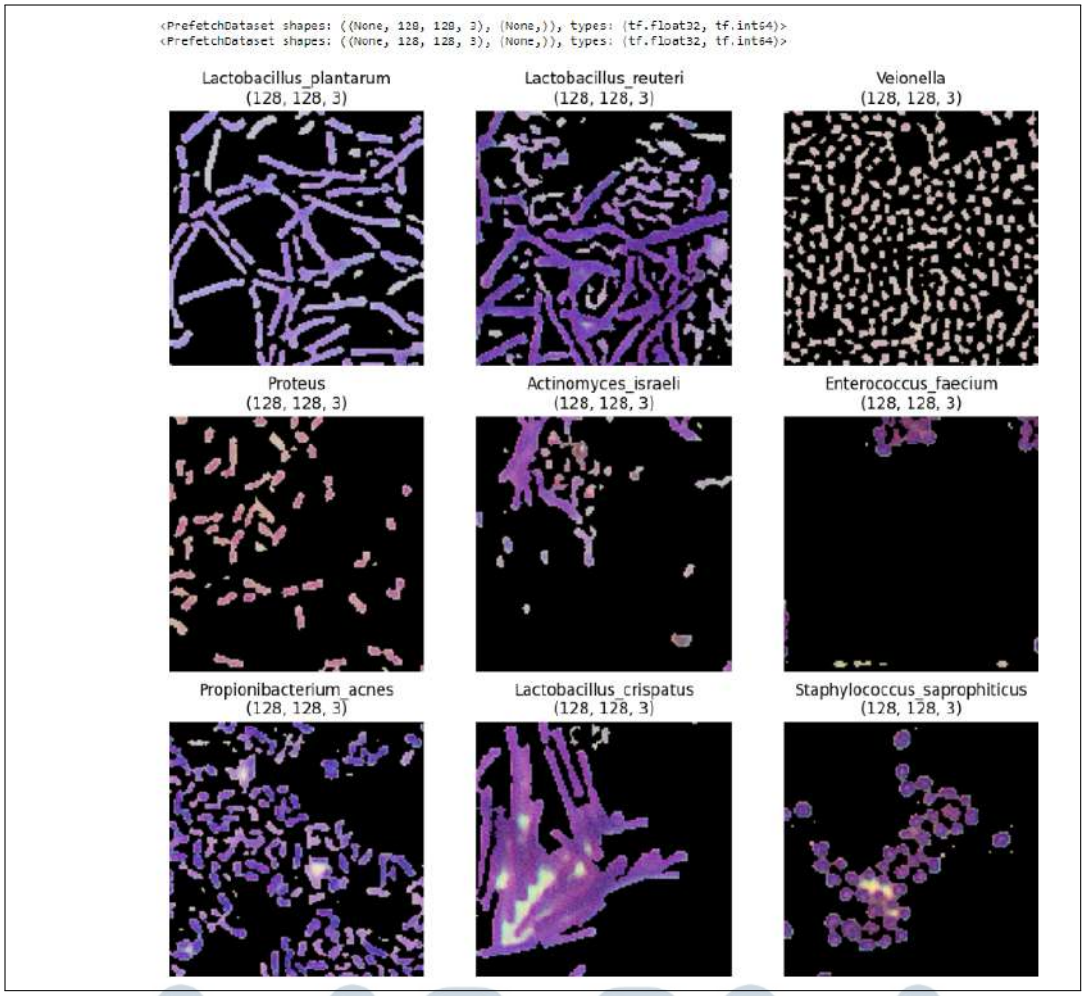
Kode 3.9: Potongan kode fungsi *extractFeature*.

Pada kode 3.10 adalah konfigurasi himpunan data dengan menggunakan fungsi *process_path* dan fungsi *tr_extractFeature*. kemudian himpunan data *train* dilakukan pengacakan sementara itu, data *validation* tidak. Ukuran *batch* yang di konfigurasi dalam data *train* dan data *validation* adalah 128. Hasil gambar ekstraksi

fitur dapat dilihat pada Gambar 3.5.

```
#the batch is set 128
AUTOTUNE = tf.data.AUTOTUNE
batch_size = 128

train_dataset = train_dataset.map(process_path, num_parallel_calls
    =AUTOTUNE)
train_dataset = train_dataset.map(tf_extractFeature,
    num_parallel_calls=AUTOTUNE)
train_dataset = train_dataset.shuffle(100)
train_dataset = train_dataset.batch(batch_size)
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)

validation_dataset = validation_dataset.map(process_path,
    num_parallel_calls=AUTOTUNE)
validation_dataset = validation_dataset.map(tf_extractFeature,
    num_parallel_calls=AUTOTUNE)
validation_dataset = validation_dataset.batch(batch_size)
validation_dataset = validation_dataset.prefetch(buffer_size=
    AUTOTUNE)

print(train_dataset)
print(validation_dataset)

img, label = next(iter(train_dataset))
plt.figure(figsize=(12, 12))
for i in range(9):
    class_name = class_names[label[i]]
    plt.subplot(3, 3, i + 1)
    plt.imshow(img[i])
    plt.title(class_name+'\n'+str(img[i].shape))
    plt.axis("off")
```

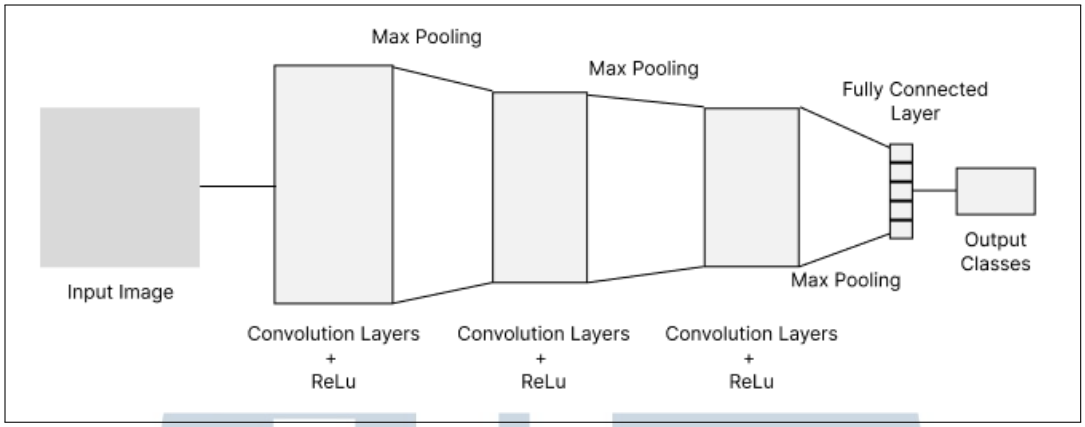
Kode 3.10: Potongan kode konfigurasi data *train* dan data *validation*.



Gambar 3.6. Sembilan gambar ekstraksi fitur.

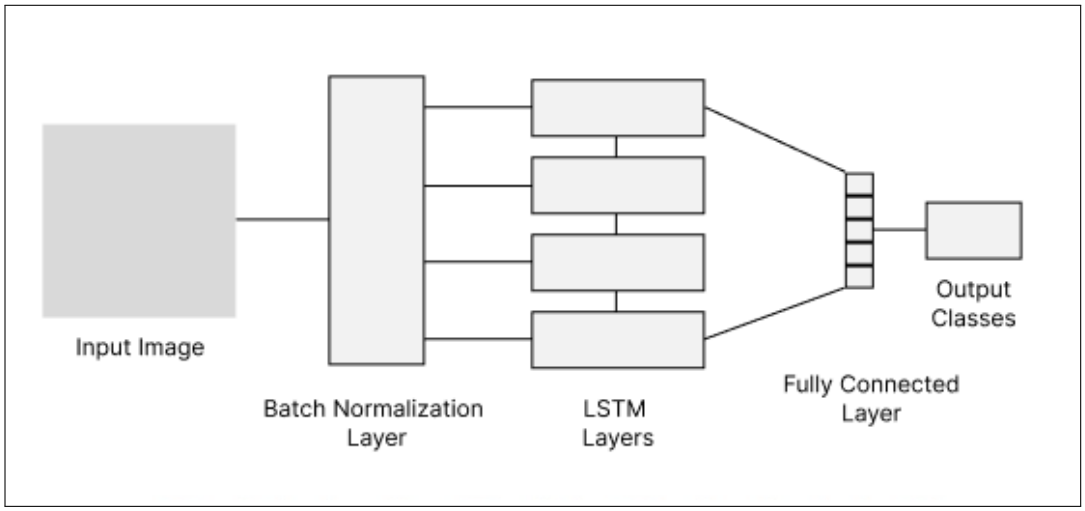
3.2.3 arsitektur model

Pada Gambar 3.7 adalah model arsitektur CNN yang akan digunakan dalam penelitian ini. Model CNN menerima *Input shape* (None, 128, 128, 3). Dalam model CNN terdapat tiga CNN layers yang telah diberikan fungsi aktivasi ReLu dan *MaxPooling* dengan ukuran (2,2) dan jumlah *units* berbeda, pada *layer* pertama dimulai dari 32 *units*, *layer* kedua terdapat 64 *units*, dan *layer* ketiga terdapat 128 *units*. Terakhir diberikan *Fully-Connected Layer* untuk menghubungkan semua neuron. Hasil potongan kode pembuatan model CNN dapat dilihat pada Kode 5.1.



Gambar 3.7. Arsitektur model CNN.

Pada Gambar 3.8 adalah model arsitektur LSTM yang akan digunakan dalam penelitian ini. Model LSTM menerima *Input shape* (None, 128, 128, 3). Dalam model LSTM diawali dari *Batch Normalization Layer*, lalu mengubah bentuk dimensi *input* yang dapat digunakan dalam LSTM *layer*. LSTM diberikan 32 jumlah *unit*, beserta fungsi aktivasi tanh, dan nilai *dropout* 0.2. Selanjutnya, diberikan *Fully-Connected Layer* untuk menghubungkan semua neuron. Hasil potongan kode pembuatan model CNN dapat dilihat pada Kode 5.2.

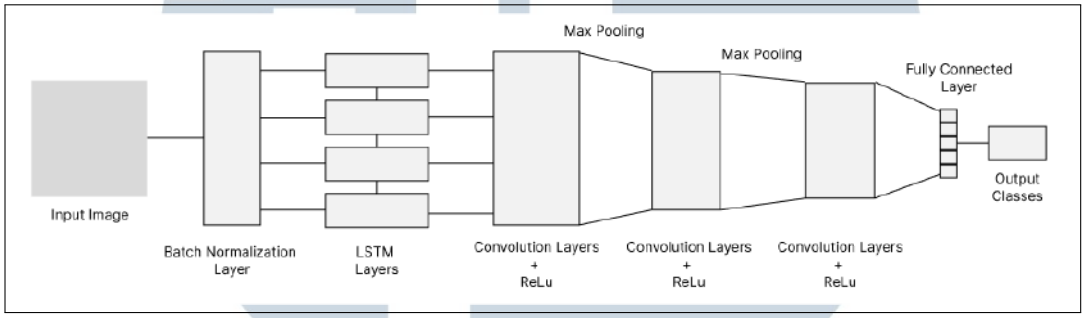


Gambar 3.8. Arsitektur model LSTM.

Pada Gambar 3.9 adalah model arsitektur LSTM-CNN yang akan digunakan dalam penelitian ini. Model LSTM-CNN menerima *Input shape* (None, 128, 128, 3). Model LSTM-CNN adalah hasil pengabungan model LSTM dan CNN. Pada sel LSTM layer dikonfigurasi seperti pembangunan model LSTM sebelumnya dengan return state menjadi true, yaitu diberikan 32 jumlah *unit*, fungsi aktivasi tanh, dan

nilai *dropout* 0.2. Sebelum memasuki ke CNN *layers*, hasil seluruh *output sequence* LSTM diperluas terlebih dahulu dan dapat digunakan dalam CNN *layers*.

Pada konfigurasi CNN *layers* sama dengan konfigurasi model CNN sebelumnya, yaitu pada *layer* pertama terdapat 32 *units*, layer kedua terdapat 64 *units*, dan *layer* ketiga terdapat 128 *units*. Setiap CNN *layer* diberikan fungsi aktivasi ReLu dan *MaxPooling* dengan ukuran *pool* (2,2). Terakhir diberikan *Fully-Connected Layer* untuk menghubungkan semua neuron. Hasil potongan kode pembuatan model CNN dapat dilihat pada Kode 5.3.



Gambar 3.9. Arsitektur model LSTM-CNN.

3.3 Implementasi

Setelah melakukan tahap pengolahan data, tahap selanjutnya adalah membuat membangun tiga model dan mendapatkan hasil akurasi *training* dan akurasi *validation* yaitu CNN, LSTM, dan LSTM-CNN *hybrid model*. Kemudian menampilkan grafik hasil akurasi *training* dan *validation* serta nilai *loss*.

3.4 Pengujian

Pada tahap pengujian, model yang telah dibuat dan dilatih akan di uji apakah hasil *input* gambar beberapa bakteri yang diberikan sesuai dengan nama jenis bakteri. Kemudian mendapatkan hasil akurasi *confidence* dari setiap model.

3.5 Evaluasi

Evaluasi hasil nilai akurasi *training*, *validation*, dan *confidence* yang dihasilkan dan melakukan perbandingan hasil nilai akurasi dari tiga model CNN, LSTM, dan LSTM-CNN.

3.6 Penulisan Laporan

Setelah melakukan evaluasi dan mendapatkan hasil, tahap terakhir yang dilakukan adalah pembuatan laporan. Laporan dibuat untuk melakukan dokumentasi data-data yang telah diolah. Mulai dari Penulisan landasan teori hingga tahap data sudah dapat divisualisasi dengan baik.

3.7 Spesifikasi Sistem

Berikut adalah spesifikasi sistem komputer pribadi yang digunakan dalam penelitian:

1. Perangkat keras
    - Prosesor: Intel(R) Core(TM) I7-12700K CPU @ 3.60GHz (20 CPUs), 3.6GHz.
    - Memori: 32768 MB RAM DDR5.
    - VGA: NVIDIA GeForce RTX 3070 Ti.
    - Solid State Drive: 222.98 GB.
    - Hard Disk Drive: 1862.89 GB.
2. Perangkat lunak
    - Sistem Operasi: Windows 10 Pro 64-bit (10.0, Build 19045).
    - Bahasa Pemrograman: LaTeX (Overleaf), Python.
    - IDE: Jupyter Notebook.
    - library: Numpy, Matplotlib, scikit-learn, Scipy, Tensorflow, Tensorflow-GPU, opencv-python.

3.8 Rencana Waktu Penelitian

Pada bagian ini dijelaskan rencana waktu penelitian menggunakan Gantt Chart. Rencana waktu penelitian dapat dilihat pada Tabel 3.1:

Tabel 3.1. Rencana waktu penelitian

Kegiatan	Minggu ke-													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Studi Litelatur														
Pengolahan data														
Implementasi														
Pengujian														
Evaluasi														
Penulisan Laporan														