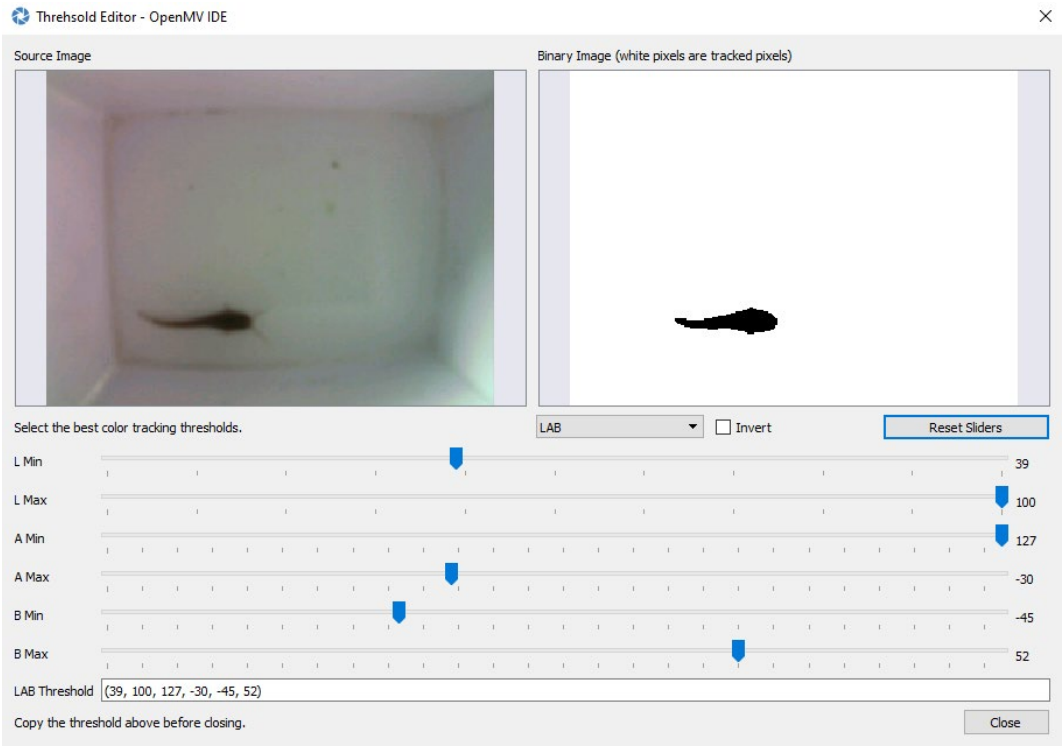
BAB V. IMPLEMENTASI DAN PENGUJIAN

Bab ini menjelaskan tentang implementasi dan pengujian sistem klasifikasi ukuran bibit ikan lele menggunakan Convolutional Neural Network (CNN) dengan OpenMV Cam. Bagian ini menjelaskan langkah-langkah teknis yang akan diambil dalam mengimplementasikan rancangan dan kemudian dilakukan pengujian terhadap rancangan tersebut. Menggunakan arsitektur MobileNetV2 dengan TensorFlow dan mengintegrasikannya dengan OpenMV Cam. Setelah tahap implementasi rancangan selesai, dilakukan pengujian sistem hasil dari rancangan tersebut menggunakan *dataset* bibit ikan lele yang telah didapat. Dari pengujian tersebut, didapatkan hasil akurasi dari sistem berupa metrik evaluasi seperti nilai akurasi, presisi, *recall*, atau F1-*score* dapat digunakan untuk mengukur kinerja model.

5.1 Pengambilan Dataset

Pengambilan *dataset* dengan menggunakan OpenMV Cam H7+. Untuk mendeteksi dan *tracking* bibit ikan lele, menggunakan fitur yang didukung oleh OpenMV Cam H7+. Fitur tersebut adalah *blob tracking*, yang mana fitur ini dapat melakukan deteksi dan *tracking* bibit ikan lele yang tertangkap oleh kamera. Dalam persiapan penggunaan *blob tracking*, perlu dilakukan penyesuaian nilai *threshold*. Nilai *threshold* pada OpenMV Cam adalah skala yang digunakan untuk merubah tingkat citra berwarna menjadi citra biner.

Dengan mengatur nilai *threshold*, dapat dilakukan pemisahan objek yang diminati dari latar belakang dengan cara mengubah semua piksel di atas nilai ambang menjadi putih dan yang di bawahnya menjadi hitam, atau sebaliknya. Nilai ambang yang dipilih bergantung pada karakteristik citra dan objek yang ingin dipisahkan. Jika citra memiliki kontras yang baik antara objek dan latar belakang, maka nilai ambang yang cukup sederhana dapat memberikan hasil yang baik. Namun, jika citra memiliki kontras rendah atau pencahayaan yang tidak merata, maka pemilihan *threshold* yang tepat menjadi lebih sulit. Pengaturan nilai *threshold* pada OpenMV Cam dapat diperoleh melalui menu pada aplikasi OpenMV IDE (*Integrated Development Environment*), seperti yang dapat dilihat pada gambar 5.1 di bawah ini.



Gambar 5.1 Menu Pengaturan *Threshold* pada OpenMV IDE

Setelah nilai *threshold* didapatkan, maka pengambilan *dataset* dapat dilakukan. Dalam penelitian ini, dengan menggunakan fitur *blob tracking*, pengambilan *dataset* dilakukan dengan pengembangan dari fitur tersebut. Pengembangan fitur tersebut berupa membuat *blob tracking* dengan 2 format, seperti yang telah dijelaskan sebelumnya. Format pertama adalah blobs.fit dan blobs.rect. *Source Codes* dari kedua format tersebut dapat dilihat pada tabel 5.1 dan tabel 5.2 di bawah ini.

Tabel 5.1 *Source Codes* blobs.fit

```
import pyb # Import module for board related functions
import sensor # Import the module for sensor related functions
import image # Import module containing machine vision
algorithms
import time # Import module for tracking elapsed time

sensor.reset() # Resets the sensor
sensor.set_pixformat(sensor.RGB565) # Sets the sensor to RGB
sensor.set_framesize(sensor.QVGA) # Sets the resolution to
320x240 px
#sensor.set_vflip(True) # Flips the image vertically
#sensor.set_hmirror(True) # Mirrors the image horizontally
sensor.skip_frames(time = 2000) # Skip some frames to let the
image stabilize
sensor.set_framerate(40)
```

```
# Define the min/max LAB values we're looking for
thresholds = (39, 100, 127, -30, -45, 52)
record_time = 10000 # 10 seconds in milliseconds
stream = image.ImageIO("image/fit.baru/video/gradeA209.bin",
"w")

clock = time.clock() # Instantiates a clock object

start = pyb.millis()
while pyb.elapsed_millis(start) < record_time:
    clock.tick() # Advances the clock
    img = sensor.snapshot() # Takes a snapshot and saves it in
memory

    # Overlapping blobs will be merged
    blobs = img.find_blobs([thresholds], invert=True,
merge=True)

    # Draw blobs
    for blob in blobs:
        roi = blob.rect()
        imgroi = img.copy(roi=roi)
        # Draw a rectangle where the blob was found
        img.draw_rectangle(blob.rect(), color=(0,255,0))
        #Draw a cross in the middle of the blob
        img.draw_cross(blob.cx(), blob.cy(), color=(0,255,0))

        filename = "imageC_" + str(pyb.millis()) + ".bmp"
        imgroi.save("image/rect.fit/grade c/" + filename)

    stream.write(img)
    print(clock.fps()) # Prints the framerate to the serial
console
```

Tabel 5.2 *Source Codes* blobs.rect

```
import sensor, image, time, pyb

sensor.reset()                         # Reset sensor
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565
sensor.set_framesize(sensor.QVGA)    # Set frame size to QVGA
(320x240)
sensor.set_auto_gain(False)          # must be turned off for
color tracking
sensor.set_auto_whitebal(False)      # must be turned off for
color tracking
sensor.skip_frames(time = 2000)      # Wait for sensor to adjust

record_time = 10000 # 10 seconds in milliseconds
clock = time.clock() # Instantiates a clock object
start = pyb.millis()

# Set color thresholds
thresholds = [(39, 100, 127, -30, -45, 52)] # You may need to
adjust these thresholds for your colorspace
```

```
while pyb.elapsed_millis(start) < record_time:
    # Advances the clock
    clock.tick()
    # Capture image
    img = sensor.snapshot()

    # Find blobs
    blobs = img.find_blobs(thresholds, pixels_threshold=200,
area_threshold=200, invert=True)

    # Draw blobs
    for blob in blobs:
        # Calculate square size based on the maximum of width
and height of the blob
        square_size = max(blob.w(), blob.h())

        # Calculate the top-left corner coordinates of the
square
        x = blob.cx() - square_size // 2
        y = blob.cy() - square_size // 2

        roi = [x,y,square_size,square_size]
        roi_img = img.copy(roi=roi)

        # Increase the width of the square by multiplying the
square_size with a factor
        width_factor = 1.2  # Adjust this factor according to
your desired width increase
        square_size = int(square_size * width_factor)

        # Menggambar persegi menggunakan koordinat yang telah
dihitung
        img.draw_rectangle((x, y, square_size,square_size),
color=(0, 255, 0))
        img.draw_cross(blob.cx(), blob.cy(), color=(0, 255, 0))

        # Save the ROI image
        filename = "imageC_" + str(pyb.millis()) + ".bmp"
        roi_img.save("image/rect.otamatis/grade c/" + filename)

    # Print FPS to serial console
    print(clock.fps())
```

Dengan menjalankan *source codes* tersebut, dihasilkan kumpulan citra *frame* yang terdapat bibit ikan lele. Total *dataset* yang didapatkan dalam proses pengambilan *dataset* pada masing-masing format dapat dilihat pada tabel 5.3 dan 5.4 di bawah ini.

Tabel 5.3 Jumlah *Dataset* blobs.fit

No	Grade	Jumlah Data
1	A	210
2	B	210
3	C	210
Total		630

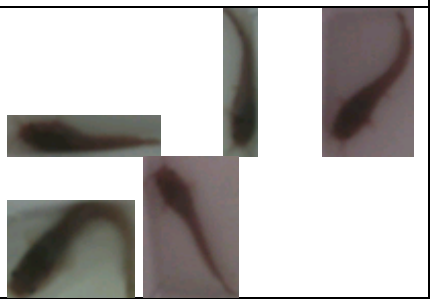
Tabel 5.4 Jumlah *Dataset* blobs.rect

No	Grade	Jumlah Data
1	A	580
2	B	580
3	C	580
Total		1740

Dari kedua tabel tersebut, dapat diketahui jika terdapat perbedaan pada perolehan jumlah *dataset* bibit ikan lele. Hal ini terjadi karena adanya perbedaan format deteksi dan perbedaan tersebut mempengaruhi *frame per second* pada performa OpenMV Cam H7+ untuk melakukan pendeteksian terhadap objek yang ditangkap. Format blobs.fit memiliki *dataset* yang lebih sedikit daripada blobs.rect karena bujur sangkar dari blobs.fit mengikuti ukuran bibit ikan lele sesuai dengan ukuran bibit ikan lele yang dideteksi.

Tabel 5.5 dan tabel 5.6 merupakan contoh citra *frame* yang telah diperoleh. Gambar yang ditampilkan merupakan ukuran sebenarnya dari hasil yang didapat.

Tabel 5.5 *Dataset* blobs.fit

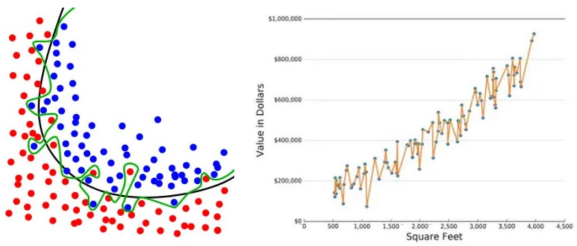
Grade A	Grade B	Grade C
		

Tabel 5.6 *Dataset* blobs.rect

Grade A	Grade B	Grade C
		

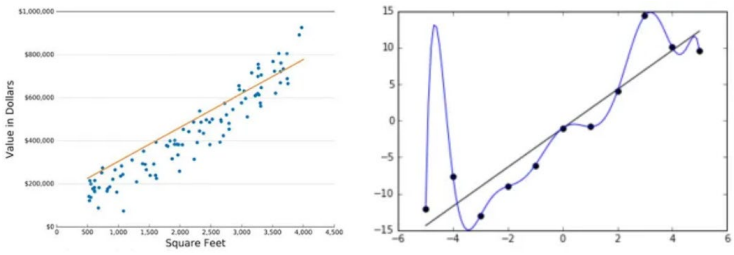
5.2 Training Process

Tahap ini merupakan tahap untuk melatih model menggunakan *dataset* yang telah dikumpulkan sebelumnya. Hasil dari proses *training* yang baik adalah model yang tidak mengalami *overfitting* dan *underfitting*. *Overfitting* adalah suatu kedaan dimana data yang digunakan pada *data training* memiliki prediksi yang terlalu baik, namun prediksinya buruk pada data testing. Ketika sebuah model *overfit*, model tidak dapat melakukan generalisasi dengan baik sehingga apabila dilakukan tes dengan menggunakan data yang berbeda dapat mengurangi akurasi (hasil yang dibuat tidak sesuai yang diharapkan). *Underfitting* adalah keadaan dimana model pelatihan data yang dibuat tidak mewakilkan keseluruhan data yang akan digunakan nantinya. *Underfitting* dapat terjadi ketika model terlalu sederhana dan tidak mampu untuk menyesuaikan pola yang terdapat pada *data training*, sehingga menghasilkan peforma yang buruk dalam *data training*. Berikut gambar 5.2 yang menampilkan contoh grafik yang mengalamni *overfitting* dan gambar 5.3 yang mengalami *underfitting*.



Gambar 5.2 Contoh Grafik *Overfitting*

(Sumber: Overfitting dan Underfitting. Salah satu hal yang terpenting untuk… | by Arief Ruslidiantoro | Medium)



Gambar 5.3 Contoh Grafik *Underfitting*

(Sumber: Overfitting dan Underfitting. Salah satu hal yang terpenting untuk… | by Arief Ruslidiantoro | Medium)

Dalam penelitian ini, selain dengan menggunakan blobs.fit dan blobs.rect sebagai *dataset*, ukuran *image* yang akan diproses menggunakan 2 ukuran yang berbeda yaitu 96 x 96 dan 128 x 128. Sehingga, ada 4 (empat) jenis bahan penelitian:

a. Ukuran 96 x 96 blobs.fit
b. Ukuran 96 x 96 blobs.rect
c. Ukuran 128 x 128 blobs.fit
d. Ukuran 128 x 128 blobs.rect

5.2.1 Import Library

Dalam persiapan menjalan program untuk proses *training*, pengunduhan dan instalasi *library* diperlukan untuk dapat menjalankan CNN. *Library* merupakan kumpulan kode atau modul yang dapat digunakan kembali dalam pengembangan perangkat lunak. *Library* ini berisi fungsi-fungsi untuk mendukung dan dapat menjalankan metode tertentu. Berikut tabel 5.7 yang berisikan *source codes* untuk meng-*import* library yang dibutuhkan.

Tabel 5.7 *Source Codes* Import Library

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf

from PIL import Image
from numpy import asarray
```

```
from tensorflow import keras
from keras.utils import np_utils
from tensorflow.keras import datasets,layers, models
```

5.2.2 Persiapan Direktori

Pengaturan direktori yang baik dapat memudahkan untuk menemukan dan mengakses *file* yang dibutuhkan. Struktur direktori yang terorganisir dapat dengan cepat menavigasi melalui *folder* dan *subfolder* untuk mengakses *file* yang dibutuhkan. *Folder* yang dimaksud seperti *folder dataset*, *data training, validation*, dan *testing*. *Folder* tersebut berisikan *folder* untuk masing-masing *grade*. Setelah *folder* tersebut dibuat, maka pengkoneksian melalui *text editor* dapat dilakukan. Karena dalam penelitian ini menggunakan 2 jenis format data yang berbeda yaitu blobs.fit dan blobs.rect, maka koneksi direktori ke *dataset* ada 2 yang berbeda. Perbedaan tersebut dapat dilihat pada tabel 5.8 dan 5.9 di bawah ini.

Tabel 5.8 Koneksi Direktori blobs.fit

```
base_dir = "blobs.fit/coba3/"
latih =  "fit.dataset/latih"
validasi = "fit.dataset/validasi"
testing = "fit.dataset/testing"
```

Tabel 5.9 Koneksi Direktori blobs.fit

```
base_dir = "blobs.rect/coba2/"
latih =  "rect.dataset/latih"
validasi = "rect.dataset/validasi"
testing = "rect.dataset/testing"
```

Setelah menyesuaikan kebutuhan direktori *dataset* yang diinginkan, maka melanjutkan perintah untuk koneksi dengan direktori. Berikut tabel 5.10 berisikan *source codes* untuk melanjutkan koneksi dengan *folder* tersebut.

Tabel 5.10 *Source Codes* Persiapan dan Koneksi ke Direktori

```
gradeA_dir = os.path.join(bahan_dir, 'gradeA/')
gradeB_dir = os.path.join(bahan_dir, 'gradeB/')
gradeC_dir = os.path.join(bahan_dir, 'gradeC/')

print("Jumlah data tiap kelas")
print("Jumlah gambar grade  A : ", len(os.listdir(gradeA_dir)))
```

```
print("Jumlah gambar grade  B : ", len(os.listdir(gradeB_dir)))
print("Jumlah gambar grade  C : ", len(os.listdir(gradeC_dir)))
print("Total dataset : ", len(os.listdir(gradeA_dir)) +
len(os.listdir(gradeB_dir)) + len(os.listdir(gradeC_dir)))

# direktori isi latih/training
train_gradeA = os.path.join(train_dir, 'gradeA/')
train_gradeB = os.path.join(train_dir, 'gradeB/')
train_gradeC = os.path.join(train_dir, 'gradeC/')

# direktori isi validasi
valid_gradeA = os.path.join(valid_dir, 'gradeA/')
valid_gradeB = os.path.join(valid_dir, 'gradeB/')
valid_gradeC = os.path.join(valid_dir, 'gradeC/')

# direktori isi test
test_gradeA = os.path.join(test_dir, 'gradeA/')
test_gradeB = os.path.join(test_dir, 'gradeB/')
test_gradeC = os.path.join(test_dir, 'gradeC/')
```

5.2.3 Membagi Dataset

Dataset yang telah didapatkan sebelumnya, kemudia dilakukan pembagian untuk *data training, data validation*, dan *data testing*. Hasil dari pembagian tersebut akan tersimpan ke *folder* yang telah disiapkan sebelumnya sesuai dengan tempatnya. Berikut tabel 5.11 yang berisikan *source codes* untuk menjalankan perintah tersebut.

Tabel 5.11 *Source Codes* Membagi Dataset

```
import random
from shutil import copyfile

def train_val_split(source, train, val, test, train_ratio,
val_ratio):
  total_size = len(os.listdir(source))
  train_size = int(train_ratio * total_size)
  val_size = int(val_ratio * total_size)
  test_size = total_size - train_size - val_size

  randomized  = random.sample(os.listdir(source), total_size)
  train_files = randomized[0:train_size]
  val_files   = randomized[train_size:train_size+val_size]
  test_files = randomized[train_size+val_size:total_size]
```

```
  for i in train_files:
    i_file = source + i
    destination = train + i
    copyfile(i_file, destination)

  for i in val_files:
    i_file = source + i
    destination = val + i
    copyfile(i_file, destination)
  for i in test_files:
    i_file = source + i
    destination = test + i
    copyfile(i_file, destination)

# jumlah pembagian data training dan testing
train_ratio = 0.9
val_ratio = 0.05

#pembagian training dan validasi
#gradeA
source_00 = gradeA_dir
train_00  = train_gradeA
val_00    = valid_gradeA
test_00   = test_gradeA
train_val_split(source_00, train_00, val_00, test_00, train_ratio,
val_ratio )

#gradeB
source_01 = gradeB_dir
train_01  = train_gradeB
val_01    = valid_gradeB
test_01   = test_gradeB
train_val_split(source_01, train_01, val_01, test_01, train_ratio,
val_ratio )

#gradeC
source_02 = gradeC_dir
train_02  = train_gradeC
val_02    = valid_gradeC
test_02   = test_gradeC
train_val_split(source_02, train_02, val_02, test_02, train_ratio,
val_ratio)

print("Jumlah all grade A   : ", len(os.listdir(gradeA_dir)))
print("Jumlah train grade A : ", len(os.listdir(train_gradeA)))
```

```
print("Jumlah validation grade A   : ",
len(os.listdir(valid_gradeA)))
print("Jumlah testing grade A   : ", len(os.listdir(test_gradeA)))

print("\nJumlah all grade B   : ", len(os.listdir(gradeB_dir)))
print("Jumlah train grade B : ", len(os.listdir(train_gradeB)))
print("Jumlah validation grade B   : ",
len(os.listdir(valid_gradeB)))
print("Jumlah testing grade B   : ", len(os.listdir(test_gradeB)))

print("\nJumlah all grade C   : ", len(os.listdir(gradeC_dir)))
print("Jumlah train grade C : ", len(os.listdir(train_gradeC)))
print("Jumlah validation grade C   : ",
len(os.listdir(valid_gradeC)))
print("Jumlah testing grade C   : ", len(os.listdir(test_gradeC)))
```

5.2.4 Pre-processing Image

Pre-processing image pada CNN adalah serangkaian langkah untuk mempersiapkan gambar sebelum digunakan sebagai input dalam jaringan saraf tiruan CNN. *Pre-processing* ini bertujuan untuk meningkatkan kualitas dan keseragaman gambar, serta mempermudah pemrosesan oleh CNN. Tahap awal *Pre-processing image* dengan melakukan perintah ImageDataGenerator. Berikut tabel 5.12 yang berisikan perintah untuk menjalankannya.

Tabel 5.12 *Source Codes* ImageDataGenerator

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
train_datagen       = ImageDataGenerator(
    rescale         = 1./255
)

val_datagen = ImageDataGenerator(
    rescale         = 1./255
)
test_datagen = ImageDataGenerator(
    rescale         = 1./255
)
```

Setelah ImageDataGenerator, merupakan tahap augmentasi gambar. Berikut tabel 5.13 yang berisikan perintah untuk menjalankannya.

Tabel 5.13 Augmentasi Gambar

```
# UNTUK UKURAN 96 X 96
IMG_SIZE = (96,96)
# UNTUK UKURAN 128 X 128
IMG_SIZE = (128,128)
# ------------------------------------------------

BATCH_SIZE = 32

train_dataset = train_datagen.flow_from_directory(
    train_dir,
    target_size = IMG_SIZE,
    batch_size = BATCH_SIZE,
    shuffle    = False,
    class_mode = 'categorical',
    color_mode = 'rgb'
)
valid_dataset = val_datagen.flow_from_directory(
    valid_dir,
    target_size = IMG_SIZE,
    batch_size = BATCH_SIZE,
    shuffle    = False,
    class_mode = 'categorical',
    color_mode = 'rgb'
)
test_dataset = test_datagen.flow_from_directory(
    test_dir,
    target_size = IMG_SIZE,
    batch_size = BATCH_SIZE,
    shuffle    = False,
    class_mode = 'categorical',
    color_mode = 'rgb'
)
```

Karena dalam penelitian ini menggunakan 2 ukuran gambar yang berbeda, maka pada *source codes* di atas, terdapat *codes* dengan pengaturan 2 ukuran gambar berbeda (pada baris pertama dan kelima). Jalankan perintah tersebut setelah

menghapus atau mematikan keaktifan *source codes* salah satu deklarasi ukuran gambar, 96 x 96 atau 128 x 128.

Penentuan nilai *batch size* peneliti menggunakan nilai secara *default* yaitu 32. Peneliti juga melakukan percobaan dalam menentukan nilai *batch size* untuk menemukan hasil terbaik dalam proses *training*, dengan menggunakan nilai *batch size* lainnya seperti 16, 32, dan 64 pada bahan penelitian blobs.rect 128x128 dengan nilai *base learning rate* 0,0001 dan epoch sejumlah 100. Hasilnya dapat dilihat pada tabel 5.14.

Tabel 5.14 *Batch Size* 16

Batch 16	Precision	Recall	F1-score
Grade A	0,90	0,93	0,92
Grade B	0,89	0,86	0,88
Grade C	0,97	0,97	0,97
Accuracy	0,92	0,92	0,92
Total Waktu	23 menit 22,4 detik		

Tabel 5.15 *Batch Size* 32

Batch 32	Precision	Recall	F1-score
Grade A	0,97	0,98	0,97
Grade B	0,98	0,96	0,97
Grade C	1,00	1,00	1,00
Accuracy	0,98	0,98	0,98
Total Waktu	21 menit 12 detik		

Tabel 5.16 *Batch Size* 64

Batch 64	Precision	Recall	F1-score
Grade A	0,93	0,97	0,95
Grade B	0,96	0,90	0,93
Grade C	0,97	1,00	0,98
Accuracy	0,95	0,96	0,95
Total Waktu	19 menit 58 detik		

Dari penghitungan tersebut, diketahui jika nilai *batch size* 32 memiliki akurasi yang lebih baik daripada 16 dan 64. Sehingga, peneliti menentukan nilai *batch size* sama seperti dengan nilai *default*, yaitu 32.

Kemudian ada fungsi train_datagen.flow_from_directory(). Fungsi ini digunakan untuk memuat *data training* dari direktori yang telah dibuat sebelumnya.

Data training akan diambil dari direktori train_dir. Parameter lain yang diatur antara lain:

- target_size

  target_size merupakan ukuran target untuk gambar yang akan dihasilkan.

- batch_size

  batch_size untuk menentukan jumlah sampel gambar yang akan dihasilkan dalam satu *batch*.

- shuffle

  shuffle untuk menentukan apakah data akan diacak sebelum setiap epoch.

- class_mode

  class_mode diatur sebagai 'categorical' untuk tugas klasifikasi multikelas (*grade* A, *grade* B, *grade* C)

- color_mode

  color_mode yang diatur sebagai 'rgb' untuk menunjukkan bahwa gambar akan diubah menjadi *mode* warna RGB.

5.2.5 Pembuatan Model

Pada tahap ini, memulai untuk pembuatan model dengan arsitektur MobileNetV2. Berikut *source codes* untuk menjalankan perintahnya yang dapat dilihat pada tabel 5.17 di bawah ini :

Tabel 5.17 *Source Codes* Arsitektur MobileNetV2

```
IMG_SHAPE = IMG_SIZE + (3,)
MobileNetV2 =
tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
include_top=False, weights='imagenet')

for layer in MobileNetV2.layers:
    layer.trainable = False

x = MobileNetV2.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
output = tf.keras.layers.Dense(3, activation='softmax')(x)
modelku = tf.keras.Model(inputs=MobileNetV2.input, outputs =
output)
```

Source codes di atas memiliki fungsi :

1. Membentuk model MobileNetV2 dengan menggunakan *input shape* yang telah ditentukan. Pada baris pertama, IMG_SHAPE didefinisikan sebagai IMG_SIZE + (3,), yang berarti menggabungkan ukuran gambar dengan tuple (3,) yang mewakili saluran warna RGB. Kemudian, pada baris kedua, MobileNetV2 dibuat menggunakan tf.keras.applications.MobileNetV2 dengan parameter :
   a input_shape=IMG_SHAPE
      input_shape digunakan untuk menentukan ukuran input model.
   b include_top=False
      Menunjukkan bahwa lapisan teratas (seperti lapisan klasifikasi) tidak akan disertakan dalam model.
   c weights='imagenet'
      mengindikasikan bahwa bobot pre-trained dari model MobileNetV2 akan digunakan.
2. Melakukan pengaturan agar semua layer di dalam model MobileNetV2 tidak dapat dilatih (non-trainable). Pada baris keenam, *loop* for digunakan untuk mengiterasi setiap layer dalam MobileNetV2.layers. Setiap *layer* kemudian diatur menjadi tidak dapat dilatih dengan mengatur atribut *trainable* menjadi *False*. Dengan mengatur semua *layer* menjadi *non-trainable*, berarti tidak akan melatih atau memperbarui bobot *layers* tersebut selama proses pelatihan model.
3. "x = MobileNetV2.output", untuk mengambil output dari MobileNetV2 sebagai *input* untuk lapisan berikutnya
4. "x = tf.keras.layers.GlobalAveragePooling2D()(x)", untuk Menambahkan lapisan *Global Average Pooling* 2D setelah *output* MobileNetV2. Ini mengubah setiap fitur spasial dalam gambar menjadi vektor satu dimensi dengan mengambil rata-rata semua fitur spasial. Hal ini berguna untuk mengurangi dimensi data sebelum memasukkannya ke lapisan klasifikasi berikutnya.
5. "modelku = tf.keras.Model(inputs=MobileNetV2.input, outputs=output)", merupakan *variable* yang menampung hasil dari dijalankannya proses tersebut.

Kemudian, dilakukan pengaturan konfigurasi dan kompilasi model dalam TensorFlow menggunakan Keras API. Berikut tabel 5.18 yang berisikan perintah untuk menjalankannya.

Tabel 5.18 TensorFlow dengan Keras API

```
base_learning_rate = 0.0001
modelku.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=b
ase_learning_rate),
                loss="categorical_crossentropy",
                metrics=['accuracy'])
```

Pada *source codes* di atas, terdapat deklarasi "base_learning_rate". "base_learning_rate = 0.0001" mendefinisikan tingkat pembelajaran awal (*learning rate*) yang akan digunakan oleh *optimizer* untuk meng-*update* bobot model selama proses *training*.

"modelku.compile(...)" digunakan untuk mengompilasi model yang sebelumnya telah dibuat dengan nama "modelku" dan menentukan pengaturan penting seperti *optimizer*, fungsi kerugian (l*oss function)*, dan metrik evaluasi yang akan digunakan selama *training*.

5.2.6 Training Model

Setelah model dibuat, dilakukanlah proses *training model*. Berikut tabel 5.19 yang berisikan *source codes* untuk menjalankan perintah *training model*.

Tabel 5.19 *Training Model*

```
history = modelku.fit(train_dataset,
epochs=100,validation_data=valid_dataset)
```

Pada tahap ini, dengan adanya 4 jenis bahan penelitian maka dilakukan *training* sebanyak 4 kali pada masing-masing jenis bahan penelitian. Perbedaan tersebut terletak pada jumlah epoch yang dijalankan, yaitu 10 (sepuluh), 25 (dua puluh lima), 50 (lima puluh), dan 100 (seratus). Sehingga, hasil *training* yang didapat sejumlah 16 (enam belas). Hasil tersebut dapat dilihat pada tabel 5.20, 5.21, 5.22, dan 5.23.

- Ukuran 96 x 96 blobs.fit

Tabel 5.20 Hasil *Training* Ukuran 96 x 96 blobs.fit

Representasi			
epoch = 10			
	Precision	Recall	F1-score
Grade A	0,73	0,80	0,76
Grade B	0,77	1,00	0,87
Grade C	0,67	0,40	0,50
Accuracy	0,72	0,73	0,71
epoch 25			
	Precision	Recall	F1-score
Grade A	1,00	0,70	0,82
Grade B	0,77	1,00	0,87
Grade C	0,90	0,90	0,90
Accuracy	0,89	0,87	0,86
epoch = 50			
	Precision	Recall	F1-score
Grade A	1,00	1,00	1,00
Grade B	1,00	1,00	1,00
Grade C	1,00	1,00	1,00
Accuracy	1,00	1,00	1,00
epoch 100			
Validation			
	Precision	Recall	F1-score
Grade A	1,00	1,00	1,00
Grade B	1,00	1,00	1,00
Grade C	1,00	1,00	1,00
Accuracy	1,00	1,00	1,00

- Ukuran 96 x 96 blobs.rect

Tabel 5.21 Hasil *Training* Ukuran 96 x 96 blobs.rect

Representasi			
epoch = 10			
	Precision	Recall	F1-score
Grade A	1,00	0,86	0,93
Grade B	0,81	0,86	0,83
Grade C	0,87	0,93	0,90
Accuracy	0,89	0,88	0,89
epoch 25			
	Precision	Recall	F1-score
Grade A	1,00	0,90	0,95
Grade B	0,88	1,00	0,94
Grade C	1,00	0,97	0,98
Accuracy	0,96	0,96	0,96
epoch = 50			
	Precision	Recall	F1-score
Grade A	1,00	0,97	0,98
Grade B	0,97	1,00	0,98
Grade C	1,00	1,00	1,00
Accuracy	0,99	0,99	0,99
epoch 100			
	Precision	Recall	F1-score
Grade A	1,00	0,97	0,98
Grade B	0,94	1,00	0,97
Grade C	1,00	0,97	0,98
Accuracy	0,98	0,98	0,98

- Ukuran 128 x 128 blobs.fit

Tabel 5.22 Hasil *Training* Ukuran 128 x 128 blobs.fit

Representasi			
epoch = 10			
	Precision	Recall	F1-score
Grade A	0,89	0,80	0,84
Grade B	0,71	1,00	0,83
Grade C	0,86	0,60	0,71
Accuracy	0,82	0,80	0,79
epoch 25			
	Precision	Recall	F1-score
Grade A	0,90	0,90	0,90
Grade B	1,00	0,90	0,95
Grade C	0,91	1,00	0,95
Accuracy	0,94	0,93	0,93
epoch = 50			
	Precision	Recall	F1-score
Grade A	1,00	0,90	0,95
Grade B	0,91	1,00	0,95
Grade C	1,00	1,00	1,00
Accuracy	0,97	0,97	0,97
epoch 100			
	Precision	Recall	F1-score
Grade A	1,00	1,00	1,00
Grade B	1,00	1,00	1,00
Grade C	1,00	1,00	1,00
Accuracy	1,00	1,00	1,00

- Ukuran 128 x 128 blobs.rect

Tabel 5.23 Hasil *Training* Ukuran 128 x 128 blobs.rect

Representasi			
epoch 10			
	Precision	Recall	F1-score
Grade A	0,87	0,90	0,88
Grade B	0,82	0,62	0,71
Grade C	0,80	0,97	0,88
Accuracy	0,83	0,83	0,82
epoch 25			
	Precision	Recall	F1-score
Grade A	0,90	0,97	0,93
Grade B	0,91	0,72	0,81
Grade C	0,85	0,97	0,90
Accuracy	0,89	0,89	0,88
epoch 50			
	Precision	Recall	F1-score
Grade A	0,87	0,93	0,90
Grade B	0,88	0,79	0,84
Grade C	0,93	0,97	0,95
Accuracy	0,89	0,90	0,90
epoch 100			
	Precision	Recall	F1-score
Grade A	0,90	0,93	0,92
Grade B	0,89	0,86	0,88
Grade C	0,97	0,97	0,97
Accuracy	0,92	0,92	0,92

5.3 Save dan Convert Model

Dengan menyimpan model CNN yang telah di-*training* memungkinkan untuk menggunakannya kembali tanpa perlu *training* ulang dari awal. Model yang disimpan juga dapat dilanjutkan untuk di-*export* dari berformat .h5 menjadi .tflite yang dapat dijalankan di OpenMV Cam H7+. Model yang disimpan dalam penelitian ini merupakan model yang memiliki jumlah epoch sebanyak 100 pada

tiap masing-masing jenis bahan penelitian ketika proses *training*. Sehingga, didapatkan jumlah 4 model.

5.3.1 Save Model

Berikut tabel 5.24 yang berisikan *source codes* untuk menjalankan perintah penyimpanan model.

Tabel 5.24 *Source Codes Save Model*

```
modelku.save('model/FIT_coba3_BL00001_no-dropout_size96.h5')
```

Pada *source codes* di atas, *variable* "modelku" yang merupakan *variable* pada proses sebelumnya dideklarasikan sebagai model yang akan disimpan. Kemudian, menentukan direktori dan nama *file* ketika nantinya berhasil disimpan.

5.3.2 Convert Model

Model yang berhasil disimpan kemudian di-*export* ke .tflite agar dapat berjalan ketika digunakan di OpenMV Cam yang mendukung *file* berformat .tflite. Berikut *source codes* untuk menjalankan perintah *exporting* yang dapat dilihat pada tabel 5.25.

Tabel 5.25 *Source Codes Convert Model*

```
import tensorflow as tf

# Load the Keras model
model = tf.keras.models.load_model('model\FIT_coba3_BL00001_no-
dropout_size96.h5')

# Convert TFLITE
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Menyimpan Hasil Convert
with open('tflite/FIT_coba3_BL00001_no-dropout_size96.tflite',
'wb') as f:
    f.write(tflite_model)
```

Pada bagian "Load the Keras Model", dilakukan pemilihan direktori terhadap model yang akan dilakukan *exporting*. Kemudian, pada bagian "Convert TFLITE" merupakan perintah untuk meng-*export* model yang telah dipilih untuk di-*conver*

menjadi .tflite. Bagian "Menyimpan Hasil Convert" merupakan tahap untuk memilih direktori dan nama *file* yang akan disimpan sebagai hasil dari *converting*.

5.4 Pengujian Model

Pengujian model dilakukan untuk mengetahui performa model yang telah di-*training*. Pengujian dilakukan menggunakan program Python dengan *text editor* Visual Studio Code. Model yang telah di-*training* diuji menggunakan *data testing* yang memiliki *variable* bernama "test_dataset" ketika dijalankan pada saat *pre-processing*. Untuk mengetahuinya, dapat membuat *calculation report* yang menyediakan informasi tentang kinerja model CNN. Informasi ini mencakup metrik evaluasi seperti nilai akurasi, presisi, *recall*, dan F1-score. Dengan melihat laporan perhitungan ini, dapat mengetahui sejauh mana model CNN bekerja dengan baik dan apakah ada area di mana model tersebut perlu ditingkatkan. *Calculation report* yang akan digunakan adalah berupa grafik dengan tampilan matplotlib dan modul sklearn.metrics yang menghasilkan laporan klasifikasi mencakup berbagai metrik evaluasi seperti nilai akurasi, presisi, *recall*, dan f1-*score*.

Laporan klasifikasi berupa *confusion matrix* dapat diperoleh dengan menjalankan *source codes* yang ada pada tabel 5.26.

Tabel 5.26 *Source Code Confusion Matrix*

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
class_list = os.listdir(train_dir)
# membuat prediksi pada data validasi
y_pred = modelku.predict(test_dataset)
y_pred = np.argmax(y_pred, axis=1)

# mengambil label yang sebenarnya
y_true = test_dataset.classes

# membuat confusion matrix
cm = confusion_matrix(y_true, y_pred)

# menampilkan confusion matrix dengan heatmap
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g',
xticklabels=class_list, yticklabels=class_list)
plt.title('Confusion Matrix TESTING')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Laporan klasifikasi berupa metrik evaluasi seperti nilai akurasi, presisi, *recall*, dan f1-*score* dapat diperoleh dengan menjalankan *source codes* yang ada pada tabel 5.27.

Tabel 5.27 S*ource Codes Classification Report*

```
import numpy as np
from sklearn.metrics import classification_report

# Evaluasi model
y_true = test_dataset.classes
Y_pred = modelku.predict(test_dataset, verbose=1)
y_pred = np.argmax(Y_pred, axis=1)

# Tampilkan laporan klasifikasi
target_names = list(test_dataset.class_indices.keys())
print(classification_report(y_true, y_pred,
target_names=target_names))
```

Hasil dari menjalankan *source codes* dari tabel 5.27 di atas dapat dilihat pada tabel 5.28, 5.29, 5.30, dan 5.31.

- Ukuran 96 x 96 blobs.fit

Tabel 5.28 Hasil *Testing* 96 x 96 blobs.fit epoch 100

epoch 100			
Testing			
	Precision	Recall	F1-score
Grade A	0,91	0,91	0,91
Grade B	0,91	0,91	0,91
Grade C	1,00	1,00	1,00
Accuracy	0,94	0,94	0,94

- Ukuran 96 x 96 blobs.rect

Tabel 5.29 Hasil *Testing* 96 x 96 blobs.rect epoch 100

epoch 100			
Testing			
	Precision	Recall	F1-score
Grade A	0,94	1,00	0,97
Grade B	1,00	0,93	0,96
Grade C	1,00	1,00	1,00
Accuracy	0,98	0,98	0,98

- Ukuran 128 x 128 blobs.fit

Tabel 5.30 Hasil *Testing* 128 x 128 blobs.fit epoch 100

epoch 100			
Testing			
	Precision	Recall	F1-score
Grade A	0,92	1,00	0,96
Grade B	1,00	0,91	0,95
Grade C	1,00	1,00	1,00
Accuracy	0,97	0,97	0,97

- Ukuran 128 x 128 blobs.rect

Tabel 5.31 Hasil *Testing* 128 x 128 blobs.rect epoch 100

epoch 100			
Testing			
	Precision	Recall	F1-score
Grade A	0,97	1,00	0,98
Grade B	0,97	0,97	0,97
Grade C	1,00	0,97	0,98
Accuracy	0,98	0,98	0,98

5.5 Pengujian Dengan OpenMV Cam

Pengujian model dilakukan untuk mengetahui performa model yang telah *convert* menjadi berformat .tflite. Pengujian ini dilakukan dengan 2 metode, yaitu pengujian dengan gambar statis dan pengujian secara *real-time*. Pada tahap ini, pengujian dilakukan menggunakan OpenMV Cam.

5.5.1 Pengujian Gambar Statis

Pengujian ini menggunakan OpenMV Cam, dengan model yang berformat .tflite dan *data testing* yang berisikan 11 gambar pada setiap *grade* yang telah ditempatkan ke dalam direktori OpenMV Cam. Kemudian, model tersebut di-*running* menggunakan aplikasi OpenMV IDE dengan *data testing* sebagai bahan pengujian modelnya. Berikut tabel 5.32 yang berisikan *source codes* untuk melakukan pengujian menggunakan OpenMV IDE.

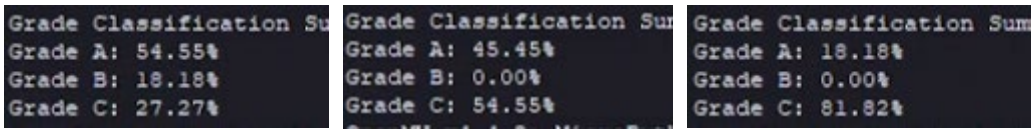
Tabel 5.32 *Source Codes* Pengujian Gambar Statis

```
# Untitled - By: Bedman - Mon Jun 26 2023

import image, tf, os

# Get the class labels
class_labels = ['Grade A', 'Grade B', 'Grade C']

# Inisialisasi model Keras
tfmodel = tf.load('titipan/tflite/FIT_coba3_BL00001_no-
dropout_size96.tflite', True)

image_dir = "titipan/fit/testing/gradeC"

#img = image.Image("image/testing rect/grade A/A (252).bmp")

# Initialize counters for each grade
grade_counts = [0, 0, 0]

# Loop melalui setiap gambar dalam direktori
for filename in os.listdir(image_dir):
    img = image.Image("titipan/fit/testing/gradeC/"+filename)
    for obj in tfmodel.classify(img):
        # Get the predicted class label and confidence
        predicted_class = obj.output()
        max_result_value = max(predicted_class)
        most_likely_idx =
predicted_class.index(max_result_value)

        # Get the predicted class label
        predicted_label = class_labels[most_likely_idx]

        # Get the confidence score
        confidence_score = max_result_value

        # Increment the counter for the predicted grade
        grade_counts[most_likely_idx] += 1

        # Print the predicted class label and confidence score
        print("--------------------------")
        print(filename)
        print("Predicted Label: %s" % predicted_label)
        print("Confidence Score: %.2f" % confidence_score)

# Calculate and print the percentage for each grade
total_count = sum(grade_counts)
print("\nGrade Classification Summary:")
for i, grade_count in enumerate(grade_counts):
    grade_name = class_labels[i]
    percentage = (grade_count / total_count) * 100
    print("%s: %.2f%%" % (grade_name, percentage))

        # Draw a rectangle where the blob was found
        img.draw_rectangle(blob.rect(), color=(0,255,0))
        # Draw a rectangle and a cross on the original image
        img.draw_cross(blob.cx(), blob.cy(), color=(0, 255, 0))

    # Calculate total number of fish seeds
```

```
    total_bibit_lele = count_grade_a + count_grade_b +
count_grade_c

    # Print results
    print("Grade A: %d" % count_grade_a)
    print("Grade B: %d" % count_grade_b)
    print("Grade C: %d" % count_grade_c)

    # Print total results
    print("Total Bibit Lele: %d" % total_bibit_lele)

    # Print FPS
    print(clock.fps())
```

Dengan menjalankan *source codes* dari tabel 5.32 di atas, maka didapatkan hasil pengujian yang dapat dilihat pada gambar 5.4, 5.5, 5.6, dan 5.7.

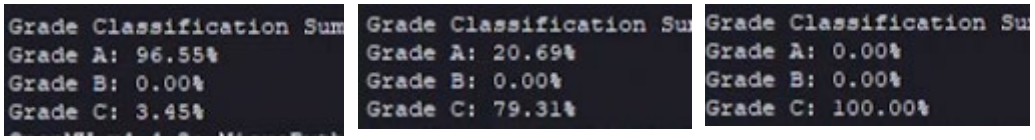
- Ukuran 96 x 96 blobs.fit



Grade A *Grade* B *Grade* C

Gambar 5.4 Hasil Pengujian Gambar Statis 96 x 96 blobs.fit

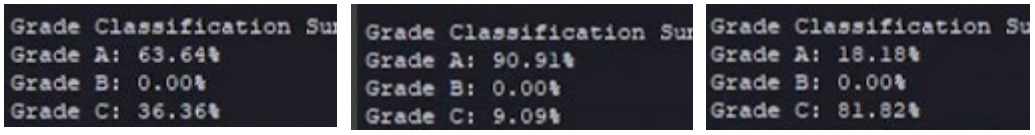
- Ukuran 96 x 96 blobs.rect



Grade A *Grade* B *Grade* C

Gambar 5.5 Hasil Pengujian Gambar Statis 96 x 96 blobs.rect

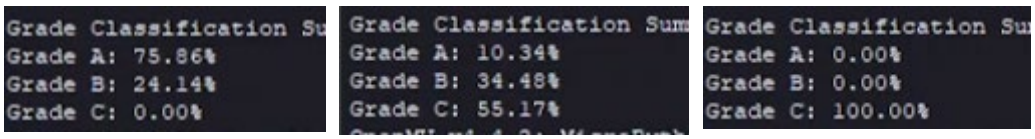
- Ukuran 128 x 128 blobs.fit



Grade A *Grade* B *Grade* C

Gambar 5.6 Hasil Pengujian Gambar Statis 128 x 128 blobs.fit

- Ukuran 128 x 128 blobs.rect



Grade A *Grade* B *Grade* C

Gambar 5.7 Hasil Pengujian Gambar Statis 128 x 128 blobs.rect

Dari hasil pengujian tersebut, disimpulkan bahwa bahan penelitian dengan ukuran 96 x 96 memiliki ketidakmampuan untuk mengidentifikasi *grade* B. *Grade* B tidak dapat diidentifikasi dengan format blobs.fit dan blobs.rect dan hanya terdeteksi sebesar 18,18% pada bahan uji *grade* A.

5.5.2 Pengujian Secara Real-Time

Pengujian ini menggunakan OpenMV Cam, dengan model yang berformat .tflite dan *data testing* yang berisikan 11 gambar pada setiap *grade* yang telah ditempatkan ke dalam direktori OpenMV Cam. Kemudian, model tersebut di-*running* menggunakan aplikasi OpenMV IDE dengan objek *real-time* yang ada di aquarium sebagai bahan pengujian modelnya. Berikut tabel 5.33 dan 5.34 yang berisikan *source codes* untuk melakukan pengujian blobs.fit dan blobs.rect menggunakan OpenMV IDE.

Tabel 5.33 *Source Codes* Pengujian *Real-Time* blobs.fit

```
import sensor
import image
import time
import tf
import pyb

# Inisialisasi kamera OpenMV
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_auto_gain(False) # must be turned off for color
tracking
sensor.set_auto_whitebal(False) # must be turned off for color
tracking
sensor.skip_frames(time=2000)  # Wait for sensor to adjust

# Inisialisasi model Keras
tfmodel = tf.load('titipan/tflite/FIT_coba3_BL00001_no-
dropout.tflite', True)

# Threshold yang digunakan untuk find_blobs
thresholds = [(29, 100, 127, -30, -45, 52)]

# Get the class labels
class_labels = ['Grade A', 'Grade B', 'Grade C']

# Inisialisasi variabel untuk perhitungan jumlah bibit lele
total_bibit_lele = 0

# Loop utama
clock = time.clock()
while (True):
    clock.tick()
```

```
    img = sensor.snapshot()
    # Find blobs
    blobs = img.find_blobs(thresholds, invert=True)

    count_grade_a = 0
    count_grade_b = 0
    count_grade_c = 0

    for blob in blobs:
        roi = blob.rect()
        roi_img = img.copy(roi=roi)
        # Draw a rectangle where the blob was found


        pyb.delay(1000) #delay untuk menangkap gambar
        for obj in tfmodel.classify(roi_img):
             # Get the predicted class label and confidence
             predicted_class = obj.output()
             max_result_value = max(predicted_class)
             most_likely_idx =
predicted_class.index(max_result_value)

             # Get the predicted class label
             predicted_label = class_labels[most_likely_idx]

             # Get the confidence score
             confidence_score = max_result_value

             # Update counts based on the predicted class
             if most_likely_idx == 0:
                if max_result_value >= 0.5:
                    count_grade_a =+ 1
             elif most_likely_idx == 1:
                if max_result_value >= 0.5:
                    count_grade_b =+ 1
             elif most_likely_idx == 2:
                if max_result_value >= 0.5:
                    count_grade_c =+ 1

             print("****************")
             # Print the predicted class label and confidence
score
             print("Predicted Label: %s" % predicted_label)
             print("Confidence Score: %.2f" % confidence_score)


        # Draw a rectangle where the blob was found
        img.draw_rectangle(blob.rect(), color=(0,255,0))
        # Draw a rectangle and a cross on the original image
        img.draw_cross(blob.cx(), blob.cy(), color=(0, 255, 0))
    # Calculate total number of fish seeds
    total_bibit_lele = count_grade_a + count_grade_b +
count_grade_c

    # Print results
    print("Grade A: %d" % count_grade_a)
    print("Grade B: %d" % count_grade_b)
    print("Grade C: %d" % count_grade_c)
```

```
    # Print total results
    print("Total Bibit Lele: %d" % total_bibit_lele)
    # Print FPS
    print(clock.fps())
```

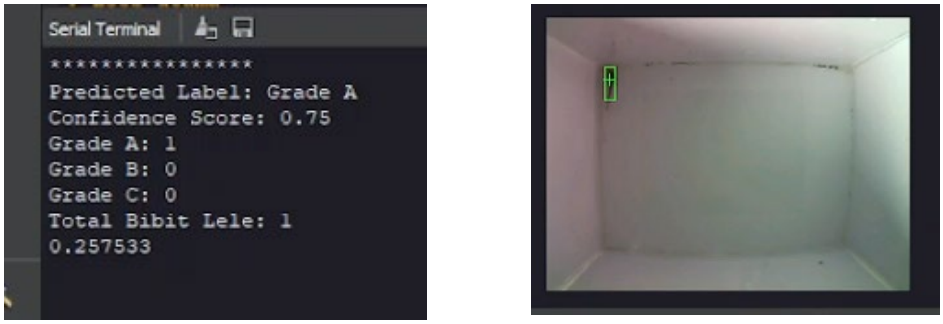
Tabel 5.34 *Source Codes* Pengujian *Real-Time* blobs.rect

```
import sensor
import image
import time
import tf
import pyb

# Inisialisasi kamera OpenMV
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_auto_gain(False) # must be turned off for color
tracking
sensor.set_auto_whitebal(False) # must be turned off for color
tracking
sensor.skip_frames(time=2000)  # Wait for sensor to adjust

# Inisialisasi model Keras
tfmodel = tf.load('titipan/tflite/RECT_coba2_BL00001_no-
dropout.tflite', True)

# Threshold yang digunakan untuk find_blobs
thresholds = [(35, 100, 127, -30, -45, 52)]

# Get the class labels
class_labels = ['Grade A', 'Grade B', 'Grade C']

# Inisialisasi variabel untuk perhitungan jumlah bibit lele
total_bibit_lele = 0

# Loop utama
clock = time.clock()
while (True):
    clock.tick()

    img = sensor.snapshot()
    # Find blobs
    blobs = img.find_blobs(thresholds, invert=True)

    count_grade_a = 0
    count_grade_b = 0
    count_grade_c = 0

    for blob in blobs:

        square_size = max(blob.w(), blob.h())

        # Calculate the top-left corner coordinates of the
square
        x = blob.cx() - square_size // 2
        y = blob.cy() - square_size // 2
```

```
        roi = [x,y,square_size, square_size]

        roi_img = img.copy(roi=roi)

        pyb.delay(1000) #delay untuk menangkap gambar
        for obj in tfmodel.classify(roi_img):
             # Get the predicted class label and confidence
             predicted_class = obj.output()
             max_result_value = max(predicted_class)
             most_likely_idx =
predicted_class.index(max_result_value)

             # Get the predicted class label
             predicted_label = class_labels[most_likely_idx]

             # Get the confidence score
             confidence_score = max_result_value

             # Update counts based on the predicted class
             if most_likely_idx == 0:
                if max_result_value >= 0.5:
                    count_grade_a =+ 1
             elif most_likely_idx == 1:
                if max_result_value >= 0.5:
                    count_grade_b =+ 1
             elif most_likely_idx == 2:
                if max_result_value >= 0.5:
                    count_grade_c =+ 1
             print("****************")
             # Print the predicted class label and confidence
score
             print("Predicted Label: %s" % predicted_label)
             print("Confidence Score: %.2f" % confidence_score)

        # Draw a rectangle and a cross on the original image
        img.draw_rectangle((x, y, square_size,square_size),
color=(0, 255, 0))
        img.draw_cross(blob.cx(), blob.cy(), color=(0, 255, 0))

    # Calculate total number of fish seeds
    total_bibit_lele = count_grade_a + count_grade_b +
count_grade_c

    # Print results
    print("Grade A: %d" % count_grade_a)
    print("Grade B: %d" % count_grade_b)
    print("Grade C: %d" % count_grade_c)

    # Print total results
    print("Total Bibit Lele: %d" % total_bibit_lele)

    # Print FPS
    print(clock.fps())
```

Dengan menjalankan *source codes* dari tabel 5.33 dan 5.34, maka didapatkan hasil pengujian yang dapat dilihat pada gambar 5.8, 5.9, 5.10 untuk bahan penelitian blobs.fit dan 5.11, 5.12, 5.13 untuk bahan penelitian blobs.rect.

- blobs.fit



Gambar 5.8 Hasil Pengujian *Real-Time* blobs.fit *Grade* A

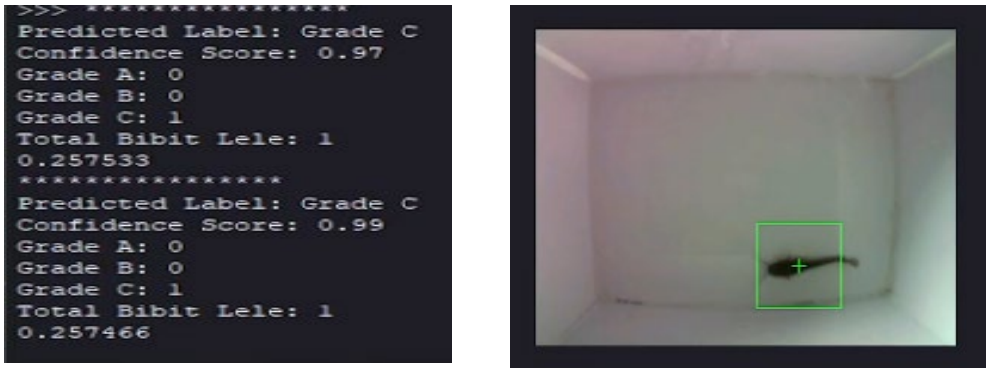
Gambar 5.9 Hasil Pengujian *Real-Time* blobs.fit *Grade* B

Gambar 5.10 Hasil Pengujian *Real-Time* blobs.fit *Grade* C

- blobs.rect

Gambar 5.11 Hasil Pengujian *Real-Time* blobs.rect *Grade* A

Gambar 5.12 Hasil Pengujian *Real-Time* blobs.rect *Grade* B



Gambar 5.13 Hasil Pengujian *Real-Time* blobs.rect *Grade* C

5.6 Confusion Matrix

Pada tahap ini, setelah melakukan pengujian gambar statis dan secara *real-time*, akan dilakukan evaluasi terhadap kinerja model dengan *confusion matrix*. *Confusion matrix* memungkinkan untuk melihat jumlah prediksi yang benar dan salah dalam bentuk matriks. Dengan menggunakan *confusion matrix,* bisa didapatkan representasi klasifikasi pada *confusion matrix*.

5.6.1 Confusion Matrix Pengujian Gambar Statis

Pada *confusion matrix* pengujian gambar statis, terdapat hasil dari 4 jenis bahan penelitian, yaitu:

- Ukuran 96 x 96 blobs.fit

Tabel 5.35 *Confusion Matrix* 96 x 96 blobs.fit

		Nilai Prediksi		
		A	B	C
Nilai Sebenarnya	A	6	2	3
	B	5	0	6
	C	2	0	9

Dari tabel 5.35, diketahui hasil dari pengujian gambar statis pada setiap *grade*. Untuk pengujian *grade* A, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* A sejumlah 6 gambar. Tetapi terdapat 5 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* B, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* B sejumlah 0. Terdapat 11 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* C, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* C sejumlah 9. Terdapat 2 gambar yang diprediksi bernilai salah. Untuk representasi klasifikasi dari *confusion matrix* tersebut, dapat dilihat pada tabel 5.36.

Tabel 5.36 Representasi Klasifikasi 96 x 96 blobs.fit

	TP	TN	FP	FN
Grade A	6	15	7	5
Grade B	0	20	2	11
Grade C	9	13	9	2

Dari tabel 5.36, diketahui *grade* A memiliki nilai TP sejumlah 6 gambar, TN sejumlah 15 gambar, FP sejumlah 7 gambar, dan FN sejumlah 5 gambar. *Grade* B memiliki nilai TP sejumlah 0 gambar, TN sejumlah 20 gambar, FP sejumlah 2 gambar, dan FN sejumlah 11 gambar. *Grade* C memiliki nilai TP sejumlah 9 gambar, TN sejumlah 13 gambar, FP sejumlah 9 gambar, dan FN sejumlah 2 gambar. Setelah diketahui representasi klasifikasi dari *confusion matrix*, selanjutnya adalah penghitungan nilai akurasi, *precision*, *recall*, dan f1-*score*.

- Akurasi

Grade A	*Grade* B	*Grade* C
$\frac{6+15}{6+15+7+5} = 0{,}64$	$\frac{0+20}{0+20+2+11} = 0{,}61$	$\frac{9+13}{9+13+9+2} = 0{,}67$

Gambar 5.14 Akurasi 96 x 96 blobs.fit

- Precision

Grade A	*Grade* B	*Grade* C
$\frac{6}{6+7} = 0{,}46$	$\frac{0}{0+2} = 0{,}00$	$\frac{9}{9+9} = 0{,}50$

Gambar 5.15 *Precision* 96 x 96 blobs.fit

- Recall

Grade A	*Grade* B	*Grade* C
$\frac{6}{6+5} = 0{,}55$	$\frac{0}{0+11} = 0{,}00$	$\frac{9}{9+2} = 0{,}82$

Gambar 5.16 *Recall* 96 x 96 blobs.fit

- F1-Score

Grade A	*Grade* B	*Grade* C
$\frac{2 \times 0{,}55 \times 0{,}46}{0{,}55+0{,}46} = 0{,}50$	$\frac{2 \times 0{,}00 \times 0{,}00}{0{,}00+0{,}00} = 0{,}00$	$\frac{2 \times 0{,}82 \times 0{,}50}{0{,}82+0{,}50} = 0{,}62$

Gambar 5.17 F1-*Score* 96 x 96 blobs.fit

- Average

Tabel 5.37 *Average* 96 x 96 blobs.fit

	Akurasi	Precision	Recall	F1-score
Grade A	0,64	0,46	0,55	0,50
Grade B	0,61	0,00	0,00	0,00
Grade C	0,67	0,50	0,82	0,62
Average	0,64	0,32	0,45	0,00

Berdasarkan data tabel 5.37, menunjukkan jika model masih mengalami kegagalan dalam mengidentifikasi nilai *grade* dari bibit ikan lele yang ditunjukkan dengan adanya nilai 0,00 f1-*score*. Rata-rata *precision* dan *recall* pun memiliki nilai kurang dari 50%.

- Ukuran 96 x 96 blobs.rect

Tabel 5.38 *Confusion Matrix* 96 x 96 blobs.rect

		Nilai Prediksi		
		A	B	C
Nilai Sebenarnya	A	10	0	1
	B	2	0	9
	C	0	0	11

Dari tabel tersebut, diketahui hasil dari pengujian gambar statis pada setiap *grade*. Untuk pengujian *grade* A, terdapat hasil prediksi yang memiliki nilai benar

terprediksi sebagai *grade* A sejumlah 10 gambar. Tetapi terdapat 1 gambar yang diprediksi bernilai salah.

Untuk pengujian *grade* B, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* B sejumlah 0. Terdapat 11 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* C, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* C sejumlah 11. Terdapat 0 gambar yang diprediksi bernilai salah. Untuk representasi klasifikasi dari *confusion matrix* tersebut, dapat dilihat pada tabel 5.39.

Tabel 5.39 Representasi Klasifikasi 96 x 96 blobs.rect

	TP	TN	FP	FN
A	10	20	2	1
B	0	22	0	11
C	11	12	10	0

Dari tabel 5.39, diketahui *grade* A memiliki nilai TP sejumlah 10 gambar, TN sejumlah 20 gambar, FP sejumlah 2 gambar, dan FN sejumlah 1 gambar. *Grade* B memiliki nilai TP sejumlah 0 gambar, TN sejumlah 22 gambar, FP sejumlah 0 gambar, dan FN sejumlah 11 gambar. *Grade* C memiliki nilai TP sejumlah 11 gambar, TN sejumlah 12 gambar, FP sejumlah 10 gambar, dan FN sejumlah 0 gambar. Setelah diketahui representasi klasifikasi dari *confusion matrix*, selanjutnya adalah penghitungan nilai akurasi, *precision*, *recall*, dan f1-*score*.

- Akurasi

Grade A	*Grade* B	*Grade* C
$\frac{10+20}{10+20+2+1} = 0{,}91$	$\frac{0+22}{0+22+0+11} = 0{,}67$	$\frac{11+12}{11+12+10+0} = 0{,}70$

Gambar 5.18 Akurasi 96 x 96 blobs.rect

- Precision

Grade A	*Grade* B	*Grade* C
$\frac{10}{10+2} = 0{,}83$	$\frac{0}{0+0} = 0{,}00$	$\frac{11}{11+10} = 0{,}52$

Gambar 5.19 *Precision* 96 x 96 blobs.rect

- Recall

Grade A	*Grade* B	*Grade* C
$\frac{10}{10+1} = 0{,}91$	$\frac{0}{0+11} = 0{,}00$	$\frac{11}{11+0} = 1{,}00$

Gambar 5.20 *Recall* 96 x 96 blobs.rect

- F1-Score

Grade A	*Grade* B	*Grade* C
$\frac{2 \times 0{,}91 \times 0{,}83}{0{,}91+0{,}83} = 0{,}87$	$\frac{2 \times 0{,}00 \times 0{,}00}{0{,}00+0{,}00} = 0{,}00$	$\frac{2 \times 1{,}00 \times 0{,}52}{1{,}00+0{,}52} = 0{,}69$

Gambar 5.21 F1-*Score* 96 x 96 blobs.rect

- Average

Tabel 5.40 *Average* 96 x 96 blobs.rect

	Akurasi	Precision	Recall	F1-score
Grade A	0,91	0,83	0,91	0,87
Grade B	0,67	0,00	0,00	0,00
Grade C	0,70	0,52	1,00	0,69
Average	0,76	0,00	0,64	0,00

Berdasarkan data tabel 5.40, menunjukkan jika model masih mengalami kegagalan dalam mengidentifikasi nilai *grade* dari bibit ikan lele yang ditunjukkan dengan adanya nilai 0,00 pada nilai *precision* dan f1-*score*.

- Ukuran 128 x 128 blobs.fit

Tabel 5.41 *Confusion Matrix* 128 x 128 blobs.fit

		Nilai Prediksi		
		A	B	C
Nilai Sebenarnya	A	7	0	4
	B	10	0	1
	C	2	0	9

Dari tabel 5.41, diketahui hasil dari pengujian gambar statis pada setiap *grade*. Untuk pengujian *grade* A, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* A sejumlah 7 gambar. Tetapi terdapat 5 gambar yang

diprediksi bernilai salah. Untuk pengujian *grade* B, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* B sejumlah 0. Terdapat 11 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* C, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* C sejumlah 9. Terdapat 2 gambar yang diprediksi bernilai salah. Untuk representasi klasifikasi dari *confusion matrix* tersebut, dapat dilihat pada tabel 5.42.

Tabel 5.42 Representasi Klasifikasi 128 x 128 blobs.fit

	TP	TN	FP	FN
Grade A	7	10	12	4
Grade B	0	22	0	11
Grade C	9	17	5	2

Dari tabel 5.42, diketahui *grade* A memiliki nilai TP sejumlah 7 gambar, TN sejumlah 10 gambar, FP sejumlah 12 gambar, dan FN sejumlah 4 gambar. *Grade* B memiliki nilai TP sejumlah 0 gambar, TN sejumlah 22 gambar, FP sejumlah 0 gambar, dan FN sejumlah 11 gambar. *Grade* C memiliki nilai TP sejumlah 9 gambar, TN sejumlah 17 gambar, FP sejumlah 5 gambar, dan FN sejumlah 2 gambar. Setelah diketahui representasi klasifikasi dari *confusion matrix*, selanjutnya adalah penghitungan nilai akurasi, *precision*, *recall*, dan f1-*score*.

- Akurasi

Grade A	*Grade* B	*Grade* C
$\frac{7+10}{7+10+12+4} = 0{,}52$	$\frac{0+22}{0+22+0+11} = 0{,}67$	$\frac{9+17}{9+17+5+2} = 0{,}79$

Gambar 5.22 Akurasi 128 x 128 blobs.fit

- Precision

Grade A	*Grade* B	*Grade* C
$\frac{7}{7+12} = 0{,}37$	$\frac{0}{0+0} = 0{,}00$	$\frac{9}{9+5} = 0{,}64$

Gambar 5.23 Precision 128 x 128 blobs.fit

- Recall

Grade A	*Grade* B	*Grade* C
$\frac{7}{7+4} = 0{,}64$	$\frac{0}{0+11} = 0{,}00$	$\frac{9}{9+2} = 0{,}82$

Gambar 5.24 *Recall* 128 x 128 blobs.fit

- F1-Score

Grade A	*Grade* B	*Grade* C
$\frac{2 \times 0{,}64 \times 0{,}37}{0{,}64+0{,}37} = 0{,}47$	$\frac{2 \times 0{,}00 \times 0{,}00}{0{,}00+0{,}00} = 0{,}00$	$\frac{2 \times 0{,}82 \times 0{,}64}{0{,}82+0{,}64} = 0{,}72$

Gambar 5.25 F1-*Score* 128 x 128 blobs.fit

- Average

Tabel 5.43 *Average* 128 x 128 blobs.fit

	Akurasi	Precision	Recall	F1-score
Grade A	0,52	0,37	0,64	0,47
Grade B	0,67	0,00	0,00	0,00
Grade C	0,79	0,64	0,82	0,72
Average	0,66	0,00	0,48	0,00

Berdasarkan data tabel 5.43, menunjukkan jika model masih mengalami kegagalan dalam mengidentifikasi nilai *grade* dari bibit ikan lele yang ditunjukkan dengan adanya nilai 0,00 pada nilai *precision* dan f1-*score*. Rata-rata dari *recall* pun memiliki nilai kurang dari 50%.

- Ukuran 128 x 128 blobs.rect

Tabel 5.44 *Confusion Matrix* 128 x 128 blobs.rect

		Nilai Prediksi		
		A	B	C
Nilai Sebenarnya	A	8	3	0
	B	1	4	6
	C	0	0	11

Dari tabel 5.44, diketahui hasil dari pengujian gambar statis pada setiap *grade*. Untuk pengujian *grade* A, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* A sejumlah 8 gambar. Tetapi terdapat 3 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* B, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* B sejumlah 4. Terdapat 7 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* C, terdapat hasil prediksi

yang memiliki nilai benar terprediksi sebagai *grade* C sejumlah 11. Terdapat 0 gambar yang diprediksi bernilai salah. Untuk representasi klasifikasi dari *confusion matrix* tersebut, dapat dilihat pada tabel 5.45.

Tabel 5.45 Representasi Klasifikasi 128 x 128 blobs.rect

	TP	TN	FP	FN
Grade A	8	21	1	3
Grade B	4	19	3	7
Grade C	11	16	6	0

Dari tabel tersebut, diketahui *grade* A memiliki nilai TP sejumlah 8 gambar, TN sejumlah 21 gambar, FP sejumlah 1 gambar, dan FN sejumlah 3 gambar. *Grade* B memiliki nilai TP sejumlah 4 gambar, TN sejumlah 19 gambar, FP sejumlah 3 gambar, dan FN sejumlah 7 gambar. *Grade* C memiliki nilai TP sejumlah 11 gambar, TN sejumlah 16 gambar, FP sejumlah 6 gambar, dan FN sejumlah 0 gambar. Setelah diketahui representasi klasifikasi dari *confusion matrix*, selanjutnya adalah penghitungan nilai akurasi, *precision*, *recall*, dan f1-*score*.

- Akurasi

Grade A	*Grade* B	*Grade* C
$\frac{8+21}{8+21+1+3} = 0{,}88$	$\frac{4+19}{4+19+3+7} = 0{,}70$	$\frac{11+16}{11+16+6+0} = 0{,}82$

Gambar 5.26 Akurasi 128 x 128 blobs.rect

- Precision

Grade A	*Grade* B	*Grade* C
$\frac{8}{8+1} = 0{,}89$	$\frac{4}{4+3} = 0{,}57$	$\frac{11}{11+6} = 0{,}65$

Gambar 5.27 *Precision* 128 x 128 blobs.rect

- Recall

Grade A	*Grade* B	*Grade* C
$\frac{8}{8+3} = 0{,}73$	$\frac{4+19}{4+7} = 0{,}36$	$\frac{11}{11+0} = 1{,}00$

Gambar 5.28 *Recall* 128 x 128 blobs.fit

- F1-Score

Grade A	*Grade* B	*Grade* C
$\frac{2 \times 0{,}73 \times 0{,}89}{0{,}73 + 0{,}89} = 0{,}80$	$\frac{2 \times 0{,}36 \times 0{,}57}{0{,}36 + 0{,}57} = 0{,}44$	$\frac{2 \times 1{,}00 \times 0{,}65}{1{,}00 + 0{,}65} = 0{,}79$

Gambar 5.29 F1-Score 128 x 128 blobs.rect

- Average

Tabel 5.46 Average 128 x 128 blobs.rect

	Akurasi	Precision	Recall	F1-score
Grade A	0,88	0,89	0,73	0,80
Grade B	0,70	0,57	0,36	0,44
Grade C	0,82	0,65	1,00	0,79
Average	0,80	0,70	0,70	0,68

Berdasarkan data tabel 5.46, menunjukkan jika model memiliki kinerja yang cukup baik dalam mengidentifikasi nilai *grade* dari bibit ikan lele. Dengan memiliki rata-rata nilai dari akurasi, *precision*, *recall*, f1-*score* lebih dari 50%.

5.6.2 Confusion Matrix Pengujian Secara Real-Time

Pada *confusion matrix* pengujian secara *real-time*, terdapat hasil dari 2 jenis bahan penelitian, yaitu:

- Ukuran 128 x 128 blobs.fit

Tabel 5.47 *Confusion Matrix Real-Time* 128 x 128 blobs.fit

		Nilai Prediksi		
		A	B	C
Nilai Sebenarnya	A	4	0	6
	B	0	0	10
	C	0	0	10

Dari tabel 5.47, diketahui hasil dari pengujian gambar statis pada setiap *grade*. Untuk pengujian *grade* A, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* A sejumlah 4 gambar. Tetapi terdapat 6 gambar yang

diprediksi bernilai salah. Untuk pengujian *grade* B, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* B sejumlah 0. Terdapat 10 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* C, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* C sejumlah 10. Terdapat 0 gambar yang diprediksi bernilai salah. Untuk representasi klasifikasi dari *confusion matrix* tersebut, dapat dilihat pada 5.48.

Tabel 5.48 Representasi Klasifikasi *Real-Time* 128 x 128 blobs.fit

	TP	TN	FP	FN
Grade A	4	20	0	6
Grade B	0	20	0	10
Grade C	10	4	16	0

Dari tabel 5.48, diketahui *grade* A memiliki nilai TP sejumlah 4 gambar, TN sejumlah 20 gambar, FP sejumlah 0 gambar, dan FN sejumlah 6 gambar. *Grade* B memiliki nilai TP sejumlah 0 gambar, TN sejumlah 20 gambar, FP sejumlah 0 gambar, dan FN sejumlah 10 gambar. *Grade* C memiliki nilai TP sejumlah 10 gambar, TN sejumlah 4 gambar, FP sejumlah 16 gambar, dan FN sejumlah 0 gambar. Setelah diketahui representasi klasifikasi dari *confusion matrix*, selanjutnya adalah penghitungan nilai akurasi, *precision*, *recall*, dan f1-*score*.

- Akurasi

Grade A	*Grade* B	*Grade* C
$\frac{4+20}{4+20+0+6} = 0{,}80$	$\frac{0+20}{0+20+0+10} = 0{,}67$	$\frac{10+4}{10+4+16+0} = 0{,}47$

Gambar 5.30 Akurasi *Real-Time* 128 x 128 blobs.fit

- Precision

Grade A	*Grade* B	*Grade* C
$\frac{4}{4+0} = 1{,}00$	$\frac{0}{0+0} = 0{,}00$	$\frac{10}{10+16} = 0{,}38$

Gambar 5.31 *Precision Real-Time* 128 x 128 blobs.fit

- Recall

Grade A	*Grade* B	*Grade* C
$\frac{4}{4+6} = 0{,}40$	$\frac{0}{0+10} = 0{,}00$	$\frac{10}{10+0} = 1{,}00$

Gambar 5.32 *Recall Real-Time* 128 x 128 blobs.fit

- F1-Score

Grade A	*Grade* B	*Grade* C
$\frac{2 \times 0{,}40 \times 1{,}00}{0{,}55+0{,}46} = 0{,}57$	$\frac{2 \times 0{,}00 \times 0{,}00}{0{,}00+0{,}00} = 0{,}00$	$\frac{2 \times 1{,}00 \times 0{,}38}{1{,}00+0{,}38} = 0{,}56$

Gambar 5.33 F1-Score *Real-Time* 128 x 128 blobs.fit

- Average

Tabel 5.49 Average *Real-Time* 128 x 128 blobs.fit

	Akurasi	Precision	Recall	F1-score
Grade A	0,80	1,00	0,40	0,57
Grade B	0,67	0,00	0,00	0,00
Grade C	0,47	0,38	1,00	0,56
Average	0,64	0,00	0,47	0,00

Berdasarkan data tabel 5.49, menunjukkan jika model masih mengalami kegagalan dalam mengidentifikasi nilai *grade* dari bibit ikan lele yang ditunjukkan dengan adanya nilai 0,00 pada *precision* dan f1-*score*. Rata- *recall* pun memiliki nilai kurang dari 50%.

- Ukuran 128 x 128 blobs.rect

Tabel 5.50 *Confusion Matrix Real-Time* 128 x 128 blobs.rect

		Nilai Prediksi		
		A	B	C
Nilai Sebenarnya	A	10	0	0
	B	3	5	2
	C	0	0	10

Dari tabel 5.50, diketahui hasil dari pengujian gambar statis pada setiap *grade*. Untuk pengujian *grade* A, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* A sejumlah 10 gambar. Tetapi terdapat 0 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* B, terdapat hasil prediksi yang

memiliki nilai benar terprediksi sebagai *grade* B sejumlah 5. Terdapat 5 gambar yang diprediksi bernilai salah. Untuk pengujian *grade* C, terdapat hasil prediksi yang memiliki nilai benar terprediksi sebagai *grade* C sejumlah 10. Terdapat 0 gambar yang diprediksi bernilai salah. Untuk representasi klasifikasi dari *confusion matrix* tersebut, dapat dilihat pada tabel 5.51.

Tabel 5.51 Representasi Klasifikasi *Real-Time* 128 x 128 blobs.rect

	TP	TN	FP	FN
Grade A	10	17	3	0
Grade B	5	20	0	5
Grade C	10	18	2	0

Dari tabel 5.51, *grade* A memiliki nilai TP sejumlah 10 gambar, TN sejumlah 17 gambar, FP sejumlah 3 gambar, dan FN sejumlah 0 gambar. *Grade* B memiliki nilai TP sejumlah 5 gambar, TN sejumlah 20 gambar, FP sejumlah 0 gambar, dan FN sejumlah 5 gambar. *Grade* C memiliki nilai TP sejumlah 10 gambar, TN sejumlah 18 gambar, FP sejumlah 2 gambar, dan FN sejumlah 0 gambar. Setelah diketahui representasi klasifikasi dari *confusion matrix*, selanjutnya adalah penghitungan nilai akurasi, *precision*, *recall*, dan f1-*score*.

- Akurasi

Grade A	*Grade* B	*Grade* C
$\frac{10+17}{10+17+3+0} = 0{,}90$	$\frac{5+20}{5+20+0+5} = 0{,}83$	$\frac{10+18}{10+18+2+0} = 0{,}93$

Gambar 5.34 Akurasi *Real-Time* 128 x 128 blobs.rect

- Precision

Grade A	*Grade* B	*Grade* C
$\frac{10}{10+3} = 0{,}77$	$\frac{5}{5+0} = 1{,}00$	$\frac{10}{10+2} = 0{,}83$

Gambar 5.35 *Precision Real-Time* 128 x 128 blobs.rect

- Recall

Grade A	*Grade* B	*Grade* C
$\frac{10}{10+0} = 1{,}00$	$\frac{5}{5+5} = 0{,}50$	$\frac{10}{10+0} = 1{,}00$

Gambar 5.36 *Recall Real-Time* 128 x 128 blobs.rect

- F1-Score

Grade A	*Grade* B	*Grade* C
$\frac{2 \times 1{,}00 \times 0{,}77}{1{,}00 + 0{,}77} = 0{,}87$	$\frac{2 \times 0{,}50 \times 1{,}00}{0{,}50 + 1{,}00} = 0{,}67$	$\frac{2 \times 1{,}00 \times 0{,}83}{1{,}00 + 0{,}83} = 0{,}91$

Gambar 5.37 F1-*Score Real-Time* 128 x 128 blobs.rect

- Average

Tabel 5.52 *Average Real-Time* 128 x 128 blobs.rect

	Akurasi	Precision	Recall	F1-score
Grade A	0,90	0,77	1,00	0,87
Grade B	0,83	1,00	0,50	0,67
Grade C	0,93	0,83	1,00	0,91
Average	0,89	0,87	0,83	0,82

Berdasarkan data tabel 5.52, menunjukkan jika model memiliki kinerja yang cukup baik dalam mengidentifikasi nilai *grade* dari bibit ikan lele. Dengan memiliki rata-rata nilai dari akurasi, *precision*, *recall*, f1-*score* lebih dari 50%.