LAMPIRAN

Lampiran 1 Dokumentasi Pengambilan Dataset

Lampiran 2 *Source Code* Pengambilan Dataset

- 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
```

- 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 = [(30, 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())
```

Lampiran 3 *Source Code* Persiapan Direktori

```
bahan_dir = os.path.join(base_dir)
train_dir = os.path.join(latih)
valid_dir = os.path.join(validasi)
test_dir = os.path.join(testing)

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/')
```

Lampiran 4 *Source Code* 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)))
```

Lampiran 5 *Source Code Pre-processing Image*

```
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
)

IMG_SIZE = (96,96)
BATCH_SIZE = 32

# train_dir = os.path.join(base_dir, 'latih')
# validation_dir = os.path.join(base_dir, 'validasi')

# train_dataset =
tf.keras.utils.image_dataset_from_directory(train_dir,
shuffle=True, batch_size=BATCH_SIZE, image_size=IMG_SIZE,
class_mode= 'categorical')
# validation_dataset =
tf.keras.utils.image_dataset_from_directory(validation_dir,
shuffle=True, batch_size=BATCH_SIZE, image_size=IMG_SIZE,
class_mode= 'categorical')

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'
)
```

Lampiran 6 *Source Code* Pembuatan Model

```
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)
# x = tf.keras.layers.Dropout(0.2)(x)
output = tf.keras.layers.Dense(3, activation='softmax')(x)
modelku = tf.keras.Model(inputs=MobileNetV2.input, outputs =
output)

modelku.summary()
```

Lampiran 7 *Source Code Training Model*

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

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

Lampiran 8 *Source Code* Pengujian Dengan *Data Testing*

```
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()

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))
```

Lampiran 9 *Source Code* Pengujian Dengan Gambar Statis

```
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))
```

Lampiran 10 *Source Code* Pengujian Secara *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())
```

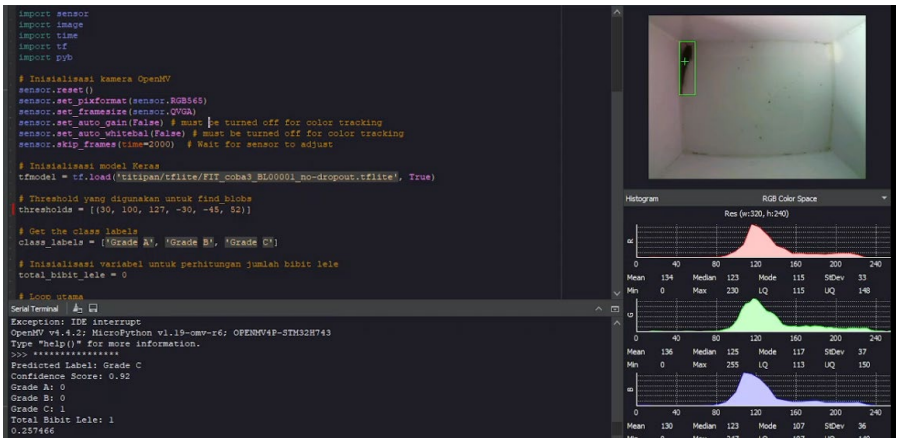
- Blobs.rect

```
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)
```

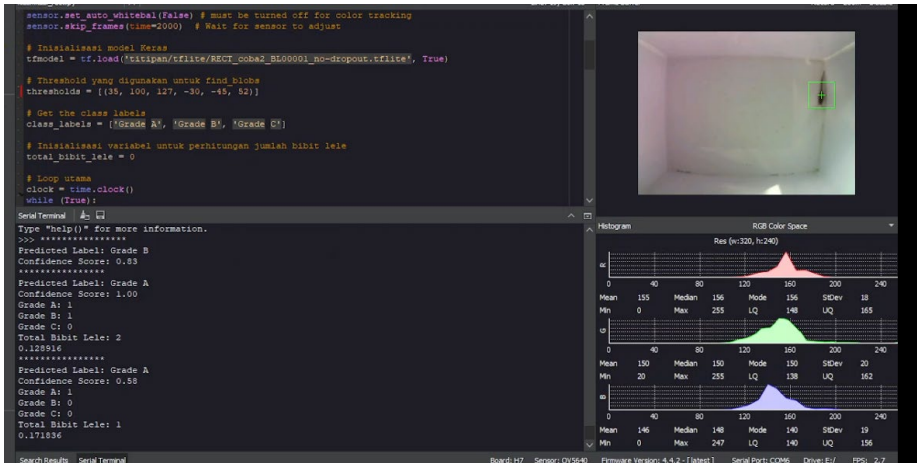
```
        pyb.delay(1000) #delay untuk menangkap gambar
        for obj in tfmodel.classify(roi_img):
             # Get the predicted class label and confidence
             predicted_class = obj.output()
             max_result_value = max(predicted_class)
             most_likely_idx =
predicted_class.index(max_result_value)

             # 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())
```

Lampiran 11 *Output* Pengujian Secara *Real-Time* di OpenMV IDE

- Blobs.fit



- Blobs.rect



Lampiran 12 Berita Acara Verifikasi Abstrak

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
POLITEKNIK NEGERI MALANG
JURUSAN TEKNOLOGI INFORMASI
PROGRAM STUDI TEKNIK INFORMATIKA
JL. Soekarno Hatta PO Box 04 Malang Telp. (0341) 404424 pes. 1122

BERITA ACARA VERIFIKASI ABSTRAK SKRIPSI PROGRAM STUDI D4-TI

Saya yang bertanda tangan di bawah ini:

NIM	:	1941720117
Nama	:	RONALDO FIRMANSYAH
ID Proposal	:	1200
Judul Skripsi	:	MENUJU GRADING BIBIT IKAN LELE OTOMATIS: KLASIFIKASI UKURAN BIBIT IKAN LELE MENGGUNAKAN CONVOLUTIONAL NEURAL NETWORK (CNN) DENGAN OPENMV CAM

Abstrak Final Bahasa Indonesia:

Penyortiran bibit ikan lele (grading) biasa dilakukan oleh para peternak bibit ikan lele. Tahap grading dilakukan untuk memisahkan bibit ikan lele berdasarkan ukurannya karena setiap bibit ikan lele dapat memiliki pertumbuhan yang berbeda dan mengurangi kanibalisme bibit ikan lele pada ukuran yang lebih kecil. Tujuan penelitian ini untuk dapat mengklasifikasikan bibit ikan lele menggunakan metode Convolutional Neural Network (CNN) dengan arsitektur MobileNetV2 dan OpenMV Cam H7+ secara otomatis. Terdapat 3 grade bibit ikan lele pada penelitian ini, antara lain: grade A (2-3 cm), grade B (4-5 cm) dan grade C (>6 cm). Dataset untuk proses training dan testing model diambil dengan 4 bahan penelitian yaitu ukuran gambar 96 x 96 blobs.fit, 96 x 96 blobs.rect, 128 x 128 blobs.fit, dan 128 x 128 blobs.rect. Dalam pengujian dengan gambar statis dan secara real-time, sistem dapat mengenali perbedaan grade dalam memprediksi grade bibit ikan lele yang dideteksi. Hasil paling optimal diantara 4 bahan penelitian adalah 128 x 128 blobs.rect yang memilki akurasi sebesar 89%. Penelitian ini dapat membantu para peternak bibit ikan lele dan memberikan pengetahuan mengenai penerapan metode CNN untuk pengembangan teknologi lebih lanjut di masa mendatang.

Kata Kunci:	**Ikan lele, bibit, klasifikasi, OpenMV, *deep learning*, CNN**

Abstrak Final Bahasa Inggris:

Sorting catfish seedlings (grading) are commonly done by catfish seedling farmers. The grading stage is done to separate catfish seedlings based on their size because each catfish seedling can have different growth and reduce cannibalism of catfish seedlings at a smaller size. The purpose of this research is to be able to classify catfish seedlings using the Convolutional Neural Network (CNN) method with MobileNetV2 architecture and OpenMV Cam H7+ automatically. There are 3 grades of catfish seedlings in this study, including: grade A (2-3 cm), grade B (4-5 cm) and grade C (>6 cm). Datasets for training and testing models are taken with 4 research materials, namely image sizes 96 x 96 blobs.fit, 96 x 96 blobs.rect, 128 x 128 blobs.fit, and 128 x 128 blobs.rect. In testing with static images and in real-time, the system can recognize grade differences in predicting the grade of catfish seedlings detected. The most optimal result among the 4 research materials is 128 x 128 blobs.rect which has an accuracy of 89%. This research is to help catfish farmers and provide knowledge about the application of CNN methods for further technology development in the future.

Keywords:	***Catfish, seedlings, classification, OpenMV, deep learning, CNN***

menyatakan bahwa: pada tanggal 28 bulan Agustus tahun 2023 abstrak dari judul skripsi saya sebagaimana tersebut di atas, telah benar-benar diperbaiki dan difinalisasi sesuai dengan supervisi serta arahan dari dosen pembimbing abstrak yang nama dan tanda tangannya tertera pada berita acara ini.

Yang membuat pernyataan,

(Ronaldo Firmansyah)
NIM. 1941720117

Dosen Pembimbing Abstrak,

(Faiz Ushbah Mubarok, S.Pd., M.Pd.)
NIP 199305052019031018