Lampiran 3 Souce Code Pembuatan Model YOLO

```
importimport locale
locale.getpreferredencoding = lambda: "UTF-8"

!pip install ultralytics

import warnings
warnings.filterwarnings('ignore')

import os
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
import cv2
import yaml
from PIL import Image
from collections import deque
from ultralytics import YOLO
from IPython.display import Video

# Load the pre-trained YOLOv8 nano segmentation model
model = YOLO('yolov8n.pt')

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

dataset = '/content/drive/MyDrive/Pothole_Dataset'


yaml_file_path = os.path.join(dataset, 'data.yaml')

with open(yaml_file_path, 'r') as file:
    yaml_content = yaml.load(file, Loader=yaml.FullLoader)
    print(yaml.dump(yaml_content, default_flow_style=False))

# Set paths for training and validation image sets
train_images_path = os.path.join(dataset, 'train', 'images')
valid_images_path = os.path.join(dataset, 'valid', 'images')

# Initialize counters for the number of images
num_train_images = 0
num_valid_images = 0

# Initialize sets to hold the unique sizes of images
train_image_sizes = set()
valid_image_sizes = set()

# Check train images sizes and count
for filename in os.listdir(train_images_path):
    if filename.endswith('.jpg'):
```

```
        num_train_images += 1
        image_path = os.path.join(train_images_path, filename)
        with Image.open(image_path) as img:
            train_image_sizes.add(img.size)

# Check validation images sizes and count
for filename in os.listdir(valid_images_path):
    if filename.endswith('.jpg'):
        num_valid_images += 1
        image_path = os.path.join(valid_images_path, filename)
        with Image.open(image_path) as img:
            valid_image_sizes.add(img.size)

# Print the results
print(f"Number of training images: {num_train_images}")
print(f"Number of validation images: {num_valid_images}")

# Check if all images in training set have the same size
if len(train_image_sizes) == 1:
    print(f"All training images have the same size:
{train_image_sizes.pop()}")
else:
    print("Training images have varying sizes.")

# Check if all images in validation set have the same size
if len(valid_image_sizes) == 1:
    print(f"All validation images have the same size:
{valid_image_sizes.pop()}")
else:
    print("Validation images have varying sizes.")

# Set the seed for the random number generator
random.seed(0)

# Create a list of image files
image_files = [f for f in os.listdir(train_images_path) if
f.endswith('.jpg')]

# Randomly select 15 images
random_images = random.sample(image_files, 15)

# Create a new figure
plt.figure(figsize=(12, 7))

# Loop through each image and display it in a 3x5 grid
for i, image_file in enumerate(random_images):
    image_path = os.path.join(train_images_path, image_file)
    image = Image.open(image_path)
    plt.subplot(3, 5, i + 1)
    plt.imshow(image)
    plt.axis('off')

# Add a suptitle
plt.suptitle('Random Selection of Dataset Images', fontsize=24)
```

```
# Show the plot
plt.tight_layout()
plt.show()

# Deleting unnecessary variable to free up memory
del image_files

results = model.train(
    data=yaml_file_path,     # Path to the dataset configuration
file
    epochs=10,              # Number of epochs to train for
    imgsz=640,               # Size of input images as integer
    patience=15,             # Epochs to wait for no observable
improvement for early stopping of training
    batch=16,                # Number of images per batch
    optimizer='Adam',        # Optimizer to use, choices=[SGD,
Adam, Adamax, AdamW, NAdam, RAdam, RMSProp, auto]
    lr0=0.001,               # Initial learning rate
    lrf=0.01,                # Final learning rate (lr0 * lrf)
    dropout=0.25,            # Use dropout regularization
    device=0,                # Device to run on, i.e. cuda device=0
    seed=40                 # Random seed for reproducibility
)

# Create the full file path by joining the directory path with the
filename
results_file_path = os.path.join(post_training_files_path,
'results.png')

# Read the image using cv2
image = cv2.imread(results_file_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image using matplotlib
plt.figure(figsize=(20, 8))
plt.imshow(image)
plt.title('Training and Validation Loss Trends', fontsize=24)
plt.axis('off')
plt.show()

results_csv_path = os.path.join(post_training_files_path,
'results.csv')

# Load the CSV file from the constructed path into a pandas
DataFrame
df = pd.read_csv(results_csv_path)

# Remove any leading whitespace from the column names
df.columns = df.columns.str.strip()

# Plot the learning curves for each loss
plot_learning_curve(df, 'train/box_loss', 'val/box_loss', 'Bounding
Box Loss Learning Curve')
```

```
plot_learning_curve(df, 'train/cls_loss', 'val/cls_loss',
'Classification Loss Learning Curve')
plot_learning_curve(df, 'train/dfl_loss', 'val/dfl_loss',
'Distribution Focal Loss Learning Curve')
plot_learning_curve(df, 'train/seg_loss', 'val/seg_loss',
'Segmentation Loss Learning Curve', ylim_range=[0,5])

# Construct the path to the confusion matrix images
confusion_matrix_path = os.path.join(post_training_files_path,
'confusion_matrix.png')
confusion_matrix_normalized_path =
os.path.join(post_training_files_path,
'confusion_matrix_normalized.png')

# Create a 1x2 subplot
fig, axs = plt.subplots(1, 2, figsize=(20, 10))

# Read and convert both images
cm_img = read_and_convert_image(confusion_matrix_path)
cm_norm_img =
read_and_convert_image(confusion_matrix_normalized_path)

# Display the images
axs[0].imshow(cm_img)
axs[0].set_title('Confusion Matrix', fontsize=24)
axs[0].axis('off')

axs[1].imshow(cm_norm_img)
axs[1].set_title('Normalized Confusion Matrix', fontsize=24)
axs[1].axis('off')

plt.tight_layout()
plt.show()

# Construct the path to the best model weights file using
os.path.join
best_model_path = os.path.join(post_training_files_path,
'weights/best.pt')

# Load the best model weights into the YOLO model
best_model = YOLO(best_model_path)

# Validate the best model using the validation set with default
parameters
metrics = best_model.val(split='val')

metrics_df = pd.DataFrame.from_dict(metrics.results_dict,
orient='index', columns=['Metric Value'])

# Display the DataFrame
metrics_df.round(3)

# Define the path to the validation images
valid_images_path = os.path.join(dataset, 'valid', 'images')
```

```
# List all jpg images in the directory
image_files = [file for file in os.listdir(valid_images_path) if
file.endswith('.jpg')]

# Select 9 images at equal intervals
num_images = len(image_files)
selected_images = [image_files[i] for i in range(0, num_images,
num_images // 9)]

# Initialize the subplot
fig, axes = plt.subplots(3, 3, figsize=(20, 21))
fig.suptitle('Validation Set Inferences', fontsize=24)

# Perform inference on each selected image and display it
for i, ax in enumerate(axes.flatten()):
    image_path = os.path.join(valid_images_path,
selected_images[i])
    results = best_model.predict(source=image_path, imgsz=640)
    annotated_image = results[0].plot()
    annotated_image_rgb = cv2.cvtColor(annotated_image,
cv2.COLOR_BGR2RGB)
    ax.imshow(annotated_image_rgb)
    ax.axis('off')

plt.tight_layout()
plt.show()

# Define the path to the sample video in the dataset
dataset_video_path =
'/content/drive/MyDrive/Pothole_Dataset/test_video1.mp4'

# Define the destination path in the working directory
video_path =
'/content/drive/MyDrive/Pothole_Dataset/output/test_video1.mp4'

# Copy the video file from its original location in the dataset to
the current working directory in Kaggle
shutil.copyfile(dataset_video_path, video_path)

# Initiate vehicle detection on the sample video using the best
performing model and save the output
best_model.predict(source=video_path, save=True)

import locale
locale.getpreferredencoding = lambda: "UTF-8"

# Convert the .avi video generated by the YOLOv8 prediction to .mp4
format for compatibility with notebook display
!ffmpeg -y -loglevel panic -i
/content/runs/detect/predict/test_video1.avi
processed_test_video1.mp4

# Embed and display the processed sample video within the notebook
```

```
Video("processed_test_video1.mp4", embed=True, width=960)

from ultralytics import YOLO

# Load model YOLOv8
model = YOLO('/content/runs/detect/train5/weights/best.pt')  #
Ganti dengan path ke model YOLOv8 Anda

# Export model ke TFLite
model.export(format='tflite', optimize=True)  # optimize=True jika
Anda ingin mengoptimalkan model
```

Lampiran 4 Source Code Deteksi Objek Pada Aplikasi

```
import 'dart:async';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter_application_1/controller/firestore.dart';
import
'package:flutter_application_1/module/map/model/pothole_model.dart'
;
import 'package:flutter_application_1/utils/boundingbox.dart';
import 'package:flutter_application_1/utils/popupmessage.dart';
import
'package:flutter_application_1/views/firebase_services.dart';
import 'package:flutter_vision/flutter_vision.dart';
import 'package:camera/camera.dart';
import 'package:latlong2/latlong.dart';
import 'package:location/location.dart';
import 'package:provider/provider.dart';
import 'package:sliding_up_panel/sliding_up_panel.dart';
import 'dart:io';

class DetectedObjectsProvider extends ChangeNotifier {
  List<dynamic> _detectedObjects = [];

  List<dynamic> get detectedObjects => _detectedObjects;

  void updateDetectedObjects(List<dynamic> newObjects) {
    _detectedObjects = newObjects;
    notifyListeners();
  }
}

class ObjectDetectionScreen extends StatefulWidget {
  @override
  _ObjectDetectionScreenState createState() =>
_ObjectDetectionScreenState();
}

class _ObjectDetectionScreenState extends
State<ObjectDetectionScreen> {
  FlutterVision vision = FlutterVision();
  CameraController? _cameraController;
  bool isDetecting = false;
  DetectedObjectsProvider _detectedObjectsProvider =
DetectedObjectsProvider();
  Location location = Location();
  LocationData? _locationData;
  PopUpMessage popUpMessage = PopUpMessage();
  File? imageFile;

  @override
  void initState() {
    super.initState();
```

```
    _initializeCamera();
    retrieveLocation();
    location.onLocationChanged.listen((LocationData
currentLocation) {
      _locationData = currentLocation;
      print('ini ini');
      print(_locationData);
    });
  }

  @override
  void dispose() {
    _cameraController?.dispose();
    vision.closeYoloModel();
    super.dispose();
  }

  Future<void> _initializeCamera() async {
    final cameras = await availableCameras();
    final firstCamera = cameras.first;

    _cameraController = CameraController(
      firstCamera,
      ResolutionPreset.high,
    );

    await _cameraController?.initialize();

    if (mounted) {
      setState(() {});
      _startObjectDetection();
    }
  }

  FirebaseServicesUpload firebaseServices =
FirebaseServicesUpload();

  Future<void> _startObjectDetection() async {
    await vision.loadYoloModel(
      labels: 'assets/yolov8.txt',
      modelPath: 'assets/newlite.tflite',
      modelVersion: 'yolov8',
      quantization: false,
      numThreads: 1,
      useGpu: false,
    );

    if (_cameraController != null) {
      _cameraController!.startImageStream((CameraImage image) async
{
        if (!isDetecting) {
          isDetecting = true;
          final result = await vision.yoloOnFrame(
```

```
            bytesList: image.planes.map((plane) =>
plane.bytes).toList(),
            imageHeight: image.height,
            imageWidth: image.width,
            iouThreshold: 0.01,
            confThreshold: 0.01,
            classThreshold: 0.01,
          );

          isDetecting = false;
          if (result.isNotEmpty) {
            print(result);

            _locationData = await location.getLocation();
            print(
                'Lokasi saat ini: Lat: ${_locationData!.latitude},
Long: ${_locationData!.longitude}');

            // Ambil gambar dari kamera
            final image = await _cameraController?.takePicture();
            if (image != null) {
              imageFile = File(image.path);
            }

            firebaseServices.uploadData(
                detail: 'Try',
                latitude: _locationData!.latitude!,
                longitude: _locationData!.longitude!,
                imageFile: imageFile!,
                title: 'Coba');
            print('ini woi');
            _detectedObjectsProvider.updateDetectedObjects(result);
          }
        }
      });
    }
  }

  Future<void> retrieveLocation() async {
    bool _serviceEnabled;
    PermissionStatus _permissionGranted;

    _serviceEnabled = await location.serviceEnabled();
    if (!_serviceEnabled) {
      _serviceEnabled = await location.requestService();
      if (!_serviceEnabled) {
        print('Location services are disabled.');
        return;
      }
    }

    _permissionGranted = await location.hasPermission();
    if (_permissionGranted == PermissionStatus.denied) {
      _permissionGranted = await location.requestPermission();
```

```
      if (_permissionGranted != PermissionStatus.granted) {
        print('Location permission denied.');
        return;
      }
    }

    _locationData = await location.getLocation();
    setState(() {
      _locationData = _locationData;
    });
  }

  SendDetails sendDeteils = SendDetails();
  location_uploader() {
    Timer.periodic(Duration(seconds: 5), (Timer timer) {
      print('Timer fired at ${DateTime.now()}');
      _detectedObjectsProvider.detectedObjects.isNotEmpty &&
              _detectedObjectsProvider.detectedObjects[0]['tag'] ==
'pothole'
          ? sendDeteils.sendDetails(context,
              LatLng(_locationData!.latitude!,
_locationData!.longitude!))
          : print('no pothole');
      print('apasi');
      _detectedObjectsProvider.detectedObjects.clear();
    });
  }

  @override
  Widget build(BuildContext context) {
    if (_cameraController == null ||
!_cameraController!.value.isInitialized) {
      return Container();
    }

    return Scaffold(
      floatingActionButton: FloatingActionButton(
        backgroundColor: Colors.purple.shade900,
        onPressed: () {
          location_uploader();
        },
        child: const Icon(Icons.upload),
      ),
      body: ChangeNotifierProvider(
        create: (_) => _detectedObjectsProvider,
        child: SlidingUpPanel(
          color: Color.fromARGB(255, 255, 255, 255),
          minHeight: 250,
          maxHeight: 700,
          borderRadius: const BorderRadius.only(
            topLeft: Radius.circular(30),
            topRight: Radius.circular(30),
          ),
          panel: Center(
```

```
            child: Consumer<DetectedObjectsProvider>(
              builder: (context, provider, _) {
                if (provider.detectedObjects.isEmpty) {
                  return const Text('No objects detected.');
                } else {
                  return ListView.builder(
                    itemCount: provider.detectedObjects.length,
                    itemBuilder: (BuildContext context, int index)
{
                      return Padding(
                        padding: const EdgeInsets.fromLTRB(10, 0,
10, 0),
                        child: Container(
                          decoration: BoxDecoration(
                            borderRadius:
BorderRadius.circular(10),
                            color:
                                '${provider.detectedObjects[index][
'tag']}' ==
                                        'pothole'
                                    ? Colors.red
                                    : Colors.purple.shade900,
                          ),
                          child: ListTile(
                            leading: Text(
                              'Lat:
${_locationData!.latitude?.toString()}\nLong:
${_locationData!.longitude?.toString()}',
                              style: const TextStyle(
                                color: Colors.white,
                                fontSize: 12,
                                fontWeight: FontWeight.bold,
                              ),
                            ),
                            selectedColor: Colors.black,
                            trailing: Text(
                              'Conf:
${double.parse(provider.detectedObjects[index]['box'][4].toStringAs
Fixed(3))}',
                              style: const TextStyle(
                                color: Colors.white,
                                fontSize: 15,
                                fontWeight: FontWeight.bold,
                              ),
                            ),
                            tileColor: Colors.black12,
                            title: Center(
                              child: Text(
                                '${provider.detectedObjects[index][
'tag']} detected'
                                    .toUpperCase(),
                                style: const TextStyle(
                                  color: Colors.black,
                                  fontSize: 16,
```

```
                                    fontWeight: FontWeight.bold,
                                  ),
                                ),
                              ),
                            ),
                          ),
                        );
                      },
                    );
                  }
                },
              ),
            ),
            body: Stack(
              children: [
                CameraPreview(_cameraController!),
                Consumer<DetectedObjectsProvider>(
                    builder: (context, provider, _) {
                  if (provider.detectedObjects.isNotEmpty) {
                    return CustomPaint(
                      size: Size(
                        _cameraController!.value.previewSize!.height,
                        _cameraController!.value.previewSize!.width,
                      ),
                      painter: BoundingBoxPainter(
                          provider.detectedObjects,
                          Size(
                            _cameraController!.value.previewSize!.wid
th
                                .toDouble(),
                            _cameraController!.value.previewSize!.hei
ght
                                .toDouble(),
                          )),
                    );
                  }
                  return const SizedBox.shrink();
                })
              ],
            ),
          ),
        ),
      );
  }
}
```

Lampiran 5 Source Code Konfigurasi Server Firebase

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_storage/firebase_storage.dart';
import
'package:flutter_application_1/module/map/model/pothole_model.dart'
;
import
'package:flutter_application_1/module/map/view/rerport_screen.dart'
;
import 'package:geolocator/geolocator.dart';
import 'package:geocoding/geocoding.dart';
import 'package:permission_handler/permission_handler.dart';
import 'dart:io';
import 'package:get/get.dart';

class FirebaseServicesUpload {
  final CollectionReference potHoleCollection =
      FirebaseFirestore.instance.collection('pothole');
  final FirebaseStorage storage = FirebaseStorage.instance;

  final controller = Get.find<MapController>();

  Future<String> getLocation() async {
    try {
      Position position = await Geolocator.getCurrentPosition(
          desiredAccuracy: LocationAccuracy.high);
      List<Placemark> placemarks =
          await placemarkFromCoordinates(position.latitude,
position.longitude);
      Placemark place = placemarks[0];

      final jalan = place.street ?? 'Unknown';
      final kota = place.locality ?? 'Unknown';
      print('Location: ${jalan}, $kota');
      return '$jalan, $kota';
    } catch (e) {
      print('Error: $e');
      return '';
    }
  }

  Future<String> requestLocationPermission() async {
    PermissionStatus status = await Permission.location.request();
    if (status.isGranted) {
      final data = getLocation();
      return data;
    } else {
      return '';
    }
  }

  Future<String> uploadImage(File image) async {
    try {
```

```
      String fileName =
          DateTime.now().millisecondsSinceEpoch.toString() +
'.jpg';
      Reference reference =
          storage.ref().child("pothole_images").child(fileName);
      UploadTask uploadTask =
          reference.putFile(image, SettableMetadata(contentType:
'image/jpeg'));
      TaskSnapshot snapshot = await uploadTask;
      String downloadUrl = await snapshot.ref.getDownloadURL();
      return downloadUrl;
    } catch (e) {
      print("Error uploading image: $e");
      return '';
    }
  }
Future<void> uploadData({
  required String detail,
  required double latitude,
  required double longitude,
  required File imageFile,
  required String title,
}) async {
  try {
    // Dapatkan lokasi saat ini
    final dataLoc = await requestLocationPermission();
    // Unggah gambar dan dapatkan URL
    final imageUrl = await uploadImage(imageFile);
    // Buat model data pothole
    final data = PotholeModel(
      id: '', // ID akan dihasilkan secara otomatis oleh Firestore
      details: dataLoc,
      location: GeoPoint(latitude, longitude),
      pict: imageUrl,
      title: 'Pothole', // Menggunakan judul yang diberikan
      createdAt: Timestamp.now(), // Tambahkan timestamp createdAt
    );

    // Tambahkan dokumen baru ke koleksi dengan ID yang dihasilkan
secara otomatis
    await potHoleCollection.add(data.toJson());
    await controller.getPotholeData();
    print('Data terupdate');
  } catch (e) {
    // Tangani kesalahan
    print(e.toString());
  }
}

  Future<String> _getNextId() async {
    QuerySnapshot snapshot = await potHoleCollection.get();
    int totalDocuments = snapshot.size;
    return (totalDocuments + 1).toString();
  }
```