LAMPIRAN

Data.yaml

```
train:
D:\LatihModel\images\dataX\train\image
s
val:
D:\LatihModel\images\dataX\valid\image
s
test:
D:\LatihModel\images\dataX\test\images
nc: 8
names: ['Naga Mentah','Naga
Matang','Pepaya Matang','Pepaya
Mentah','Semangka Mentah','Semangka
Matang','Tomat Matang','Tomat Mentah']
```

Image.py

```
from ultralytics import YOLO
import cv2

model =
YOLO('runs/detect/train/weights/best.pt')
results =
model("./images/dataX/train/images/21.j
pg", show=True)
cv2.waitKey(0)
```

TrainYolo.py

```
from ultralytics import YOLO

# Memuat model YOLOv8 pralatih dari
berkas 'yolov8n.pt'
model = YOLO("yolov8n.pt")

# Melatih model dengan dataset yang
ditentukan dalam 'data.yaml' selama 10
epoch dan ukuran gambar 640x640 piksel
# 'data.yaml' berisi path ke gambar
pelatihan dan validasi, jumlah kelas, dan
nama kelas
results = model.train(data="data.yaml",
epochs=1)
```

Yolo-Webcam.py

```
from ultralytics import YOLO
import cv2
import cvzone
import math

cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)

model = YOLO("best.pt")

classNames =
['nagamentah','nagamatang','pepayamata
ng','pepayamentah','semangkamentah','s
emangkamatang','tomatmatang','tomatm
entah']

while True:
   success, img = cap.read()
   results = model(img,stream=True)
   for r in results:
      boxes = r.boxes
      for box in boxes:

         # Bounding Box
         x1, y1, x2, y2 = box.xyxy[0]
         x1, y1, x2, y2 =
int(x1),int(y1),int(x2),int(y2)
         w, h = x2-x1, y2-y1
         cvzone.cornerRect(img,(x1,y1,w,h))
         # Confidence
         conf =
math.ceil((box.conf[0]*100))/100
         # Class Name
         cls = int(box.cls[0])

         cvzone.putTextRect(img,f'{classNa
mes[cls]} {conf}',(max(0,x1), max(35,y1)))


   cv2.imshow("Image",img)
   cv2.waitKey(1)
```

EksporModel.py

```
from ultralytics import YOLO

import cv2
```

```
model =
YOLO('runs/detect/train/weights/best.pt')
results =
model("./images/dataX/train/images/21.j
pg", show=True)
cv2.waitKey(0)
```

BoundingBox

```
package com.example.yolov8tflite

data class BoundingBox(
   val x1: Float,
   val y1: Float,
   val x2: Float,
   val y2: Float,
   val cx: Float,
   val cy: Float,
   val w: Float,
   val h: Float,
   val cnf: Float,
   val cls: Int,
   val clsName: String
)
```

constants

```
package com.example.yolov8tflite

object Constants {
   const val MODEL_PATH =
"best_float32.tflite"
   const val LABELS_PATH = "buah.txt"
}
```

Detector

```
package com.example.yolov8tflite

import android.content.Context
import android.graphics.Bitmap
import android.os.SystemClock
import org.tensorflow.lite.DataType
import org.tensorflow.lite.Interpreter
import
org.tensorflow.lite.gpu.CompatibilityList
import
org.tensorflow.lite.gpu.GpuDelegate
import
org.tensorflow.lite.support.common.FileU
til
import
org.tensorflow.lite.support.common.ops.C
astOp
import
org.tensorflow.lite.support.common.ops.
NormalizeOp
import
org.tensorflow.lite.support.image.ImagePr
ocessor
import
org.tensorflow.lite.support.image.TensorI
mage
import
org.tensorflow.lite.support.tensorbuffer.T
ensorBuffer
import java.io.BufferedReader
import java.io.IOException
import java.io.InputStream
import java.io.InputStreamReader

class Detector(
   private val context: Context,
   private val modelPath: String,
   private val labelPath: String,
   private val detectorListener:
DetectorListener,
) {

   private var interpreter: Interpreter
   private var labels =
mutableListOf<String>()

   private var tensorWidth = 0
   private var tensorHeight = 0
   private var numChannel = 0
   private var numElements = 0

   private val imageProcessor =
ImageProcessor.Builder()
      .add(NormalizeOp(INPUT_MEAN,
INPUT_STANDARD_DEVIATION))
```

```
        .add(CastOp(INPUT_IMAGE_TYPE))
        .build()

    init {
        val compatList = CompatibilityList()

        val options =
Interpreter.Options().apply{

if(compatList.isDelegateSupportedOnThis
Device){
                val delegateOptions =
compatList.bestOptionsForThisDevice

this.addDelegate(GpuDelegate(delegateO
ptions))
            } else {
                this.setNumThreads(4)
            }
        }

        val model =
FileUtil.loadMappedFile(context,
modelPath)
        interpreter = Interpreter(model,
options)

        val inputShape =
interpreter.getInputTensor(0)?.shape()
        val outputShape =
interpreter.getOutputTensor(0)?.shape()

        if (inputShape != null) {
            tensorWidth = inputShape[1]
            tensorHeight = inputShape[2]

            // If in case input shape is in format
of [1, 3, ..., ...]
            if (inputShape[1] == 3) {
                tensorWidth = inputShape[2]
                tensorHeight = inputShape[3]
            }
        }

        if (outputShape != null) {
            numChannel = outputShape[1]
            numElements = outputShape[2]
        }

        try {
            val inputStream: InputStream =
context.assets.open(labelPath)
            val reader =
BufferedReader(InputStreamReader(input
Stream))

            var line: String? = reader.readLine()
            while (line != null && line != "") {
                labels.add(line)
                line = reader.readLine()
            }

            reader.close()
            inputStream.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }

    fun restart(isGpu: Boolean) {
        interpreter.close()

        val options = if (isGpu) {
            val compatList = CompatibilityList()
            Interpreter.Options().apply{

if(compatList.isDelegateSupportedOnThis
Device){
                    val delegateOptions =
compatList.bestOptionsForThisDevice

this.addDelegate(GpuDelegate(delegateO
ptions))
                } else {
                    this.setNumThreads(4)
                }
            }
        } else {
            Interpreter.Options().apply{
                this.setNumThreads(4)
            }
        }

        val model =
```

```
FileUtil.loadMappedFile(context,
modelPath)
      interpreter = Interpreter(model,
options)
   }

   fun close() {
      interpreter.close()
   }

   fun detect(frame: Bitmap) {
      if (tensorWidth == 0) return
      if (tensorHeight == 0) return
      if (numChannel == 0) return
      if (numElements == 0) return

      var inferenceTime =
SystemClock.uptimeMillis()

      val resizedBitmap =
Bitmap.createScaledBitmap(frame,
tensorWidth, tensorHeight, false)

      val tensorImage =
TensorImage(INPUT_IMAGE_TYPE)
      tensorImage.load(resizedBitmap)
      val processedImage =
imageProcessor.process(tensorImage)
      val imageBuffer =
processedImage.buffer

      val output =
TensorBuffer.createFixedSize(intArrayOf(1
, numChannel, numElements),
OUTPUT_IMAGE_TYPE)
      interpreter.run(imageBuffer,
output.buffer)

      val bestBoxes =
bestBox(output.floatArray)
      inferenceTime =
SystemClock.uptimeMillis() -
inferenceTime

      if (bestBoxes == null) {
         detectorListener.onEmptyDetect()
         return
      }

      detectorListener.onDetect(bestBoxes,
inferenceTime)
   }

   private fun bestBox(array: FloatArray) :
List<BoundingBox>? {

      val boundingBoxes =
mutableListOf<BoundingBox>()

      for (c in 0 until numElements) {
         var maxConf =
CONFIDENCE_THRESHOLD
         var maxIdx = -1
         var j = 4
         var arrayIdx = c + numElements * j
         while (j < numChannel){
            if (array[arrayIdx] > maxConf) {
               maxConf = array[arrayIdx]
               maxIdx = j - 4
            }
            j++
            arrayIdx += numElements
         }

         if (maxConf >
CONFIDENCE_THRESHOLD) {
            val clsName = labels[maxIdx]
            val cx = array[c] // 0
            val cy = array[c + numElements]
// 1
            val w = array[c + numElements *
2]
            val h = array[c + numElements *
3]
            val x1 = cx - (w/2F)
            val y1 = cy - (h/2F)
            val x2 = cx + (w/2F)
            val y2 = cy + (h/2F)
            if (x1 < 0F || x1 > 1F) continue
            if (y1 < 0F || y1 > 1F) continue
            if (x2 < 0F || x2 > 1F) continue
            if (y2 < 0F || y2 > 1F) continue

            boundingBoxes.add(
```

```
                BoundingBox(
                    x1 = x1, y1 = y1, x2 = x2, y2 =
y2,
                    cx = cx, cy = cy, w = w, h = h,
                    cnf = maxConf, cls = maxIdx,
clsName = clsName
                )
            )
        }
    }

    if (boundingBoxes.isEmpty()) return
null

    return applyNMS(boundingBoxes)
  }

  private fun applyNMS(boxes:
List<BoundingBox>) :
MutableList<BoundingBox> {
    val sortedBoxes =
boxes.sortedByDescending { it.cnf
}.toMutableList()
    val selectedBoxes =
mutableListOf<BoundingBox>()

    while(sortedBoxes.isNotEmpty()) {
      val first = sortedBoxes.first()
      selectedBoxes.add(first)
      sortedBoxes.remove(first)

      val iterator = sortedBoxes.iterator()
      while (iterator.hasNext()) {
        val nextBox = iterator.next()
        val iou = calculateIoU(first,
nextBox)
        if (iou >= IOU_THRESHOLD) {
          iterator.remove()
        }
      }
    }

    return selectedBoxes
  }

  private fun calculateIoU(box1:
BoundingBox, box2: BoundingBox): Float {
    val x1 = maxOf(box1.x1, box2.x1)
    val y1 = maxOf(box1.y1, box2.y1)
    val x2 = minOf(box1.x2, box2.x2)
    val y2 = minOf(box1.y2, box2.y2)
    val intersectionArea = maxOf(0F, x2 -
x1) * maxOf(0F, y2 - y1)
    val box1Area = box1.w * box1.h
    val box2Area = box2.w * box2.h
    return intersectionArea / (box1Area +
box2Area - intersectionArea)
  }

  interface DetectorListener {
    fun onEmptyDetect()
    fun onDetect(boundingBoxes:
List<BoundingBox>, inferenceTime: Long)
  }

  companion object {
    private const val INPUT_MEAN = 0f
    private const val
INPUT_STANDARD_DEVIATION = 255f
    private val INPUT_IMAGE_TYPE =
DataType.FLOAT32
    private val OUTPUT_IMAGE_TYPE =
DataType.FLOAT32
    private const val
CONFIDENCE_THRESHOLD = 0.3F
    private const val IOU_THRESHOLD =
0.5F
  }
}
```

MainActivity

```
package com.example.yolov8tflite

import android.Manifest
import
android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.Matrix
import android.os.Bundle
import android.util.Log
import
androidx.activity.result.contract.ActivityR
esultContracts
```

```
import
androidx.appcompat.app.AppCompatActi
vity
import androidx.camera.core.AspectRatio
import androidx.camera.core.Camera
import
androidx.camera.core.CameraSelector
import
androidx.camera.core.ImageAnalysis
import androidx.camera.core.Preview
import
androidx.camera.lifecycle.ProcessCamera
Provider
import androidx.core.app.ActivityCompat
import
androidx.core.content.ContextCompat
import
com.example.deteksibuah.databinding.Ac
tivityMainBinding
import
com.example.yolov8tflite.Constants.LABE
LS_PATH
import
com.example.yolov8tflite.Constants.MOD
EL_PATH
import
java.util.concurrent.ExecutorService
import java.util.concurrent.Executors

class MainActivity : AppCompatActivity(),
Detector.DetectorListener {
    private lateinit var binding:
ActivityMainBinding
    private var isFrontCamera = false

    private var preview: Preview? = null
    private var imageAnalyzer:
ImageAnalysis? = null
    private var camera: Camera? = null
    private var cameraProvider:
ProcessCameraProvider? = null
    private var detector: Detector? = null

    private lateinit var cameraExecutor:
ExecutorService

    override fun
onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding =
ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        cameraExecutor =
Executors.newSingleThreadExecutor()

        cameraExecutor.execute {
            detector = Detector(baseContext,
MODEL_PATH, LABELS_PATH, this)
        }

        if (allPermissionsGranted()) {
            startCamera()
        } else {

ActivityCompat.requestPermissions(this,
REQUIRED_PERMISSIONS,
REQUEST_CODE_PERMISSIONS)
        }

    }

    private fun startCamera() {
        val cameraProviderFuture =
ProcessCameraProvider.getInstance(this)
        cameraProviderFuture.addListener({
            cameraProvider =
cameraProviderFuture.get()
            bindCameraUseCases()
        },
ContextCompat.getMainExecutor(this))
    }

    private fun bindCameraUseCases() {
        val cameraProvider =
            cameraProvider ?: throw
IllegalStateException("Camera
initialization failed.")

        val rotation =
binding.viewFinder.display.rotation

        val cameraSelector = CameraSelector
            .Builder()
```

```
        .requireLensFacing(if
(isFrontCamera)
CameraSelector.LENS_FACING_FRONT
else CameraSelector.LENS_FACING_BACK)
        .build()

      preview = Preview.Builder()

.setTargetAspectRatio(AspectRatio.RATIO
_4_3)
        .setTargetRotation(rotation)
        .build()

      imageAnalyzer =
ImageAnalysis.Builder()

.setTargetAspectRatio(AspectRatio.RATIO
_4_3)

.setBackpressureStrategy(ImageAnalysis.S
TRATEGY_KEEP_ONLY_LATEST)

.setTargetRotation(binding.viewFinder.dis
play.rotation)

.setOutputImageFormat(ImageAnalysis.O
UTPUT_IMAGE_FORMAT_RGBA_8888)
        .build()


imageAnalyzer?.setAnalyzer(cameraExecu
tor) { imageProxy ->
        val bitmapBuffer =
          Bitmap.createBitmap(
            imageProxy.width,
            imageProxy.height,
            Bitmap.Config.ARGB_8888
          )
        imageProxy.use {
bitmapBuffer.copyPixelsFromBuffer(imag
eProxy.planes[0].buffer) }
        imageProxy.close()

        val matrix = Matrix().apply {

postRotate(imageProxy.imageInfo.rotatio
nDegrees.toFloat())

          if (isFrontCamera) {
            postScale(
              -1f,
              1f,
              imageProxy.width.toFloat(),
              imageProxy.height.toFloat()
            )
          }
        }

        val rotatedBitmap =
Bitmap.createBitmap(
          bitmapBuffer, 0, 0,
bitmapBuffer.width, bitmapBuffer.height,
          matrix, true
        )

        detector?.detect(rotatedBitmap)
      }

      cameraProvider.unbindAll()

      try {
        camera =
cameraProvider.bindToLifecycle(
          this,
          cameraSelector,
          preview,
          imageAnalyzer
        )


preview?.setSurfaceProvider(binding.view
Finder.surfaceProvider)
      } catch (exc: Exception) {
        Log.e(TAG, "Use case binding
failed", exc)
      }
    }

    private fun allPermissionsGranted() =
REQUIRED_PERMISSIONS.all {

ContextCompat.checkSelfPermission(base
Context, it) ==
PackageManager.PERMISSION_GRANTED
```

```
    }

    private val requestPermissionLauncher
= registerForActivityResult(

ActivityResultContracts.RequestMultipleP
ermissions()
    ) {
        if (it[Manifest.permission.CAMERA]
== true) {
            startCamera()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        detector?.close()
        cameraExecutor.shutdown()
    }

    override fun onResume() {
        super.onResume()
        if (allPermissionsGranted()) {
            startCamera()
        } else {

requestPermissionLauncher.launch(REQUI
RED_PERMISSIONS)
        }
    }

    companion object {
        private const val TAG = "Camera"
        private const val
REQUEST_CODE_PERMISSIONS = 10
        private val REQUIRED_PERMISSIONS
= mutableListOf(
            Manifest.permission.CAMERA
        ).toTypedArray()
    }

    override fun onEmptyDetect() {
        runOnUiThread {
            binding.overlay.clear()
        }
    }

    override fun onDetect(boundingBoxes:
List<BoundingBox>, inferenceTime: Long)
{
        runOnUiThread {
            binding.overlay.apply {
                setResults(boundingBoxes)
                invalidate()
            }
        }
    }
}
```

OverlayView

```
package com.example.yolov8tflite

import android.content.Context
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.graphics.Rect
import android.util.AttributeSet
import android.view.View
import
androidx.core.content.ContextCompat
import com.example.deteksibuah.R

class OverlayView(context: Context?,
attrs: AttributeSet?) : View(context, attrs)
{

    private var results =
listOf<BoundingBox>()
    private var boxPaint = Paint()
    private var textBackgroundPaint =
Paint()
    private var textPaint = Paint()

    private var bounds = Rect()

    init {
        initPaints()
    }

    fun clear() {
        results = listOf()
        textPaint.reset()
```

```
        textBackgroundPaint.reset()
        boxPaint.reset()
        invalidate()
        initPaints()
    }

    private fun initPaints() {
        textBackgroundPaint.color =
Color.BLACK
        textBackgroundPaint.style =
Paint.Style.FILL
        textBackgroundPaint.textSize = 50f

        textPaint.color = Color.WHITE
        textPaint.style = Paint.Style.FILL
        textPaint.textSize = 50f

        boxPaint.color =
ContextCompat.getColor(context!!,
R.color.bounding_box_color)
        boxPaint.strokeWidth = 8F
        boxPaint.style = Paint.Style.STROKE
    }

    override fun draw(canvas: Canvas) {
        super.draw(canvas)

        results.forEach {
            val left = it.x1 * width
            val top = it.y1 * height
            val right = it.x2 * width
            val bottom = it.y2 * height

            canvas.drawRect(left, top, right,
bottom, boxPaint)
            val drawableText = it.clsName


textBackgroundPaint.getTextBounds(draw
ableText, 0, drawableText.length, bounds)
            val textWidth = bounds.width()
            val textHeight = bounds.height()
            canvas.drawRect(
                left,
                top,
                left + textWidth +
BOUNDING_RECT_TEXT_PADDING,
                top + textHeight +
BOUNDING_RECT_TEXT_PADDING,
                textBackgroundPaint
            )
            canvas.drawText(drawableText,
left, top + bounds.height(), textPaint)

        }
    }

    fun setResults(boundingBoxes:
List<BoundingBox>) {
        results = boundingBoxes
        invalidate()
    }

    companion object {
        private const val
BOUNDING_RECT_TEXT_PADDING = 8
    }
}
```

SplashActivity

```
package com.example.yolov8tflite

import android.annotation.SuppressLint
import android.content.Intent
import android.os.Bundle
import android.os.Handler
import
androidx.activity.enableEdgeToEdge
import
androidx.appcompat.app.AppCompatActi
vity
import com.example.deteksibuah.R

@SuppressLint("CustomSplashScreen")
class SplashActivity : AppCompatActivity()
{
    override fun
onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

setContentView(R.layout.activity_splash)
        Handler(mainLooper).postDelayed({
```

```
        startActivity(Intent(this,
MainActivity::class.java))
      }, 3000)
   }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.Constr
aintLayout
xmlns:android="http://schemas.android.c
om/apk/res/android"

xmlns:app="http://schemas.android.com/
apk/res-auto"

xmlns:tools="http://schemas.android.com
/tools"
   android:id="@+id/camera_container"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@color/black">

   <androidx.camera.view.PreviewView
      android:id="@+id/view_finder"
      android:layout_width="0dp"
      android:layout_height="0dp"

app:layout_constraintBottom_toBottomO
f="parent"

app:layout_constraintEnd_toEndOf="pare
nt"

app:layout_constraintStart_toStartOf="pa
rent"

app:layout_constraintTop_toTopOf="pare
nt"
      app:scaleType="fillStart" />

   <com.example.yolov8tflite.OverlayView
      android:id="@+id/overlay"
      android:layout_width="0dp"
      android:layout_height="0dp"
      android:translationZ="5dp"

app:layout_constraintBottom_toBottomO
f="parent"

app:layout_constraintEnd_toEndOf="pare
nt"

app:layout_constraintStart_toStartOf="pa
rent"

app:layout_constraintTop_toTopOf="pare
nt" />

</androidx.constraintlayout.widget.Const
raintLayout>
```

Activity_splash.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.Constr
aintLayout
xmlns:android="http://schemas.android.c
om/apk/res/android"

xmlns:app="http://schemas.android.com/
apk/res-auto"

xmlns:tools="http://schemas.android.com
/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"

android:background="?attr/colorPrimaryC
ontainer">

   <ImageView
      android:id="@+id/iv_logo"
      android:layout_width="100dp"
      android:layout_height="100dp"

android:src="@drawable/deteksi_buah_lo
go"

app:layout_constraintBottom_toBottomO
f="parent"

app:layout_constraintEnd_toEndOf="pare
```

```
nt"

app:layout_constraintStart_toStartOf="pa
rent"

app:layout_constraintTop_toTopOf="pare
nt"
      />


</androidx.constraintlayout.widget.Const
raintLayout>
```