// ---- Kolom inisiasi TensorFlow Lite Interpreter ----

void init() {

try {

Interpreter.Options opt = new Interpreter.Options();

tflite = new Interpreter(loadmodelfile(context), opt);

} catch (Exception e) {

e.printStackTrace();

}

}

// ----------------------------------------------------

// ---- Kolom preprocessing gambar ----

private TensorImage loadImage(final Bitmap bitmap) {

// Loads bitmap into a TensorImage.

inputImageBuffer.load(bitmap);

// Creates processor for the TensorImage.

int cropSize = Math.*min*(bitmap.getWidth(), bitmap.getHeight());

// *TODO(b/143564309): Fuse ops inside ImageProcessor.*

ImageProcessor imageProcessor =

new ImageProcessor.Builder()

.add(new ResizeWithCropOrPadOp(cropSize, cropSize))

.add(new ResizeOp(imageSizeX, imageSizeY, ResizeOp.ResizeMethod.*NEAREST\_NEIGHBOR*))

.add(getPreprocessNormalizeOp())

.build();

return imageProcessor.process(inputImageBuffer);

}

// ----------------------------------------------------

// ---- Kolom pemanggilan model tflite ----

private MappedByteBuffer loadmodelfile(Activity activity) throws IOException {

String MODEL\_NAME = "vegs.tflite";

AssetFileDescriptor fileDescriptor = activity.getAssets().openFd(MODEL\_NAME);

FileInputStream inputStream = new FileInputStream(fileDescriptor.getFileDescriptor());

FileChannel fileChannel = inputStream.getChannel();

long startoffset = fileDescriptor.getStartOffset();

long declaredLength = fileDescriptor.getDeclaredLength();

return fileChannel.map(FileChannel.MapMode.*READ\_ONLY*, startoffset, declaredLength);

}

void classifyImage(Bitmap bitmap){

int imageTensorIndex = 0;

int[] imageShape = tflite.getInputTensor(imageTensorIndex).shape(); // {1, height, width, 3}

imageSizeY = imageShape[1];

imageSizeX = imageShape[2];

DataType imageDataType = tflite.getInputTensor(imageTensorIndex).dataType();

int probabilityTensorIndex = 0;

int[] probabilityShape =

tflite.getOutputTensor(probabilityTensorIndex).shape(); // {1, NUM\_CLASSES}

DataType probabilityDataType = tflite.getOutputTensor(probabilityTensorIndex).dataType();

inputImageBuffer = new TensorImage(imageDataType);

outputProbabilityBuffer = TensorBuffer.*createFixedSize*(probabilityShape, probabilityDataType);

probabilityProcessor = new TensorProcessor.Builder().add(getPostprocessNormalizeOp()).build();

inputImageBuffer = loadImage(bitmap);

tflite.run(inputImageBuffer.getBuffer(),outputProbabilityBuffer.getBuffer().rewind());

}

private TensorOperator getPreprocessNormalizeOp() {

return new NormalizeOp(*IMAGE\_MEAN*, *IMAGE\_STD*);

}

// ----------------------------------------------------

// ---- Kolom postprocessing ----

public String showresult() {

try {

labels = FileUtil.*loadLabels*(context, "vegs.txt");

} catch (Exception e) {

e.printStackTrace();

return null;

}

Map<String, Float> labeledProbability =

new TensorLabel(labels, probabilityProcessor.process(outputProbabilityBuffer))

.getMapWithFloatValue();

float maxValueInMap = (Collections.*max*(labeledProbability.values()));

String result = null;

for (Map.Entry<String, Float> entry : labeledProbability.entrySet()) {

if (entry.getValue() == maxValueInMap) {

result = entry.getKey();

}

}

return result;

}

private TensorOperator getPostprocessNormalizeOp() {

return new NormalizeOp(*PROBABILITY\_MEAN*, *PROBABILITY\_STD*);

}

// ----------------------------------------------------