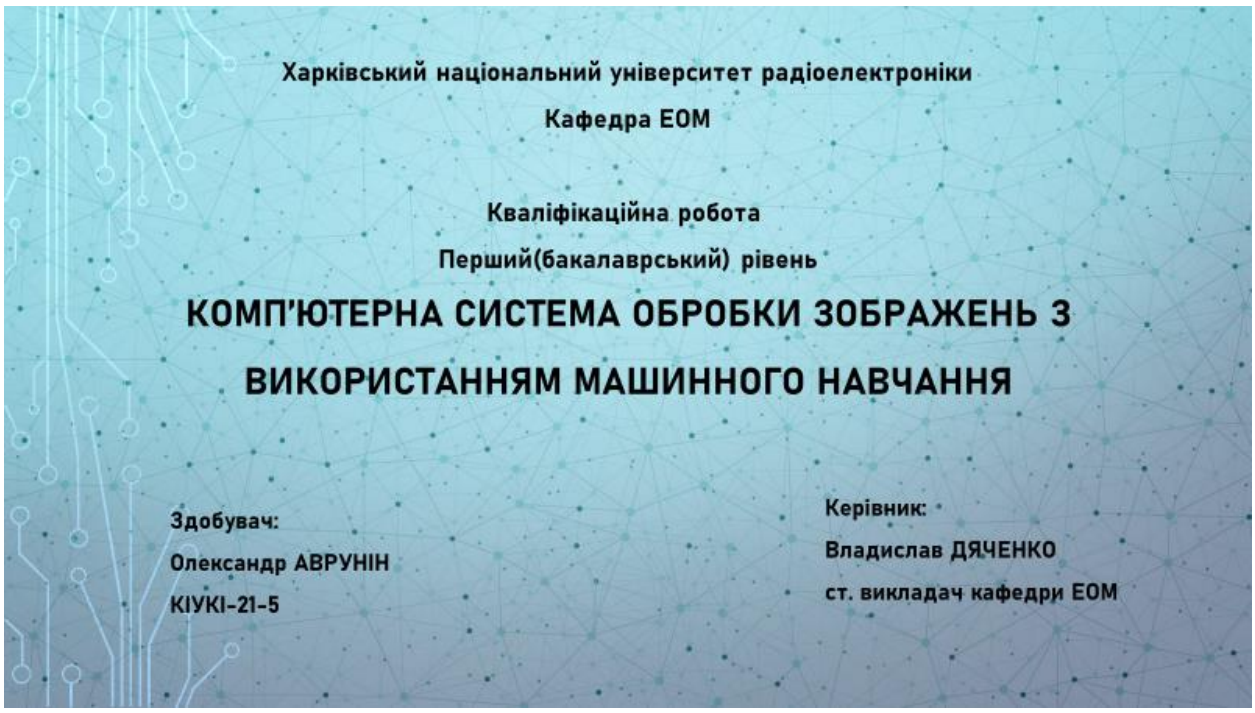
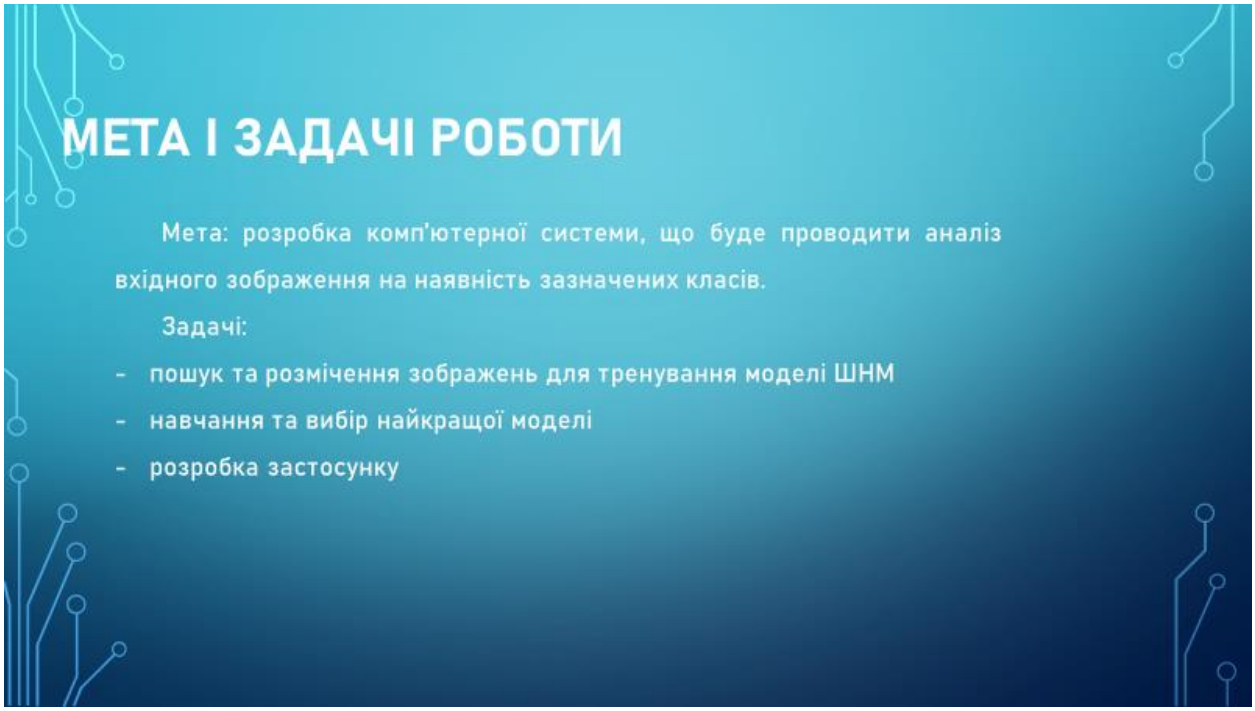
ДОДАТОК А
ГРАФІЧНИЙ МАТЕРІАЛ КВАЛІФІКАЦІЙНОЇ РОБОТИ

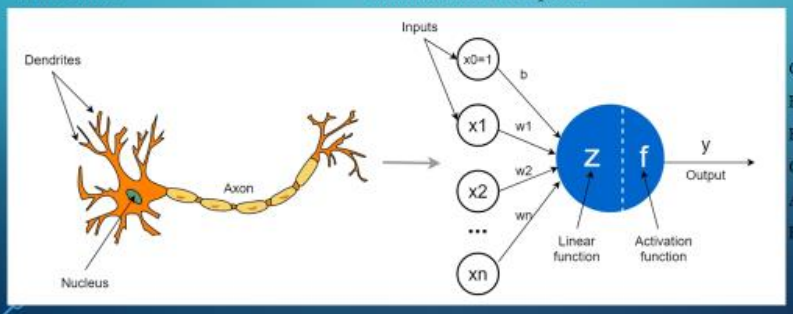

Харківський національний університет радіоелектроніки
Кафедра ЕОМ
Кваліфікаційна робота
Перший(бакалаврський) рівень
КОМП'ЮТЕРНА СИСТЕМА ОБРОБКИ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МАШИННОГО НАВЧАННЯ
Здобувач:
Олександр АВРУНІН
КІУКІ-21-5
Керівник:
Владислав ДЯЧЕНКО
ст. викладач кафедри ЕОМ


МЕТА І ЗАДАЧІ РОБОТИ
Мета: розробка комп'ютерної системи, що буде проводити аналіз вхідного зображення на наявність зазначених класів.
Задачі:
- пошук та розмічення зображень для тренування моделі ШНМ
- навчання та вибір найкращої моделі
- розробка застосунку

НЕЙРОМЕРЕЖА

Нейрон – основна структурно-функціональна одиниця нервової системи. Складається з тіла ті відростків: дендритів та аксонів.

Перцептрон — це найпростіший тип штучної нейронної мережі. Це базовий елемент нейронних мереж, який імітує функціонування біологічного нейрона.



Складається з наступних компонентів:

Входи (Inputs)

Ваги (Weights)

Сума (Summation):

Активаційна функція

Вихід (Output)

OPENCV

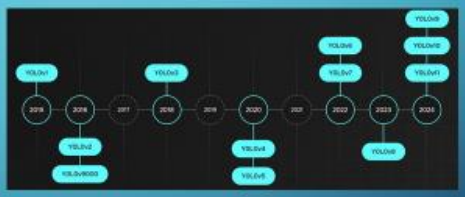
- це відкрита бібліотека комп'ютерного зору та машинного навчання, розроблена для забезпечення простих у використанні інтерфейсів для обробки зображень та відео

У бібліотеці існує такий інструмент аналізу вхїдниз зображень як Cascade Classifier – метод розпізнавання об'єктів, заснований на каскадному підході Хаара. Це надає можливість розпізнавати такі об'єкти як:

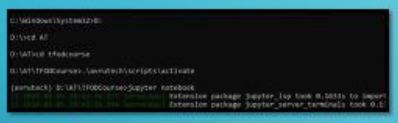
- **Обличчя**
- **Очі**
- **Автомобілі**

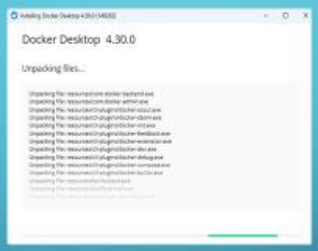
YOLO

(англ. You Only Live Once, укр. «Живеш лише раз») – швидкий та ефективний алгоритм виявлення об'єктів в реальному часі. Розроблений Йозефом Редмоном та Алі Фархаді у Вашингтонському Університеті в 2015р.



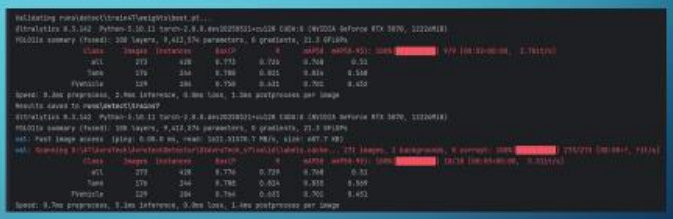
РОЗРОБКА МОДЕЛІ ШНМ



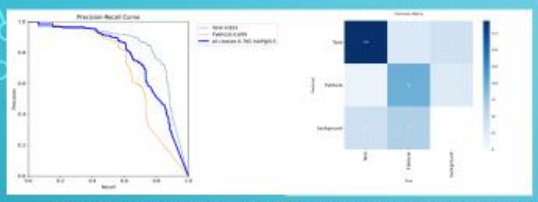


РОЗРОБКА МОДЕЛІ ШНМ

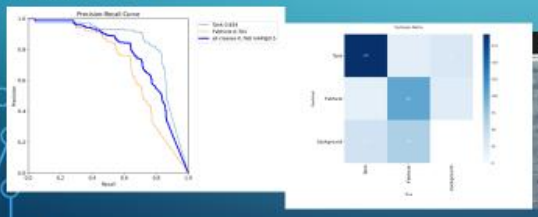
model = YOLO("yolov8s.pt") model.train(data="config.yaml", epochs=120, imgsz=1024) metrics = model.val()

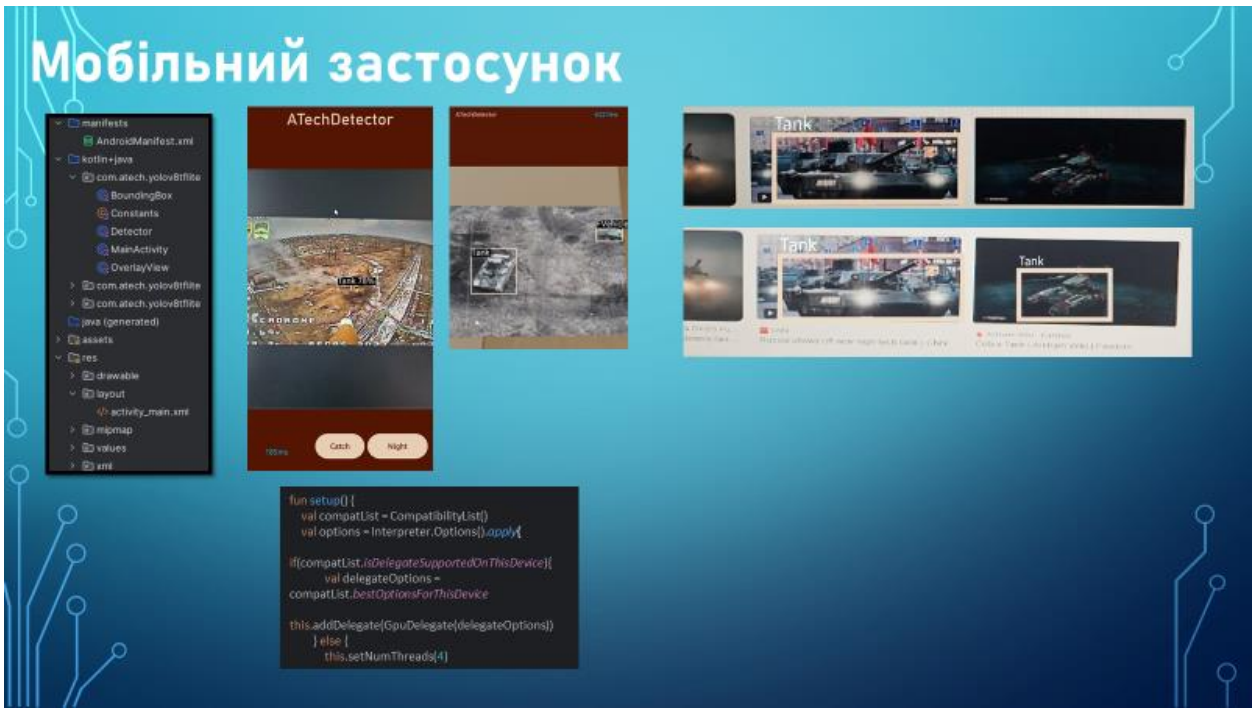


ПОРІВНЯННЯ МОДЕЛЕЙ 8 ТА 11 ВЕРСІЙ

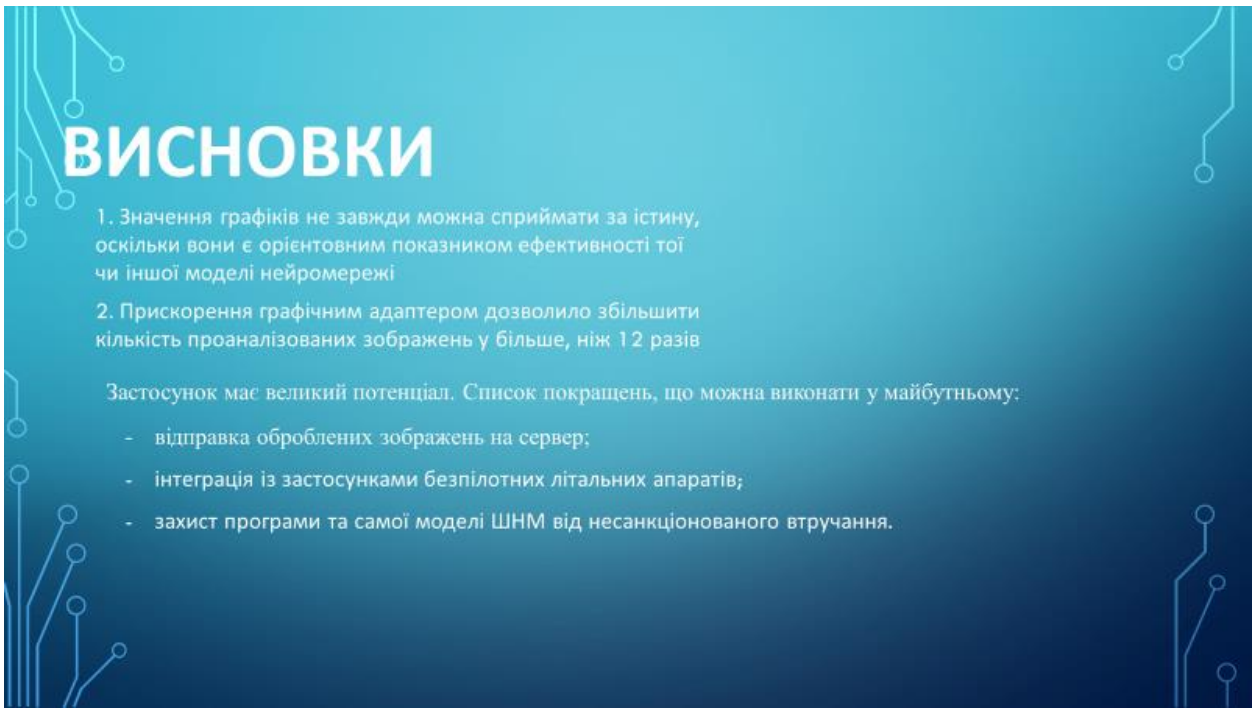


Крива точності-повноти (Precision-Recall Curve) ілюструє середнє значення успішних детекцій серед усіх. Модель, яка досягає ближче до верхнього правого кута цієї кривої, вважається кращою, оскільки вона одночасно досягає високої точності і високої повноти.




Мобільний застосунок
manifests
AndroidManifest.xml
kotlin+java
com.atech.yolov8tflite
BoundingBox
Constants
Detector
MainActivity
OverlayView
com.atech.yolov8tflite
com.atech.yolov8tflite
java (generated)
assets
res
drawable
layout
activity_main.xml
mipmap
values
xml
ATechDetector
Catch
Night
Tank
Tank
Tank
fun setup() {
val compatList = CompatibilityList()
val options = Interpreter.Options().apply{
if(compatList.isDelegateSupportedOnThisDevice){
val delegateOptions =
compatList.bestOptionsForThisDevice
this.addDelegate(GpuDelegate(delegateOptions))
} else {
this.setNumThreads(4)


Комп'ютерна система
BEGIN
DATABASE
TRAINING
MODEL 1
MODEL 2
COMPARISON
BEST MODEL
END
COMPUTER, SMARTPHONE, DRONE
DATA
LOCAL PROCESSING
LOCAL/ SERVER PROCESSING
ENCRYPTION
DATA TRANSIT
PROCESSING
RESULT
RESULT


ВИСНОВКИ
1. Значення графіків не завжди можна сприймати за істину, оскільки вони є орієнтовним показником ефективності тої чи іншої моделі нейромережі
2. Прискорення графічним адаптером дозволило збільшити кількість проаналізованих зображень у більше, ніж 12 разів
Застосунок має великий потенціал. Список покращень, що можна виконати у майбутньому:
- відправка оброблених зображень на сервер;
- інтеграція із застосунками безпілотних літальних апаратів;
- захист програми та самої моделі ШНМ від несанкціонованого втручання.

ДОДАТОК Б
КОД

Б.1 Навчання та верифікація моделі YOLO

Лістинг Б.1 – Код для запуску навчання моделі:

```
from ultralytics import YOLO
def main():
model = YOLO("yolov8s.pt")
model.train(data="config.yaml", epochs=160, imgsz=1024)
metrics = model.val()
model.save("AT_Weights/final_5070_yolv8s")
print("Saved as test5070_8l.pt")

if __name__ == '__main__':
main()
```

Лістинг Б.2 – Код для перевірки роботи моделі на відео чи фото

```
from ultralytics import YOLO
import cv2
import math

model =
YOLO('D:/AT/AvruTech/AvrutechDetector/AT_Weights/final_5070_yolv
8s.pt')
classNames = ["tank", "vehicle"]

image_path = "../image/66666.jpeg"
frame = cv2.imread(image_path)

if frame is None:
print("Failed to load image.")
exit()

results = model(frame, imgsz=1280)

for r in results:
boxes = r.boxes
for box in boxes:
conf = math.ceil((box.conf[0] * 100)) / 100
if conf >= 0.2:
x1, y1, x2, y2 = map(int, box.xyxy[0])
cv2.rectangle(frame, (x1, y1), (x2, y2), (5, 5, 80), 3)
```

```
cls = int(box.cls[0])
cv2.putText(frame, f'{classNames[cls]} {conf}', (max(0, x1),
max(35, y1)),
cv2.FONT_HERSHEY_TRIPLEX, 1, (181, 206, 231), thickness=1)

cv2.namedWindow("Detection", cv2.WINDOW_NORMAL)
cv2.resizeWindow("Detection", 1280, 720)
cv2.imshow("Detection", frame)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Лістинг Б.3 – Код конвертації моделі YOLO у формат TFLite

```
from ultralytics import YOLO
# Load the YOLO11 model
model = YOLO("yolo11n.pt")
# Export the model to TFLite format
model.export(format="tflite")  # creates
'yolo11n_float32.tflite'
# Load the exported TFLite model
tflite_model = YOLO("yolo11n_float32.tflite")
# Run inference
results = tflite_model("https://ultralytics.com/images/bus.jpg")
```

Б.2 Мобільний застосунок

Лістинг Б.4 – Запит дозволів, цільовий API

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools">

<uses-feature android:name="android.hardware.camera" />
<uses-permission android:name="android.permission.CAMERA" />
<application
android:allowBackup="true"
android:dataExtractionRules="@xml/data_extraction_rules"
android:fullBackupContent="@xml/backup_rules"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.YOLOv8TfLite"
tools:targetApi="31">
```

```
<activity
android:name=".MainActivity"
android:exported="true">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
```

Лістинг Б.5 – Метод Detect(), що відповідає за обробку кадрів:

```
fun detect(frame: Bitmap) {

interpreter ?: return
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 processedBitmap = applyPreProBitmapFilter(resizedBitmap)
lastProcessedBitmap = resizedBitmap

val tensorImage = TensorImage(DataType.FLOAT32)
tensorImage.load(processedBitmap)
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.isNullOrEmpty()) {
detectorListener.onEmptyDetect()
return
}
detectorListener.onDetect(bestBoxes, inferenceTime)
}
```

Лістинг Б.6 – Метод bestBox(), відповідає за вибір найкращого з прямокутників шляхом фільтрації порогом довіри і координати фігури:

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

val boundingBoxes = mutableListOf<BoundingBox>()

for (c in 0 until numElements) {
var maxConf = -1.0f
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)}
```

Лістинг Б.7 – Перевірка можливості прискорення графічним адаптером пристрою:

```
val compatList = CompatibilityList()
val options = Interpreter.Options().apply{
if(compatList.isDelegateSupportedOnThisDevice){
// if the device has a supported GPU, add the GPU delegate
val delegateOptions = compatList.bestOptionsForThisDevice
this.addDelegate(GpuDelegate(delegateOptions))
} else {
```

```
// if the GPU is not supported, run on 4 threads
this.setNumThreads(4)
}
}
```

Лістинг Б.8 – Метод onDetect(), що відповідає за малювання границь об’єктів:

```
override fun onDetect(boundingBoxes: List<BoundingBox>,
timeToProcess: Long) {
runOnUiThread {
hasDetectedOnce = true
lastDetectionTime = System.currentTimeMillis()
binding.inferenceTime.text = "${timeToProcess}ms"
binding.lastSeenText.visibility = View.GONE
binding.overlay.apply {
setResults(boundingBoxes)
invalidate()}
val bitmap = detector.getLastProcessedBitmap()
val bitmapWithBoxes = bitmap.copy(Bitmap.Config.ARGB_8888, true)
val canvas = Canvas(bitmapWithBoxes)
val boxPaint = Paint().apply {
color = Color.parseColor("#E7CEB5")
strokeWidth = 8f
style = Paint.Style.STROKE}
val textPaint = Paint().apply {
color = Color.WHITE
style = Paint.Style.FILL}
for (box in boundingBoxes) {
val left = (box.x1 * bitmap.width).coerceIn(0f,
bitmap.width.toFloat())
val top = (box.y1 * bitmap.height).coerceIn(0f,
bitmap.height.toFloat())
val right = (box.x2 * bitmap.width).coerceIn(0f,
bitmap.width.toFloat())
val bottom = (box.y2 * bitmap.height).coerceIn(0f,
bitmap.height.toFloat())
canvas.drawRect(left, top, right, bottom, boxPaint)
val boxWidth = right - left
val textSize = boxWidth / 10f
textPaint.textSize = textSize.coerceAtLeast(20f)
val labelText = box.clsName
canvas.drawText(labelText, left, top - 8, textPaint)}
lastBitmapWithBoxes = bitmapWithBoxes}}}
```

Лістинг Б.9 – Метод takeScreenshot(), що збегігає останнє оброблене зображення. Виконується лише за умови успішного знаходження об’єкту:

```
private fun takeScreenshot() {
try {
val rawBitmap = detector.getLastProcessedBitmap()
if (rawBitmap != null) {
val bitmap = if (isNightModeEnabled) enhanceForNight(rawBitmap)
else rawBitmap
saveBitmap(bitmap)
} else {
Toast.makeText(this, "Image is unavailable",
Toast.LENGTH_SHORT).show()}}
catch (e: Exception) {
e.printStackTrace()
Toast.makeText(this, "Error! File was not saved",
Toast.LENGTH_SHORT).show()    }}
```

Лістинг Б.10 – Метод enhanceForNight(), що викликається натисканням кнопки з метою підвищення яскравості вхідного зображення шляхом гамма-корекції:

```
private fun enhanceForNight(input: Bitmap): Bitmap {
val gamma = 1.4
val gammaCorrection = FloatArray(256) { i ->
((255.0 * Math.pow(i / 255.0, 1.0 / gamma)).toInt()).coerceIn(0,
255).toFloat()}
val result = Bitmap.createBitmap(input.width, input.height,
Bitmap.Config.ARGB_8888)
val canvas = Canvas(result)
val paint = Paint()
val colorMatrix = ColorMatrix(floatArrayOf(
1f, 0f, 0f, 0f, 0f,
0f, 1f, 0f, 0f, 0f,
0f, 0f, 1f, 0f, 0f,
0f, 0f, 0f, 1f, 0f))
paint.colorFilter = ColorMatrixColorFilter(colorMatrix)
val gammaCorrected = Bitmap.createBitmap(input.width,
input.height, Bitmap.Config.ARGB_8888)
for (x in 0 until input.width) {
for (y in 0 until input.height) {
val pixel = input.getPixel(x, y)
val r = gammaCorrection[Color.red(pixel)].toInt()
val g = gammaCorrection[Color.green(pixel)].toInt()
val b = gammaCorrection[Color.blue(pixel)].toInt()
val a = Color.alpha(pixel)
gammaCorrected.setPixel(x, y, Color.argb(a, r, g, b))}}
canvas.drawBitmap(gammaCorrected, 0f, 0f, paint)
return result}
```