ДОДАТОК А

Код програми

```
  private func configureCaptureSession() ->
AVCaptureVideoDataOutput? {
        disableCaptureSession()
        guard isEnabled else { return nil }
        defer { enableCaptureSession() }
        defer { captureSession.commitConfiguration() }

        captureSession.beginConfiguration()

        let modelFrameRate = BoxModel.frameRate
        let input =
AVCaptureDeviceInput.createCameraInput(position:
cameraPosition,

frameRate: modelFrameRate)

        let output =
AVCaptureVideoDataOutput.withPixelFormatType(kCVPixelFormatTyp
e_32BGRA)
        let success = configureCaptureConnection(input,
output)
        return success ? output : nil
    }

    private func enableCaptureSession() {
        if !captureSession.isRunning
{ captureSession.startRunning() }
    }

    private func disableCaptureSession() {
```

```
        if captureSession.isRunning
{ captureSession.stopRunning() }
    }

     private func configureCaptureConnection(_ input:
AVCaptureDeviceInput?,
                                             _ output:
AVCaptureVideoDataOutput?) -> Bool {

             guard let input = input else { return false }
             guard let output = output else { return false }


captureSession.inputs.forEach(captureSession.removeInput)

captureSession.outputs.forEach(captureSession.removeOutput)

             guard captureSession.canAddInput(input) else {
                 return false
             }

             guard captureSession.canAddOutput(output) else {
                 return false
             }

             captureSession.addInput(input)
             captureSession.addOutput(output)

             guard captureSession.connections.count == 1 else {
                 let count = captureSession.connections.count
                 print("The capture session has \(count)
connections instead of 1.")
                 return false
             }
```

```
            guard let connection =
captureSession.connections.first else {
                print("Getting the first/only capture-session
connection shouldn't fail.")
                return false
            }

            if connection.isVideoOrientationSupported {
                connection.videoOrientation = orientation
            }

            if connection.isVideoMirroringSupported {
                connection.isVideoMirrored = horizontalFlip
            }

            if connection.isVideoStabilizationSupported {
                if videoStabilizationEnabled {
                    connection.preferredVideoStabilizationMode
= .standard
                } else {
                    connection.preferredVideoStabilizationMode
= .off
                }
            }

            output.alwaysDiscardsLateVideoFrames = true

            return true
        }



    import UIKit
    import Vision
```

```
@available(iOS 14.0, *)
class MainViewController: UIViewController {
    @IBOutlet var imageView: UIImageView!
    @IBOutlet weak var labelStack: UIStackView!
    @IBOutlet weak var actionLabel: UILabel!
    @IBOutlet weak var confidenceLabel: UILabel!
    @IBOutlet weak var buttonStack: UIStackView!
    @IBOutlet weak var summaryButton: UIButton!
    @IBOutlet weak var cameraButton: UIButton!

    var videoCapture: VideoCapture!
    var videoProcessingChain: VideoProcessingChain!
    var actionFrameCounts = [String: Int]()
}

// MARK: - View Controller Events
extension MainViewController {

    override func viewDidLoad() {
        super.viewDidLoad()


        UIApplication.shared.isIdleTimerDisabled = true


        let views = [labelStack, buttonStack,
cameraButton, summaryButton]
        views.forEach { view in
            view?.layer.cornerRadius = 10
            view?.overrideUserInterfaceStyle = .dark
        }


        videoProcessingChain = VideoProcessingChain()
        videoProcessingChain.delegate = self
```

```
            videoCapture = VideoCapture()
            videoCapture.delegate = self

            updateUILabelsWithPrediction(.startingPrediction)
        }

        override func viewDidAppear(_ animated: Bool) {
            super.viewDidAppear(animated)

            videoCapture.updateDeviceOrientation()
        }


        override func viewWillTransition(to size: CGSize,
                                         with coordinator:
UIViewControllerTransitionCoordinator) {

            videoCapture.updateDeviceOrientation()
        }
    }

    // MARK: - Button Events
    extension MainViewController {

        @IBAction func onCameraButtonTapped(_: Any) {
            videoCapture.toggleCameraSelection()
        }


        @IBAction func onSummaryButtonTapped() {
            let main = UIStoryboard(name: "Main", bundle: nil)
```

```
            let vcName = "SummaryViewController"
            let viewController =
main.instantiateViewController(identifier: vcName)


            guard let summaryVC = viewController as?
SummaryViewController else {
                fatalError("Couldn't cast the Summary View
Controller.")
            }


            summaryVC.actionFrameCounts = actionFrameCounts


            modalPresentationStyle = .popover
            modalTransitionStyle = .coverVertical


            summaryVC.dismissalClosure = {

                self.videoCapture.isEnabled = true
            }

            present(summaryVC, animated: true)

            videoCapture.isEnabled = false
        }
    }

    // MARK: - Video Capture Delegate
    extension MainViewController: VideoCaptureDelegate {


        func videoCapture(_ videoCapture: VideoCapture,
```

```
                            didCreate framePublisher:
FramePublisher) {
            updateUILabelsWithPrediction(.startingPrediction)

            videoProcessingChain.upstreamFramePublisher =
framePublisher
        }
    }

    // MARK: - video-processing chain Delegate
    extension MainViewController: VideoProcessingChainDelegate
{

        func videoProcessingChain(_ chain:
VideoProcessingChain,
                                  didPredict actionPrediction:
ActionPrediction,
                                  for frameCount: Int) {

            if actionPrediction.isModelLabel {

                addFrameCount(frameCount, to:
actionPrediction.label)
            }

            updateUILabelsWithPrediction(actionPrediction)
        }

        func videoProcessingChain(_ chain:
VideoProcessingChain,
                                  didDetect poses: [Pose]?,
                                  in frame: CGImage) {

            DispatchQueue.global(qos: .userInteractive).async
{
```

```
                    self.drawPoses(poses, onto: frame)
                }
            }
        }

        // MARK: - Helper methods
        extension MainViewController {

            private func addFrameCount(_ frameCount: Int, to
actionLabel: String) {

                let totalFrames =
(actionFrameCounts[actionLabel] ?? 0) + frameCount


                actionFrameCounts[actionLabel] = totalFrames
            }

            private func updateUILabelsWithPrediction(_
prediction: ActionPrediction) {

                DispatchQueue.main.async { self.actionLabel.text =
prediction.label }


                let confidenceString = prediction.confidenceString
?? "Observing..."
                DispatchQueue.main.async
{ self.confidenceLabel.text = confidenceString }
            }

            private func drawPoses(_ poses: [Pose]?, onto frame:
CGImage) {
```

```
            let renderFormat = UIGraphicsImageRendererFormat()
            renderFormat.scale = 1.0

            let frameSize = CGSize(width: frame.width, height:
frame.height)
            let poseRenderer = UIGraphicsImageRenderer(size:
frameSize,
                                                       format:
renderFormat)


            let frameWithPosesRendering = poseRenderer.image {
rendererContext in

                let cgContext = rendererContext.cgContext

                let inverse = cgContext.ctm.inverted()

                cgContext.concatenate(inverse)

                let imageRectangle = CGRect(origin: .zero,
size: frameSize)
                cgContext.draw(frame, in: imageRectangle)

                let pointTransform = CGAffineTransform(scaleX:
frameSize.width,
                                                       y:
frameSize.height)

                guard let poses = poses else { return }


                for pose in poses {
```

```
                    pose.drawWireframeToContext(cgContext,
applying: pointTransform)
                }
            }


            DispatchQueue.main.async { self.imageView.image =
frameWithPosesRendering }
        }
    }
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	75 с.
	Інші документи	
2.	Презентаційні матеріали	20 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Кусков Р.В.			Дослідження методів для аналізу рухів людини та реалізація мобільного застосунку на основі штучного інтелекту.	Шифр групи	Код напр./спец.
Перевірив		Вітько О.В.				СШІм-19-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					