```
//ResNet50                                    //check if added to Target Membership
//ViewController.swift
import UIKit
import SceneKit
import ARKit
import Vision
class ViewController: UIViewController, ARSCNViewDelegate{
    @IBOutlet var sceneView: ARSCNView!
    private var resnetModel = Resnet50()
    private var hitTestResult: ARHitTestResult!
    private var visionRequests = [VNRequest]()

    override func viewDidLoad(){
        super.viewDidLoad()
        sceneView.delegate = self
        sceneView.showStatistics = true

        let scene = SCNScene()
        secenView.scene = scene

        registerGestureRecognizers()

    }

    private func registerGestureRecognizers()
        let tapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(tapped))
        self.sceneView.addGestureRecognizer(tapGestureRecognizer)
}

@objc func tapped(recognizer: UIGestureRecognizer){
        let sceneView = recognizer.view as! ARSCNView
        let touchLocation = self.sceneView.center
        guard let currentFrame = sceneView.session.currentFrame else{
            return
        }
        let hitTestResults = sceneView.hitTest(touchLocation, types: .featurePoint)
        if hitTestResults.isEmpty{
            return
        }
        self.hitTestResult = hitTestResult
        let pixelBuffer = currentFrame.capturedImage
        performVisionRequest(pixelBuffer: pixelBuffer)
}
private func performVisionRequest(pixelBuffer: CVPixelBuffer){
        let visionModel = try! VNCoreMLModel(for: self.resnetModel.model)
        let request = VNCoreMLRequest(model: visionModel){ request, error in
            if error != nil{
                return
            }
            guard let observations = request.results else{
                return
```

```
        }
        let observation = observation.first as! VNClassificationObservation

        DispatchQueue.main.async{
            self.displayPredictions(text: observation.identifier)
        }
    }
    request.imageCropAndScaleOption = .centerCrop
    self.visionRequests = [request]
    let imageRequestHandler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, orientation: .upMirrored,
                                                    options: [:])
                                  ⇒ in the back
    DispatchQueue.global().async{
        try! imageRequestHandler.perform(self.visionRequests)
    }
}

private func displayPredictions(text:String){
    let node = createText(text: text)
    node.position = SCNVector3(self.hitTestResult.worldTransform.columns.3.x,
                               self.hitTestResult.worldTransform.columns.3.y,
                               self.hitTestResult.worldTransform.columns.3.z)

    self.sceneView.scene.rootNode.addChildNode(node)

}
private func createText(text:String) -> SCNode{
    let parentNode = SCNNode()
    let sphere = SCNSphere(radius:0.01)
    let sphereMaterial = SCNMaterial()
    sphereMaterial.diffuse.contents = UIColor.orange
    sphere.firstMaterial = sphereMaterial
    let sphereNode = SCNNode(geometry: sphere)
    let textGeometry = SCNText(string: text, extrusionDepth:0)

    textGeometry.alignmentMode = kCAAlignmentCenter
    textGeometry.firstMaterial?.diffuse.contents = UIColor.orange
    textGeometry.firstMaterial?.specular.contents = UIColor.white
    textGeometry.firstMaterial?.isDoubleSided = true
    var font = UIFont(name:"Futura", size:0.15)
    textGeometry.font =font
    let textNode =SCNNode(geometry: textGeometry)
    textNode.scale = SCNVector3Make(0.2, 0.2, 0.2)

    parentNode.addChildNode(sphereNode)
    parentNode.addChildNode(textNode)

    return parentNode
}
```

// Occlusion

```
private func setup(){
    self.planeGeometry = SCNPlane(width: CGFloat(self.anchor.extent.x), height: CGFloat(self.anchor.extent.z))
    let material = SCNMaterial()
    self.planeGeometry.materials = [material]
    let planeNode = SCNNode(geometry: self.planeGeometry)
    planeNode.physicsBody = SCNPhysicsBody(type: .static, shape: SCNPhysicsShape(geometry: self.planeGeometry, options: nil))

     self.planeGeometry.firstMaterial?.colorBufferWriteMask = []      // those two lines most important
     planeNode.renderingOrder = -1

    planeNode.position = SCNVector3Make(anchor.center.x, 0, anchor.center.z)
    planeNode.transform = SCNMatrix4MakeRotation(Float(-Double.pi/2.0), 1.0, 0.0, 0.0)

    self.addChildNode(planeNode)
}
```