COREML入門篇（建立一個圖像識別 APP）

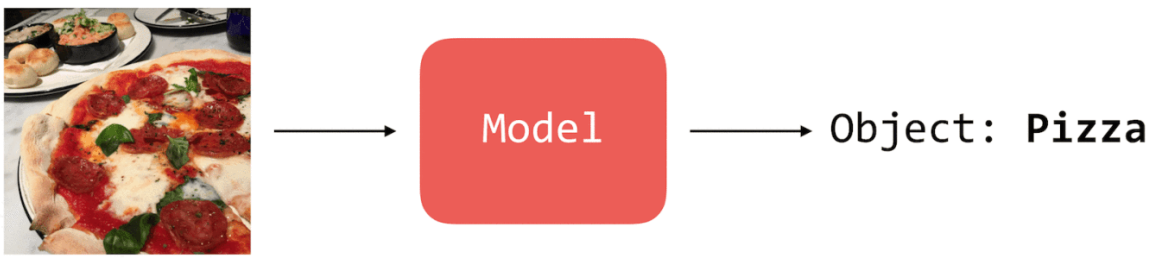
最近有小伙伴来根 Vergil 说：
Vergil也要做 AI 了？
难道是看到AI圈比较有钱，所以出轨了？
然而 Vergil 并没有。。

实在是AR 和 AI 关系太纠缠，
很多AR产品中都有结合了AI。

在 WWDC 2017 中，Apple 发表了许多令开发者们为之振奋的新框架（Framework） 及 API 。而在这之中，最引人注目的莫过于 Core ML 了。借由 Core ML，你可以为你的 App 添增机器学习(Machine Learning)的能力。而最棒的是你不需要深入的了解关于神经网络(Neural Network)以及机器学习(Machine Learning)的相关知识。接下来我们将会使用 Apple 开发者网站上提供的 Core ML 模型来制作范例 App。话不多说，Let's Start To Learn Core ML！

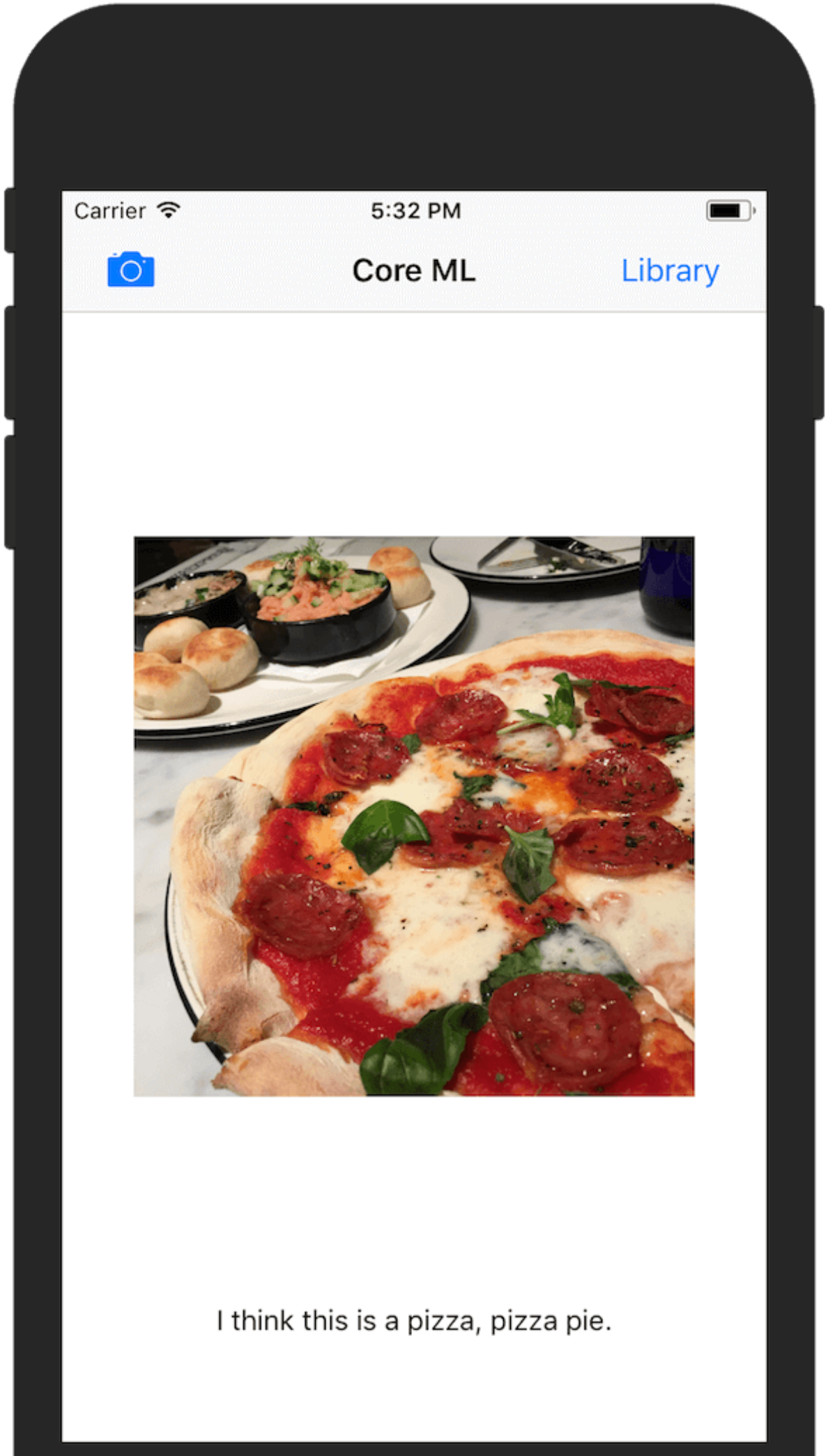
註: 接下來的教學會使用 Xcode 9 beta 作為開發工具，同時需要有 iOS 11 beta 的裝置以便測試其中的功能。Xcode 9 支援 Swift 3.2 及 4.0，本教學使用 Swift 4.0 開發。

Core ML 是在今年 WWDC 中发表的全新机器学习框架，将会随着 iOS 11 正式发布。借着 Core ML，你可以将机器学习整合进自己的 App 之中。 在这边我们先停一下，什么是机器学习(Machine Learning)呢？简单来说，机器学习是给予电脑可以在不明确撰写程式的情况下学习能力的应用。而一个完成训练的模型便是指将资料经由演算法结合后的成果。

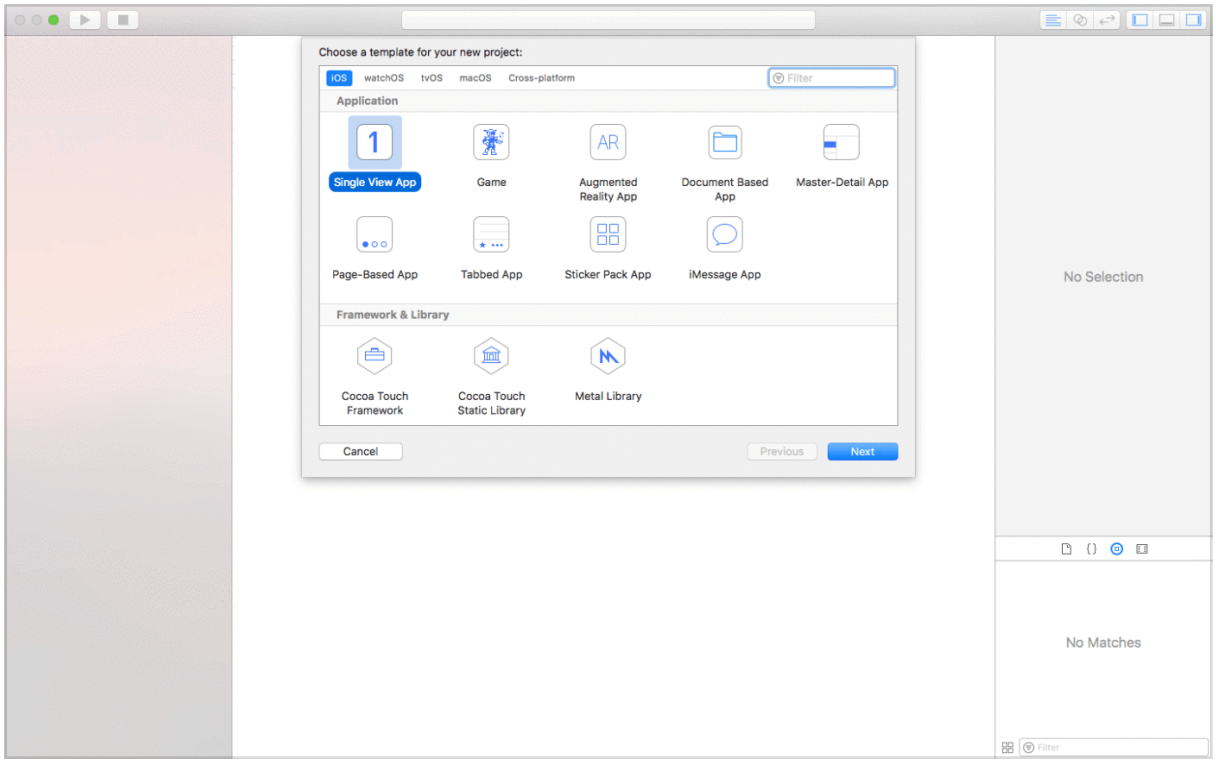


作为开发者，我们主要关心的是如何使用机器学习模型来做出有趣的玩意。幸运的是，Apple 让 Core ML 可以很简单的将不同的机器学习模型整合进我们的 App 中。如此一来一般的开发者们也将能够制作出图像识别、语言处理、输入预测等等功能。
听起来是不是很酷呢？让我们开始吧。

接下来要制作的 App 相当地简单。这个 App 能够让使用者拍照或是从相簿中选择一张相片，然后机器学习演算法将会试着辨识出相片中的物品是什么。虽然可能无法每次都识别成功，但你可以借此思考出如何在你 App 里使用 Core ML。


Carrier
5:32 PM
Core ML
Library
I think this is a pizza, pizza pie.

现在就开始吧！
首先，开启 Xcode 9 然后建立一个新专案。选择 Single View App，接着确认程式语言为 Swift。



制作使用者介面

一开始我们要做的是打开 Main.storyboard 然后加入几个 UI 元件到 View 之中。因此我们先点选 StoryBoard 中的 ViewController，然后到 Xcode 的功能列中点选 Editor-> Embed In-> Navigation Controller。当完成后你会看到 Navigation Bar 出现在 View 之上，接着我们将这个 Navigation Bar 的标题命名为 Core ML（或是任何你觉得适合的文字）。

接下来，拖曳两个按钮到 Navigation Bar 里头，一个放在标题左边一个放右边。接着点选左边的按钮然后到右侧的 Attributes Inspector 里将按钮由 System Item 改为 “Camera”。右边的按钮则修改文字为 “Library”。这两个按钮的用途是让使用者可以从相簿中选取相片或开启相机拍照。
最后我们还需要加入两个元件，分别是 UILabel 及 UIImageView。拖曳 UIImageView 到 View 里设定垂直水平置中以及长宽为 299，让 UIImageView 看起来是个正方形。现在轮到 UILabel，将其放入到 View 的底部并延伸两端到 View 的两侧。这样我们完成这个 App 的 UI 了。
虽然没有提到设定这些 View 的 Auto Layout，但很推荐你尝试设定 Auto Layout 以避免 UI 元件的错置。如果你不了解如何设定，也可以将 Storyboard 的尺寸设定为你要运行的装置尺寸。

实作相机以及相簿功能
现在我们已经完成 UI 了，接下来往实作功能的方向前进吧。在这个段落中，我们将会实作相簿以及相机按钮功能。首先在 ViewController.swift 中，我们要先调用 UINavigationControllerDelegate ，因为后续的 UIImagePickerController 会需要用到这部份。

class ViewController: UIViewController, UINavigationControllerDelegate

接著為畫面上的 UILabel 及 UIImageView 加上 IBoutlet。為了方便起見，我將 UIImageView 命名為 imageView，UILabel 則命名為 classifier。完成後的程式碼應該會如下面所呈現的樣子：

```
import UIKit

class ViewController: UIViewController, UINavigationControllerDelegate {
    @IBOutlet weak var imageView: UIImageView!
    @IBOutlet weak var classifier: UILabel!

     override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

接下来，你需要为两个按钮分别建立 IBAction 。请将以下的 Action 方法加入至 Viewcontroller 中吧：

```
@IBAction func camera(_ sender: Any) {

    if !UIImagePickerController.isSourceTypeAvailable(.camera) {
        return
    }
```

```
    let cameraPicker = UIImagePickerController()
    cameraPicker.delegate = self
    cameraPicker.sourceType = .camera
    cameraPicker.allowsEditing = false

    present(cameraPicker, animated: true)
}

@IBAction func openLibrary(_ sender: Any) {
    let picker = UIImagePickerController()
    picker.allowsEditing = false
    picker.delegate = self
    picker.sourceType = .photoLibrary
    present(picker, animated: true)
}
```

到这边我们先了解一下上述的 Action 方法。我们各产生了一个 UIImagePickerController 常数，然后将其设定为不允许编辑图像（不论是相机拍摄或是相簿选取），接着将 Delegate 指向为自己。最后呈现 UIImagePickerController 给使用者。

因为我们尚未将 UIImagePickerControllerDelegate 的方法们加入至 ViewController.swift 中，所以会发生错误。我们另外建立 Extension 来调用 delegate：

```
extension ViewController: UIImagePickerControllerDelegate {
    func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
        dismiss(animated: true, completion: nil)
    }
}
```

上面的程式码处理了使用者取消选取图像的动作，同时也指派了 UIImagePickerControllerDelegate 的类别方法到我们的 Swift 档案中。现在，你的程式码会如同下面所示：

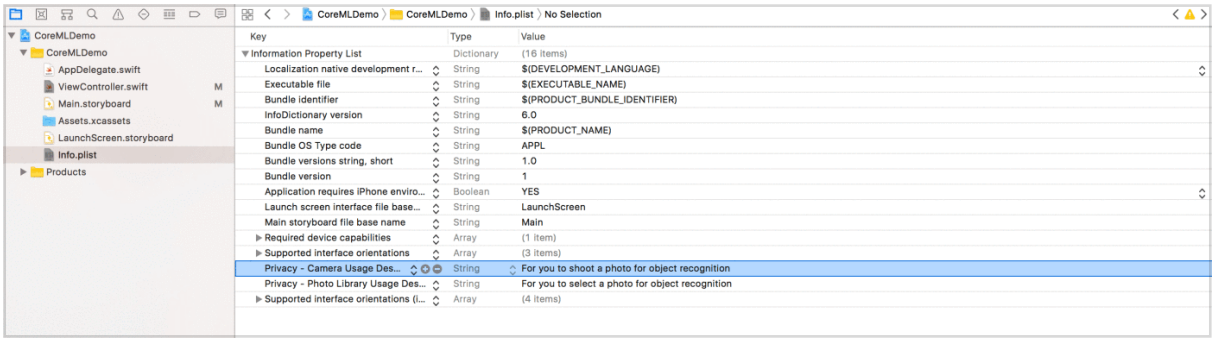
```
import UIKit
```

```
class ViewController: UIViewController, UINavigationControllerDelegate {

    @IBOutlet weak var imageView: UIImageView!
    @IBOutlet weak var classifier: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a
nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func camera(_ sender: Any) {

        if !UIImagePickerController.isSourceTypeAvailable(.camera) {
            return
        }

        let cameraPicker = UIImagePickerController()
        cameraPicker.delegate = self
        cameraPicker.sourceType = .camera
        cameraPicker.allowsEditing = false

        present(cameraPicker, animated: true)
    }

    @IBAction func openLibrary(_ sender: Any) {
        let picker = UIImagePickerController()
        picker.allowsEditing = false
        picker.delegate = self
        picker.sourceType = .photoLibrary
        present(picker, animated: true)
    }
```

```
}

extension ViewController: UIImagePickerControllerDelegate {
    func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
        dismiss(animated: true, completion: nil)
    }
}
```

现在回头确认一下 Storyboard 上的 UI 元件是否有与 Outlet 辨识及 Action 方法确实连结。

为了使用手机上的相机以及相簿，还有一项必需要做的事。前往 Info.plist 然后新增 Privacy – Camera Usage Description 及 Privacy – Photo Library Usage Description。从 iOS 10 开始，你需要添注说明为何你的 App 需要使用相机及相簿功能。

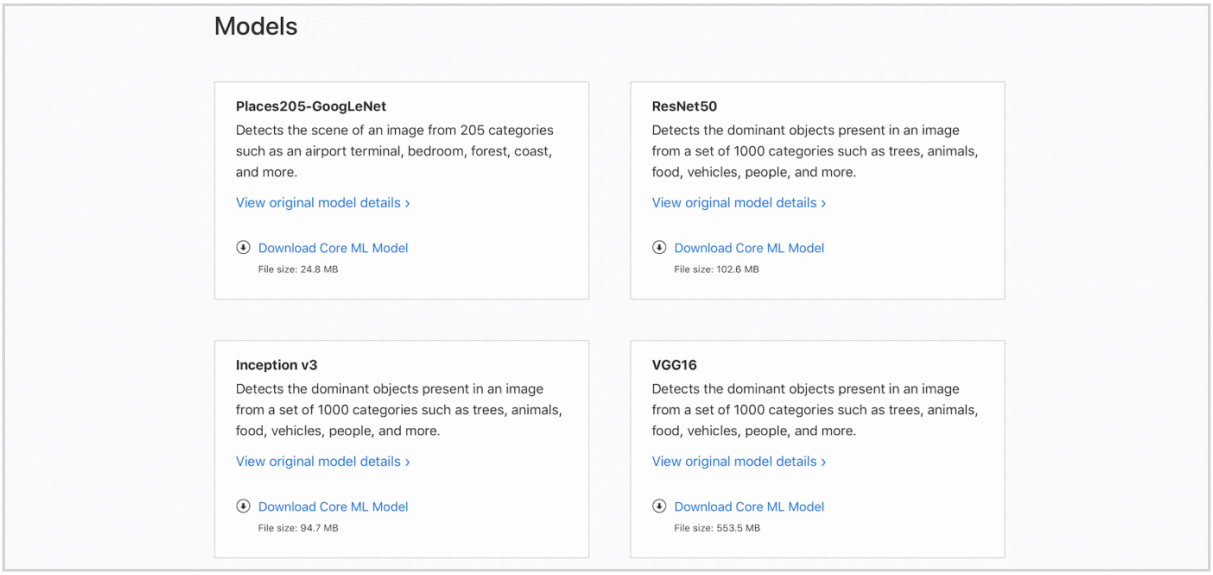



```
 <key>NSCameraUsageDescription</key>
<string>This application will use the camera for Augmented Reality.</string>
```

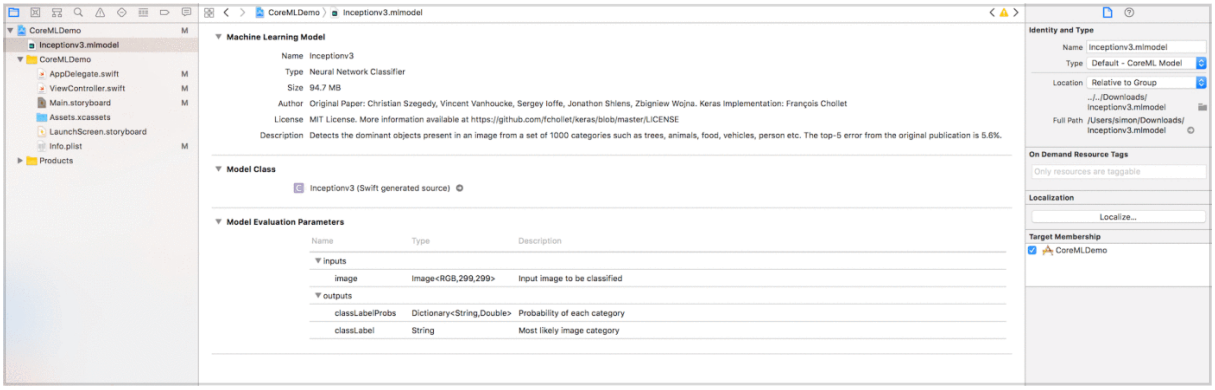
整合 CORE ML DATA 模型

现在让我们转换一下开始整合 Core ML 资料模型到我们的 App。如同早先提到的，我们需要一份预先训练的资料模型来与 Core ML 合作。虽然你也可以自己建立一份资料模型，但在本次范例里我们会使用由 Apple 开发者网站所提供预先训练完毕的资料模型。

前往 Apple 开发者网站的 Machine Learning 页面然后拉到最底下，你会找到四个已预先训练好的 Core ML 资料模型。



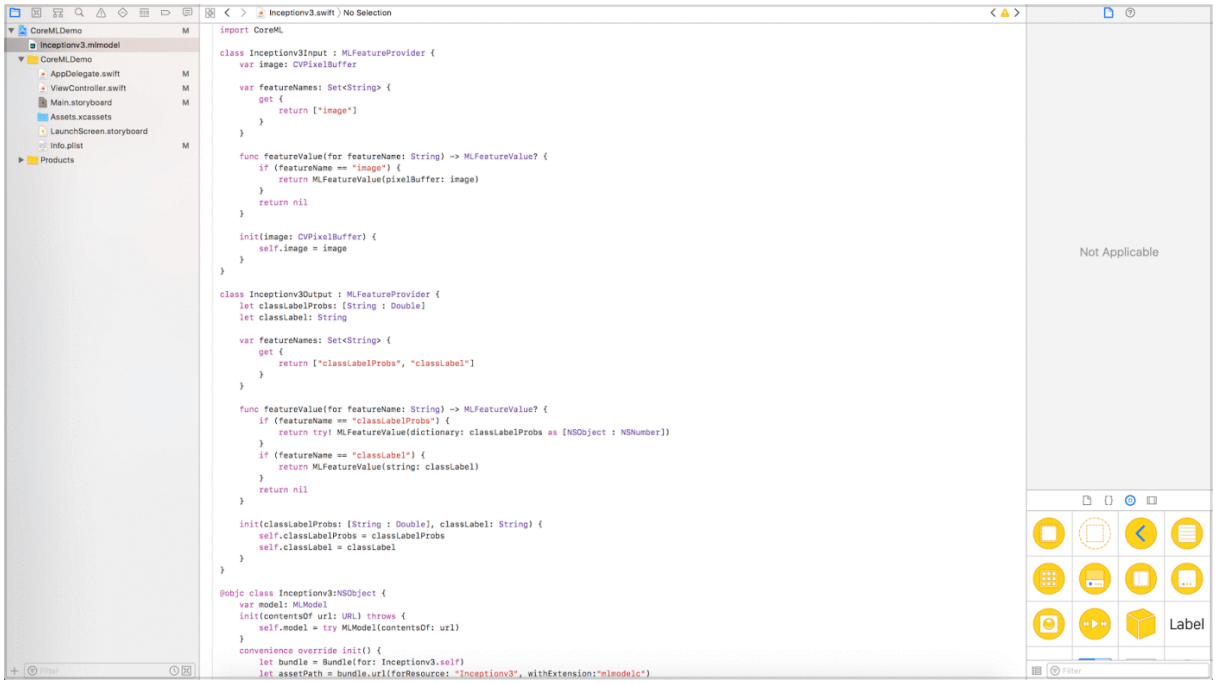
在本次教学里，我们使用了 Inception v3 模型。当然，你也可以程式其他另外三种的资料模型。当你下载完 Inception v3 后，将它放入 Xcode 专案中，然后看一下他显示了哪些东西。



注:请确认已选择了专案的 Target Membership，否则你的 App 将无法存取档案。

从上面的画面中，你可以看到资料模型的类型也就是神经网络（Neural Networks）的分类器。其他你需要注意的资讯有模型评估参数（Model Evaluation Parameters），这告诉你模型放入的是什么，输出的又是什么。以这来说，这个模型可以放入一张 299×299 的图像，然后回传给你这张图像最有可能的分类以及每种分类的可能性。

另外一个你会注意到的是模型的类别（Model Class）。这个模型类别（Inceptionv3）是由机器学习模型中产生出来并且可以让我们直接在程式码里使用。如果点击 Inceptionv3 旁的箭头，你可以看到这个类别的原始码。



现在，让我们把资料模型加入至我们的程式码中吧。回到 ViewController.swift，将 CoreML 引入：

```
import CoreML
```

接着，为 Inceptionv3 宣告一个 model 变数并且在 viewWillAppear() 中初始化。

```
var model: Inceptionv3!

override func viewWillAppear(_ animated: Bool) {
    model = Inceptionv3()
}
```

我知道你现在在想什么。

“为何我们不更早一点初始化呢？”

“在 viewWillAppear 中定义的要点是什么?”

这要点是当你的 App 试着识别你的图像里有哪些物件时，会快上许多。

现在，回头看一下 Inceptionv3.mlmodel，我们看到这个模型只能放入尺寸为 299×299 的图像。所以，我们该如何让一张图像符合这样的尺寸呢？这就是我们接下来要做的。

图像转换

在 ViewController.swift 的 Extension 中，添加下述的程式码。在新增的程式码里，我们实作了 imagePickerController(_:didFinishPickingMediaWithInfo) 来处理选取完照片的后续动作。

```
extension ViewController: UIImagePickerControllerDelegate {
    func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
        dismiss(animated: true, completion: nil)
    }

    func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [String : Any]) {
        picker.dismiss(animated: true)
        classifier.text = "Analyzing Image..."
        guard let image = info["UIImagePickerControllerOriginalImage"] as?
UIImage else {
            return
        }

        UIGraphicsBeginImageContextWithOptions(CGSize(width: 299, height: 299),
true, 2.0)
        image.draw(in: CGRect(x: 0, y: 0, width: 299, height: 299))
        let newImage = UIGraphicsGetImageFromCurrentImageContext()!
        UIGraphicsEndImageContext()

        let attrs = [kCVPixelBufferCGImageCompatibilityKey: kCFBooleanTrue,
kCVPixelBufferCGBitmapContextCompatibilityKey: kCFBooleanTrue] as CFDictionary
        var pixelBuffer : CVPixelBuffer?
        let status = CVPixelBufferCreate(kCFAllocatorDefault,
Int(newImage.size.width), Int(newImage.size.height), kCVPixelFormatType_32ARGB,
attrs, &pixelBuffer)
        guard (status == kCVReturnSuccess) else {
            return
```

```
        }

        CVPixelBufferLockBaseAddress(pixelBuffer!,
CVPixelBufferLockFlags(rawValue: 0))
        let pixelData = CVPixelBufferGetBaseAddress(pixelBuffer!)

        let rgbColorSpace = CGColorSpaceCreateDeviceRGB()
        let context = CGContext(data: pixelData, width:
Int(newImage.size.width), height: Int(newImage.size.height), bitsPerComponent:
8, bytesPerRow: CVPixelBufferGetBytesPerRow(pixelBuffer!), space:
rgbColorSpace, bitmapInfo: CGImageAlphaInfo.noneSkipFirst.rawValue) //3

        context?.translateBy(x: 0, y: newImage.size.height)
        context?.scaleBy(x: 1.0, y: -1.0)

        UIGraphicsPushContext(context!)
        newImage.draw(in: CGRect(x: 0, y: 0, width: newImage.size.width,
height: newImage.size.height))
        UIGraphicsPopContext()
        CVPixelBufferUnlockBaseAddress(pixelBuffer!,
CVPixelBufferLockFlags(rawValue: 0))
        imageView.image = newImage
    }
}
```

在上述程式码中被标记起来的部分：

第 7-11 行: 我们从 info 这个 Dictionary (使用 UIImagePickerControllerOriginalImage 这个 key)里取回了选取的的图像。同时我们让 UIImagePickerController 在我们选取图像后消失。
第 13-16 行: 因为我们使用的模型只接受 299×299 的尺寸，所以将图像转换为正方形，并将这个新的正方形图像指定给另个常数 newImage。
第 18-23 行: 我们把 newImage 转换为 CVPixelBuffer。 给对于 CVPixelBuffer 不熟悉的人，CVPixelBuffers 是一个将像数（Pixcel）存在主记忆体里的图像缓冲器。你可以从这里了解更多关于 CVPixelBuffers 的资讯
第 31-32 行: 然后我们取得了这个图像里的像数并转换为装置的 RGB 色彩。接着把这些资料作成 CGContext。这样一来每当我们需要渲染（或是改变）一些底层属性时可以很轻易的呼叫使用。最后的两行程式码即是以此进行翻转以及缩放。

第 34-38 行: 最后，我们完成新图像的绘制并把旧的资料移除，然后将 newImage 指定给 imageView.image。

如果你有点不明白上面的程式码，别担心。这些是有点进阶的 Core Image 语法，并不在这次教学范围内。你只要明白这些是要将选取的图像转换为资料模型可以接受的资料即可。不过推荐你可以换个数值执行几次，看看执行结果以更进一步的了解。

使用 CORE ML

无论如何，让我们把注意力拉回到 Core ML 上吧。我们使用 Inceptionv3 模型来作物件识别。借由 Core ML，我们只需几行程式码就可以完成工作了。贴上下述的程式码到 imageView.image = newImage 底下吧。

```
guard let prediction = try? model.prediction(image: pixelBuffer!) else {
    return
}
classifier.text = "I think this is a \(prediction.classLabel)."
```

没错，就是这样！Inceptionv3 类别已经产生了名为 prediction(image:) 的方法，它被用来预测所提供的图像里的物件。这里我们把 pixelBuffer 变数放入方法中，这个变数代表的是缩放后的图像。一旦完成预测会以字串形式回传结果，我们把 classifier 的文字内容更新为收到的结果文字。

当测试 App 时，你可能注意到 App 并不能很正确的预测出内容。这并不是你的程式码有问题，而是出在这份资料模型上。