Swift App 進階教師專班

AI 機器學習與設備(BLE & IOT)通訊

Day 1 – 使用機器學習模型

主辦：新北市政府教育局
日期：2023.07.19 (三)
講師：友教有限公司 Michael

Swift 物件導向觀念學習

Struct V.S. Class

我們來仔細觀察這兩種物件導向的寫法，一個是 struct S_Rect 另一個是 class C_Rect。

```
struct S_Rect {
    var width = 200.0
    var height = 100.0
    func area() -> Double {
        return width * height
    }
}

class C_Rect {
    var width = 200.0
    var height = 100.0
    func area() -> Double {
        return width * height
    }
}
```

乍看之下，兩個寫法只有 struct 和 class 關鍵字不同。但其實在使用變數和物件的時候有大大的差異，首先看到 S_Rect 的例子，如下：

```
var s1 = S_Rect()
var s2 = s1
s2.width = 500.0
print(s1.width)
```

一開始 s1 內容包含 width 為 200.0 和 height 為 100.0 ，這個從 S_Rect() 得到的結果，接著 s2 會從 s1 得到一份 copy，兩者有相同的內容，但是 s2 和 s1 代表不一樣的記憶體空間。所以當 s2.width 被更改成 500.0 時候，s1 的內容沒有被更動，所以執行到最後一個 line 當我們 print(s1.width) 的內容我們會得到 200.0 。

這個是 struct 特有的 copy 的特質， 等號 = 代表的意思是 assignment ， 這個動作是把 = 右邊的內容 copy 一份到左邊的變數，如果 = 右邊是 struct 類型。

例如： s2 = s1 ，這代表把 s1 的內容完整複製給 s2 。

如果同樣的操作應用在 class 身上，如下顯示。

```
var c1 = C_Rect()
var c2 = c1
c2.width = 500.0
print(c1.width)
```

一開始 c1 有 width 200.0 和 height 為 100.0 的內容，接著 c2 = c1 的運算會把 c1 所參考的記憶體內容，傳給 c2 這個時候 c2 和 c1 會指向同一個記憶體， c2.width 和 c1.width 會代表同一個實體，也就是說當我們用 c2.width = 500.0 把 200.0 改成 500.0 之後，c1.width 拿到的內容也是 500.0 。改變 c2.width 內容 c1.width 也改變了。這個就是 class 類型的特色， = 不會 copy 一份全新的內容給左邊的變數。

到這裡我們學到，struct 和 class 的不同，但其實這個觀念可以延伸應用到所有我們用的的類型，比如：Int ，Double 或 String 其實他們都是 struct 類型的型別。

我們從 Documentation 定義就可以看到。如下，從 Documentation 找到 Int 其實是一種 struct。

Structure

Int

A signed integer value type.

iOS 8.0+ | macOS 10.10+ | Mac Catalyst 13.0+ | tvOS 9.0+ | watchOS 2.0+

Declaration

```
@frozen struct Int
```

Overview

On 32-bit platforms, `Int` is the same size as `Int32`, and on 64-bit platforms, `Int` is the same size as `Int64`.

另個例子 FileManager 就是一個 class ，如下：

Class

FileManager

A convenient interface to the contents of the file system, and the primary means of interacting with it.

iOS 2.0+ | iPadOS 2.0+ | macOS 10.0+ | Mac Catalyst 13.0+ | tvOS 9.0+ | watchOS 2.0+

```
class FileManager : NSObject
```

在使用 class 類型的型別時候，要特別注意，傳入 function 運算時候，也有可能在 function 更改了，function 外面變數對應的內容。

語法筆記：

建構子 Initializer, Constructor

回到 Struct 和 Class 的差別，如果我們用具有參數值的方式來產生物件如下。

```
var s3 = S_Rect(width: 30.0, height: 50.0)
```

對於 Struct 來說是合法的，只要 S_Rect 裡面有相對應的 width 和 height ，而且產生的順序和 () 裡面呼叫的順序是一樣的，就是合法。

但是同樣的寫法在 class 就會有問題，如下：

```
• var c3 = C_Rect(width: 30.0, height: 50.0)
```

C_Rect 裡面的結構和 S_Rect 完全一樣，不論是變數名稱，順序和 function 名稱都一樣，但是就不可以用上方的寫法，這是 class 和 struct 設計理念不同的結果。如果要讓上方的紅色點消失(也就是 Error)，我們要在 C_Rect 裡面，新增一個，名稱固定的 function 叫 init 如下。

```
init(width:Double, height:Double){
    self.width = width
    self.height = height
}
```

如此一來原本的 error 就消失，如下：

```
var c3 = C_Rect(width: 30.0, height: 50.0)
```

這個 function 叫做 init 也就是 initializer 或是 constructor，參數的組合和內容要和 C_Rect(width: 30.0, height: 50.0) 一致，裡面用到 self 這個關鍵字，代表 width 這個參數內容要給 self.width 代表的是 init 外部的 width ，而不是 init 裡面的 width。

但不幸的是會產生另一個問題，如下：

```
• var c1 = C_Rect()
```

原本 C_Rect() 竟然錯誤，原因是少了一個 init，如下：

```
init(){

}
```

也就是對應到 C_Rect() 的呼叫，至於為什麼 init() 裡面是空的，是因為 width 和 height 這兩個變數都有給初始值，就不用在 init() 裡面給，如果一開始宣告 width 和 height 沒有給初始值，就要在 init() 裡面給，原本 init 的目的就是要把所有 class 裡面的變數都有初始值才可以正常使用。

語法上 struct 可以直接使用 S_Rect(width: 30.0, height: 50.0) ，前題是 width 和 height 的宣告順序一致， class 需要在 init 內定義使用 width 與 height 初始化，這樣才能使用 C_Rect(width: 30.0, height: 50.0)，其他初始化的時機原則是一樣的。

若要同時使用不同種方法來使用這個class ，其中需要涵蓋不同種初始化方法：

```
class C_Rect {
    var width = 200.0
    var height = 100.0
    func area() -> Double {
        return width * height
    }
    init(){

    }
    init(width:Double, height:Double){
        self.height = height
        self.width = width
    }
}
var c1 = C_Rect()
var c3 = C_Rect(width: 30.0, height: 50.0)
```

語法筆記：

Function 觀念進階

有參數 parameters 的函數

目前有一個 add 的函數：

```
func add(a:Int, b:Int) {
    print("\(a+b)")
}
```

呼叫 function add 時候必須把 a 和 b 兩個參數名稱一併寫上。

```
add(a:5,b:8)
```

自定義參數名的函數

使用函數時，可以自訂呼叫時候的名稱，如下：

```
func divide(dividend a:Double, divider b:Double){
    print(a/b)
}
```

此時 a, b 為內部使用的變數，dividend 和 divider 稱之為 external parameter 用來呼叫時使用，如下：

```
divide(dividend: 500.0, divider: 200.0)
```

可省略參數名的函數

如果將 divide 改成如下：

```
func divide(_ a:Double, divider b:Double){
    print(a/b)
}
```

external parameter 為一個下底線 _ ，呼叫的時候，可以省略第一個參數的名稱，如下：

```
divide(500.0, divider: 200.0)
```

有回傳值的函數

如果 function 需要回傳值，寫法如下：

```
func multiply(a:Double, b:Double) -> Double {
    return a*b
}
```

呼叫時候可以把，回傳值指定給給某個變數，如下：

```
let m = multiply(a: 5.0, b: 4.0)
```

Function Type

Swift 這個語言的變數 Type 可以用 function 的輸入與輸出來宣告，如下：

```
var sum:(Double,Double) -> Double
```

sum 是一個變數，用 var 宣告，其型別由 function 而來，(Double,Double) -> Double ，這就是一個 Function Type。

將 sum 的變數內容直接指定給一個正確型別的 function name，例如把 multiply 這個 function 的內容給 sum ：

```
sum = multiply
```

此時 sum 可以成為 function 一般的執行內容，如下：

```
let r = sum(800.0,900.0)
print(r)
```

function type 的變數，小括號裡面不需要加入參數名稱。

語法筆記：

動手做看看 – 模擬 AP CS 考題

題目 ： 最大公因數

問題描述：

產生一個名為 gcd 的 fucntion ，其中有 2 個 Int 輸入和一個 Int 輸出，輸出內容為這個兩個輸入的最大公因數，並要成功執行下面兩個範例結果。

題示：使用輾轉相除法與餘數的運算子是 %

範例一 ：

```
let r = gcd(a:6, b:8)
print(r)
```

輸出結果為 2

範例二：

```
let r = gcd(a:12, b:8)
print(r)
```

輸出結果為 4

打開 Swift Playgrounds 裡面的 Playground 練習即可。

語法筆記：

章節筆記

語法疑問：

教學重點：

反思回饋：

AI 機器學習

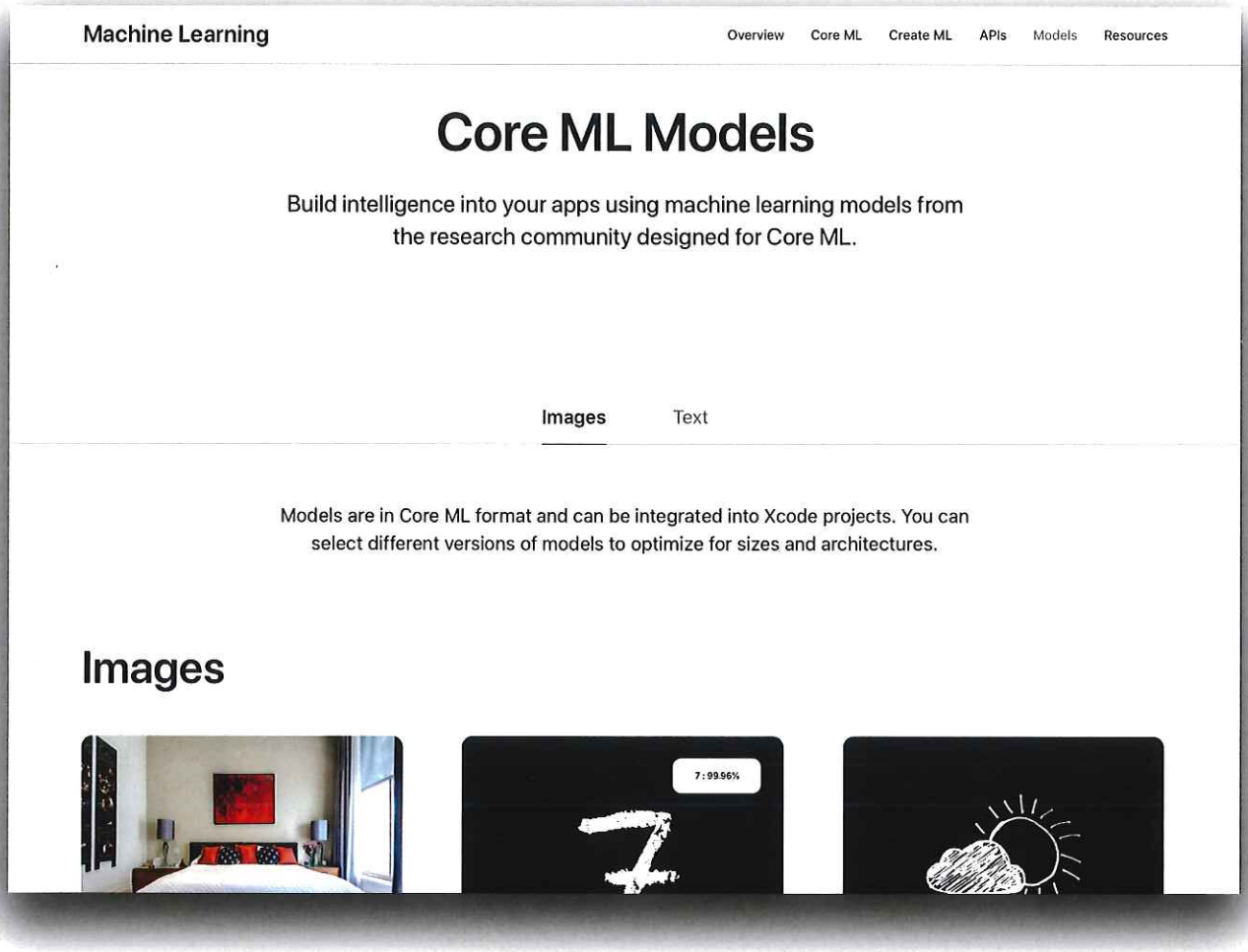
關鍵字：機器學習Machine learning、Core ML 框架、Core ML Models

機器學習（Machine learning）是一種電腦科學，讓電腦有能力從資料中學習並做出預測或決策，而不需要明確的程式設計。你可以把它想像成一個大腦：就像我們從經驗中學習，機器學習讓電腦從數據中學習。它被廣泛應用在各種領域，像是語音助理（Siri 或 Google Assistant）、推薦系統（Netflix 或 YouTube）和自駕車等。

Core ML 是 Apple 提供的機器學習框架，讓開發者能在 Apple 的各種裝置上，輕鬆地整合並使用各種預先訓練好的機器學習模型。這些模型可以用來處理各種任務，如圖像識別、語音識別、自然語言處理等。

Core ML Models 則是 Apple 提供的預訓練機器學習模型庫，它包含了多種模型，這些模型已經訓練好並優化，能直接在各種應用程式中使用。開發者可以根據需要，直接從官方網站下載並導入這些模型，而不需要自己從頭開始訓練，節省大量時間和資源。

Core ML Models



接下來我們要使用一個以訓練好的機器模型 Core ML Models ，在 Swift App 中利用 Core ML 來建立一個 AI 圖片辨識的 App。

AI 圖片辨識 App

建立的使用機器學習模型（ML Model）來辨識圖片的 App 可拆解為 3 個部分：

步驟 A

下載 Core ML Models 後，在 Swift Playgrounds 開啟新的 App 專案，並將下載的機器學習模型加入 App 資源庫。

步驟 B

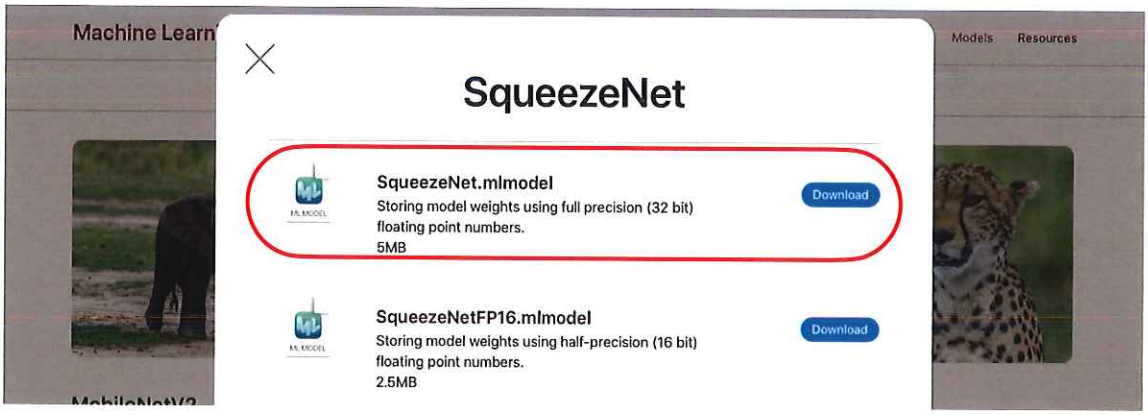
在畫面中使用 PhotosPicker 建立一個選取照片的功能，並讓圖片顯示在畫面上。

步驟 C

建立一個 function ，使用 Core ML 框架來運用機器模型判別已選取的照片，並在畫面中選取照片後使用這個函數。

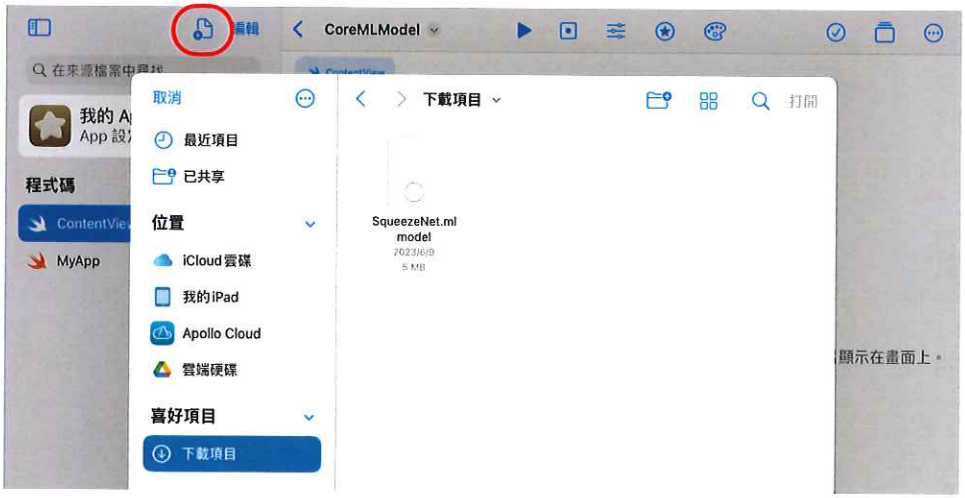
Core ML Models

在**步驟 A** 中，要到 Core ML Models 網站下載一個圖片的機器學習模型，在範例中使用了 SqueezeNet 模型。



在 Swift Playgrounds 中開啟一個新的 App 專案，命名為 CoreMLModel。

由左上角加入下載好的機器學習模型。



成功加入後可以在左方的資源，看到加入的機器學習模型。

PhotosPicker

在**步驟B**中，將要新增一個照片選取器來選取照片，可以分成5個**步驟**，分別為 B1 ~ B5 。

```
1   import SwiftUI
2   //加入所需要的框架
3   //B1.加入取用照片庫的框架                                    B1
4
5   struct ContentView: View {
6       //宣告變數位
7       //B2.宣告 selectedItem 與 selectedImage                  B2
8
9
10      var body: some View {
11          VStack {
12              //步驟B.使用 PhotosPicker 建立選取照片的功能，並讓圖片顯示在畫面上。
13              //B3.使用 PhotosPicker 選取照片                  B3
14
15              //B5.將 selectedImage 顯示在畫面上               B5
16
17
18          }..
19          //B4.偵測 selectedItem 改變，執行任務，取出照片內容給 selectedImage。
20                                                              B4
21
22      }
23
24      //步驟C.建立一個 function，使用 ML Model 判別已選取的照片。
25  }
26
```

B1 程式碼

```
import PhotosUI
```

引入 PhotosUI 的框架。

B2 程式碼

```
@State var selectedItem: PhotosPickerItem?
@State var selectedImage: UIImage?
```

加入所需要的變數。selectedItem 這個變數用於記錄 PhotosPicker 選取到的項目，其型別 Optional 的 PhotosPickerItem 。selectedImage 用於儲存選取到的相片 ，型別 Optional 的 UIImage 。宣告這兩個變數之前要加上 @State，當這兩個變數變化時，需要將畫面更新。

使用 PhotosPicker 選取到的不一定為圖片，也有可能是影片資料，所以用 PhotosPicker 選取到的項目要使用 selectedItem 來記錄，而不是直接使用 selectedImage 來記錄。

B3 程式碼

使用 PhotosPicker 選取照片，選取照片到的照片項目會存在 selectedItem 內，記得要加 $ 賦予 PhotosPicker 改變 selectedItem 的內容，選取的項目需要符合圖片類型，確保不會選擇到影片或其他類型的資料。

```
PhotosPicker("Select Photo", selection: $selectedItem,
matching: .images)
```

B4 程式碼

當使用 PhotosPicker 選取到類型為圖片的項目時，使用 .onChange 來監測 selectedItem 的改變，.onChange 是一個 closure ，接受 selectedItem 這個參數，但我們不使用這個 closure 回傳的內容，所以使用 _ in 來省略 。

```
.onChange(of: selectedItem) { _ in
    Task {


    }
}
```

偵測到 selectedItem 改變後，需要做一些資料處理的**步驟**，將 selectedItem 轉換成 Data 類型，再將 Data 轉為 UIImage 照片資料型別，這樣才可以在畫面上顯示選取的圖片。這個資料處理需要一些時間，屬於一個非同**步** (Asynchronous) 執行的操作，需要使用 await 來等待這些程式**碼**執行完成，並將這些非同**步**執行的程式**碼**放在一個 Task { } 任務內。

在 Task{ } 放入要執行資料處理的程式**碼**：

```
.onChange(of: selectedItem) { _ in
    Task {
        guard let data = try? await selectedItem?.loadTransferable(type: Data.self),
              let uiImage = UIImage(data: data) else {
            print("Failed")
            return
        }
        selectedImage = uiImage
    }
}
```

使用 guard let 來執行資料處理，當資料不為空值時，才進行資料處理，否則印出“Failed”並 return 離開這個任務。

處理 data 資料使用 data = try? await selectedItem?.loadTransferable(type: Data.self)，.loadTransferable 這個方法會拋出 error ，所以需要 try 來使用這個方法，而用 try? 執行時，若有拋出異常等號右邊將會返回 nil。

這個資料轉換需要時間，所以需要用 await 來等待結果。selectedItem 是一個 Optional，所以需要加？。將 selectedItem? 選擇的內容用 .loadTransferable(type: Data.self) 的方法轉成 Data 的資料類型，其中 Data.self 指的是 Data 型別本身，而不是指一個型別為 Data 的資料。

成功拿到 data 資料後，將 data 使用 UIImage(data: data) 轉成圖片，儲存到 uiImage。如果 data 與 uiImage 都有成功拿到資料，將 uiImage 的內容存到 selectedImage。

B5 程式碼

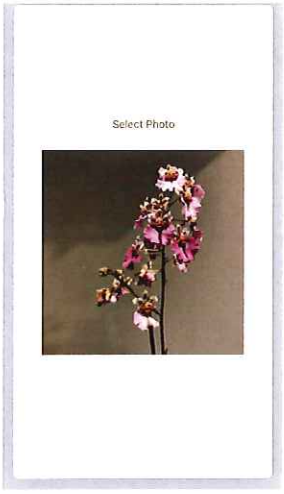
selectedImage 的型別為一個 Optional UIImage，在畫面顯示之前要先判斷不為空值再顯示在畫面上，否則會造成 App 當機。

```
if let selectedImage = selectedImage {
    Image(uiImage: selectedImage)
} else {
    Image(systemName: "photo")
}
```

另外可以在這兩個 Image 的後面加上所需要的修飾符來調整圖片大小的顯示，例如：

```
Image(systemName: "photo")
    .resizable()
    .scaledToFit()
    .frame(height: 300)
    .padding()
```

這樣就完成照片選取器的功能，測試看看是否能正常選取照片？



章節筆記

語法疑問：

教學重點：

反思回饋：

Core ML Model 與 判斷圖片

接下來要使用這一個 MLModel 來判斷前面選取的圖片。

使用以訓練好的機器學習模型來進行圖片，可以拆為以下幾個**步驟**：

1. 在引入框架與宣告變數。
2. 建立一個來執行圖片判斷函數 predictImage()，裡面包含一個 do-catch 語法。
3. 確保圖片資料不為空值。
4. 設定正確的 Model 名稱與的 URL 路徑。
5. 在 do-catch 內，初始化與設定 Model。
6. 在 do-catch 內，建立一個 Core ML 的請求 request。
7. 在 do-catch 內，建立一個處理器，執行 request 分析圖片。
8. 在選取圖片後，執行 predictImage() 來判斷圖片。

先在引入框架區域，B1 程式碼 後方加入機器學習的框架 Vision 。

```
import Vision
```

在畫面中宣告變數的位置，B2 程式碼 後加入字串變數 predictionLabel 來儲存判斷結果，再加入一個浮點數變數 probability 來儲存為此結果的機率。

```
@State var predictionLabel = ""
@State var probability: Float = 0.0
```

語法筆記：

步驟 C

將圖片判斷的程式碼另外寫在一個 predictImage() 函數，函數中包含一個 do { } chatch { }，並在 chatch 中印出 do 大括弧內拋出的錯誤，如下：

```
func predictImage() {
    //C1.確保圖片資料不為空值                                    C1 ~ C2

    //C2.設定正確的 Model 名稱與的 URL 路徑。

    //C3 ~ C5 使用到 try 語法，執行時有可能會拋出錯誤，所以要將其放置在 do-chtch 內
    do {
        //C3.初始化與設定 Model                                  C3 ~ C5

        //C4.建立一個 Core ML 的請求 request

        //C5.建立一個處理器，執行 request 分析圖片

    } catch {
        print(error)
    }


}..
```

C1 程式碼

確保圖片資料不為空值：

```
guard let selectedImage = selectedImage else {
    return
}
```

使用 guard let 確定 selectedImage 不是空值時，才執行接下來的程式碼。
若 selectedImage 為空值，則 return 離開這個函數。

C2 程式碼

設定正確的 Model 名稱與的 URL 路徑：

```
let modelName = "SqueezeNet"
guard let modelURL = Bundle.main.url(forResource: modelName, withExtension:
 "mlmodel") else {
    print("error loading model")
    return
}
```

定義一個變數 modelName 來儲存 Core ML Models 的檔名 "SqueezeNet"。

接著需要取得檔案路徑，使用 guard let 語法確保拿到的 modelURL 不為空值，若沒有正確拿到，modelURL 為空值時，印出 "error loading model" ，並 return 離開這個函數。

在前面我們把 Core ML Model 放在了 App 的專案資料夾內，利用 Bundle.main.url 的方法可以獲取此專案資料夾（應用程式 bundle）內的路徑，也就是下圖中資源內部的檔案，其中的參數為(forResource: modelName, withExtension: "mlmodel") ，modelName 是稍早宣告的檔名，"mlmodel" 是副檔名名稱。

C3 程式碼

在 do-catch 內，初始化與設定 Model：

```
let compiledUrl = try MLModel.compileModel(at:modelURL)
let mlmodel = try MLModel(contentsOf: compiledUrl)
guard let model = try? VNCoreMLModel(for: mlmodel) else {
    print("VNCoreMLModel error")
    return
}
```

let compiledUrl = try MLModel.compileModel(at: modelURL) 是將的 .mlmodel 檔案進行編譯，將機器學習模型翻譯成一種適合設備（在 iPhone 或 iPad 上）運行的語言。其中 modelURL 是 .mlmodel 機器學習模型所在的位置。

let mlmodel = try MLModel(contentsOf: compiledUrl) 它會嘗試從編譯過的 URL (compiledUrl) 讀取模型並將其加載到記憶體中。

在這邊加入以下程式碼，可以得到這個機器學習模型內，學習了哪些類別的照片資料，所以這個模型也只能判斷符合這些關鍵字的照片。

```
print(mlmodel.modelDescription.classLabels)
```

```
Optional([tench, Tinca tinca, goldfish, Carassius auratus, great white shark, white
 shark, man-eater, man-eating shark, Carcharodon carcharias, tiger shark, Galeocerdo
 cuvieri, hammerhead, hammerhead shark, electric ray, crampfish, numbfish, torpedo,
 stingray, cock, hen, ostrich, Struthio camelus, brambling, Fringilla montifringilla,
 goldfinch, Carduelis carduelis, house finch, linnet, Carpodacus mexicanus, junco,
 snowbird, indigo bunting, indigo finch, indigo bird, Passerina cyanea, robin,
 American robin, Turdus migratorius, bulbul, jay, magpie, chickadee, water ouzel,
 dipper, kite, bald eagle, American eagle, Haliaeetus leucocephalus, vulture, great
```

guard let model = try? VNCoreMLModel(for: mlmodel) else { ... } 嘗試將 Core ML 模型轉換為一個適用於 Vision 框架的 VNCoreMLModel 模型。如果轉換失敗，它將印出一個錯誤訊息並提早返回。如果成功，得到的 model 可以被用於後續的圖片分析請求。

C4 程式碼

建立一個 Core ML 的請求 request ：

```
let request = VNCoreMLRequest(model: model) { request, _ in
    if let classifications = request.results as?
     [VNClassificationObservation] {
        if let bestClassification = classifications.first {
            predictionLabel = bestClassification.identifier
            probability = bestClassification.confidence
        }
    }
}
```

request 為 VNCoreMLRequest(model: model) 的物件，這個用來處理圖片的任務，使用模型為前面建立的 model。完成分析後會傳出一個 request 的結果，在紅色方框區塊對於傳出的 request 進行一些處理，並把預測分析的結果存下來。

C4 程式碼的紅色方框內

if let classifications = request.results as? [VNClassificationObservation]，這行程式碼將分析結果 request.results 轉換為 VNClassificationObservation 的陣列。使用 as? 轉換型別時， 如果轉換成功，as? 將返回一個 Optional Type 的值。如果轉換失敗，as? 將返回 nil。

if let bestClassification = classifications.first 取得了 classifications 陣列中的第一個物件，這個 classifications 陣列是依照分析結果的 "機率" 大小來排列，第一個物件就是分析出來機率最高的結果。

最後將 bestClassification 的 identifier 儲存為 predictionLabel ，這就是模型認為圖片最有可能是的分類結果。並把可能為這個分類結果的機率 bestClassification.confidence 儲存為 probability 。

C5 程式碼

建立一個處理器，執行 request 分析圖片：

```
let handler = VNImageRequestHandler(cgImage: selectedImage.cgImage!)
try handler.perform([request])
```

用 VNImageRequestHandler 創建一個圖片請求處理器 handler ，處理的圖片用 .cgImage! 將 selectedImage 由 UIImage 轉型為 CGImage 的形式作為參數。

try handler.perform([request]) 這行程式碼告訴 handler 執行之前創建的 Core ML 請求 request。因為執行 request 可能會丟出錯誤（例如，如果圖像無法被處理），所以需要用 try 來執行請求。

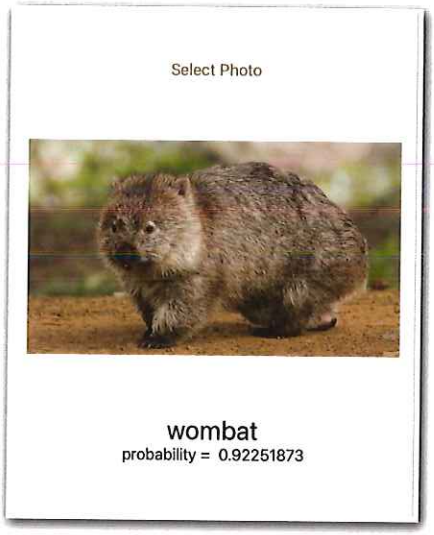
執行圖片判斷函數

在 B4 程式碼 的最後方加入執行圖片判斷的函數 predictImage() 。

```
Task {
    guard let data = try? await selectedItem?.loadTransferable(type: Data.self),
            let uiImage = UIImage(data: data) else {
        print("Failed")
        return
    }
    selectedImage = uiImage
    predictImage()
}
```

並在 B5 程式碼 內，
加入 Text() 顯示執行函數後，照片的預測結果。

```
if let selectedImage = selectedImage {
    Image(uiImage: selectedImage)
        .resizable()
        .scaledToFit()
        .frame(height: 300)
        .padding()
    Text(predictionLabel).font(.title)
    Text("probability =  \(String(probability))")
```



執行看看程式碼，試著判別不同的照片，結果如何？

除了拍照後將照片放進圖片辨識 App 中判斷，還可以從網路上搜尋其他圖片將圖片存到相簿中，測試圖片辨識 App 的準確度。

使用圖片搜尋，找到要下載的圖片後，對圖片長按後，會出現一個選單，選擇儲存到相片，這樣就可以在圖片辨識 App 中找到下載的圖片。

章節筆記

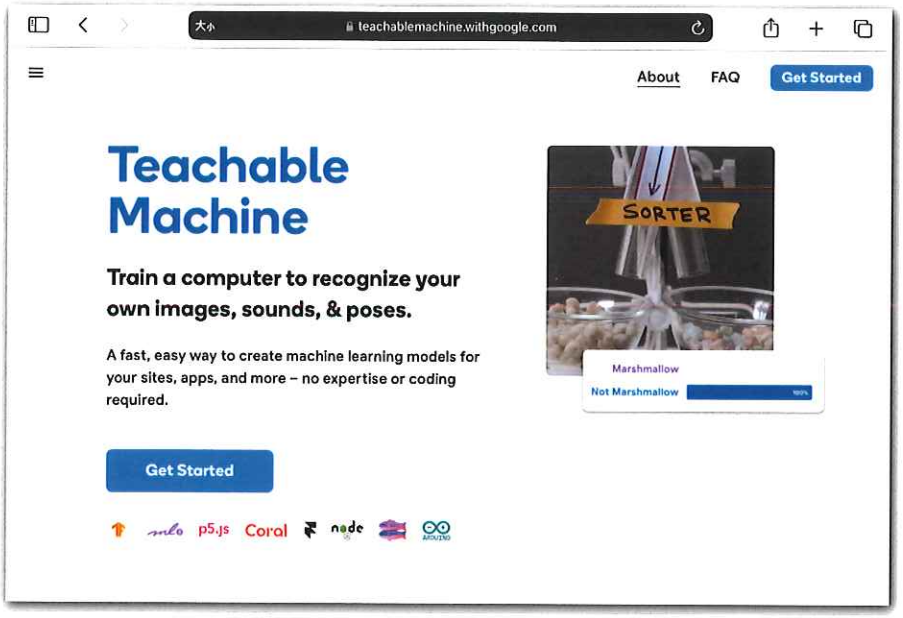
語法疑問：

教學重點：

反思回饋：

Teachable Machine

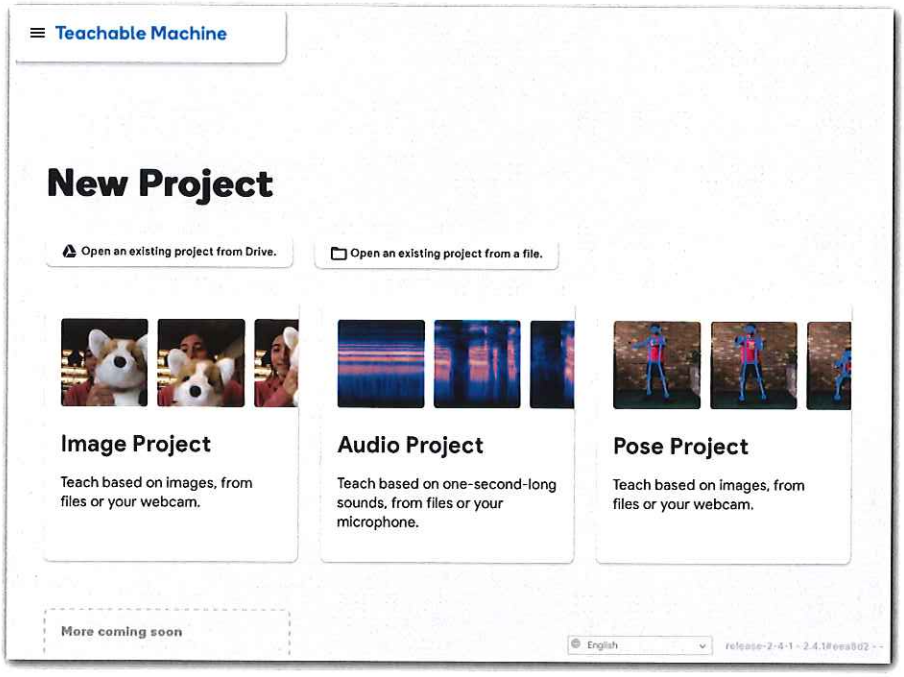
Teachable Machine 是一個 Google 推出的網頁工具，能讓使用者在不需要有機器學習的專業知識，也無需寫程式的情況下，就可以建立自己的機器學習模型。網頁中能夠使用設備鏡頭、圖像、聲音或者姿勢數據訓練模型，然後即時看到結果，進一步驗證和調整模型的性能。



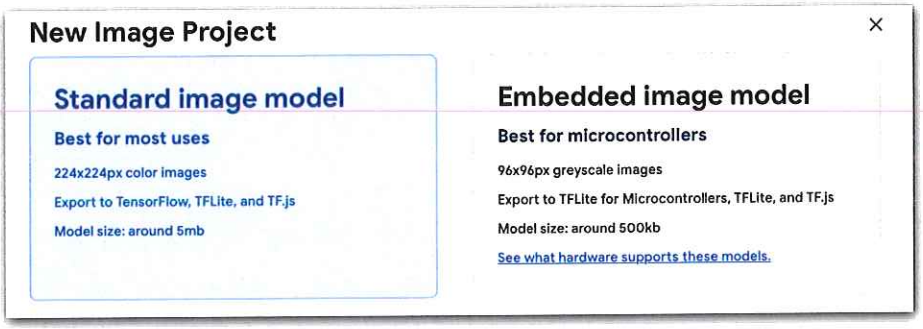
TeachableMachine App

在電腦上直接使用此網頁來訓練機器學習模型，在 iPhone 與 iPad 上需要下載 App 才能使用。到 App Store 搜尋『 TeachableMachine 』，或是掃上方的 QR Code 下載此 App。

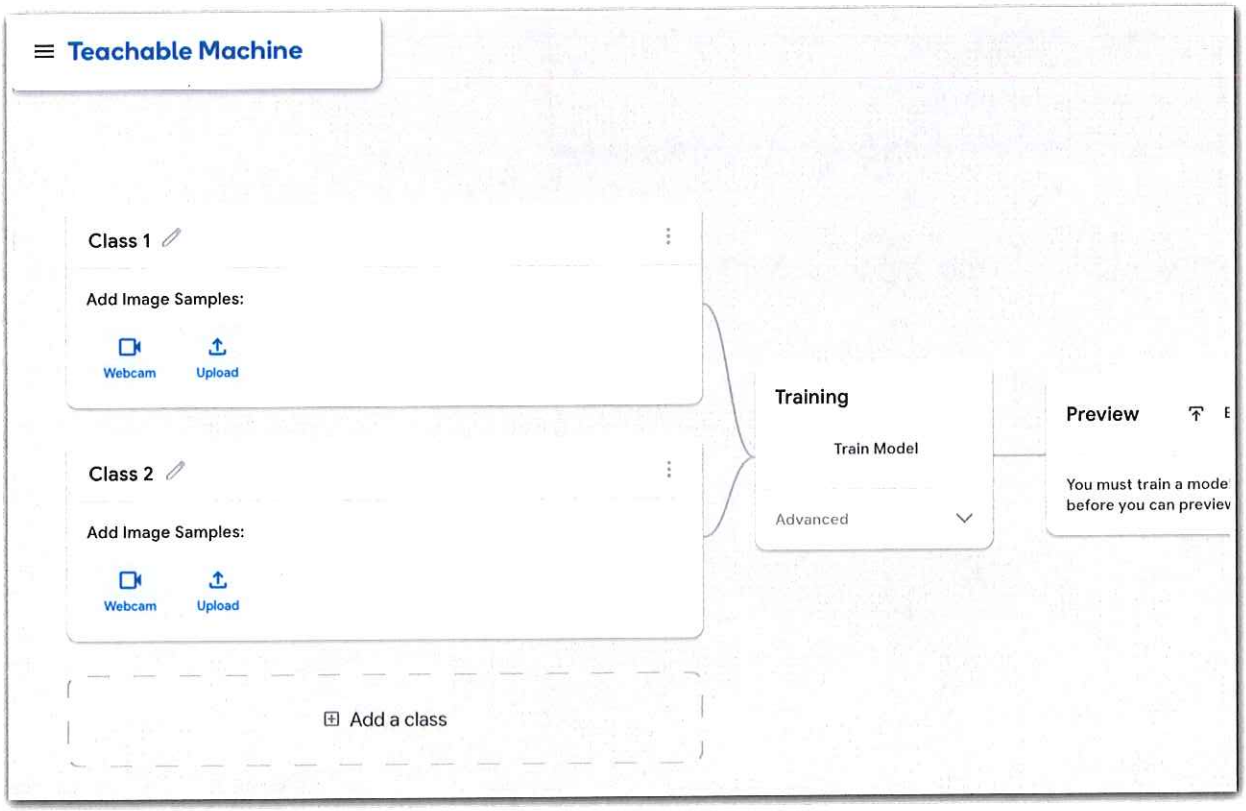
打開 App 後，或是網頁按下 「 Get Started 」後會出現下方畫面，點選 Image Project：



選擇 『Standard image model』



就可以從以下畫面來訓練自己的圖片機器學習模型：

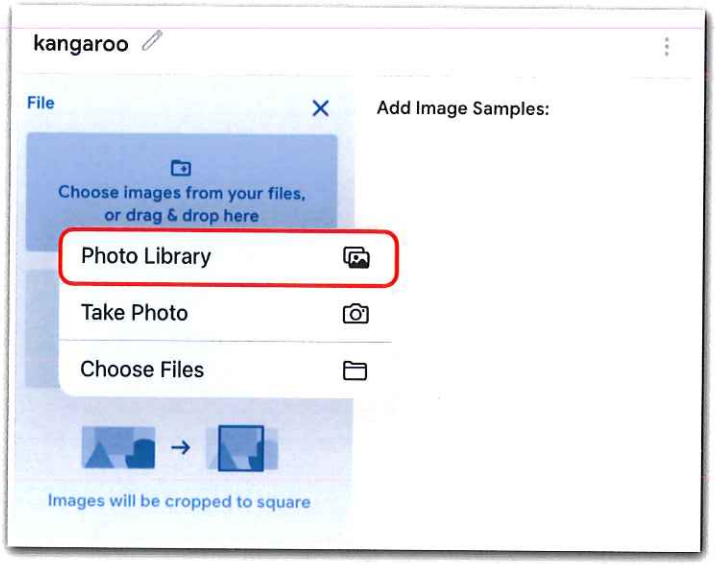


預設會給有兩個圖片類別，必須要在這兩個圖片類別內放入資料，並重新命名圖片類別的名稱。圖片資料可以從相機拍照，或是直接上傳檔案或是相簿內的照片/圖片檔。

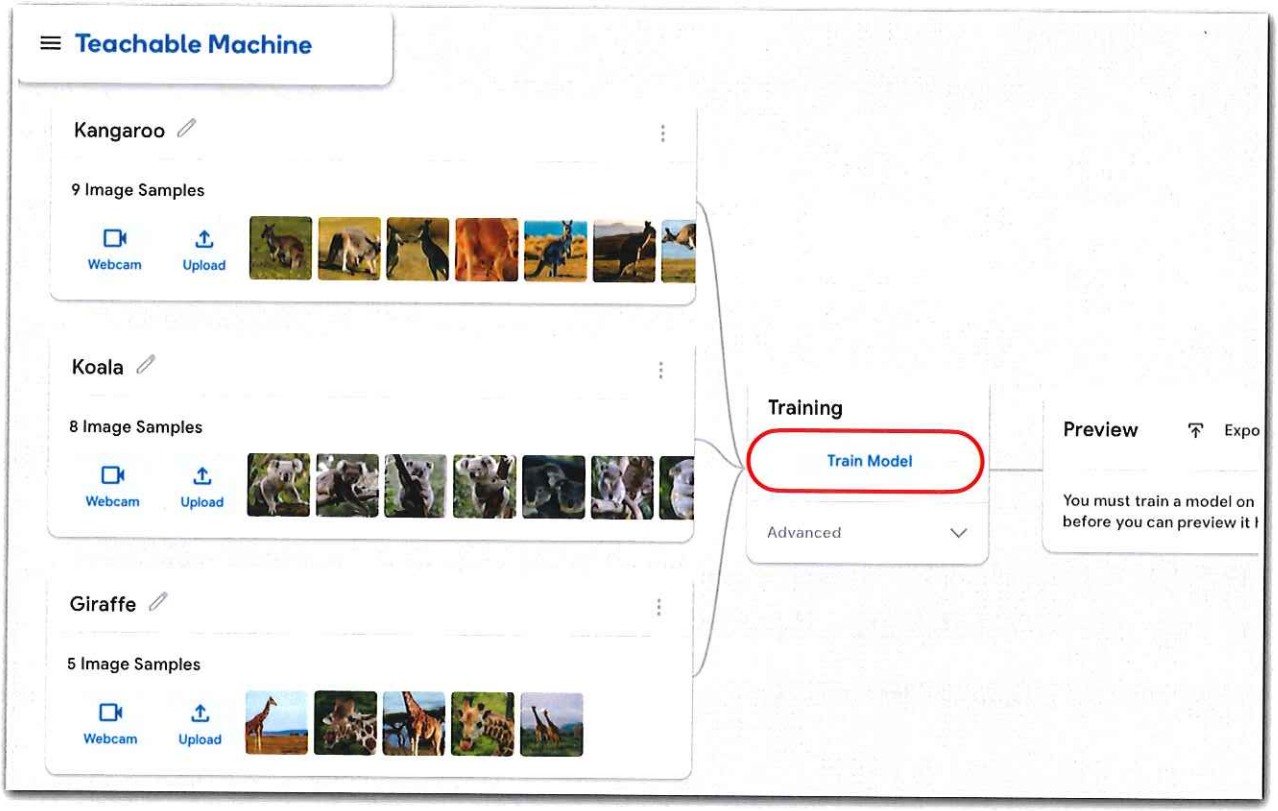
如果要增加第三種類別的圖片可以從下方的 ＋Add a Class 增加更多的圖片類別。

準備好機器學習的圖片資料之後，就可以分類放到 App 中。

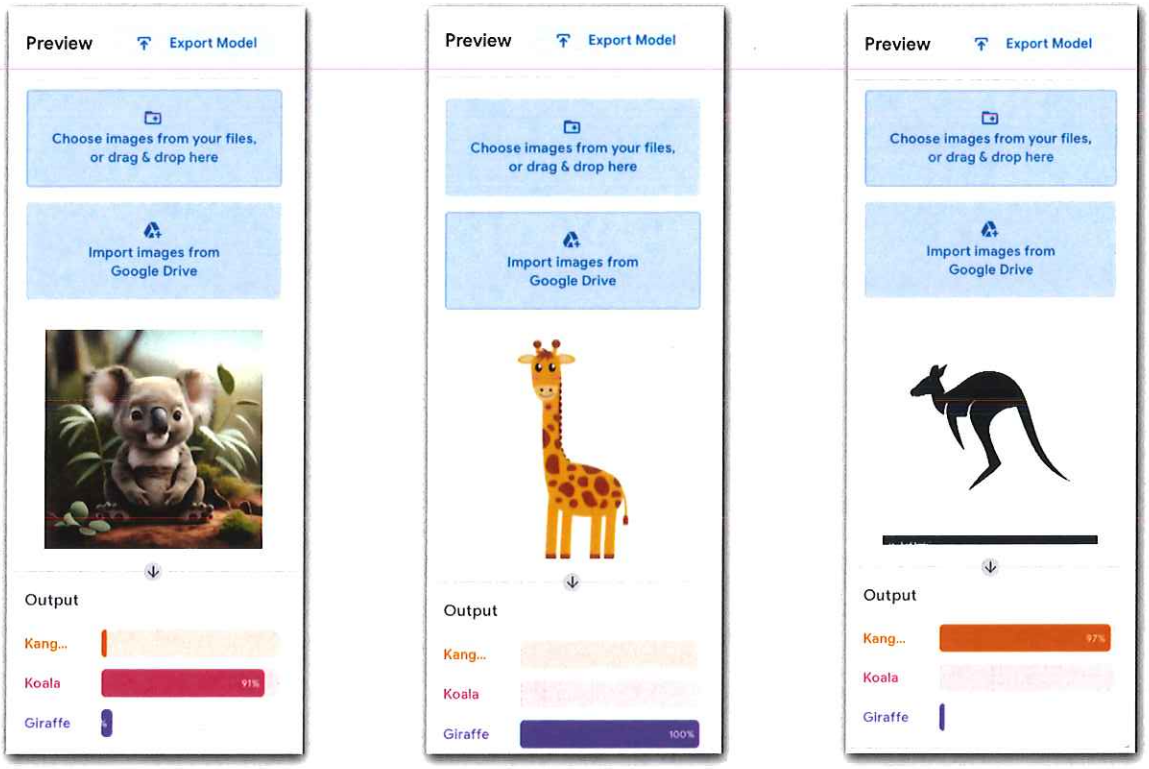
修改類別名稱，從相簿加入圖片資料：



完成後，按下 『 Train Model 』等待數分鐘後，就可以使用訓練好的模型來做圖片判斷。



model 訓練完成後，在 preview 就可以使用這個 model 來判斷圖片資料。



測試訓練的模型是否能成功判斷圖片？

在下一堂課程，我們要使用 Teachable Machine 訓練好的 model ，轉檔後放入我們的圖片判斷 App 中。

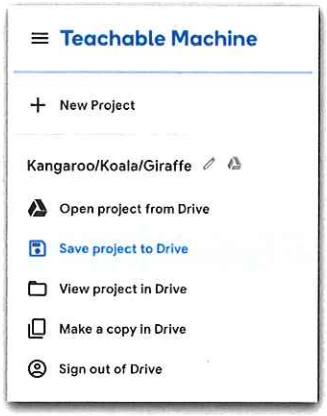
- 目前 App 版本的 Teachable Machine，沒辦法登入 google 帳號將訓練好 Model 儲存，要匯出訓練好的 Model 時會因為檔案太大無法匯出，所以只能在 App 中來使用這個 model 。
- 要下載訓練的 Model 必須使用電腦版瀏覽器的 Teachable Machine 。

訓練自己的機器學習模型

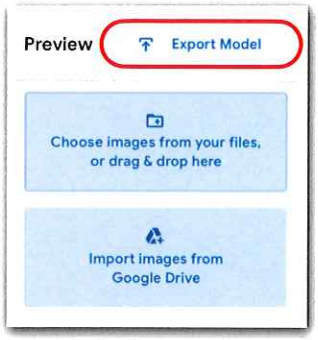
下一堂課前的準備事項

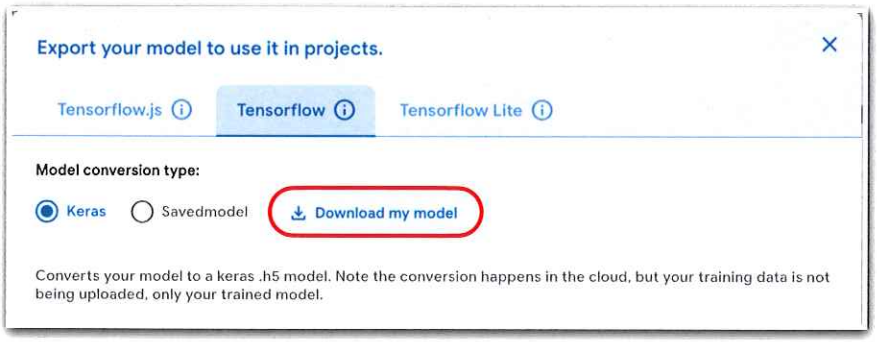
- 注意：以下準備步驟必須由電腦網頁版的 Teachable Machine 來訓練模型，建議使用 Chrome 瀏覽器。

1. 準備至少 2 個類別的圖片資料，各30張。
2. 使用網頁版 Teachable Machine 依照前面的**步驟**完成 model 的訓練。
3. 儲存專案在 google 雲端，方便未來存取或修改你的機器學習模型。



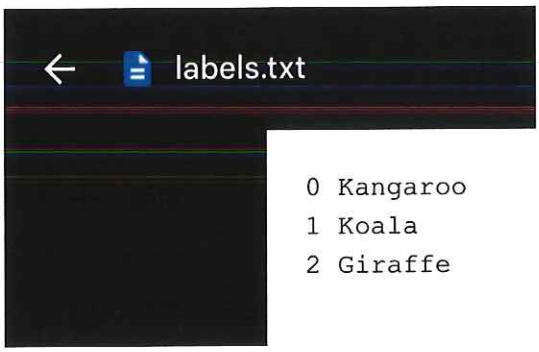
4. 在 preview 選擇 export Model –> TensorFlow –> 選擇 Keras 檔案類型 –> Download my model。下載這個檔案可能會花一點時間。





5. 下載好的檔案是一個壓縮檔，將其解壓縮後，將整個資料夾放至 google 雲端硬碟。

6. 將判斷圖片 model 的類別順序記下來，在 model 轉檔的時候會需要使用，可以從資料



Class 1 _____

Class 2 _____

Class 3 _____

夾中 labels.txt 這個檔案拿到正確的順序。

章節筆記

語法疑問：

教學重點：

反思回饋：