Swift App 進階教師專班

AI 機器學習與設備(BLE & IOT)通訊

Day 2 – 訓練機器學習模型

主辦：新北市政府教育局
日期：2023.07.20(四)
講師：友教有限公司 Michael

Closure 觀念 1

Closure 應用為 Function 參數 1

Delegate 模式 2

Delegate 模式 – Protocol 延伸 3

動手做看看 – 模擬考題 4

Delegate – Protocol App 7

使用 Teachable Machine 訓練的 Model 18

Keras Model to Core ML Model 18

Colab and coremltools 18

加入 AI 圖片辨識 App 24

Create ML 27

Closure 觀念

當我們有 Function Type 變數的觀念，如下的變數：

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

從上方程式，我們可以理解 sum 是一個 function type 變數，我們可以指定 function name 給它，在這裡的例子是 multiply 。

還有另一個寫法就是 closure，如下：

```
sum = { (a:Double, b:Double ) -> Double in
    return a * b
}
```

= 右邊的寫法就叫 Closure ， 最外圍是 { } ，裡面有 in 這個關鍵子 ，in 左邊是 Function Type ， in 右邊要換行，in 和 } 裡面其實就是類似 function 的主題，在這裡可以參考 multiply 這個 function 。

sum 內容不論是 function 還是 closure 使用的時候都是一樣的， 如下：

```
let r1 = sum(5.0,9.0)
print(r1)
```

Closure 應用為 Function 參數

以 function 為例，我們定義名為 calculator 的 function ，第三個參數型別是 Function Type 如下：

```
func calculator(a:Double, b:Double, op:(Double,Double) -> Double ) -> Double {
    return op(a,b)
}
```

定義的時候還不確定 op 的內容是什麼，上面程式碼只是把 a b 分別當做 op 的參數傳入，呼 叫的時候再指定明確的 op 的內容。如下。

```
let cr = calculator(a: 5.0, b: 8.0, op: { (a:Double,b:Double) -> Double in
    return a - b
})
print(cr)
```

如此一來 5.0 和 8.0 就會被傳入當 op 的 a 和 b 的內容，進而運算 a – b 得到 calculator 的回傳值，存入 cr 裡面。

Delegate 模式

委任模式在物件導向設計裡面是一種常見的設計方法，我們用兩個 class 來做介紹。

```
class A {
    var data = "Some Data"
}

class B {
    func gotData(data:String){
        print("Got in from B : " + data)
    }
}
```

現在有 A class 和 B class ，這個兩個 class 初始化後如下：

```
var a = A()
var b = B()
```

如果 b 這個物件，需要拿到 a 裡面 data 的內容，A 需要改寫如下：

```
class A {
    var data = "Some Data"
    var delegate: B?
    func sendData(){
        delegate?.gotData(data: data)
    }
}
```

新增一個 delegate 變數，型別是 B? ， sendData 裡面會利用 delegate 呼叫 B 型別的 gotData 並且把 data 傳入。

使用 A 和 B class 時候就要改寫如下。

```
var a = A()
var b = B()
a.delegate = b
a.sendData()
```

b 才是 B 的物件，也就是真正收到資料的地方，必須放在 a 的 delegate 裡面，然後 a 呼叫 sendData() 就會把值傳給 b。

這個寫法是標準 Delegate 的作法。

Delegate 模式 – Protocol 延伸

基本的 Delegate 模式寫法，不夠彈性。現在 A 裡面 delegate 只能是 B class ，想像一下 A 如果是 iPhone 機器提供 GPS 位置的 class，我們寫 App 畫面的 class 不能只叫 B ，我們要自己定義自己畫面的 class。為了讓 delegate pattern 更有彈性，可以用 protocol 來改寫。

首先我們要訂一個 A class 專用的 protocol：

```
protocol ADelegate {
    func gotData(data:String)
}
```

並且把 A 裡面 delegate B 換成 ADelegate 這個 Type，如下。

```
class A {
    var data = "Some Data"
    var delegate: ADelegate?
    func sendData() {
        delegate?.gotData(data: data)
    }
}
```

同一時間，要接受資料的 class 在這裡 B 要改寫成如下

```
class B: ADelegate {
    func gotData(data: String) {
        print("Got in from B : " + data)
    }
}
```

這個時候資料傳送者 A 和 接受者 B 之間的關係，就用 protocol 來當中介，這樣接受者只要採用 protocol 並實作其宣告的 gotData function 就可以。

使用上基本不變

```
var a = A()
var b = B()
a.delegate = b
a.sendData()
```

動手做看看 – 模擬考題

題目：美化的 Print 結果

問題描述

有個一個 class 名為 Car ，請完整內容使得正確定義 class 不會有錯誤。

```
class Car {
    var name: String
    var year: Int
}
```

修改 class 來讓實體化 Car 得到正確內容，如以下範例。

範例一：

```
var c = Car(name: "Toyota", year: 2000)
print(c.name)
```

輸出結果：

```
Toyota
```

語法筆記：

```
class Car {
  var name: string = " "
  var year: Int = 0

  init(){

  }

  init(name: String, year: Int){
     self.name = name
     self.year = year
  }
}
```

問題描述

承上題，採用 CustomStringConvertible 這個 Swift 的 protocol 來實作。

```
var description: String{ }
```

return 得到正確的 String 格式結果，如以下範例。

範例二：

```
var c = Car(name: "Toyota", year: 2000)
print(c)
```

輸出結果:

```
name: Toyota, year: 2000
```

語法筆記：

```
class Car: CustomStringConvertible {
    ...
    var description: String {
        "name: \(name), year: \(year)"
    }
    ...
}
```

章節筆記

語法疑問：

教學重點：

反思回饋：

Delegate – Protocol App

接下來要使用 Delegate 模式 與 Protocol 協定來實作一個 App 範例，在這個 App 中，會有一個首頁與一個設定使用者資訊的頁面。

『 首頁 』與『 設定頁面 』的資料傳輸方式，使用 Delegate 模式 與 Protocol 協定，完成這個 App 可拆解成以下幾個步驟：

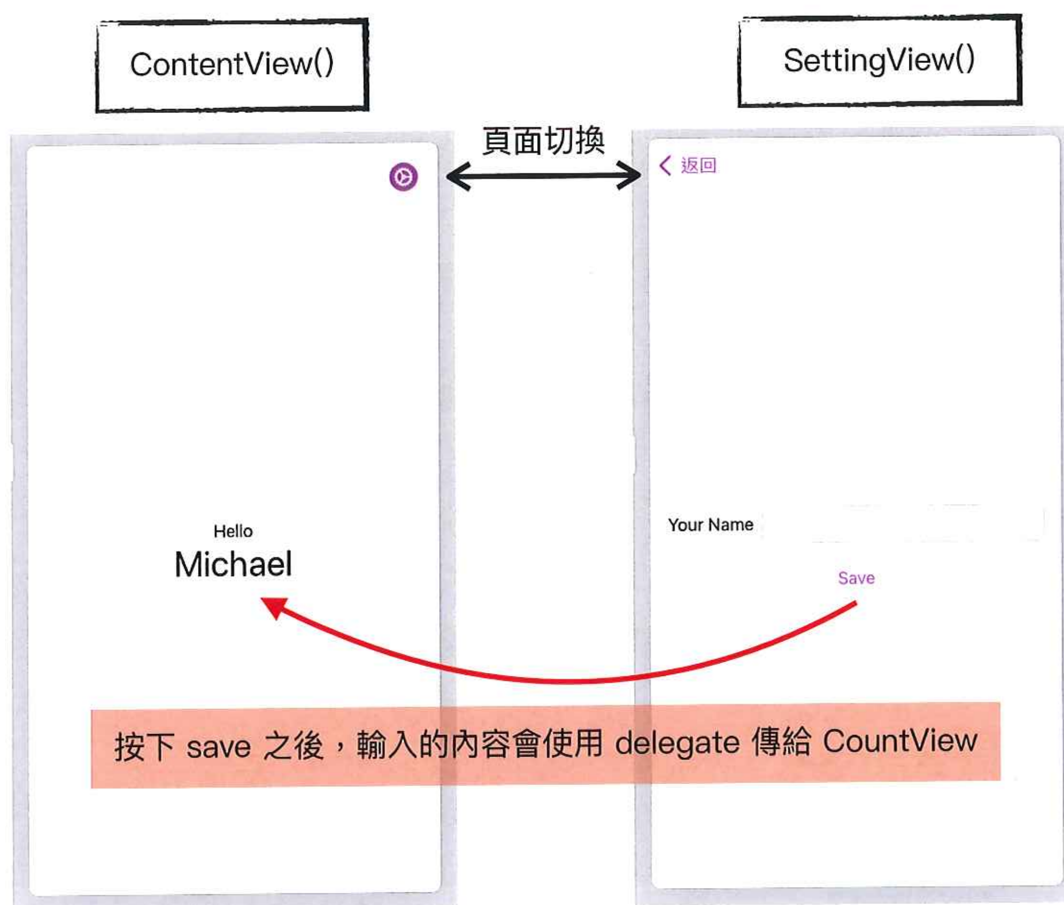
步驟A. 開啟一個新的 App 專案，完成『 首頁 』ContentView 畫面。

步驟B. 新增一個『 設定頁面 』SettingView 的檔案，畫面中可以輸入使用者名稱

步驟C. 使用 Delegate 模式 與 Protocol 協定，在『 設定頁面 』設定傳輸資料。

步驟E. 使用 NavigationView 建立『 首頁 』導往『 設定頁面 』的連結。

步驟D. 在『 首頁 』使用 Delegate 模式 與 Protocol 協定來接收資料，並顯示在畫面上。



左邊是首頁 ContentView ，按了右上角的圖示後，會切換至 SettingView 的畫面，在 SettingView 輸入名字後按下 Save ，輸入的內容會使用 delegate 傳給 ContentView，並顯示在畫面上 ，SettingView 就是 P.4 範例中的 **A**，ContentView 是 P.4 範例中的 **B**。

步驟 A

開啟一個新的 App 專案，來設定『首頁』ContentView 的畫面。

```
import SwiftUI

struct ContentView: View {
    //A1.宣告變數
                                                    A1
    var body: some View {
        VStack {
            //A3.設定頁面連結圖示                        A2 ~ A3

            //A2.首頁畫面

        }
    }
}
```

A1 程式碼

宣告一個變數 name 儲存使用者名字，這個變數會被修改後顯示在畫面上，所以記得在前面加上@State 。

```
@State var name = "Michael"
```

A2 程式碼

在畫面上使用 Text() 畫面元件顯示 Hello 與 A1 程式碼 設定的使用者名稱。

```
Text("Hello")
Text(name).font(.largeTitle)
```

A3 程式碼

在畫面右上角加入齒輪圖形，之後會在這個齒輪圖形上設定前往『設定頁面』。

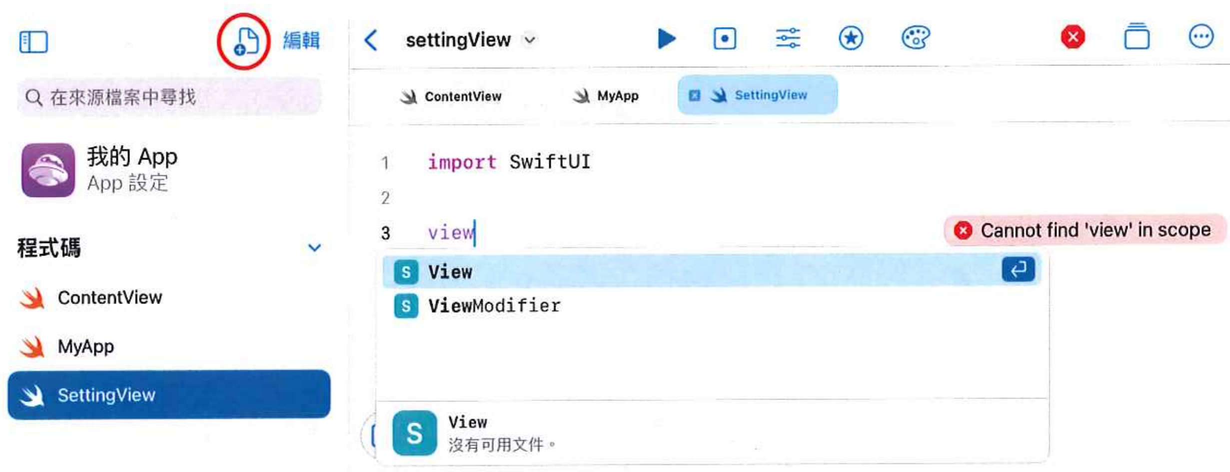
```
HStack{
    Spacer()
    Image(systemName: "gear.circle.fill")
        .font(.title)
        .padding()
}
```

為了讓齒輪可以在右上角呈現，需要加入一個 HStack{ } ，裡面放置一個Spacer() 與齒輪圖像 Image(systemName: "gear.circle.fill") 。

若要讓 A2 程式碼 的內容在畫面中間呈現，可以在前後各加一個 Spacer()

步驟 B

從左上新增一個 Swift 檔案，命名為 SettingView ，在程式碼區域加入一個 view 的結構。



將加入 view 的結構命名為 SettingView ，在此頁面將讓使用者輸入姓名，保留下方的 Preview 結構來預覽目前畫面設定的狀況。

完成『設定頁面』的程式碼，可拆成步驟 B1 ~ B3。

```
import SwiftUI

struct SettingView: View {
    //B1.宣告一個變數來儲存輸入欄位輸入的內容                B1

    var body: some View {
        VStack{
            //B2.加入Text與TextField讓使用者輸入姓名              B2 ~ B3

            //B3.新增一個按鈕來儲存設定，之後會在這邊設定將輸入的使用者姓名傳回『首頁』

        }
    }
}

struct SettingView_Previews: PreviewProvider {
    static var previews: some View {
        SettingView()
    }
}
```

B1 程式碼

宣告一個變數來儲存輸入欄位輸入的內容：

```
@State var inputName = ""
```

B2 程式碼

加入 Text() 與 TextField() 讓使用者輸入姓名，並用一個 HStack{ } 讓這兩個畫面元件水平排列放置：

```
HStack{
    Text("Your Name")
    TextField("", text: $inputName)
        .textFieldStyle(RoundedBorderTextFieldStyle())
}
```

記得在 inputName 的前面加上 $ ，讓 TextField 可以改變這個 inputName 的內容。若要在 TextField 後面加入一個修飾符 .textFieldStyle(RoundedBorderTextFieldStyle())，可以讓 TextField 有一個外框。

B3 程式碼

新增一個按鈕 "Save" 來儲存輸入設定。

目前按鈕內容是空的，之後會在這邊設定將輸入的使用者姓名傳回至『 首頁 』。

```
Button("Save"){

}
```

語法筆記：

步驟 C

使用 Delegate 模式 與 Protocol 協定，在『 設定頁面 』設定傳輸資料。

要將使用者名稱由 SettingView 傳給 ContentView ，SettingView 是資料擁有者也是傳送者，ContentView 是接收者。使用 Delegate 模式來傳資料時，SettingView 的 delegate 是 ContentView 。

可以分為 C1 ~ C3 的步驟：

ContentView SettingView

```
1   import SwiftUI
2
3   //C1. 建立一個 SettingView 的 protocol，裡面有個函數要將設定頁面的資料傳回      C1
4
5   struct SettingView: View {
6       //B1.宣告一個變數來儲存輸入欄位輸入的內容
7       @State var inputName = ""
8       //C2.建立一個 SettingView 接收者 delegate，型別為 SettingViewDelegate?
9                                                                                  C2
10      var body: some View {
11          VStack{
12              //B2.加入Text與TextField讓使用者輸入姓名
13              HStack{
14                  Text("Your Name")
15                  TextField("", text: $inputName)
16                      .textFieldStyle(RoundedBorderTextFieldStyle())
17              } .padding()
18              //B3.新增一個按鈕來儲存設定，之後會在這邊設定將輸入的使用者姓名傳回『首頁』
19              Button("Save"){
20                  //C3.按鈕按下後，執行 delegate 內的方法來傳遞資料。              C3
21
22              }
23          }
24      }
25  }
```

C1 程式碼

用 protocol 來寫 SettingView 專用的 delegate ，命名為 SettingViewDelegate，其中有個函數 sendData 用來將設定頁面的資料傳回。protocol 裡面的 function 沒有大括號，是一個還沒有確定的內容的函數：

```
protocol SettingViewDelegate {
    func sendData(name: String)
}
```

C2 程式碼

建立一個 SettingView 接收者變數，名為 delegate，型別為 SettingViewDelegate? 。

```
var delegate: SettingViewDelegate?
```

在實際寫 App 的時候，有發送者，但是不一定有接收者，所以 delegate 通常會是一個 optional 的型別。例如 iPhone 內的設備有一個發送者在傳出 GPS 位置資料，但如果 App 畫面結構不會用到 GPS 資料時，這時候就不需要有接收者，不會有 delegate 的存在。所以在 iOS 開發中，delegate 通常是 optional 的型別。

C3 程式碼

在按鈕按下時，若有接收者 delegate 存在時，會執行發送使用者名稱的資料 ~~nameInput~~ inputName，若 delegate 不存在 .sendData(name: nameInput) 就不會被執行：

```
delegate?.sendData(name: nameInput)
```

(inputName)

步驟 D

在 ContentView 中，使用 NavigationView 建立『首頁』導往『設定頁面』的連結。

```
//D1.設定頁面使用 NavigationView 來切換頁面
NavigationView {                                         D1
    VStack {
        HStack{
            Spacer()
            //D2. 設定NavigationLink
            NavigationLink(destination: SettingView()) {
                Image(systemName: "gear.circle.fill")
                    .font(.title).padding()
            }                                            D2
        }
        Spacer()
        Text("Hello")
        Text(name).font(.largeTitle)
        Spacer()
    }
}..
```

D1 程式碼 將整個 VStack 包含在一個 NavigationView{ } 中。

D2 程式碼 將 Image(systemName: "gear.circle.fill") 使用一個 NavigationLink { } 包起來，NavigationLink 的目的地 destination 為 SettingView()。測試看看頁面是否可以切換？

步驟 E

『首頁』的 ContentView 為資料接收者，使用 Delegate 模式 與 Protocol 協定來接收資料，並顯示在畫面上。

```
//E1.修改 ContentView ，符合 SettingViewDelegate 協定
struct ContentView: View, SettingViewDelegate {
    @State var name = "Michael"
    @State var isDarkMode = false
    //E2.設定 sendData 的函數內容
    func sendData(name: String) {
        self.name = name
    }
```

E1

E2

E1 程式碼

ContentView 作為資料接收者，必須符合接收者 SettingViewDelegate 的型別。
修改 ContentView，採用 SettingViewDelegate 的協議。

```
struct ContentView: View, SettingViewDelegate {
    //A1.宣告變數
    @State var name = "Michael"
```

E2 程式碼

因為 ContentView 必須符合 SettingViewDelegate 的協議，所以必須要有一個名為 sendData 的函數，否則會出現錯誤。

```
func sendData(name: String) {
    self.name = name
}
```

實作 sendData 的函數內容時，函數內的 name 是使用函數 sendData(name: String) 時傳入的 name，是一個區域變數 local variable。

self 指的是結構本身，也就是 ContentView，而 self.name 則是 ContentView 的 name。

Thinking Time：到設定頁面修改使用者名稱，測試一下首頁是否有收到資料？為什麼沒有收到資料？

修改 D2 程式碼

在 ContentView 內 NavigationLink 中， SettingView() 內加上 delegate: self ，設定 SettingView 的接收者 delegate 為 self，此時的 self 指的是 ContentView 這個結構。

```
//D2. 設定NavigationLink
NavigationLink(destination: SettingView(delegate: self)) {
    Image(systemName: "gear.circle.fill")
        .font(.title).padding()
}
```

測試看看是否能正確傳輸資料了呢？

Challenge：試著實作另外一筆資料，並讓資料內容傳到首頁後，呈現在首頁上。

章節筆記

語法疑問：

教學重點：

反思回饋：

使用 Teachable Machine 訓練的 Model

Keras Model to Core ML Model

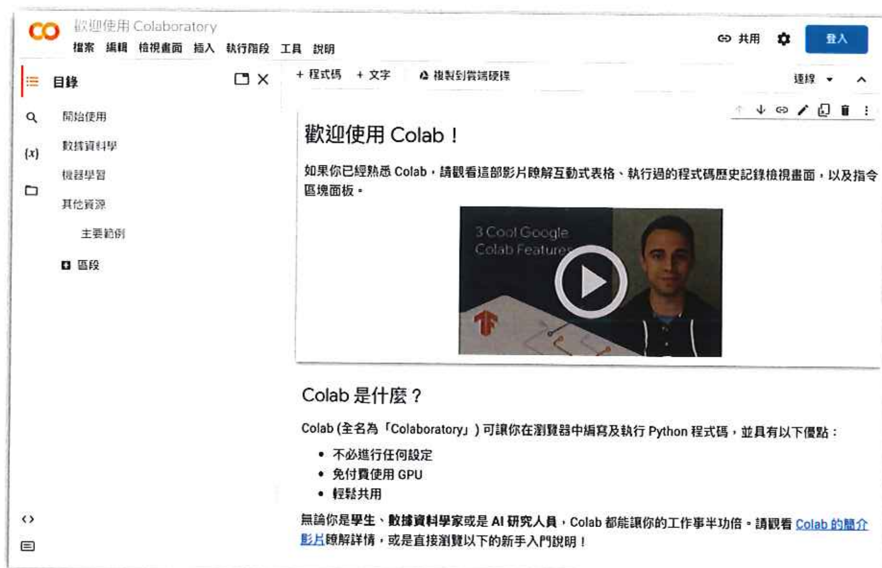
Keras 是一個用於構建和訓練深度學習模型的 Python 框架，而 Apple 使用的機器學習框架為 Core ML，用於在 iOS 和 macOS 設備上運行機器學習模型，Core ML Model 的標準檔案格式為 .mlmodel。

上一堂課中，我們將 Teachable Machine 訓練好的機器學習模型，使用 Keras Model 的格式匯出，檔案格式是 .h5。

要將這個模型放入 SwiftUI 的圖片辨識 App 中使用之前，我們需要將 Keras Model 的 .h5 檔案轉檔為 Core ML Model 的 .mlmodel 檔。我們將在 Colaboratory (Colab) 網頁上使用 coremltools 來完成轉檔。

Colab and coremltools

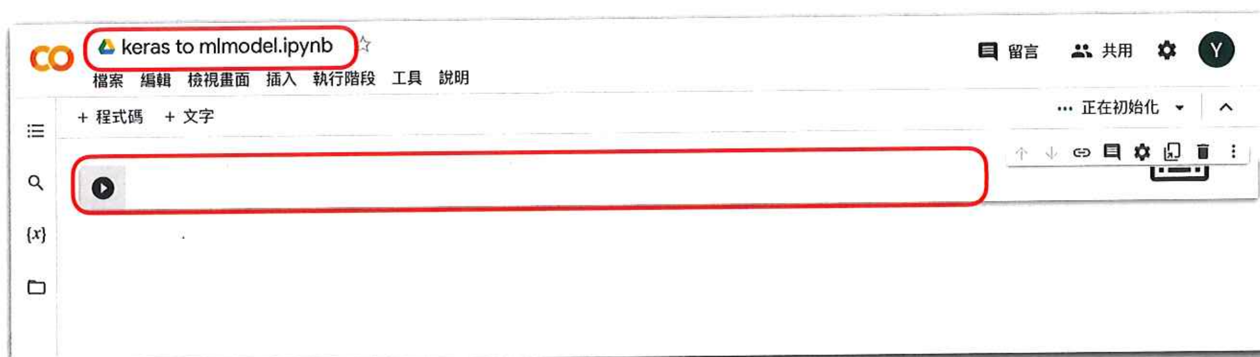
Colaboratory (Colab) 是 Google 提供的一個網頁版 Python 開發環境，它提供強大的計算資源和豐富的工具，在無需安裝任何軟體的情況下，就可以在瀏覽器中開發程式、進行機器學習.....等。 CoreMLTools 是一個 Python 套件，能將各種機器學習模型轉換成Core ML 格式，以便在 Apple 的設備上運行。



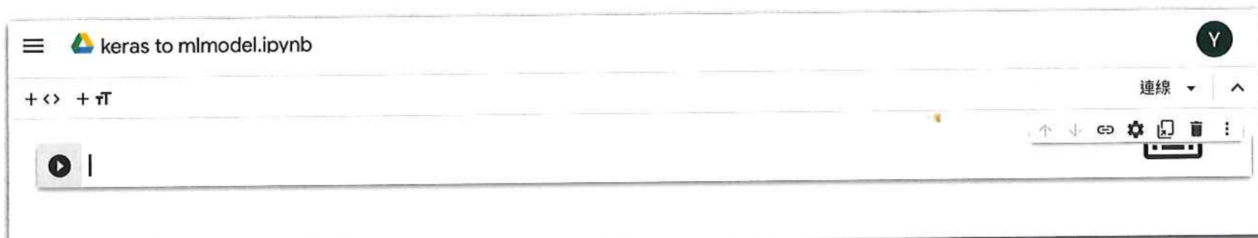
Colaboratory

首先先在按右上角登入Google 帳號，登入完成後會看到以下畫面：

新增記事本後如下圖，可點選上方檔案名字重新命名，下方區塊就是程式碼編輯區，將 python 程式碼寫好後，按下左邊的箭頭即可執行程式碼：

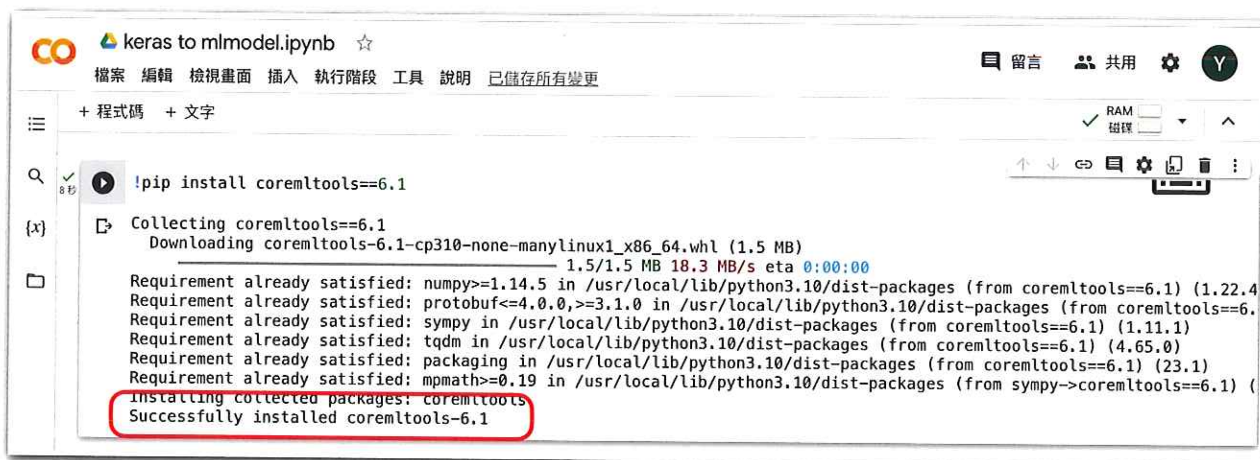


使用不同版本的瀏覽器可能會遇到不同的介面，以上是電腦版 Chrome、iPad 版 Safari 介面，以下是 iPad 版的 Chrome：



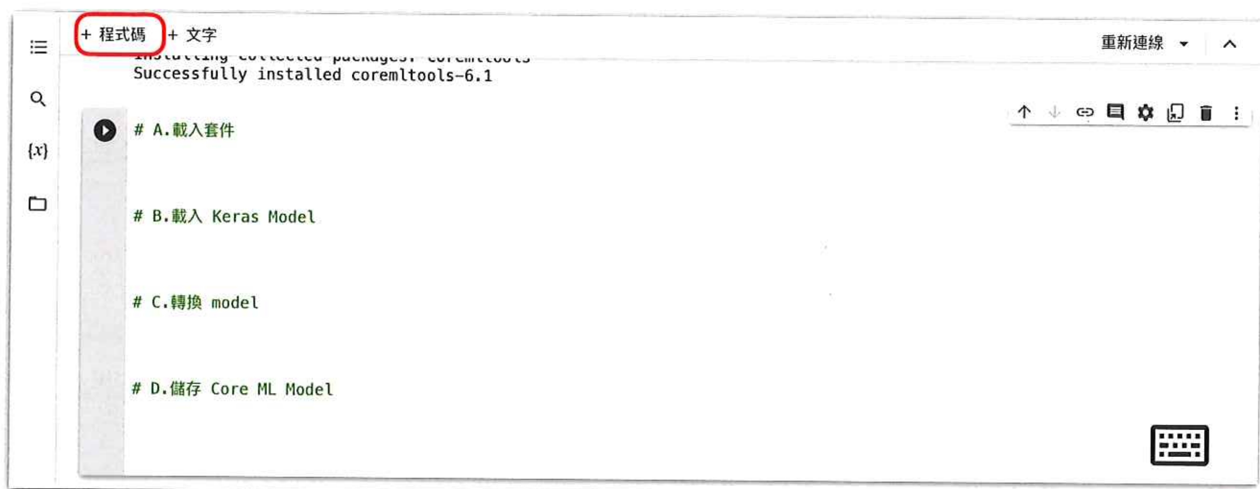
進行轉檔前，需要先安裝 coremltools 版本 6.1 的套件，在程式碼區塊輸入以下程式碼，並按箭頭執行安裝 coremltools：

```
!pip install coremltools==6.1
```



成功安裝後會看到訊息 Successfully installed coremltools–6.1。

新增一個程式碼區塊，接下來進行轉檔的工作，可分為四個步驟：



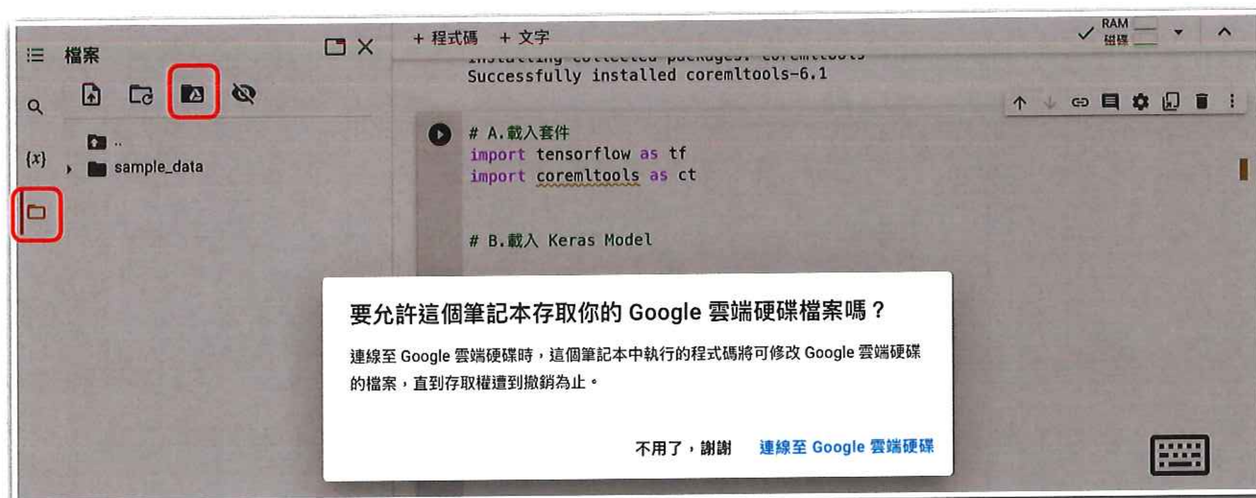
步驟 A. 載入套件

```
import tensorflow as tf
import coremltools as ct
```

載入 tensorflow 與 coremltools 套件，import tensorflow as tf 其中的 as tf 表示載入的 tensorflow 的縮寫名為 tf。

步驟 B. 載入 Keras Model

在載入 model 之前，要先找到放在雲端資料夾下的檔案路徑。開啟左邊的資料夾圖示，並加入雲端資料夾，允許『連線至 Google 雲端硬碟』。



加入雲端資料夾之後，找到 Keras Model 的檔案，副檔名為 .h5，對檔案點一下後出現右邊視窗，點選『 複製路徑 』。

宣告一個變數 keras_file 儲存檔案路徑，並將剛才複製的檔案了路徑『 貼上 』到程式碼區塊，使用雙引號包起來，路徑內容為一個字串。

```
# B.載入 Keras Model
keras_file = "/content/drive/MyDrive/Teachable_Machine_Data/converted_keras/keras_model.h5"
model = tf.keras.models.load_model(keras_file)
```

使用 TensorFlow 的 tf.keras.models.load_model 函數讀取 keras_file 檔案，儲為 model 是一個 TensorFlow Model 的物件。

此外可以使用以下程式碼印出 model 的資訊，

```
for layer in model.layers:
    print('Layer Name:', layer.name)
    print('Input Shape:', layer.input_shape)
```

印出的資訊如下，在轉檔的時候需要知道 Layer 的名稱：

```
Layer Name: sequential_1
Input Shape: (None, 224, 224, 3)
Layer Name: sequential_3
Input Shape: (None, 1280)
```

步驟 C. 轉換 model

將 TensorFlow 模型轉換為 Core ML 模型時，需要明確指定輸入的資料類型和形狀：

```
image_input =
ct.converters.mil.input_types.ImageType(name='sequential_1_input',
shape=(1,224, 224, 3))
```

定義標籤類別 class_labels，內容為一個陣列，值為你的 Model 內的類別，注意這邊陣列的順序必須按照 labels.txt 的順序。使用類別標籤做定義一個分類器 classifier_config：

```
class_labels = ["Kangaroo","Koala","Giraffe"]
classifier_config = ct.ClassifierConfig(class_labels)
```

使用 ct.convert 方法將 TensorFlow 模型轉換為 Core ML 模型。將 TensorFlow Model、定義好的輸入資料類型，以及分類器配置傳遞給這個方法：

```
coreml_model =
ct.convert(model, inputs=[image_input], classifier_config = classifier_config)
```

步驟 D. 儲存 Core ML Model

使用 coreml_model.save 的方法儲存 mlmodel ：

```
coreml_model.save('MyImageModel.mlmodel')
```

按下執行完成轉檔：

＋程式碼　＋文字

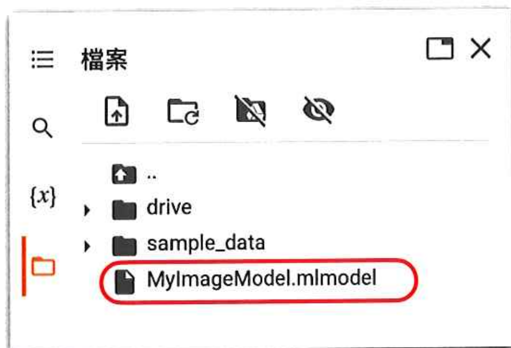
```
Successfully installed coremltools-6.1

# A.載入套件
import tensorflow as tf
import coremltools as ct


# B.載入 Keras Model
keras_file = "/content/drive/MyDrive/Teachable_Machine_Data/converted_keras/keras_model.h5"
model = tf.keras.models.load_model(keras_file)

# C.轉換 model
image_input = ct.converters.mil.input_types.ImageType(name='sequential_1_input', shape=(1,224, 224, 3))
class_labels = ["Kangaroo","Koala","Giraffe"]
classifier_config = ct.ClassifierConfig(class_labels)
coreml_model = ct.convert(model, inputs=[image_input], classifier_config = classifier_config)


# D.儲存 Core ML Model
coreml_model.save('MyImageModel.mlmodel')
```

```
WARNING:coremltools:scikit-learn version 1.2.2 is not supported. Minimum required version: 0.17. Maximum required versi
WARNING:coremltools:XGBoost version 1.7.6 has not been tested with coremltools. You may run into unexpected errors. XGB
WARNING:coremltools:TensorFlow version 2.12.0 has not been tested with coremltools. You may run into unexpected errors.
WARNING:coremltools:Torch version 2.0.1+cu118 has not been tested with coremltools. You may run into unexpected errors.
Running TensorFlow Graph Passes: 100%|██████████| 6/6 [00:00<00:00, 16.07 passes/s]
Converting TF Frontend ==> MIL Ops: 100%|██████████| 431/431 [00:00<00:00, 1114.91 ops/s]
Running MIL Common passes: 100%|██████████| 39/39 [00:01<00:00, 30.26 passes/s]
Running MIL Clean up passes: 100%|██████████| 11/11 [00:00<00:00, 105.88 passes/s]
Translating MIL ==> NeuralNetwork Ops: 100%|██████████| 496/496 [00:00<00:00, 5732.89 ops/s]
```

✓ 25 秒　完成時間：下午2:42

成功轉檔後，會在資料夾看到轉檔完成的 mlmodel 。

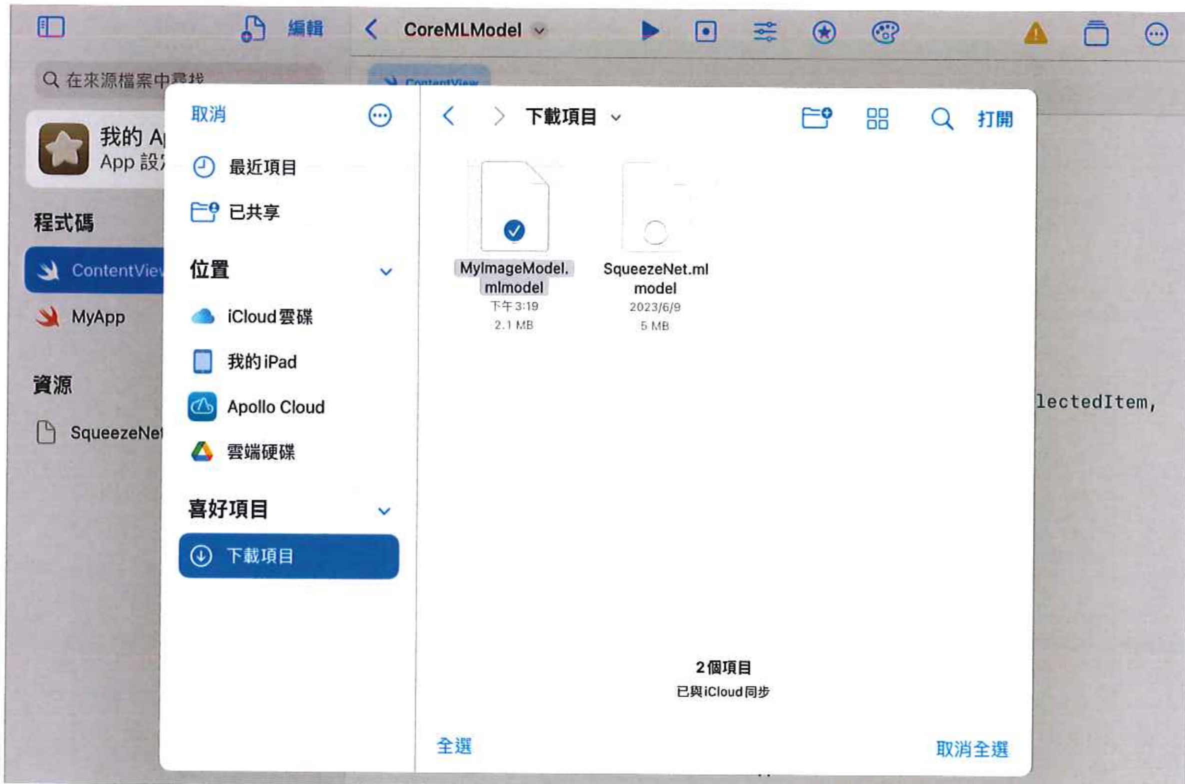
點選檔案將 mlmodel 下載至 iPad 的檔案內。



檔案轉檔完成後，我們要將這個自己訓練的機器學習模型，放入上一堂課堂中的『 AI 圖片辨識 App 』來使用。

加入 AI 圖片辨識 App

打開前一堂課的圖片辨識 App ，從左上角加入轉檔完下載至 iPad 的 mlmodel 。



成功加入 mlmodel 後，在左方可以看的你的 model。

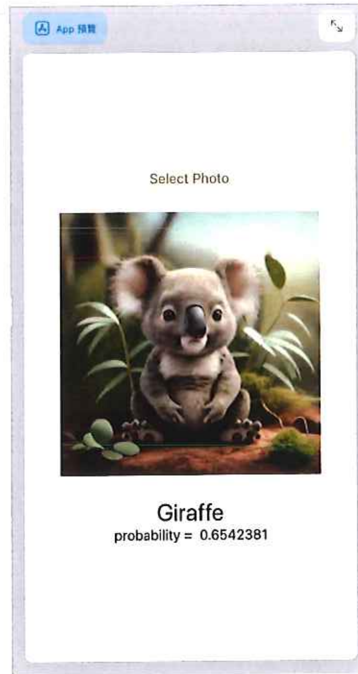
修改程式碼內載入 modelName：

程式碼
- ContentView
- MyApp

資源
- MyImageModel.mlmodel
- SqueezeNet.mlmodel

```
51      //步驟C.建立一個 function，使用 ML Model 判別已選取的照片。
52
53      func predictImage() {
54          //C1.確保圖片資料不為空值
55          guard let selectedImage = selectedImage else {
56              return
57          }
58          //C2.設定正確的 Model 名稱與的 URL 路徑。
59          let modelName = "MyImageModel"
60          guard let modelURL = Bundle.main.url(forResource:
         modelName, withExtension: "mlmodel") else {
61              print("error loading model")
62              return
63          }
```

測試看看圖片辨識的結果。



這時候會發現轉檔後的模式，準確度降低很多，在『Teachable Machine』網頁上可以判別正確的圖片，轉檔後無法正確判斷。

章節筆記

語法疑問：

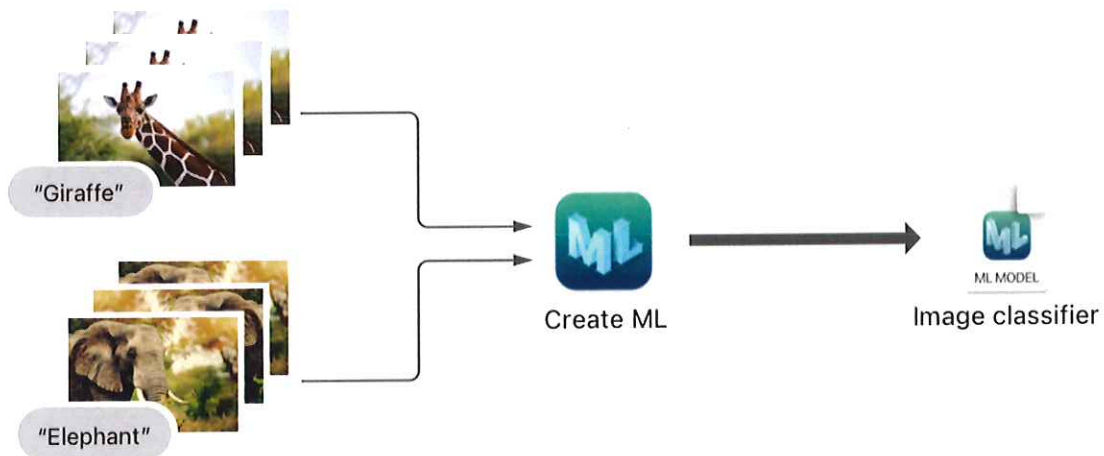
教學重點：

反思回饋：

Create ML

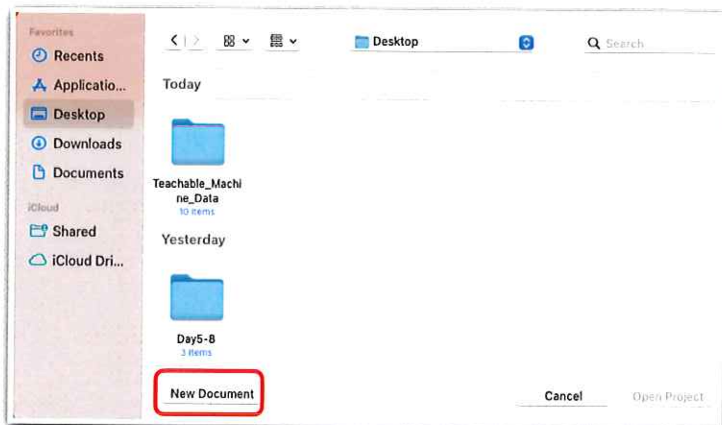
轉檔後的機器學習模型很容易不準確，可能是因為數據處理方式與模型架構.....等差異，所以有不一樣的表現結果。因此如果要在 Apple 裝置運行機器學習模型，使用工具直接訓練 mlmodel 會更好，不會因為轉檔造成模式準確度降低。

在 Mac 上的 Xcode 中，可以使用 Create ML 工具來訓練自己 Core ML models。

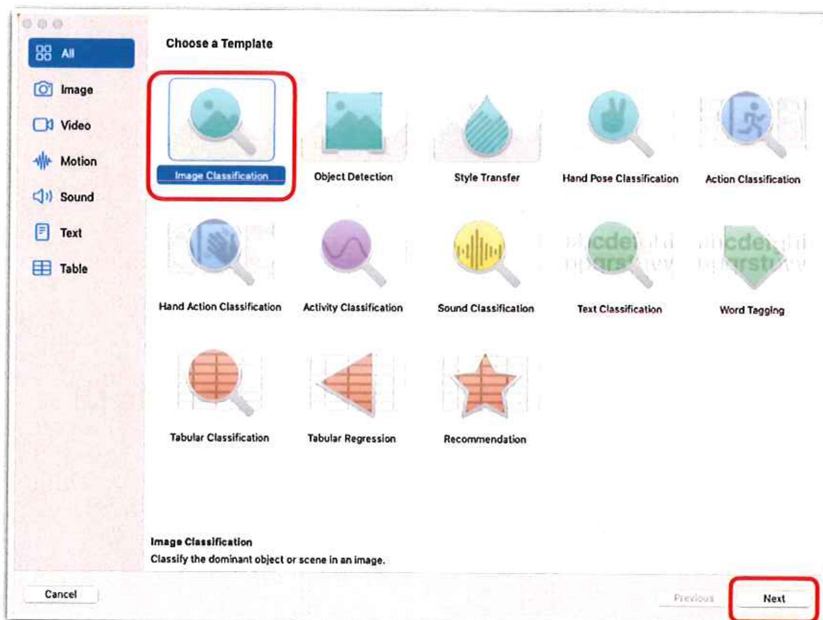


在 Xcode 左上工具列 Xcode > Open Developer Tool > Create ML 開啟 Create ML：

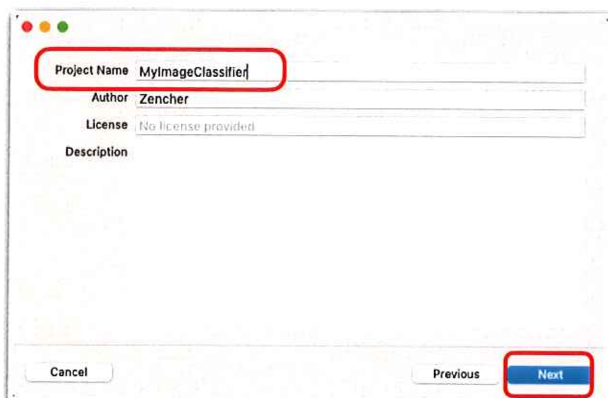
新增一個檔案 New Document：



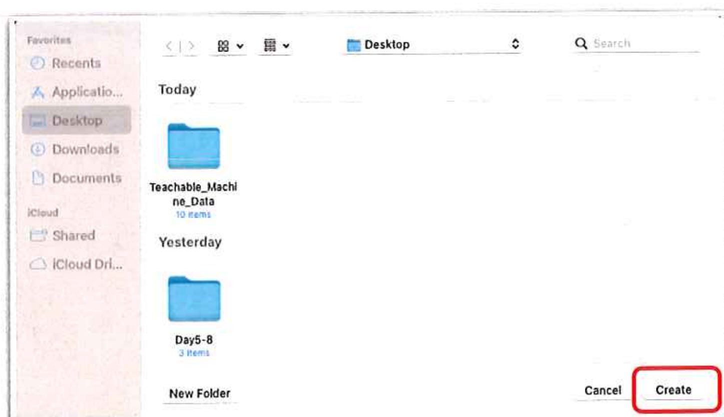
選擇圖片分類 Image Classification ：



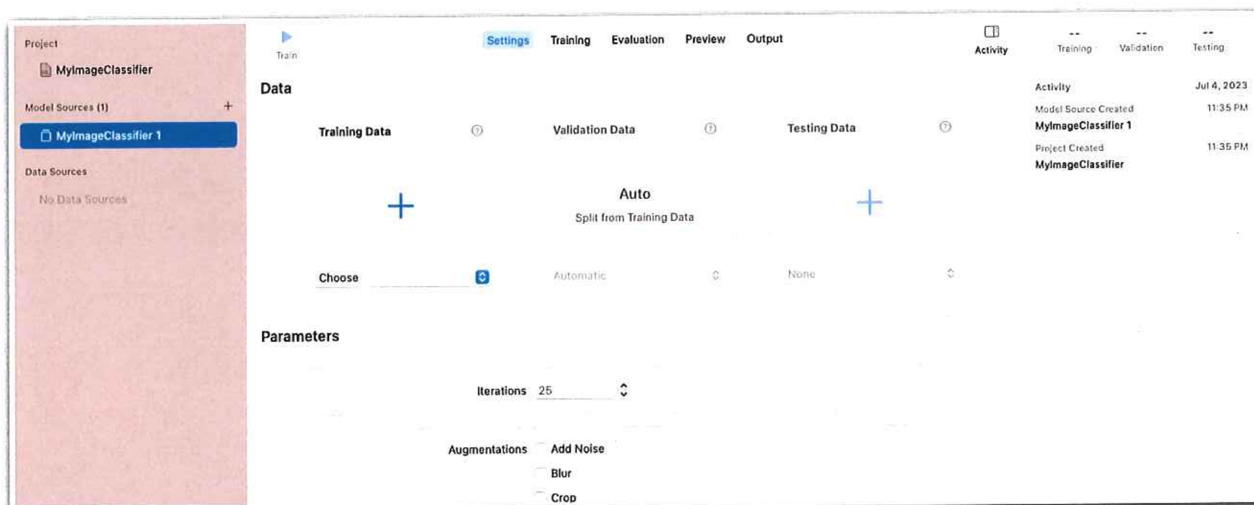
命名專案名稱：



選擇儲存位置，建立專案：



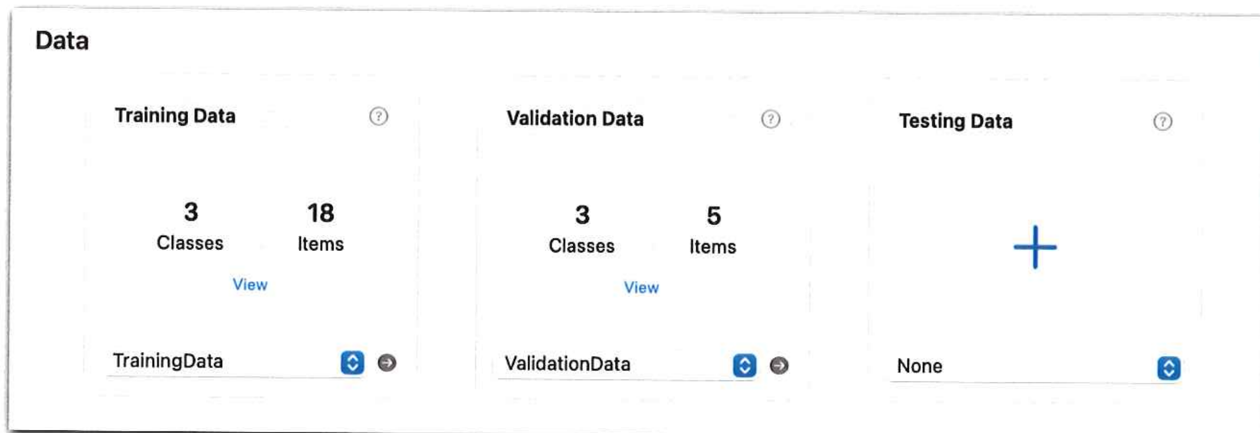
接下來會進入到訓練模型的設定頁面：



在開始訓練模型之前，要先將圖片資料放置到正確的資料夾結構。在 TrainingData 與 ValidationData 資料夾，裡面分別有三個資料夾，以類別名為資料夾名稱：

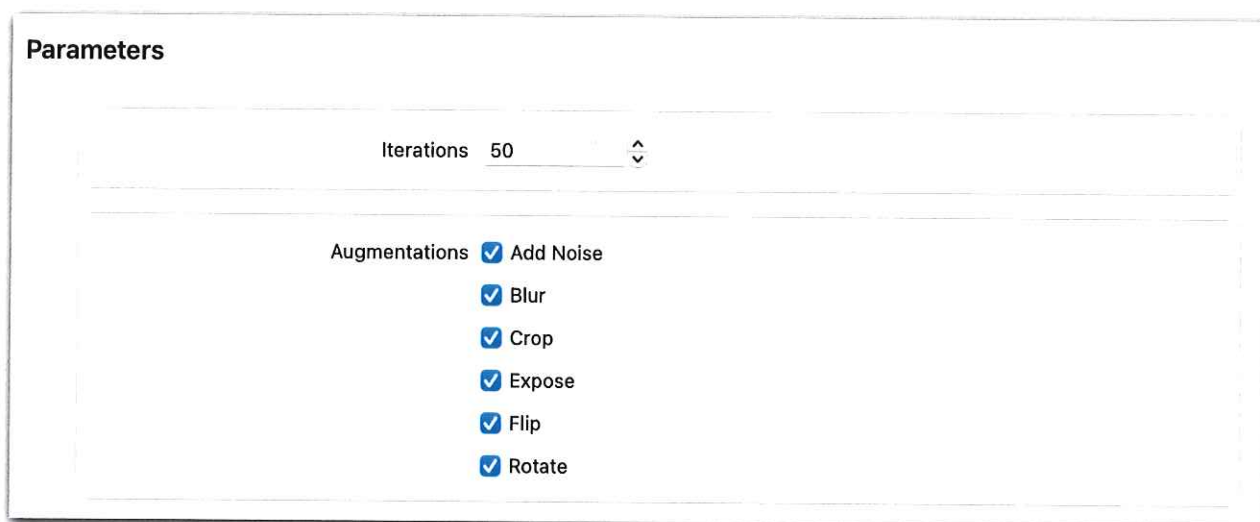
建議一個類別有 30 張的圖片資料，並要訓練的圖片資料， 80% 放在 TrainingData 底下的資料夾內，剩下的放在 ValidationData 資料夾。建議每個類別的照片數量要差不多，這樣可以提高模型的準確度。

接著回到 Create ML ，在 Training Data 與 Validation Data 的資料，分別選擇正確的資料夾。



訓練模型的資料只會使用 Training Data 與 Validation Data 的資料，而 Testing Data 是訓練完成這個模型後，用來測試模式的準確度，可以先不用放。

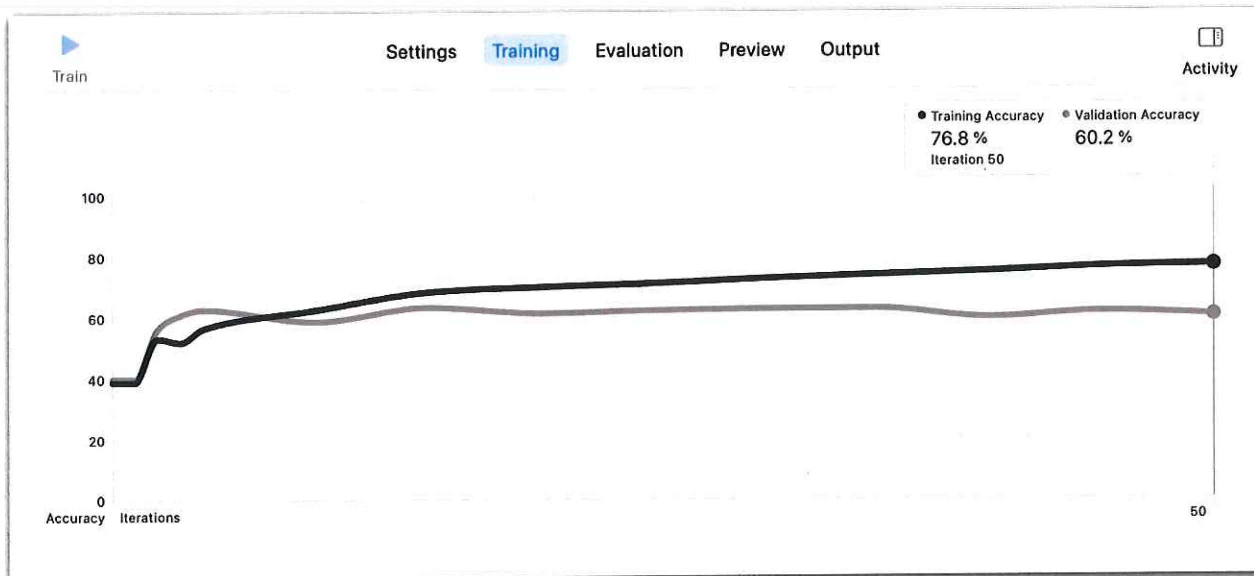
接下來要做一些參數的設定，Iterations 是迭代次數。迭代次數是用來設定模型要訓練幾次，在這個迭代的過程中，模式會去調整權重改善預測結果，恰當的訓練次數可以提高模式的精確度，過多的迭代次數會造成過度學習。



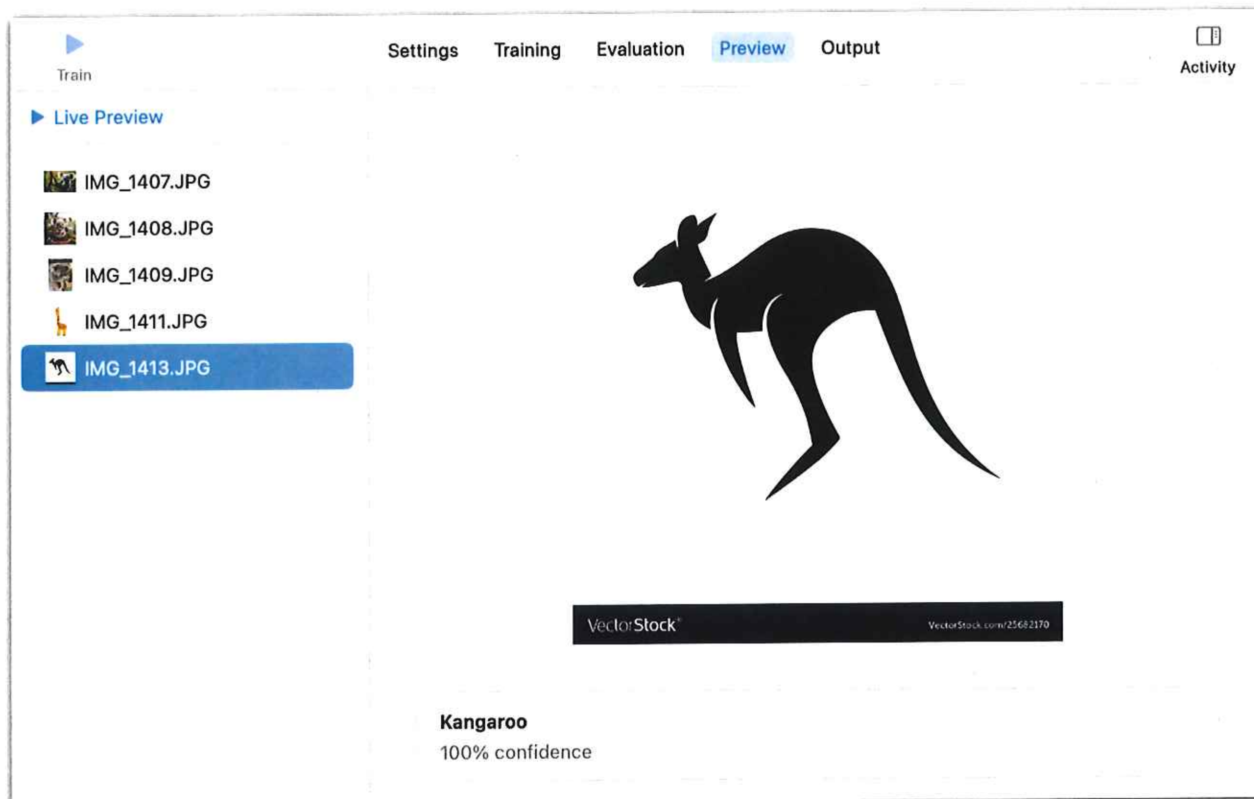
Augmentations 代表數據增強，在原始的圖片資料上，使用一些隨機變化（如旋轉、縮放、剪切、翻轉、添加雜訊等），產生更多的圖片資料來訓練模式，可以提高模式的性能。

接著就可以按下左上角 ▶ 開始訓練模型。

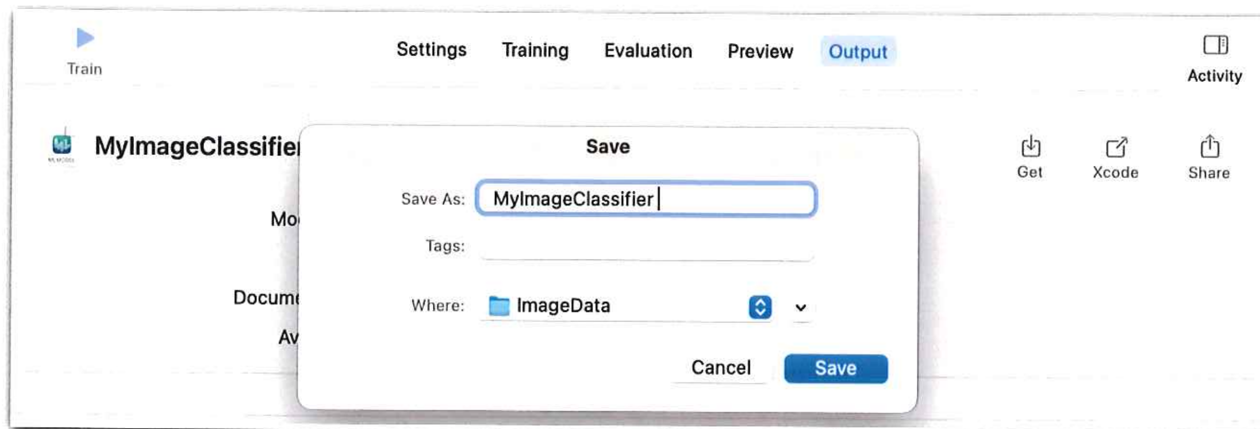
訓練完成後會看到這樣的圖：



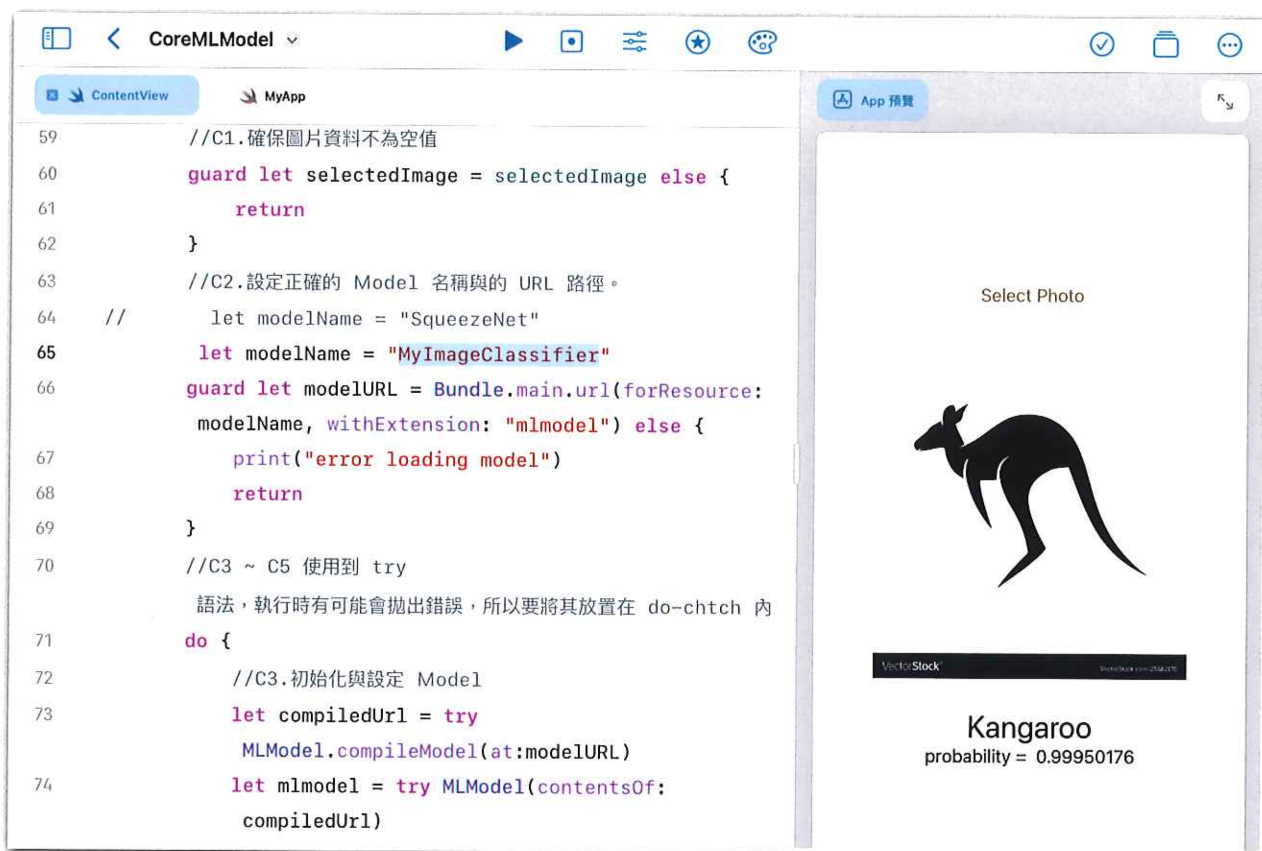
在 Create ML 內，可以直接預覽訓練模型的狀況，切換到預覽模式，將要辨別的圖片拖進資料區，點選就可以看到模式判斷的結果，如下圖：



接著就可以把訓練好的模型存檔匯出。 Output > Get > Save 來儲存 Model：



最後可以把這個 mlmodel 放到圖片辨識 App 中使用，看圖片判別結果。

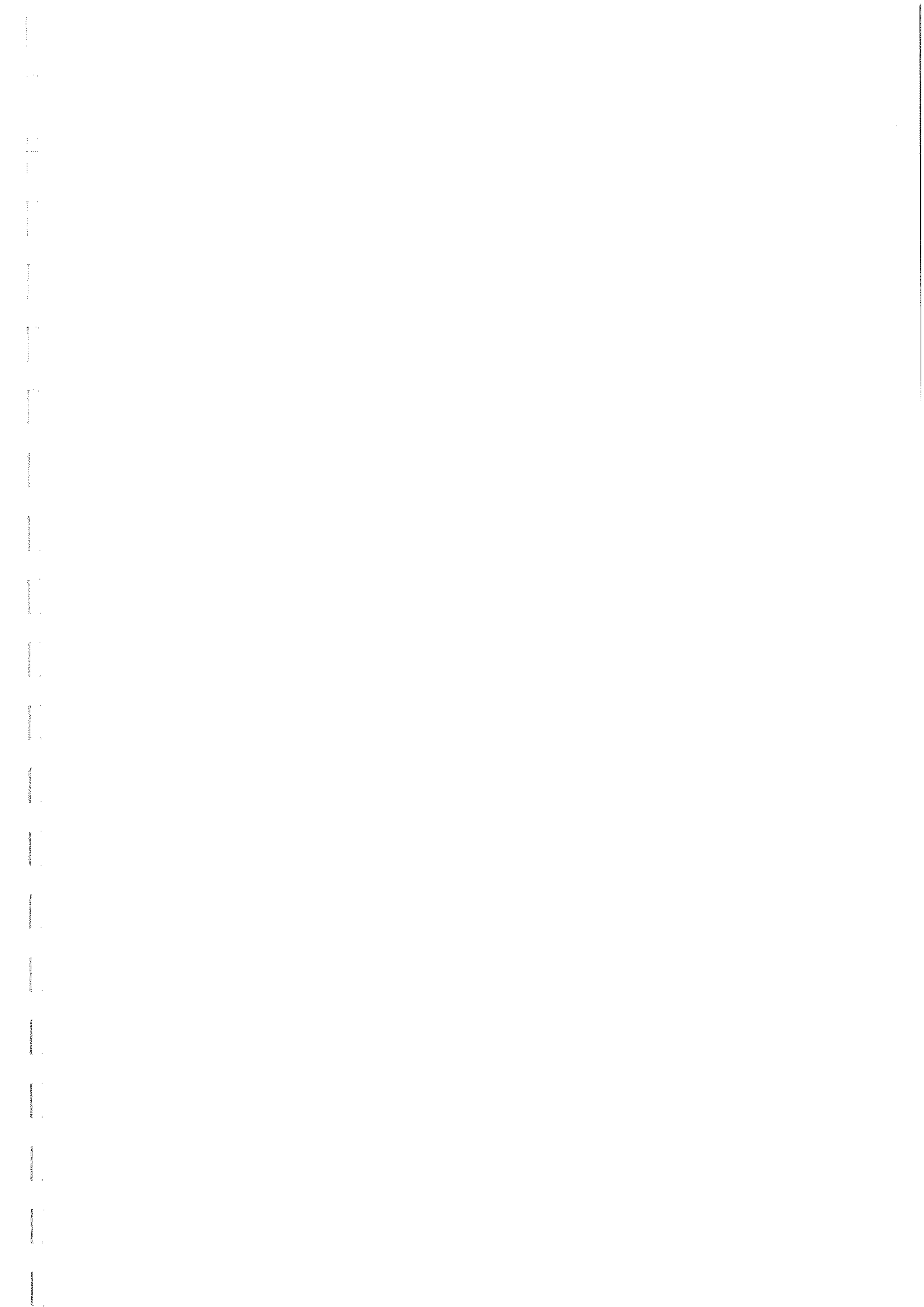


章節筆記

語法疑問：

教學重點：

反思回饋：



作者：Michael Pan
版權：Zencher Co. Ltd. 2023, 並分享給所有與會老師使用