Sample Code

Classifying Images with Vision and Core ML

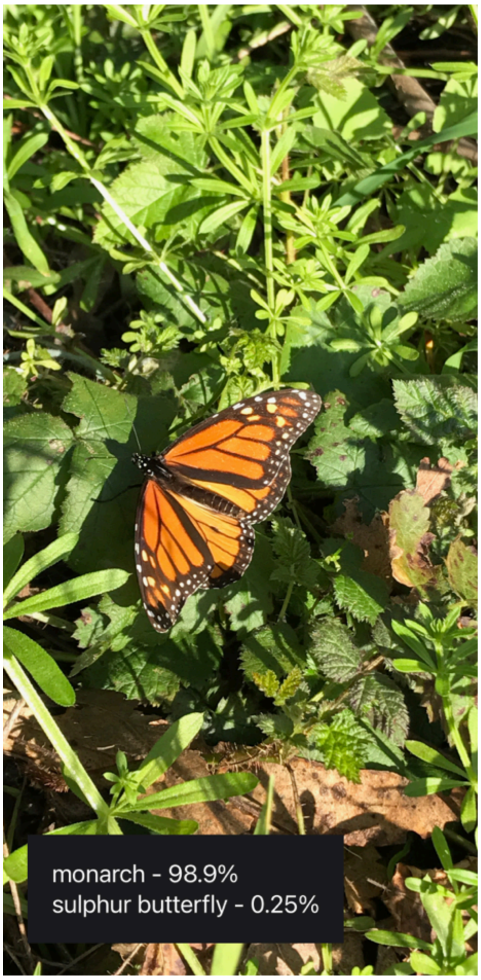
Crop and scale photos using the Vision framework and classify them with a Core ML model.

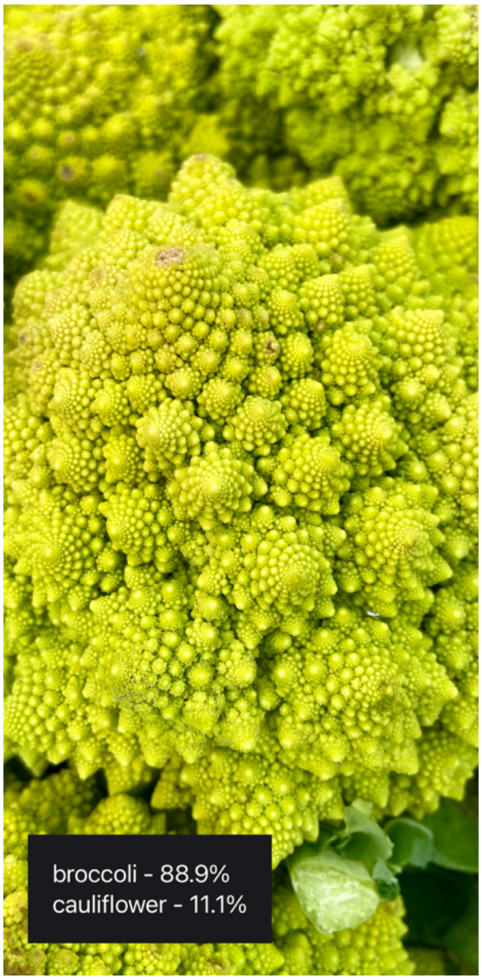
Download

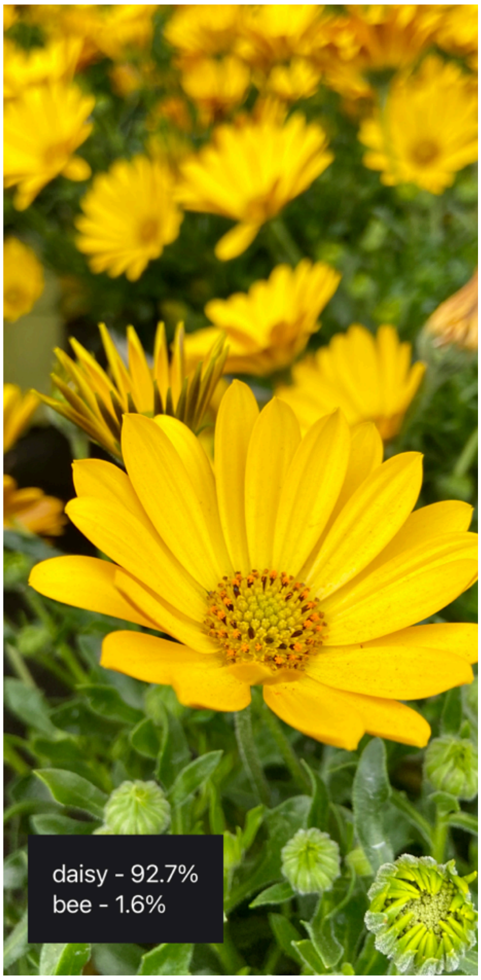
iOS 14.0+ | iPadOS 14.0+ | Xcode 13.4+ | iPad 14.0+

Overview

The app in this sample identifies the most prominent object in an image by using MobileNet, an open source image classifier model that recognizes around 1,000 different categories.







Each time a user selects a photo from the library or takes a photo with a camera, the app passes it to a Vision image classification request. Vision resizes and crops the photo to meet the MobileNet model's constraints for its image input, and then passes the photo to the model using the Core ML framework behind the scenes. Once the model generates a prediction, Vision relays it back to the app, which presents the results to the user.

The sample uses MobileNet as an example of how to use a third-party Core ML model. You can download open source models — including a newer version of MobileNet — on the Core ML model gallery.

Before you integrate a third-party model to solve a problem — which may increase the size of your app — consider using an API in the SDK. For example, the Vision framework's VNClassifyImageRequest class offers the same functionality as MobileNet, but with potentially better performance and without increasing the size of your app (see Classifying Images for Categorization and Search).

Note

You can make a custom image classifier that identifies your choice of object types with Create ML. See Creating an Image Classifier Model to learn how to create a custom image classifier that can replace the MobileNet model in this sample.

Configure the sample code project

The sample targets iOS 14 or later, but the MobileNet model in the project works with:

- iOS 11 or later
- macOS 10.13 or later

To take photos within the app, run the sample on a device with a camera. Otherwise, you can select photos from the library in Simulator.

> Note
>
> Add your own photos to the photo library in Simulator by dragging photos onto its window.

Create an image classifier instance

At launch, the `ImagePredictor` class creates an image classifier singleton by calling its `createImageClassifier()` type method.

The method creates a Core ML model instance for Vision by:

1. Creating an instance of the model’s wrapper class that Xcode auto-generates at compile time
2. Retrieving the wrapper class instance’s underlying ‘MLModel’ property
3. Passing the model instance to a `VNCoreMLModel` initializer

The Image Predictor class minimizes runtime by only creating a single instance it shares across the app.

> Note
>
> Share a single `VNCoreMLModel` instance for each Core ML model in your project.

Create an image classification request

The Image Predictor class creates an image classification request — a `VNCoreMLRequest` instance — by passing the shared image classifier model instance and a request handler to its initializer.

The method tells Vision how to adjust images that don’t meet the model’s image input constraints by setting the request’s `imageCropAndScaleOption` property to `centerCrop`.

Create a request handler

The Image Predictor's `makePredictions(for photo, ...)` method creates a `VNImageRequestHandler` for each image by passing the image and its orientation to the initializer.

Vision rotates the image based on `orientation` — a `CGImagePropertyOrientation` instance — before sending the image to the model.

If the image you want to classify has a URL, create a Vision image request handler with one of these initializers:

- `VNImageRequestHandler(url:options:)`
- `VNImageRequestHandler(url:orientation:options:)`

Start the Request

The [`makePredictions(for photo, ...)`][makePredictions] method starts the request by adding it into a `VNRequest` array and passes it to the handler's `perform(_:)` method.

> Note
>
> You can perform multiple Vision requests on the same image by adding each request to the array you pass to the `perform(_:)` method's `requests` parameter.

Retrieve the request's results

When the image classification request is finished, Vision notifies the Image Predictor by calling the request's completion handler, `visionRequestHandler(_:error:)`. The method retrieves the request's `results` by:

1. Checking the `error` parameter
2. Casting `results` to a `VNClassificationObservation` array

The Image Predictor converts each result to `Prediction` instances, a simple structure with two string properties.

The method sends the `predictions` array to the Image Predictor's client — the main view controller — by calling the client's completion handler.

Format and present the predictions

The main view controller's `imagePredictionHandler(_:)` method formats the individual predictions into a single string and updates a label in the app's UI using helper methods.

The `updatePredictionLabel(_:)` helper method safely updates the UI by updating the label's text on the main dispatch queue.

> **Important**
>
> Keep your app's UI responsive by making predictions with Core ML models off of the main thread.

See Also

Image classification models

{} Using Core ML for semantic image segmentation

Identify multiple objects in an image by using the DEtection TRansformer image-segmentation model.

{} Detecting human body poses in an image

Locate people and the stance of their bodies by analyzing an image with a PoseNet model.

{} Understanding a Dice Roll with Vision and Object Detection

Detect dice position and values shown in a camera frame, and determine the end of a roll by leveraging a dice detection model.