Sample Code

Classifying images for categorization and search

Analyze and label images using a Vision classification request.

Download

macOS 15.0+ | Xcode 16.0+

Overview

With the help of machine learning to perform image analysis, the Vision framework can identify, recognize, and label objects in an image you provide. The `ClassifyImageRequest` structure generates the image analysis results in the form of `ClassificationObservation` objects. Each object contains a classification label (such as *bicycle*), and an associated confidence value to indicate the level of certainty in the observation.

This sample app demonstrates how to set up and perform the classification request on a single image or on a collection of images. For a collection of images, the sample shows you how to process the images concurrently using Swift concurrency. Finally, the sample shows how you access and display the results from the request.

Perform the request

First, the app creates the `ClassifyImageRequest` and performs it on an image. The request returns the entire taxonomy as `ClassificationObservation` objects, and stores the objects in an array. For each observation, the request assigns a label in the form of an identifier, along with a floating-point confidence value in the range of `0.0` to `1.0`. A value of `0.0` indicates that the observation is not likely in the image, while a value of `1.0` indicates that the observation is certain to be in the image.

When the request returns the results, the app performs a high-recall `filter(_:)`, and only retains the observations that meet a confidence threshold. This threshold is set by the app with the `hasMinimumPrecision(_:forRecall:)` method from `Vision`. A high-recall filter provides a much broader range of observations, but can result in more false positive results. For example, with this filter, an ambiguous image of a bear may result in an observation with the label *dog*.

If an app can't tolerate false positive results, the `hasMinimumRecall(_:forPrecision:)` method allows for a high-precision filter. A high-precision filter retains a smaller number of observations, but less chance to contain false positives. Increasing precision decreases recall, and increasing recall decreases precision. Testing can help determine the balance point that returns the best results for a specific use case.

The app stores the final results in the `observations` dictionary. The dictionary's key is the identifier (the label of the observation), and its value is the confidence level.

```swift
// Returns an `ImageFile` object based on the `ClassifyImageRequest` results.
func classifyImage(url: URL) async throws -> ImageFile {
    var image = ImageFile(url: url)

    // Vision request to classify an image.
    let request = ClassifyImageRequest()

    // Perform the request on the image, and return an array of `ClassificationObser
    let results = try await request.perform(on: url)
        // Use `hasMinimumPrecision` for a high-recall filter.
        .filter { $0.hasMinimumPrecision(0.1, forRecall: 0.8) }
        // Use `hasMinimumRecall` for a high-precision filter.
        // .filter { $0.hasMinimumRecall(0.01, forPrecision: 0.9) }

    // Add each classification identifier and its respective confidence level into t
    for classification in results {
        image.observations[classification.identifier] = classification.confidence
    }

    return image
}
```

Processing a large collection of images can take time, so the app uses Swift concurrency to help with speed and efficiency. Using `TaskGroup`, the app processes the images in parallel instead of sequentially in a loop. When the request returns results, the app adds each image's observations to an array. For more information on using Swift concurrency, see Concurrency.

```swift
func classifyAllImages(urls: [URL]) async throws -> [ImageFile] {
    var images = [ImageFile]()

    try await withThrowingTaskGroup(of: ImageFile.self) { group in
        for url in urls {
            group.addTask {
                return try await classifyImage(url: url)
            }
        }

        for try await image in group {
            images.append(image)
        }
    }

    return images
}
```

Retrieve and search the results

The sample provides the ability to search for images by their classification labels. If the search bar is empty, the app presents all the images. If the search bar is not empty, the app only presents images with classification labels equal to the search term.

```swift
var searchResults: [ImageFile] {
    if searchTerm.isEmpty {
        // If the search bar is empty, keep all of the images available.
        return images
    } else {
        // The only images that are available are those that contain classification
        return images.filter({ $0.observations.keys.contains(searchTerm) })
    }
}
```

Clicking an image navigates to the results view and displays the results of the image analysis. Using a `ForEach` loop, the app iterates through the observations. The `sorted(by:)` method sorts the observations in descending order of their confidence levels. The `ForEach` loop accesses the values of the observation with the dictionary’s key and value keywords. Note that any confidence values of `0.0` are greater than zero due to rounding.

```
List {
    if image.observations.isEmpty {
        Text("No observations found with significant confidence.")
            .font(.title2)
            .padding(10)
    }

    ForEach(image.observations.sorted(by: { $0.value > $1.value }), id: \.key) { key
        Text("\(value, specifier: "%.2f"): \(key.capitalized)")
            .font(.title2)
            .padding(10)
    }
}
```

See Also

Still-image analysis

`struct ClassifyImageRequest`

A request to classify an image.

`protocol ImageProcessingRequest`

A type for image-analysis requests that focus on a specific part of an image.

`class ImageRequestHandler`

An object that processes one or more image-analysis requests pertaining to a single image.

`protocol VisionRequest`

A type for image-analysis requests.

`protocol VisionObservation`

A type for objects produced by image-analysis requests.

`struct DetectLensSmudgeRequest`

A request that detects a smudge on a lens from an image or video frame capture.

`struct SmudgeObservation`

An observation that provides an overall score of the presence of a smudge in an image or video frame capture.