Sample Code

Personalizing a Model with On-Device Updates

Modify an updatable Core ML model by running an update task with labeled data.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 15.2+

Overview

With the Core ML framework, you can customize an updatable model at runtime on the user's device. Using this technique, you can create a personalized experience for the user while keeping their data private.

This sample demonstrates how to update the drawing classifier with an `MLUpdateTask`. The app initiates an update task with the user's drawings paired with a string label. Once the update is complete, the app uses the updated drawing classifier to recognize similar drawings from the user and convert each into its associated string label.

> **Note**
>
> Run this sample on a device or Simulator with iOS 13 or later, or iPadOS 13 or later.

Prepare your model update data

Gather your training data:

1. Wrap each value of a datapoint in an `MLFeatureValue`, one for each model input and output.

2. Group all the feature values for a datapoint in an `MLFeatureProvider`.

3. Group all the feature providers in an MLBatchProvider.

Each time the user adds a new emoji sticker, the app prompts the user to make three drawings, and uses those drawings to update the drawing classifier. It does this by first creating an MLDictionaryFeatureProvider that contains the feature values for a drawing and its label. The app appends each feature provider to an array, which it uses to create an MLArrayBatchProvider at the end of the function.

```swift
  var featureProviders = [MLFeatureProvider]()

  let inputName = "drawing"
  let outputName = "label"

  for drawing in trainingDrawings {
      let inputValue = drawing.featureValue
      let outputValue = MLFeatureValue(string: String(emoji))

      let dataPointFeatures: [String: MLFeatureValue] = [inputName: inputValue,
                                                         outputName: outputValue]

      if let provider = try? MLDictionaryFeatureProvider(dictionary: dataPointFeature
          featureProviders.append(provider)
      }
  }

return MLArrayBatchProvider(array: featureProviders)
```

The sample makes each MLDictionaryFeatureProvider by initializing it with a dictionary of two MLFeatureValue instances keyed by strings. The feature values are:

- The underlying image of the drawing keyed by "drawing"
- The emoji character as a string keyed by "label"

The sample creates a feature value for the emoji string by using init(string:). However, to convert the drawing's underlying CGImage into a feature value, the sample acquires the image constraint of the model's image input feature.

```swift
let imageFeatureValue = try? MLFeatureValue(cgImage: preparedImage,
                                            constraint: imageConstraint)
return imageFeatureValue!
```

The sample gets the drawing classifier’s "drawing" MLImageConstraint by inspecting the MLModelDescription.

```
/// - Tag: ImageConstraintProperty
extension UpdatableDrawingClassifier {
    /// Returns the image constraint for the model's "drawing" input feature.
    var imageConstraint: MLImageConstraint {
        let description = model.modelDescription

        let inputName = "drawing"
        let imageInputDescription = description.inputDescriptionsByName[inputName]!

        return imageInputDescription.imageConstraint!
    }
}
```

Create an update task

You create an MLUpdateTask by passing the following to an initializer:

- An MLBatchProvider that contains your update data
- The location of the compiled model you’d like to update (*ModelName*.mlmodelc)
- An MLModelConfiguration, if applicable
- A completion handler with a single MLUpdateContext parameter

The sample updates the drawing classifier model it’s currently using, which could be the original drawing classifier model or a previously updated model.

```
// Create an Update Task.
guard let updateTask = try? MLUpdateTask(forModelAt: url,
                                   trainingData: trainingData,
                                   configuration: nil,
                                   completionHandler: completionHandler)
    else {
        print("Could't create an MLUpdateTask.")
        return
}
```

Important

An update task can only update a *compiled* model file—one whose name ends with `.mlmodelc`.

Run the update task

You begin an update task by calling its `resume()` method.

```
updateTask.resume()
```

Core ML updates the model on a separate thread and calls your completion handler when it finishes the update process.

Save the updated model

Use your completion handler to save the updated model in the `MLUpdateContext` to disk. The sample saves the updated model to the file system by first writing the model to a temporary location. Next, the sample moves the updated model to a permanent location, replacing any previously saved updated model.

```
let updatedModel = updateContext.model
let fileManager = FileManager.default
do {
    // Create a directory for the updated model.
    try fileManager.createDirectory(at: tempUpdatedModelURL,
                                    withIntermediateDirectories: true,
                                    attributes: nil)

    // Save the updated model to temporary filename.
    try updatedModel.write(to: tempUpdatedModelURL)

    // Replace any previously updated model with this one.
    _ = try fileManager.replaceItemAt(updatedModelURL,
                                      withItemAt: tempUpdatedModelURL)

    print("Updated model saved to:\n\t\(updatedModelURL)")
} catch let error {
    print("Could not save updated model to the file system: \(error)")
    return
}
```

Load the updated model

Use your updated model by loading it with the model's `init(contentsOf:)` initializer. The sample loads a new instance of `UpdatableDrawingClassifier` with the URL of the updated model file the app saved in the previous step.

```swift
guard FileManager.default.fileExists(atPath: updatedModelURL.path) else {
    // The updated model is not present at its designated path.
    return
}

// Create an instance of the updated model.
guard let model = try? UpdatableDrawingClassifier(contentsOf: updatedModelURL) else
    return
}

// Use this updated model to make predictions in the future.
updatedDrawingClassifier = model
```

See Also

On-device model updates

`class MLTask`

An abstract base class for machine learning tasks.

`class MLUpdateTask`

A task that updates a model with additional training data.