A
Graduate Project

REPORT ON
RAILROAD BOXCAR COREML

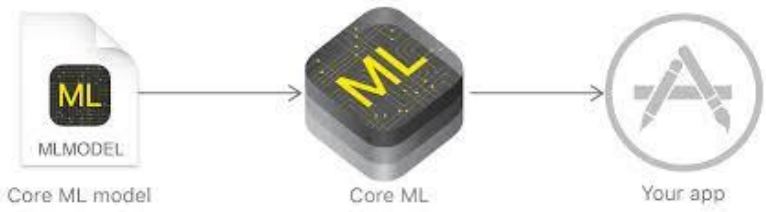
By
Jahnavi Bombothula

Table of Contents

Introduction

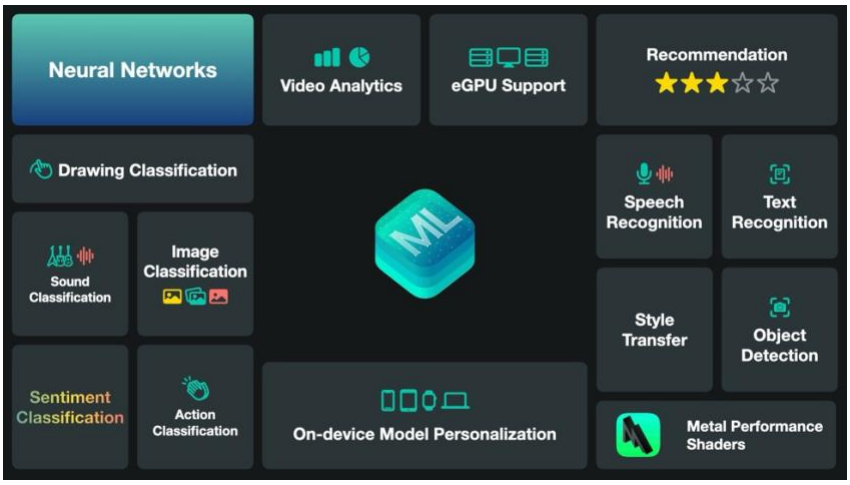
Core ML offers a powerful solution for integrating machine learning models seamlessly into your app. It provides a unified representation for all models, allowing your app to leverage Core ML APIs and user data for predictions, training, or fine-tuning models directly on a person's device.

A model, resulting from applying a machine learning algorithm to training data, serves as the foundation for making predictions based on new input data. These models can accomplish a wide range of tasks, from categorizing photos to detecting objects within images, tasks that would be challenging or impractical to achieve solely through code.

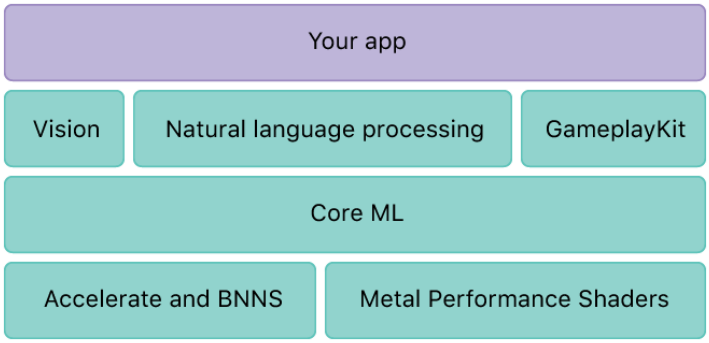


With Core ML, you can build and train models using the Create ML app bundled with Xcode, ensuring compatibility with the Core ML model format. Alternatively, you can utilize various machine learning libraries and then convert the model into the Core ML format using Core ML Tools.

Once deployed on a person's device, Core ML enables on-device retraining or fine-tuning of models using that person's data. Leveraging the CPU, GPU, and Neural Engine, Core ML optimizes on-device performance while minimizing memory footprint and power consumption. Furthermore, running models strictly on the device eliminates the need for a network connection, ensuring data privacy and app responsiveness.

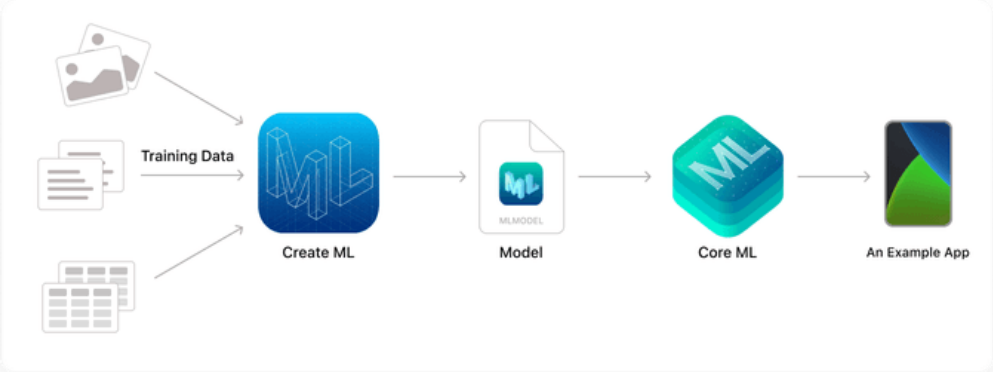


Core ML serves as the foundation for domain-specific frameworks such as Vision for image analysis, Natural Language for text processing, Speech for audio-to-text conversion, and Sound Analysis for identifying sounds in audio. Built upon low-level primitives like Accelerate and BNNS, as well as Metal Performance Shaders, Core ML empowers developers to implement advanced machine learning capabilities efficiently within their apps.

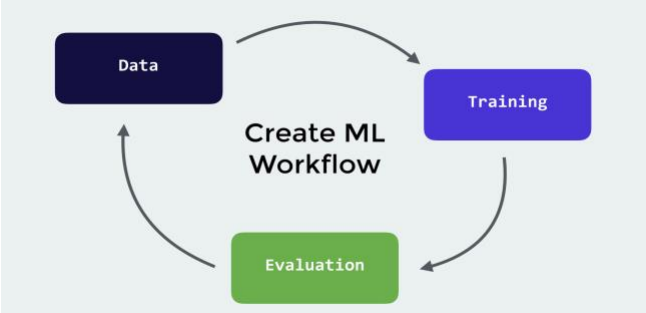


Create ML

Create ML is a powerful tool that enables developers to create and train custom machine learning models directly on their Mac using familiar tools like Swift and macOS playgrounds. With Create ML, you can train models to perform various tasks such as recognizing images, extracting meaning from text, or finding relationships between numerical values.



The process of training a model involves showing it representative samples to recognize patterns. For example, to train a model to recognize dogs, you would provide it with numerous images of different dog breeds. Once trained, the model is tested on unseen data to evaluate its performance. When the model achieves satisfactory performance, it can be seamlessly integrated into your app using Core ML.



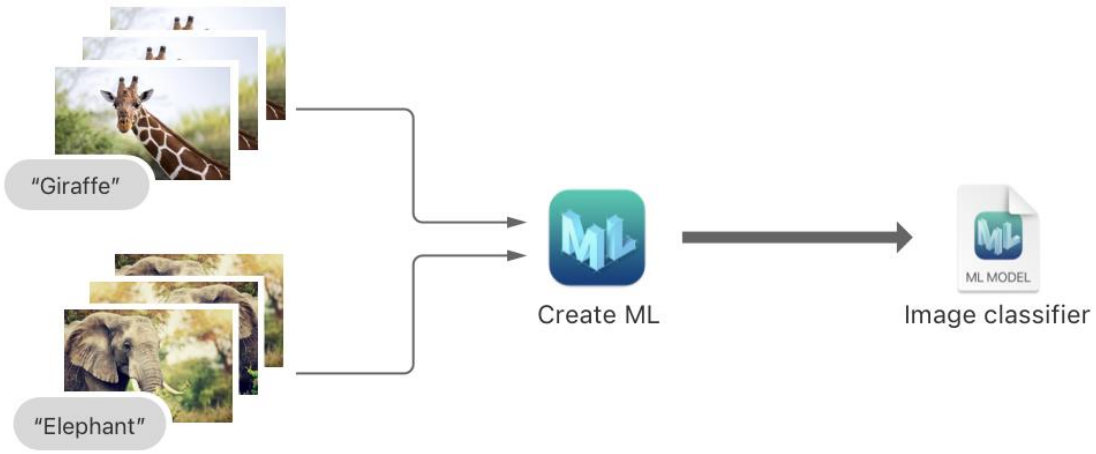
Create ML leverages the machine learning infrastructure built into Apple products like Photos and Siri, resulting in smaller model sizes and significantly reduced training times. This integration with existing Apple technologies enhances efficiency and makes it easier for developers to incorporate machine learning capabilities into their apps.

How to create an Image Classifier Model

An image classifier is a type of machine learning model designed to recognize images and assign them to specific categories. By providing an image to the classifier, it can accurately predict the category label associated with that image.

To train an image classifier, you need to supply it with a large dataset of labeled images. For instance, to train a classifier to identify animals, you would gather images of elephants, giraffes, lions, and other animals.

Once the image classifier has been trained using these labeled examples, you evaluate its accuracy. If the classifier performs satisfactorily, you save it as a Core ML model file. This file can then be imported into your Xcode project, allowing you to seamlessly integrate the image classifier into your app.



Gather Your Data

When creating your image dataset for training an image classifier, it's essential to adhere to certain guidelines to ensure optimal performance:

1. Include a minimum of 10 images per category to provide sufficient diversity for effective training. Remember that a more diverse set of images leads to better classifier performance. Consider capturing images from various angles and under different lighting conditions to enhance model robustness.

2. Maintain a balanced distribution of images across categories to prevent bias in model training. Avoid scenarios where one category has significantly more images than others, as this can skew the classifier's performance.

3. Acceptable image formats include JPEG and PNG, which can be opened in Quicktime Player. The images do not need to be of a specific size, but it's recommended to use images that are at least 299 x 299 pixels to ensure clarity and detail.

4. Ideally, select images that closely resemble the scenarios in which the model will be deployed. For example, if your app classifies outdoor scenes captured by a camera, gather outdoor images captured by a similar camera to provide accurate training data.

By following these guidelines, you can create a comprehensive image dataset that effectively trains your image classifier for accurate and reliable performance in your application.

Organize Your Training Data

To prepare your training dataset, it's essential to organize your images into subfolders, with each subfolder representing a distinct category of images. Assign meaningful names to these subfolders corresponding to the categories they represent. For instance, you could label a subfolder as "Cheetah" to contain all images of cheetahs.

Organize Your Testing Data

Testing your model with a separate testing dataset provides valuable insights into its real-world performance. If your dataset contains a sufficient number of images, typically 25 or more per category, create a testing dataset by duplicating the folder structure of the training dataset. Then, transfer approximately 20 percent of the images from each category into the corresponding category folder within the testing dataset.

Integrating a CoreML model with an iOS app

By integrating Apple's Vision framework with a Core ML model, you can significantly enhance your application's capabilities, particularly in image and video analysis. Vision simplifies the preparation and handling of image data before it undergoes processing by a Core ML model, making tasks such as image classification, object detection, and more complex image-based predictions more streamlined and efficient.

How to create an Image Classifier Instance:

Upon launch, the ImagePredictor class initializes a singleton instance of the image classifier by invoking its createImageClassifier() method.

```
/// - Tag: name
static func createImageClassifier() -> VNCoreMLModel {
    // Use a default model configuration.
    let defaultConfig = MLModelConfiguration()

    // Create an instance of the image classifier's wrapper class.
    let imageClassifierWrapper = try? MobileNet(configuration: defaultConfig)

    guard let imageClassifier = imageClassifierWrapper else {
        fatalError("App failed to create an image classifier model instance.")
    }

    // Get the underlying model instance.
    let imageClassifierModel = imageClassifier.model

    // Create a Vision instance using the image classifier's model instance.
    guard let imageClassifierVisionModel = try? VNCoreMLModel(for: imageClassifierModel) el
        fatalError("App failed to create a `VNCoreMLModel` instance.")
    }

    return imageClassifierVisionModel
}
```

How to create an Image Classification Request:

The ImagePredictor class generates an image classification request, represented by a VNCoreMLRequest instance. This request is created by passing the shared image classifier model instance and a request handler to its initializer.

```
// Create an image classification request with an image classifier model.

let imageClassificationRequest = VNCoreMLRequest(model: ImagePredictor.imageClassifier,
                                                 completionHandler: visionRequestHandler)

imageClassificationRequest.imageCropAndScaleOption = .centerCrop
```

How to create a Request Handler:

For each image, the ImagePredictor's makePredictions(for photo, ...) method generates a VNImageRequestHandler by providing the image and its orientation to the handler's initializer. This handler facilitates the processing of images through the Vision framework.

Through these steps, you can seamlessly integrate Core ML models into your iOS app, leveraging the power of Vision to enhance image and video analysis functionalities.

```
let handler = VNImageRequestHandler(cgImage: photoImage, orientation: orientation)
```

Conclusion

In conclusion, we have showcased the development process of a simple custom image classifier app for iOS using CoreML. Leveraging the CreateML app within Xcode, we were able to generate a machine learning model by providing training, testing, and configuration data. Once the model was created, we seamlessly integrated it into the iOS app by incorporating it with the Vision framework and executing the Vision request. The classification results were effectively managed and displayed on the app's frontend, allowing users to observe varying degrees of rusting in the analyzed images.

References

https://developer.apple.com/documentation/coreml/getting_a_core_ml_model
https://www.youtube.com/watch?v=3MXYwpifQOM&t=219s
https://www.youtube.com/watch?v=6_k0kXhRajo&t=853s
https://www.youtube.com/watch?v=SQFObTCQvpI