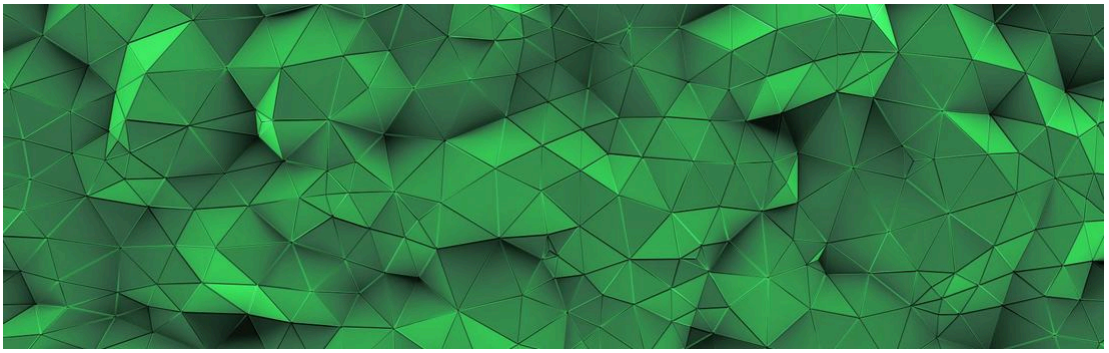
# Apple Core ML: Easily Leverage the Power of Machine Learning

Originally published at https://www.iflexion.com/blog/coreml

Published: August 5, 2020 | By Martin Anderson



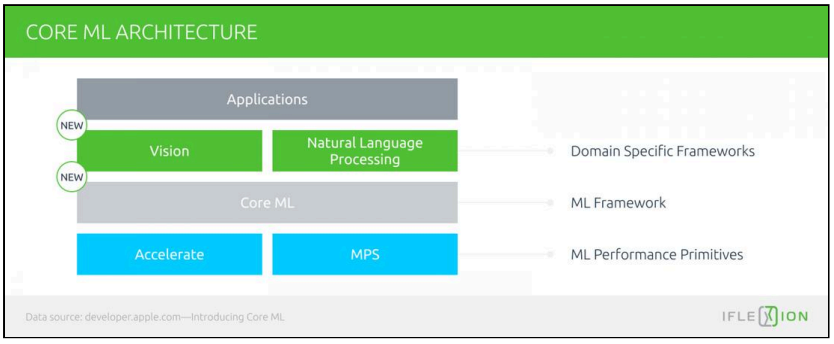
Apple's Core ML mobile machine learning framework is the user-friendly face of one of the newest sectors to draw tech headlines in recent years — machine learning on smartphone and small-form factor mobile devices.

It's a field currently torn between several business and logistical imperatives and limitations, yet which promises to transform AI development from an abstract and generalized user experience into a deeply personal and 'local' one, without the negative privacy implications of cloud-based data-gathering.

Here, we'll examine the latest capabilities of Core ML in the wider context of mobile machine learning. We'll also investigate the ways in which localized machine learning could be a benefit, but also explore some of the constraining factors that should keep our expectations realistic.

## Apple and Deep Learning

Core ML is a modular intermediary between Apple's earlier machine learning frameworks (Accelerate and Metal) and new domain-specific frameworks (Vision, Foundation, and GameplayKit). Core ML is oriented toward the four domains most relevant for local machine learning systems: vision, natural language processing (NLP), speech recognition and transcription/synthesis, and sound analysis.




Metal-based machine learning deployments would typically leverage the mobile device's GPU via low-level APIs and protocols, requiring developers to adopt processes and pipelines often as complex and rigorous as for desktop or cloud-based AI systems. Though Metal-based apps can also access Apple's dedicated neural network hardware chip, the A11-13 processor[1], the development workflow can be similarly demanding.

Nonetheless, for applications with a wider scope than the four addressed by Core ML, Metal remains the most flexible route, facilitating low-level GPU-based functions such as ray-tracing[2] and also custom machine learning model implementations that are not catered to in Core ML[3].

## A Rosetta Stone for Deep Learning Models

When Core ML was introduced at WWDC 2017, it offered native neural networks on Apple platforms, for deep, convolutional, linear and recurrent network types, as well as tree ensembles[4]. Later that year, Apple provided software to convert TensorFlow Lite ML models into native Core ML models, though this was a unilateral and 'read-only' solution.

Core ML deployments could also be trialed in an iOS simulator environment that leverages the CPU, whereas Metal applications initially needed direct GPU access for testing purposes[4,5].

The release of Core ML 2 in 2018 brought improved performance, size-optimization, and greater flexibility, with the ability to customize native Core ML models.

Meanwhile, the new Python-based Core ML Tools provided a drag-and-drop conversion facility for some of the most popular machine learning frameworks, including Caffe[6], Keras[7], XGBoost[8], Sci-Kit Learn[9], Apple's Turi[10] and LIBSVM[11].


With the new capability to quantize a machine learning model, it was now possible to slim down the number of models and weights, as well as the size of the weights, and generate reduced models at 16-bit and even 8-bit sizes, instead of the 32-bit format of the initial release.

Core ML's use of Transfer Learning was able to facilitate a reduction in model size of up to 98%, by replacing the more generic layers of a model with abstract model code already built into the device.

Considering the app-specific and very targeted nature of applications that leverage local machine learning, such aims were often achievable even with highly quantized models.

Trace the timeline of Core ML developments from 2017 till now.

## Machine Learning for The Masses, Not the Individual

However, at this stage, there was still no capability to update machine learning models locally, and the primary use was for predictive capabilities related to text analysis as well as generic inference in the areas of image recognition, recommender system templates, and audio-based processes, such as speech analysis and music-centered applications.

Personalized ongoing training was still limited to cloud-ML models, such as Spotify's AI-driven algorithm[12], which processes local input in the cloud in order to return individually weighted inference to the user experience in the form of recommendations.

Such workflows require the provider to dedicate a machine learning instance to one user out of millions, as well as encumbering themselves with the data governance, privacy and security issues of receiving the raw data over the network, sending back the weights and negotiating the legal implications of any residual usage or retention of that data.

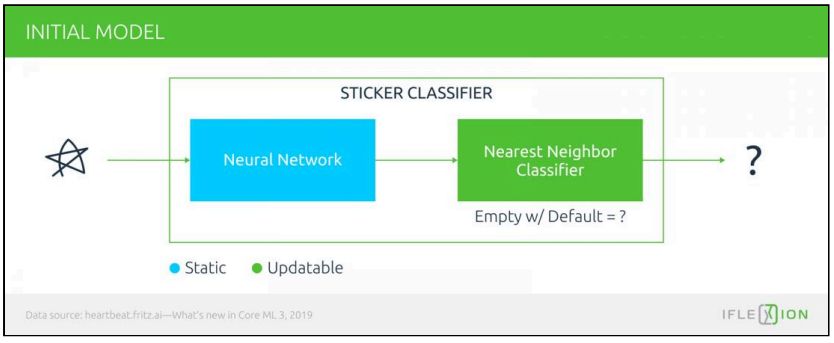
## Core ML 3

In 2019, the release of Core ML 3 brought not only the addition of more than 100 layer types[13], but also the much-anticipated capability for machine learning models to update locally and natively based solely on local input, free of any dependence on cloud-based processing services.

Core ML 3 also brought a radical expansion of compatibility features and extensibility for existing and imported models, including:

- **Symbolic linking** for models, where a virtual instance of a model can be referenced anywhere in the workflow, so that feature extraction over multiple classifiers no longer requires model duplication[14].
- **ItemSimilarityRecommender** for easy creation of recommender systems from Create ML.
- **Sound Analysis Preprocessing** in Create ML, facilitating audio feature extraction.
- The long overdue **k-Nearest Neighbors**(k-NN) classifier for statistical estimation and pattern recognition[15].
- **MLGazetteer**, a database template to aid tagging in NLP pipelines[16].
- **MLWordEmbedding**, an array of strings in the vector space to identify similar strings in adjacent data[17].

However, only models that define a neural network or utilize the k-Nearest Neighbors (see above) model can use the new *isUpdatable* property for models in Core ML 3. Additionally, training is only possible for convolutional and connected layers, although you can specify target layers in a model whose other parameters might not support on-device training and updating.




At the time of writing, the Create ML environment offers six types of model[18]:

- **Image models**, capable of object detection and classification, as well as segmentation.
- **Video models**, capable of pose detection and general movement classification, as well as style transfer.
- **Text models**, for natural language processing functions such as classification, tagging, and vector spaces for word embedding.
- **Motion models**, which can make use of the device's Core Motion gyroscope, accelerometer, pedometer, and other environmental input.

- **Sound models** for the classification of audio data.
- **Tabular Models**, including facilities for importing tabular data, estimating continuous values (regression), classifying data into categories, and developing recommender systems from a number of criteria.

Tabular models also feature a dedicated class of data structure called Tabular Data, including a model evaluation table, the MLDataValue type, which offers inspection classes for cell-based data, and data visualizations, which can stream image data into an experimental space for development purposes.

An application may make use of several model types, for instance in transcribing audio (sound analysis and NLP), gait recognition (pose detection and motion sensor classifiers), or in stabilizing video by accounting for device movement as well as visual perturbation.

One could also create a model that utilizes Vision to identify the text boundaries of signs featured in an image, and then leverage NLP to get the meaning of the text:


## Accessing the Apple Neural Engine (ANE)

Since the release of the A12 Bionic Processor in 2018, Core ML models have been able to directly access Apple's dedicated neural net hardware, the Apple Neural Engine (ANE).

However, the Core ML documentation promises only that models *can* have access to ANE, as well as the GPU and the CPU in the host device. It does not promise to grant access to the neural network-enabled hardware in all cases, or for all applications that could potentially access it.

Therefore, if a model or implementation has been built on the expectation of running on the best local hardware, performance may drop if the device sends layer processes to the GPU or CPU instead[19].

The wording of Apple's documentation in this regard indicates that the device has to negotiate and balance the needs of local machine learning applications against power consumption, temperature, and a range of other factors: *'Core ML optimizes on-device performance by leveraging the CPU, GPU, and Neural Engine while minimizing its memory footprint and power consumption.'*

Since the method implementation of ANE is not explicitly available to developers, it may be necessary to track the runtime with dedicated sniffing processes and dumping buffers until an *AppleNeuralEngine`-[_ANEClient* return reveals that the code is running on ANE[20].

## The Value of On-Device Machine Learning Systems

The current machine learning revolution is oriented toward throwing ever greater hardware, storage, and data resources at neural networks (as is the case with 2020's headline-grabbing GPT-3[21]), in the context of growing concerns and new legislation around privacy and data-sharing — and of the hard limits to computing power, storage, and battery life on hand-held devices.

Developing effective yet lightweight local networks is an excellent way to build AI applications that are genuinely personal and stay 'off the grid', while putting statistical analysis principles in favor of the user, who is normally served the 'statistical mean' recommendation based on thousands of data points from other users in centralized machine learning systems.

Unhelpful predictions and recommendations tend to occur because overly-reductive machine learning models will lead an inference system to a 'lowest common denominator' result, as the model is too optimized to return useful results for 'outliers' (especially on slimmed-down mobile machine learning systems). A great deal of human manual weighting is needed to overcome some of the inherent weaknesses in the principles of statistical analysis.

### No Room for 'Off-beat' Tastes

Since people are strange, studying them *en masse* frequently fails to reveal any useful information about any single one of them.

If ten people in a crowd of twenty prefer chocolate ice-cream, and the other ten prefer strawberry, the mean result that a centralized recommender system might produce is that they *all* like chocolate and strawberry flavor (since the results produced a zero weighting); or that *none* of them like either flavor (since each one's preference is cancelled out by another's); or that they all only like vanilla (since it is the third-favorite flavor and they don't seem to like anything else).

The weakness of AI recommender systems that use centralized, aggregated data in this way is that they are either susceptible to such logic-traps or else are easily biased by strange outliers 'blowing the curve'.

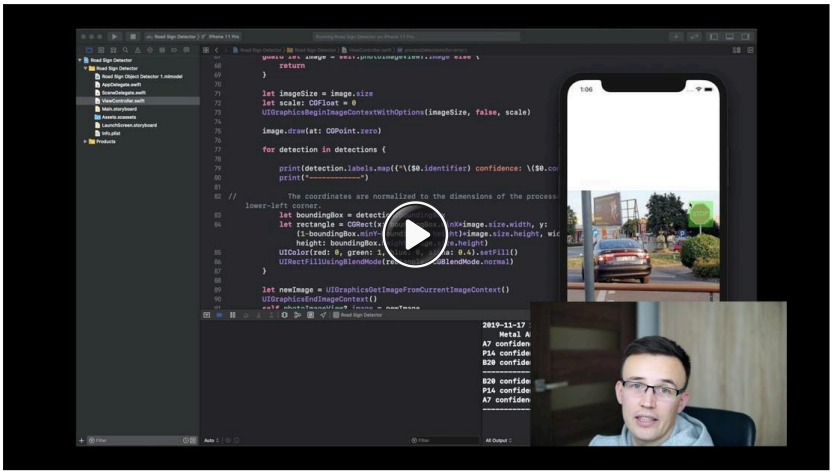
For this reason, a degree of manual weighting is usually necessary in recommender and prediction systems. Since human power resources are limited, these weights will usually also favor the most popular or generalized results.

With local updateable machine learning algorithms, as implemented via Core ML and other open-source projects, the more limited computing resources of a small device are off-set by the fact that the incoming data relates to no one else but the specific user — a level of specificity that is probably not practicable in any other scenario.

There's little room for off-beat tastes in mobile machine learning systems, unless manual weighting is applied.

## Use Cases

Core ML has been implemented with imagination since its debut in 2017. In the video below, a developer uses Core ML to train and implement an object detection system capable of recognizing road signs:

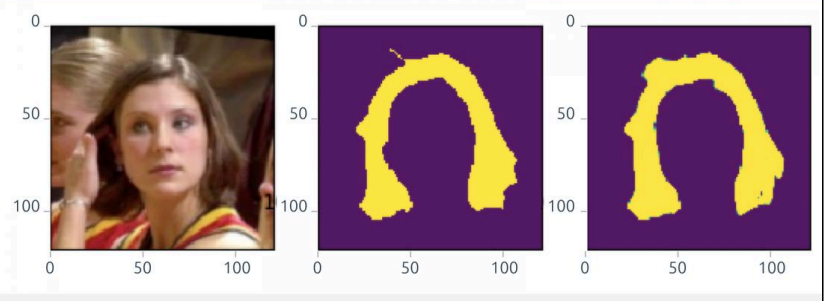


In the next video, developers integrate Apple's augmented reality framework ARKit with Core ML, to ensure that a brand app can recognize each of four different flavors of the same cider:


Other applications of Core ML include:

- Real-time semantic segmentation capable of individuating hair from faces.

## HAIR INDIVIDUATION

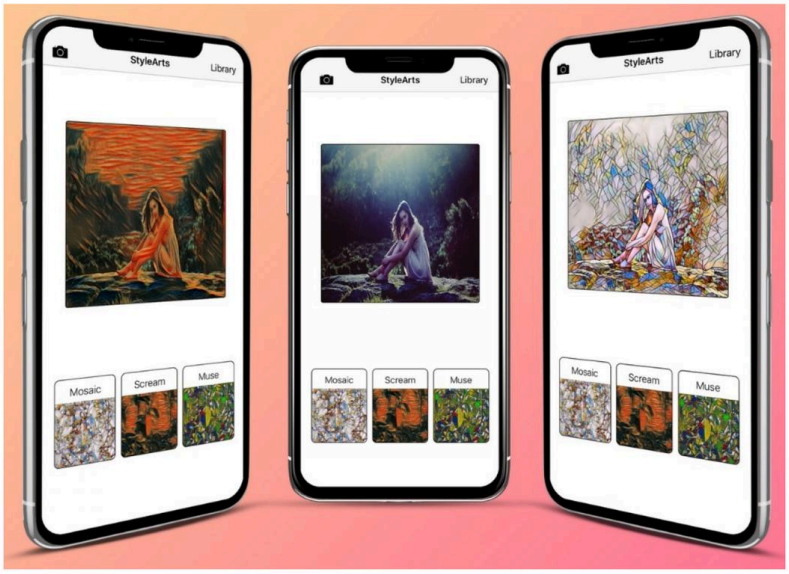



Data source: github.com—Real-Time Semantic Segmentation in Mobile device

IFLE ION

- Using the style transfer capabilities of Core ML to develop an app capable of transforming photos into styled paintings.

## STYLE TRANSFER IOS CORE ML




Data source: github.com—iLeaf Solutions, StyleArt

IFLE ION

- Image classification for Unity on ARKit.

## UNITY IMAGE CLASSIFICATION


Data source: github.com—UnityVision—iOS

IFLE ION

- An iOS camera framework featuring object detection.

## LUMINA DEMO


Data source: github.com—Lumina

IFLEXION

- An object recognition system for food.


- An iOS implementation of the heavy-duty Waifu2x AI-based upscaling algorithm[22].
- A hand gesture recognition system.

HAND RECOGNITION CORE ML


Data source: github.com—Hand Gesture Recognition

IFLEXION

## Summing It Up

The social and commercial imperatives to drive down power usage in desktop and mobile devices is set to impose an ongoing austerity regarding the machine learning capabilities of mobile devices in the future.

Therefore, the key to success in this sector, for the time being, lies in directing these highly optimized resources to the specific, local digital world of the end user and in providing a level of personalization that could never be achieved by updating centrally aggregated machine learning models.

However, companies will need to contend with the challenge of keeping their mobile machine learning deployments *on* the user's system. This is likely to make performance analysis and ongoing improvement a challenge, and will require clear and transparent sharing policies, where the host system even permits it.

This applies more to bug reporting, customer satisfaction and engagement feedback mechanisms than for exfiltrating local data with a view to analyzing it in more powerful cloud-based machine learning systems.

In any case, personal data that's highly relevant to the end user, and which can create a dazzlingly relevant local experience, would contribute little to a more homogenous dataset, for the reasons we examined earlier. Except to prove, once again, that people aren't the same all over.