Adding Intelligent Vision to Your Next Embedded Product

Radhika Jagtap, Product Manager
Alessandro Grande, Developer Advocate

White Paper

Embedded vision has proved to enhance solutions in a broad range of markets including automotive, security, medical and entertainment. Leaps in understanding of deep learning technology combined with increasing embedded compute performance will lead to an explosive growth in embedded vision products. Join this tour of hardware and software components and the trade-offs a designer has to make in order to satisfy the broad market requirements. What are the key enablers for easily adding intelligent vision to your next embedded product? Will the intelligence be local or in the cloud, and why? This paper presents answers to these questions and more.

Introduction

Computer vision (CV) is the ability of machines to understand images or video, infer information and potentially use it to make decisions. Embedded vision alliance defines embedded vision as the following.

Embedded vision refers to the practical use of computer vision in machines that understand their environment through visual means.

The major problems addressed by embedded vision technology which makes the devices intelligent are image classification, object detection, and recognition. According to our estimates, the demand for embedded devices with intelligent vision capabilities is highest for devices connected to the internet.

- The security (surveillance) and IP camera market is on pace for an annual growth rate of 20% through to 2021 reaching over 500 million shipped units;*
- Personal robots are growing at a staggering 75% y-o-y rate and are expected to reach 2 million units shipped in 2021;*
- The smart home market is growing at 14% y-o-y to 88 million in 2021, and other devices like drones, augmented reality (AR)/ mixed reality (MR) equipment and action cameras are shifting from emerging to widely adopted in terms of market category*

* These estimates have been produced by Arm based on market forecast sources from ABI Research and IHS Markit.

From a technical viewpoint, the sub-problems that embedded vision can be broken down into are capturing high-quality images and video, analyzing them, detecting and recognizing objects, and be able to give a response to human users in real-time. Object detection and recognition are hard, if not impossible to solve using classic rules-based programming. In traditional CV, feature extraction is key and algorithms for detecting features such as edges, corners and objects are based on mathematical models. One needs to be an expert to implement traditional CV and manually decide which features to extract and which algorithm to use for the specific class of objects. Deep learning has gained momentum for complex CV problems for which it is feasible to obtain a large amount of visual data. While many researchers believe deep learning will not make traditional CV obsolete, it has a clear advantage, which is doing end-to-end object detection without requiring to define features specifically.

A. End-to-end deep learning flow

Neural networks are at the core of deep learning, which is a subset of machine learning (ML). Neural networks have an architecture inspired by the structure of the brain. They transform an input through a series of layers wherein each layer comprises of neurons, each neuron has a set of weights, and the neurons belonging to different layers are connected. The final layer outputs the answer, for example, whether an object is detected or not. In practice, there are two phases of deep learning.

The first phase of deep learning is training which is when the neural network learns the features automatically. The outcome of the training phase is a neural network with weights configured so that it works for the images it was trained on, called the neural network model. Many image databases are available online to aid this, for example, ImageNet contains 10s of millions of classified images. The second phase, called inference, is deploying the network onto the embedded device that will run the vision application, for example, a smart doorbell. During the inference phase, a new image is input to the neural network, and it outputs a prediction of the answer. An important metric of design and in turn the user-experience is the accuracy of the prediction. Fig 1 shows an illustration of the end-to-end flow. Deep learning has significantly enhanced the speed, accuracy and scalability of solving computer vision problems as shown in the widely cited ImageNet paper.

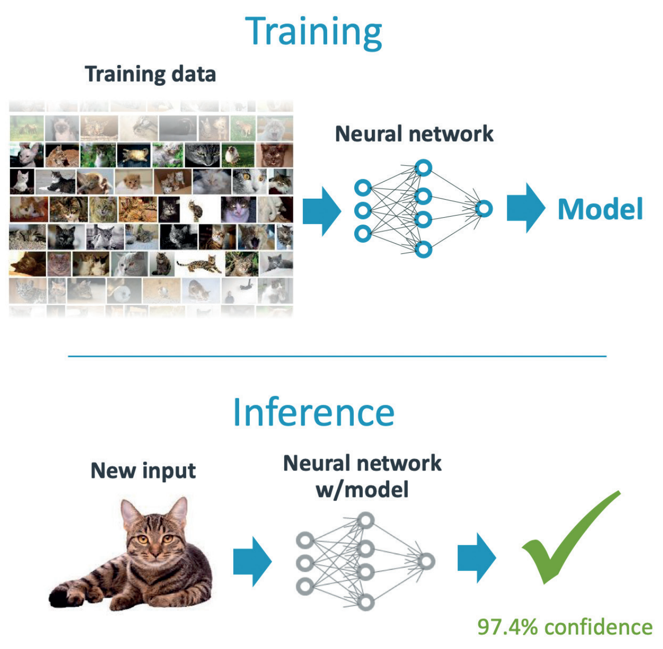


Fig 1:
The two phases of training and inference make up the end-to-end flow for embedded vision using deep learning.

An example of inference is whether the object in the image belongs to a specific class of objects. This problem called image classification is the focus of our paper. Recent research has shown that *Convolutional Neural Networks* (CNNs) are highly effective at classifying images and maps well onto embedded devices.

B. Embedded vision explosion

Advances in embedded compute capability have resulted in an explosion of embedded vision applications and products. Here are a few examples of embedded vision applications in home automation, automotive, medical and industrial.

- Intuition Robotics' ElliQ is a social companion robot that helps the elderly. Arm's partner, Brodmann17 worked with them to speed up the performance of the robot via the software solution. The SoC is a Qualcomm Snapdragon 820 which is based on Armv8-A.
- Skydio is an autonomous camera drone that senses the environment, identifies and tracks people so that they can be filmed, for example when athletes want to record their fitness activities. It maps things like trees and building which are obstacles in real-time and deploys a series of CNNs to track people using a combination of identifiers, for example, colour of clothing.
- A cucumber sorting application developed on Raspberry Pi 3 takes images of the cucumbers with a camera and runs a small-scale neural network on TensorFlow to detect whether or not the image is of a cucumber.

In all the above examples, the inference runs locally on the device, closest to the source of the input data and the user of the intelligent embedded device. There are several advantages of deploying intelligent vision locally on embedded devices compared to utilizing cloud compute:

- Bandwidth saving and lower latency as there is no need to communicate with the server
- Lower cost as embedded devices are cheaper than provisioning a machine in the cloud
- Reliability as it works in the absence of internet connectivity
- Smaller attack surface compared to exposing the device to attacks over the network connection

C. Model optimization gap

To enable the end-to-end deep learning flow in a flexible manner, layers of abstraction are designed such that neural network models trained using any framework can be deployed on a variety of hardware platforms. These layers are introduced below and are described in detail in Sections titled Hardware and Software.

- ML frameworks – these enable specifying neural networks in high-level languages and training them, for example, Caffe, TensorFlow and Torch
- Optimization libraries - typically, neural network models are implemented using libraries optimized for specific hardware targets
- Embedded hardware - from general-purpose to dedicated, providing a range of performance and cost points
  - General purpose processors, for example, Arm Cortex-A, Cortex-R and Cortex-M processors
  - Dedicated or specialized hardware, for example, neural network accelerators

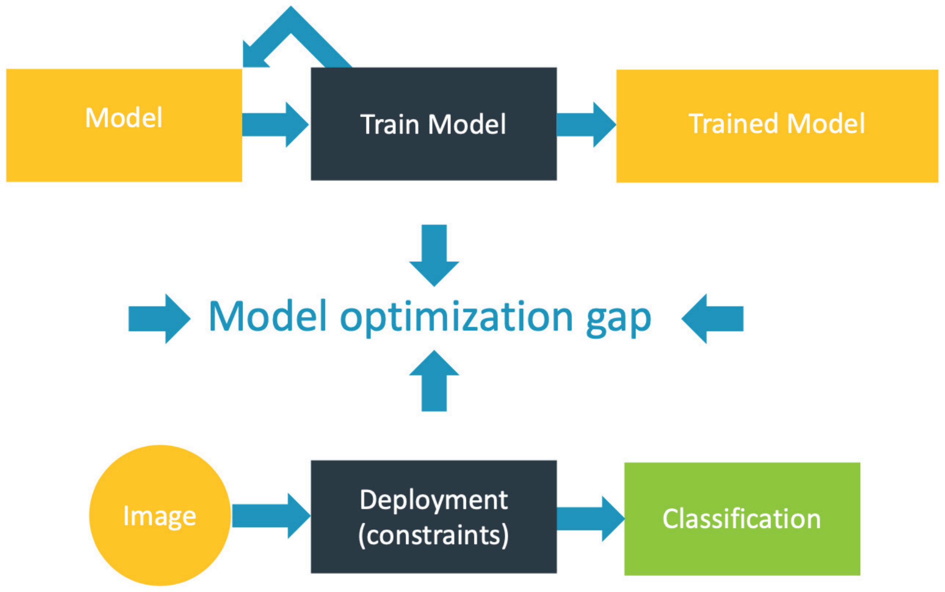
To enable developers to easily ride the wave of intelligent embedded vision, Arm is addressing

Open-source ML frameworks and pre-trained models are readily available. Alongside that, there is a strong trend that embedded hardware is becoming increasingly accessible. The layer in between, comprising of libraries that abstract away advanced optimizations that are specific to target hardware, requires deep technical expertise. Without this layer, developers who aren't experts in ML and hardware would experience a gap, as shown in Fig 3. We call this the *model optimization gap*

this gap by contributing to libraries known as CMSIS-NN, Arm-NN and Arm Compute Library.

In summary, market and technical insights indicate we are at the brink of crossing the

Fig 2:
Developers who aren't experts in ML and hardware would experience a gap when adding intelligent vision to their embedded devices. We call this the model optimization gap.



inflection point after which embedded vision will be omnipresent. This paper shows the embedded community how to add intelligent vision to your device by bringing it all together - low-cost and low-power embedded hardware, open-source ML frameworks and key enablers such as optimized libraries. In the remainder of this paper, we deep-dive into a versatile and easy-to-use smart camera module from OpenMV. We describe the steps to leverage open-source software framework and libraries

Hardware

There is a broad range of compute performance requirements for machine learning applications based on neural networks, illustrated in Fig 4. For most of the markets, vision dominates the compute requirement.

Arm IP portfolio offers a scalable platform that serves a broad range of performance and energy efficiency needs of embedded vision applications. An important consideration across the spectrum is image processing – cleaner the image, the better the accuracy of inference. In embedded devices, the key function of an Image Signal Processor (ISP) is delivering the most accurate and highest image quality when pulling data from an image sensor and processing each pixel, particularly where ML is involved. Arm Mali-C52 and Mali-C32 apply over twenty-five processing steps to each pixel, of which three critical ones deliver key differentiation in terms of image output quality. These include high-dynamic range (HDR), noise reduction and color management.

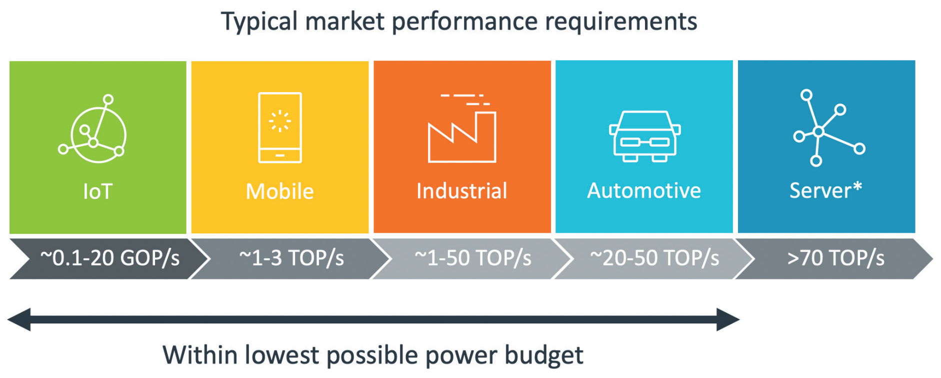


Fig 3:
There is a broad range of performance requirements for machine learning based on neural networks. For most of the markets, vision dominates the compute requirement.
* Server market is merely shown for reference.

Next, we describe the hardware components needed in an embedded vision pipeline with the help of a smart camera module that is available to purchase.

A. OpenMV Cam

The OpenMV Cam M7 shown in Fig 4 is a small, low-power, microcontroller board which allows you to implement applications using machine vision in the real-world easily. You program the OpenMV Cam in high level Python script, courtesy of the MicroPython Operating System. This makes it easier to process complex outputs of machine vision algorithms and aids working with high-level data structures. You can also readily trigger capturing photos and videos on external events or execute machine vision algorithms as you have direct control of the I/O pins.

The board allows you to capture image and video in a highly controlled way, perform vision functions and directly draw on the captured content before saving it. There is code available for face and eye detection, eye tracking, QR Code and Bar Code decoding and CNN inference. The board is available to buy online, and the software tools and scripts are open-source.

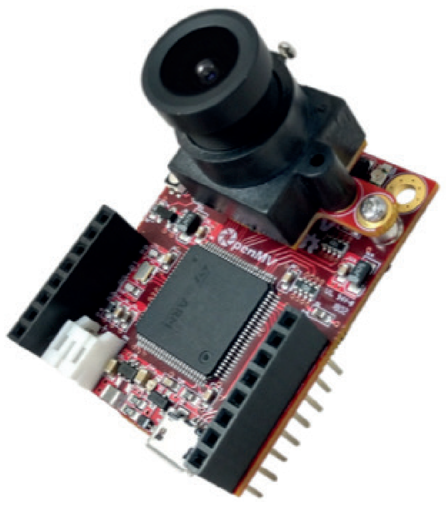


Fig 4:
OpenMV Cam M7 allows you to implement real-world embedded vision applications easily.

Here is a summary of the OpenMV Cam M7 components.

Image sensor – OV7225

- Small, low voltage (3.3 V), providing single-chip VGA camera up to 60 FPS and image signal processor
- Image processing functions including gamma curve process, white balance, saturation, color
- 640x480 16-bit RGB565 images at 60 FPS for resolutions above 320x240 and 120 FPS for below

Processor

STM32F765VI Arm Cortex M7 processor running at 216 MHz with 512KB of RAM and 2 MB of Flash. In addition, the module has a built-in camera lens, microSD card socket and a USB interface to connect to a PC.

B. Trade-offs in hardware

If the neural network model for the vision problem does not fit in the on-chip RAM memory, then it is possible to expand the memory by using an SD card, but this will result in a slower inference output. The next generation of OpenMV platform is the H7 Cam based on the STM32H743VI Arm Cortex M7 processor running at 400 MHz with 1MB of RAM and 2 MB of flash – higher speed and twice the RAM size compared to the M7 Cam. For a significant boost in compute performance and memory, application processors such as the Arm Cortex-A processors can be chosen. JeVois smart camera is a gadget that packs an impressive amount of compute capability, runs on a battery and is also available to purchase online. JeVois A33 smart camera is based on the Allwinner A33 quad-core Arm Cortex A7 processor running at 1.34GHz with VFPv4 and NEON, and a dual-core Mali-400 GPU supporting OpenGL-ES 2.0. It uses 256MB DDR3 SDRAM which enables more complex neural network models.

These are a few of the trade-offs a designer has to make when choosing a hardware platform. Having covered embedded hardware components, we turn our attention to the software needed to make applications come alive.

Software

Next, we describe in detail the layers of abstraction, what their contribution is to the flow and what are the most widely used software frameworks, libraries and toolchains.

A. ML frameworks

ML frameworks enable specifying neural networks as software models (functions in high-level languages) and training the models using large data sets. Popular frameworks include Caffe, TensorFlow and PyTorch. At the end, you get a neural network with weights configured so that it works for the data it was trained on. Caffe was developed for image classification and other vision problems using CNNs. Caffe is perhaps best known for Model Zoo, a set of pre-trained models which you can use without writing any code.

For inference, we need to select models with smaller neural network architectures such that they better match the capability of an embedded device, rather than the capabilities of workstations on which they are trained. Next, it needs to be optimized to fit an embedded device that is designed with optimal memory and compute capability. Techniques that enable this without sacrificing application accuracy or increasing hardware cost are critical to the wide deployment of embedded vision. While embedded hardware, open-source ML frameworks as well as pre-trained models are readily available, the layer in between that encapsulates technically-challenging optimizations needs attention, as there is a risk of it becoming the weakest link. If this layer is missing, developers would experience a gap, which we call the model optimisation gap, illustrated in Fig 2.

B. Addressing the model optimization gap

Libraries are an attractive solution that bridges the model optimization gap between ML frameworks and embedded devices. Libraries enable developers and the embedded community to ride the wave of embedded intelligent vision without needing to become an expert on CNN optimization.

Next, we describe a range of libraries developed by Arm to hide the complexity of neural networks and hardware inside the library, so the developer only needs to call the high-level functions in their application code. Details on the optimizations implemented in CMSIS-NN

CMSIS-NN is an open source collection of efficient neural network kernels developed to maximize the performance and minimize the memory footprint of neural networks on Arm embedded processors.

library are in the original paper. CMSIS-NN is publicly available as open-source software under Apache 2.0 license without any fee on CMSIS-NN Github. CMSIS-NN for Cortex-M processors and CMSIS-NN Cortex-A processors are open to extensions by anyone to add more functions.

Currently, translating a Caffe model to CMSIS-NN functions is supported. Support for other frameworks, such as TensorFlow and PyTorch will come in the future. Below are a few intelligent embedded devices that use CMSIS-NN.

- Real-time water quality monitoring system that detects harmful microbes and bacteria implemented using CIFAR-10 CNN and CMSIS-NN library.
- Although not a vision application, the enablement potential of CMSIS-NN has also been demonstrated for speech recognition on the Nuvoton NuMaker-PFM-M487 platform, based on the Cortex-M4 processor.

Arm NN is an inference engine for CPUs, GPUs and NPUs. It bridges the gap between existing neural network frameworks and the underlying IP. It enables efficient translation of existing neural network frameworks, such as TensorFlow and Caffe, allowing them to run efficiently – without modification – across Arm Cortex CPUs and Arm Mali GPUs.

Fig 5 and Fig 6 capture the support for the Android and the embedded Linux operating systems respectively.

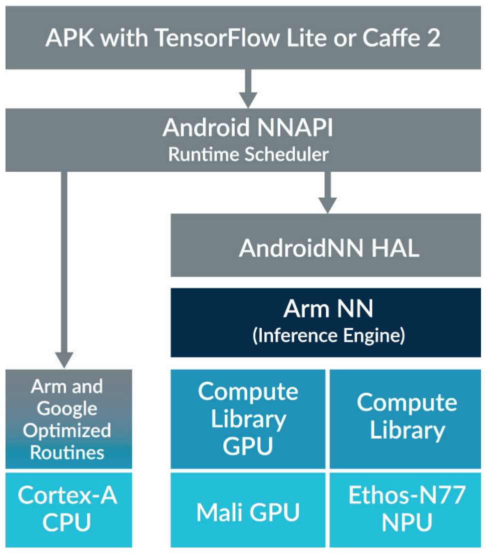


Fig 5:
Arm NN providing support for Mali GPUs under Android NNAPI.

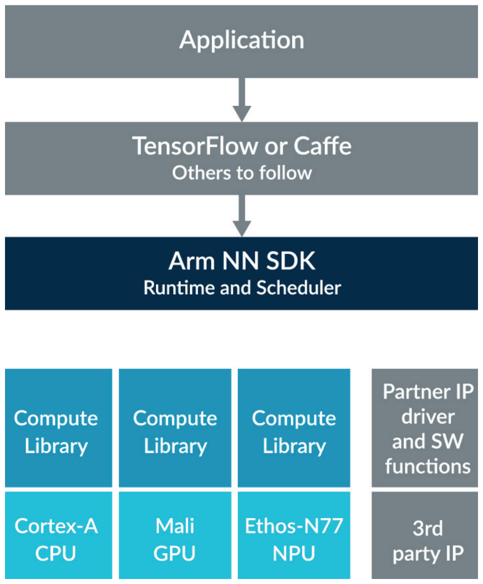


Fig 6:
Arm NN currently provides support for Cortex-A CPUs and Mali GPUs under embedded Linux while support for the Ethos-N77 NPU will come in the future.

Bringing it all together

A flowchart of the steps to deploy a CNN model on the OpenMV M7 Cam is shown in Fig 7. OpenMV is one of the first published examples of an end-to-end embedded vision solution including hardware board as well as software utilizing CMSIS-NN under the hood.

1. Train the model

Training the neural network model takes place on workstations or laptops and is beyond the scope of this paper. Plenty of pre-trained networks are publicly available which can be used as the input to the next step of quantization.

2. Quantize

Once we have a trained model, we need to shrink it to a reasonable size to be able to deploy it on an embedded device designed with optimal memory and speed to achieve high energy efficiency. A quantization script is available from Arm to convert the Caffe model weights and activations from a 32-bit floating point to an 8-bit and fixed-point format. This will not only reduce the size of the network, but also avoid floating point computations.

The NN quantizer script works by testing the network and figuring out the best format for the dynamic fixed-point representation. The output of this script is a serialized Python file which includes the network's model, quantized weights and activations, and the quantization format of each layer.

3. Convert to binary

Finally using the OpenMV NN converter script, the model can be converted into a binary format, executable by the OpenMV Cam. The converter script outputs a code for each layer type, followed by the layer's dimensions and weights (if any). On the OpenMV Cam, the firmware reads the binary file and builds the network in memory using a linked list data structure.

4. Deploy

The binary obtained from the above step is then downloaded to the Open MV board via a USB. When the model is run, it calls CMSIS-NN libraries.

For more information covering everything from installation to troubleshooting, please refer to the developer guide on the Arm website - *Deploying a Caffe Model on OpenMV using CMSIS-NN*.

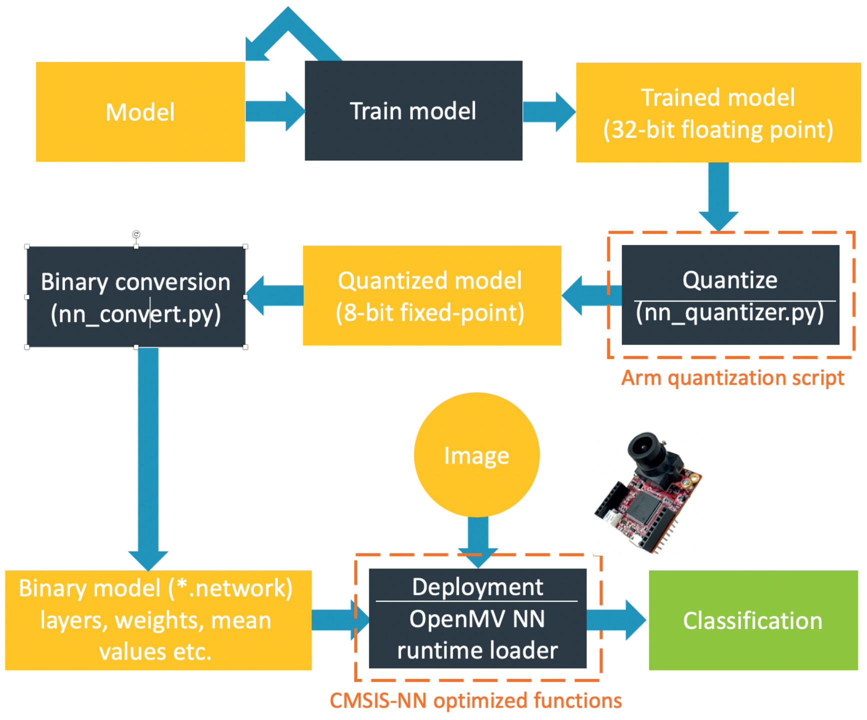
Summary

Adding intelligent vision to your next embedded devices unlocks applications in medical, industrial, smart home automation and more. There is a broad spectrum of hardware compute capabilities with trade-offs in processing speed, memory capacity and cost. Low-cost and low-energy embedded devices, such as OpenMV are available to deploy image classification models based on neural networks.

Image classification models that are optimized for running on embedded devices, offer fast classification without needing to go to the cloud. Techniques that enable energy-efficient processing of CNNs on embedded devices with limited memory, without sacrificing application accuracy or increasing hardware cost, are critical to the wide deployment of embedded vision.

Optimization techniques require deep expertise in both neural network optimization and underlying embedded hardware. A key enabler is software libraries optimized for embedded processors, for example, CMSIS-NN and Arm-NN that are open-source. These libraries address the model optimization gap between ML frameworks and underlying hardware. Arm's contribution ensures developers don't need to become an expert in both ML and embedded technologies, and we have already seen real-world examples of applications using CMSIS-NN.

Fig 7:
Deployment flow on OpenMV Cam for a model trained using the Caffe framework



Model
Train model
Trained model (32-bit floating point)
Quantize (nn_quantizer.py)
Arm quantization script
Quantized model (8-bit fixed-point)
Binary conversion (nn_convert.py)
Image
Binary model (*.network) layers, weights, mean values etc.
Deployment
OpenMV NN runtime loader
CMSIS-NN optimized functions
Classification

The combination of accessible hardware and the ease of development makes adding intelligent vision to your next embedded device highly attractive from a business and technical point of view. There is an increasing trend for delivering compute capabilities on embedded devices and soon, the advantages of local compute vs the cloud will make intelligent vision in embedded devices omnipresent.

For further reading about this topic, please visit the below:

- Machine learning on Arm
- Software for machine learning on Arm
- Guide: Deploying a Caffe Model on OpenMV using CMSIS-NN
- Arm Mali-C52 and Mali-C32 ISPs
- The Arm Innovators, including OpenMV and JeVois

Acknowledgment

We would like to thank Ibrahim Abdelkader and Kwabena Agyeman, the co-founders and Arm Innovators behind OpenMV for working with us on the guide. The authors are also immensely grateful to their Arm colleagues for giving great inputs and reviewing.