Bringing machine learning to embedded systems

Mark Nadeski
Embedded Processing
Texas Instruments

It is hard to understate the promise of machine learning, the latest evolution of which, deep learning, has been called a foundational technology that will impact the world to the same degree as the internet, or the transistor before that.

Brought on by great advancements in computing power and the availability of enormous labeled data sets, deep learning has already brought major improvements to image classification, virtual assistants and game playing, and will likely do the same for countless industries. Compared to traditional machine learning, deep learning can provide improved accuracy, greater versatility and better utilization of big data – all with less required domain expertise.

In order for machine learning to fulfill its promise in many industries, it is necessary to be able to deploy the inference (the part that executes the trained machine learning algorithm) into an embedded system. This deployment has its own unique set of challenges and requirements. This white paper will address the challenges of deploying machine learning in embedded systems and the primary considerations when choosing an embedded processor for machine learning.

Training and inference

In the subset of machine learning that is deep learning, there are two main pieces: training and inference, which can be executed on completely different processing platforms, as shown in **Figure 1**, below. The training side of deep learning usually occurs offline on desktops or in the cloud and entails feeding large labeled data sets into a deep neural network (DNN). Real-time performance or power is not an issue during this phase. The result of the training phase is a trained neural network that

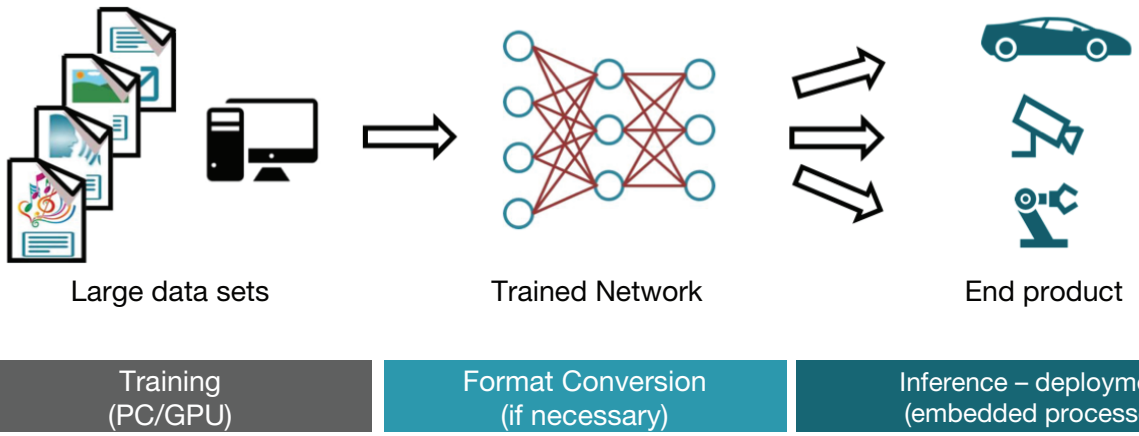

***Figure 1**. Traditional deep learning development flow.*

when deployed can perform a specific task, such as inspecting a bottle on an assembly line, counting and tracking people within a room, or determining whether a bill is counterfeit. The deployment of the trained neural network on a device that executes the algorithm is known as the inference. Given the constraints imposed by an embedded system, the neural network will often be trained on a different processing platform than the one running the inference. This paper focuses on processor selection for the inference part of deep learning. The terms "deep learning" and "machine learning" in the rest of this paper refer to the inference.

Machine learning at the edge

The concept of pushing computing closer to where sensors gather data is a central point of modern embedded systems – i.e. the edge of the network. With deep learning, this concept becomes even more important to enable intelligence and autonomy at the edge. For many applications – from automated machinery and industrial robots on a factory floor, to self-guided vacuums in the home, to an agricultural tractor in the field – the processing must happen locally.

The reasons for local processing can be quite varied depending on the application. Here are just a few of the concerns driving the need for local processing:

- **Reliability.** Relying on an internet connection is often not a viable option.
- **Low latency.** Many applications need an immediate response. An application may not be able to tolerate the time delay in sending data somewhere else for processing.
- **Privacy.** The data may be private and therefore should not be transmitted or stored externally.
- **Bandwidth.** Network bandwidth efficiency is often a key concern. Connecting to a server for every use case is not sustainable.
- **Power.** Power is always a priority for embedded systems. Moving data consumes power. The further the data needs to travel, the more energy needed.

Choosing an embedded processor for machine learning

Many of the concerns requiring local processing overlap with those inherent in embedded systems, particularly power and reliability. Embedded systems also have several other factors to consider that are related to or caused by the system's physical limitations. There are frequently inflexible requirements regarding size, memory, power, temperature, longevity and, of course, cost.

In the midst of balancing all of the requirements and concerns for a given embedded application, there are a few important factors to consider when choosing a processor to execute machine learning inference for the edge:

- **Consider the entire application.** One of the first things to understand before selecting a processing solution is the scope of the entire application. Will running the inference be the only processing required or will there be a combination of traditional machine vision with the addition of a deep learning inference? It can often be more efficient for a system to run a traditional computer vision algorithm at a high level and then run deep learning when needed. For example, an entire input image at high frames per second (fps) can run classical computer vision algorithms to perform object tracking with deep learning used on identified sub-regions of the image at a lower fps for object classification. In this example, the classification of objects across multiple subregions may require multiple instances of inference, or possibly even different inferences running on each sub-region. In the latter case, you must choose a processing solution that can run both traditional computer vision and deep learning, as well as

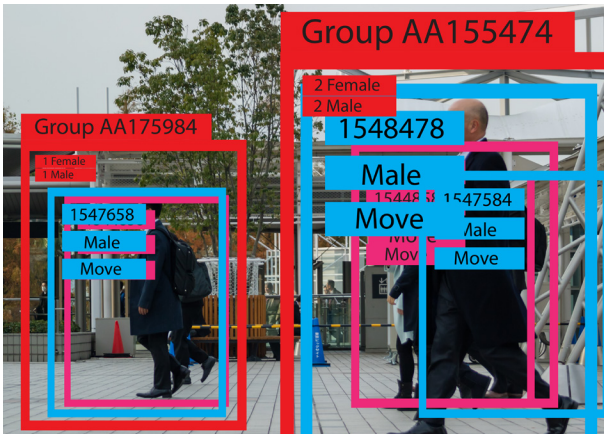

***Figure 2**. Example of object classification using embedded deep learning.*

multiple instances of different deep learning inferences. **Figure 2** shows an example usage of tracking multiple objects through sub-regions of an image and performing classification on each object being tracked.

- **Choose the right performance point.** Once you have a sense of the scope of the entire application, it becomes important to understand how much processing performance is necessary to satisfy the application needs. This can be difficult to understand when it comes to machine learning because so much of the performance is application-specific. For example, the performance of a convolutional neural net (CNN) that classifies objects on a video stream depends on what layers are used in the network, how deep the network is, the video's resolution, the fps requirement and how many bits are used for the network weights – to name just a few. In an embedded system, however, it is important to try and get a measure of the performance needed because throwing too powerful a processor at the problem generally comes at a trade-off against increased power, size and/or cost. Although a processor may be capable of 30fps at 1080p of ResNet-10, a popular neural net model used in high power, centralized deep learning applications, it's likely overkill for an application that will run a more embedded-friendly network on a 244 x 244 region of interest.
- **Think embedded.** Selecting the right network is just as important as selecting the right processor. Not every neural net architecture will fit on an embedded processor. Limiting models to those with fewer operations will help achieve real-time performance. You should prioritize benchmarks of an embedded-friendly network, one that will tradeoff accuracy for significant computational savings, instead of more well-known networks like AlexNet and GoogleNet, which were not designed for the embedded space. Similarly, look for processors capable of efficiently leveraging the tools that bring these networks into the embedded space. For example, neural networks can tolerate lots of errors; using quantization is a good way to reduce performance requirements with minimal decreases in accuracy. Processors that can support dynamic quantization and efficiently leverage other tricks like sparsity (limiting the number of non-zero weights) are good choices in the embedded space.
- **Ensure ease of use.** Ease of use refers to both ease of development and ease of evaluation. As mentioned earlier, right-sizing the processor performance is an important design consideration. The best way to do this correctly is to run the chosen network on an existing processor. Some offerings provide tools that, given a network topology, will show achievable performance and accuracy on a given processor, thus enabling a performance evaluation without the need for actual hardware or finalization of a network. For development, being able to easily import a trained network model from popular frameworks like Caffe or TensorFlow is a must.

Additionally, support for open ecosystems like ONNX (Open Neural Network eXchange) will support an even larger base of frameworks to be used for development.

There are many different types of processors to consider when choosing one for deep learning, and they all have their strengths and weaknesses. Graphics Processing Units (GPUs) are usually the first consideration because they are widely used during network training. Although extremely capable, GPUs have had trouble gaining traction in the embedded space given the power, size and cost constraints often found in embedded applications.

Power- and size-optimized "inference engines" are increasingly available as deep learning grows in popularity. These engines are specialized hardware offerings aimed specifically at performing the deep learning inference. Some engines are optimized to the point of using 1-bit weights and can perform simple functions like key phrase detection, but optimizing this much to save power and compute comes with a tradeoff of limited system functionality and precision. The smaller inference engines may not be powerful enough if the application needs to classify objects or perform fine-grain work. When evaluating these engines, make sure that they are right-sized for the application. A limitation on these inference engines comes when the application needs additional processing aside from the deep learning inference. More often than not, the engine will need to be used alongside another processor in the system, functioning as a deep learning co-processor.

An integrated system on chip (SoC) is often a good choice in the embedded space because in addition to housing various processing elements capable of running the deep learning inference, an SoC also integrates many components necessary to cover the entire embedded application. Some integrated SoCs include display, graphics, video acceleration and industrial networking capabilities, enabling a single-chip solution that does more than just run deep learning.

An example of a highly integrated SoC for deep learning is the AM5749 device from Texas Instruments, shown in **Figure 3**. The AM5749 has two Arm® Cortex®-A15 cores for system processing, two C66x digital signal processor (DSP) cores for running traditional machine vision algorithms and two Embedded Vision Engines (EVE) for running the inference. TI's deep learning (TIDL) software offering includes the TIDL library, which runs on either C66x DSP cores or the EVEs, enabling multiple inferences to run simultaneously on the device. Additionally, the AM5749 provides a rich peripheral set; an industrial communications subsystem (ICSS) for implementation of factory floor protocols such as EtherCat; and acceleration for video encode/decode and 3D and 2D graphics, facilitating the use of this SoC in an embedded space that also performs deep learning.

Choosing a processor for an embedded application is often the most critical component selection for a

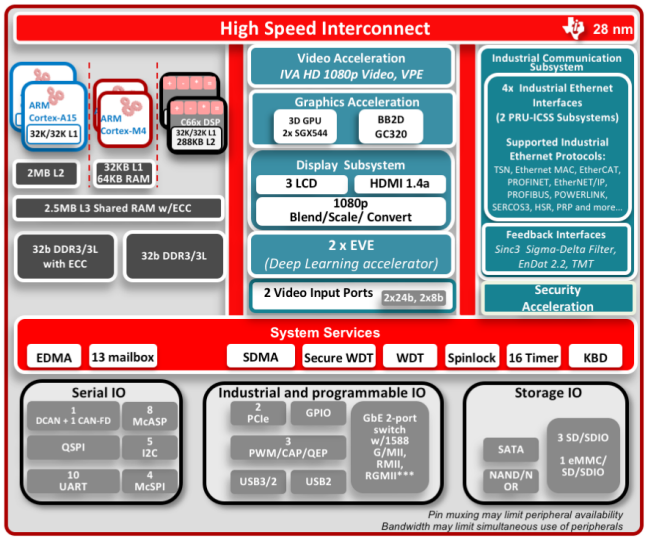

***Figure 3**. Block diagram of the Sitara™ AM5749 SoC.*

product, and this is true for many industry-changing products that will bring machine learning to the edge. Hopefully, this paper provided some insight into what you should consider when selecting a processor: consider the entire application, choose the right performance point, think embedded, and ensure ease of use.

Related websites:

- Learn more about Sitara AM57x processors.
- Download the Processor software development kit (SDK) for Sitara AM57x processors with deep learning for embedded applications.
- Download the Deep Learning Inference for Embedded Applications Reference Design

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265