Machine Learning in Embedded Vision Applications

Nick Ni and Adam Taylor

One of the hottest topics within the Embedded Vision space at the current time is machine learning. Machine learning spans several industry mega trends, playing a very prominent role within not only Embedded Vision (EV), but also Industrial Internet of Things (IIoT) and Cloud Computing. For those unfamiliar with machine learning it is most often implemented by the creation and training of a Neural Network. The term neural network is very generic and includes a significant number of distinct sub categories whose names are normally used to identify the exact type of network being implemented. These neural networks are modelled upon the cerebral cortex in that each neuron receives an input, processes it and communicates it on to another neuron. Neural Networks therefore typically consist of an input layer, several hidden internal layers and an output layer.

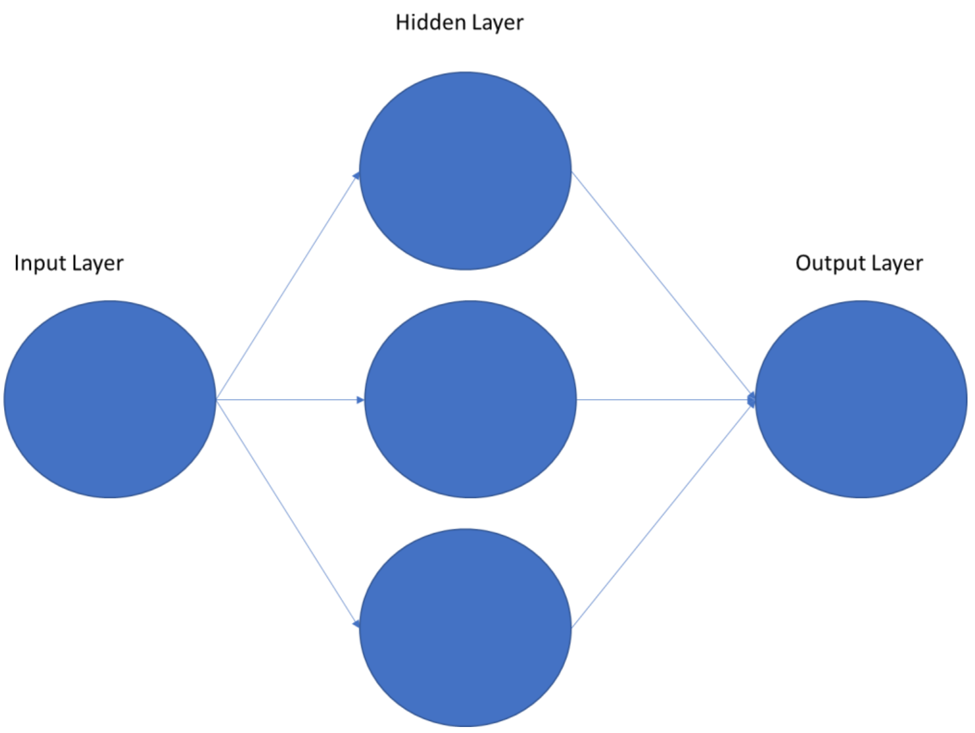


Figure One a Simple Neural Network

At the simplest level the neuron takes its input and applies a weight to it before performing a transfer function upon the sum of the weighted inputs. This result is then passed onto either another layer within the hidden layers or to the output layer. Neural networks which pass the output of one stage to another without forming a cycle are called Feed-forward Neural Networks (FNN), while those which contain directed cycles where there is feedback, for example like the Elman network, are called Recurrent Neural Networks (RNN). One very commonly used term in many machine learning applications is Deep Neural Networks (DNN). These are neural networks which have several hidden layers enabling a more complex machine learning task to be implemented. Neural networks are required to be trained to determine the value of the weights and biases used within each layer. During training, the network has a number of both correct and incorrect inputs applied with the error function being used to teach the network the desired performance. Training a DNN may require a very significant data set to correctly train the required performance.

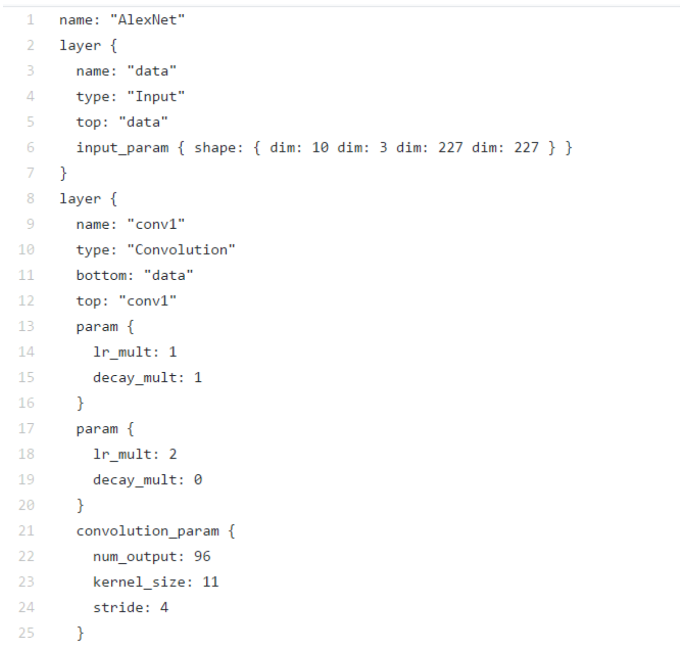
One of the most important uses of machine learning is within the embedded vision sphere where systems are evolving into vision guided autonomous systems from vision enabled systems. What separates embedded vision applications from other simpler machine learning applications is that they have a two-dimensional input format. As such, in machine learning implementations, a network structure called Convolutional Neural Networks (CNN) are used as they have the ability to process two-dimensional inputs. CNN are a type of feed-forward network that contains several convolutional and sub sampling layers along with a separate fully connected network to perform the final classification. Due to the complexity of CNNs, they also fall within the Deep Learning classification. Within the convolution layer the input image will be broken down into a number of overlapping smaller tiles. The results from this convolution is used to create an activation map by the use of an activation layer, before being subject to further sub sampling and additional stages, prior to being applied to the final fully connected network. The exact definition of the CNN network will vary depending upon the network architecture implemented, however it will typically contain at least the following elements:

- Convolution – Used to identify features within the image
- Rectified Linear Unit (reLU) – Activation layer used to create an activation map following the convolution
- Max Pooling – Performs sub sampling between layers
- Fully Connected – Performs the final classification

The weights for each of these elements is determined via training, and one of the advantages of the CNN is the relative ease of training the network. Training to generate the weights requires large image sets of both the object which is wished to be detected, and the false images. This enables us to create the weights required for the CNN. Due to the processing requirements involved in the training process, it is often run on cloud based processors which offer high performance computing.

Frameworks

Machine learning is a complex subject, especially if one had to start from the beginning each time and define the network, its architecture, and generate the training algorithms. To help engineers both implement the networks and train the network, there are a number of industry standard frameworks such as Caffe and Tensor Flow. The Caffe framework provides machine learning developers with a range of libraries, models and pre-trained weights within a C++ library, along with Python and Matlab bindings. This framework enables the user to create networks and train them to perform the operations desired without the need to start from scratch. To aid reuse, Caffe users can share their models via the model zoo, which provides several network models that can be implemented and updated for a specialised task if desired. These networks and weights are defined within a prototxt file, when deployed in the machine learning environment it is this file which is used to define the inference engine.



```
name: "AlexNet"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
  }
```

Example of Prototxt file defining the network

Implementing Embedded Vision and Machine Learning

Programmable logic based solutions such as heterogeneous Xilinx® All Programmable Zynq® -7000 SoC (System on Chip) and Multi-Processor System on Chip (MPSoC) like Zynq® UltraScale+™ MPSoC, are increasingly used in Embedded Vision applications. These devices combine Programmable Logic (PL) fabric with high performance ARM® core in the Processing System (PS). This combination allows the creation of a system which has an increased response time, is very flexible to future modification, and offers a power efficient solution. A low latency decision and response loop is of critical importance for many applications, such as vision guided autonomous robots, where the response time is critical to avoid injury or damage to people and its environment. This increased response time is enabled by the use of programmable logic to implement the vision processing pipeline, and machine learning inference engine to implement the machine learning. Using programmable logic for this reduces system bottlenecks when compared to traditional solutions. With a CPU/GPU based approach each stage of operation requires the use of external DDR as the images cannot be passed between functions within the limited internal cache. A programmable logic approach allows for a streaming method with the internal RAM providing buffering as required. This removal of the need to store intermediate elements within DDR reduces not only the latency of the image processing, but also reduces the power dissipated and increases the determinism as there is no need to share access with other system resources.

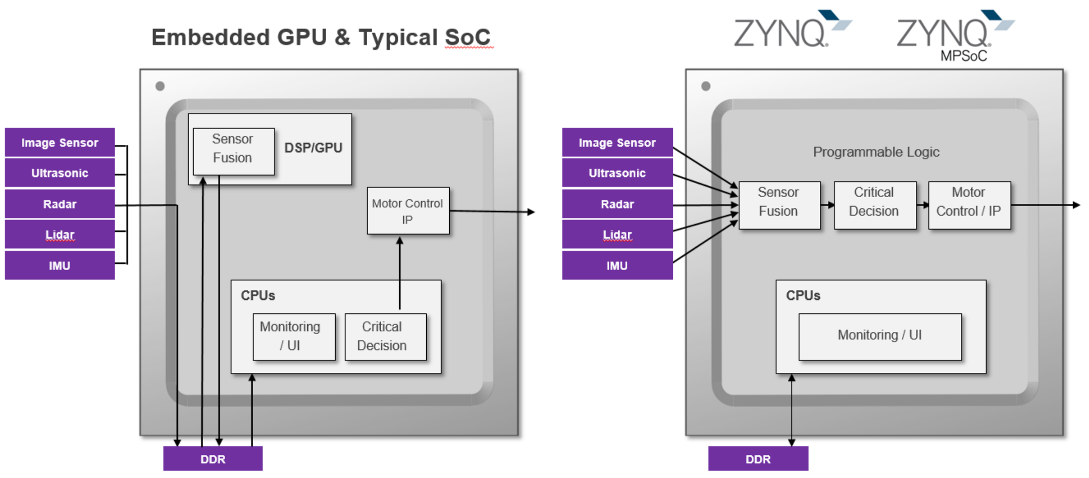
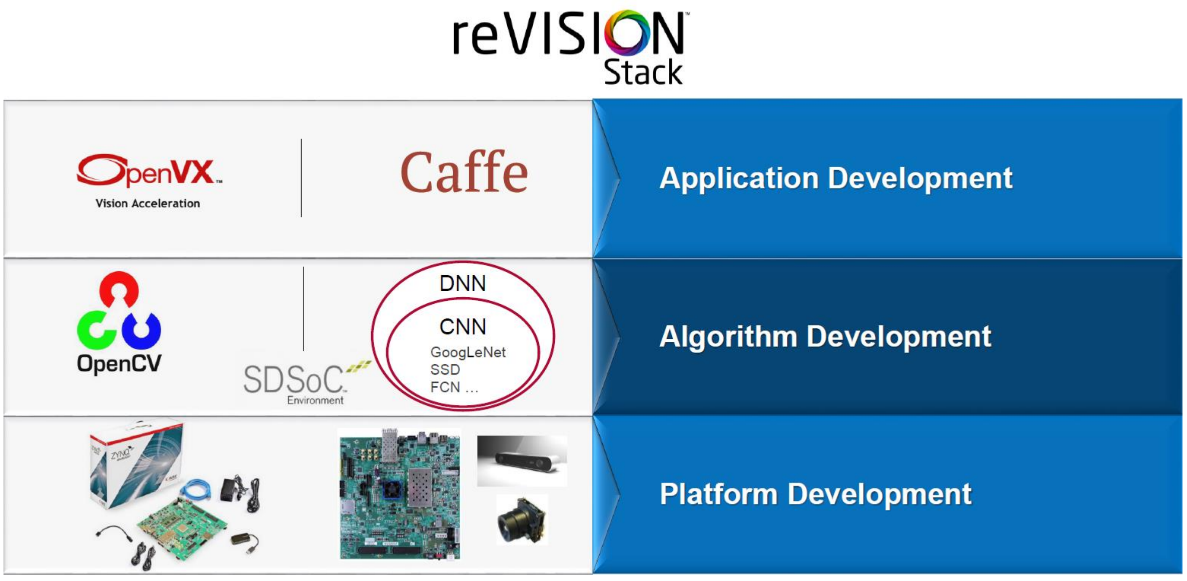


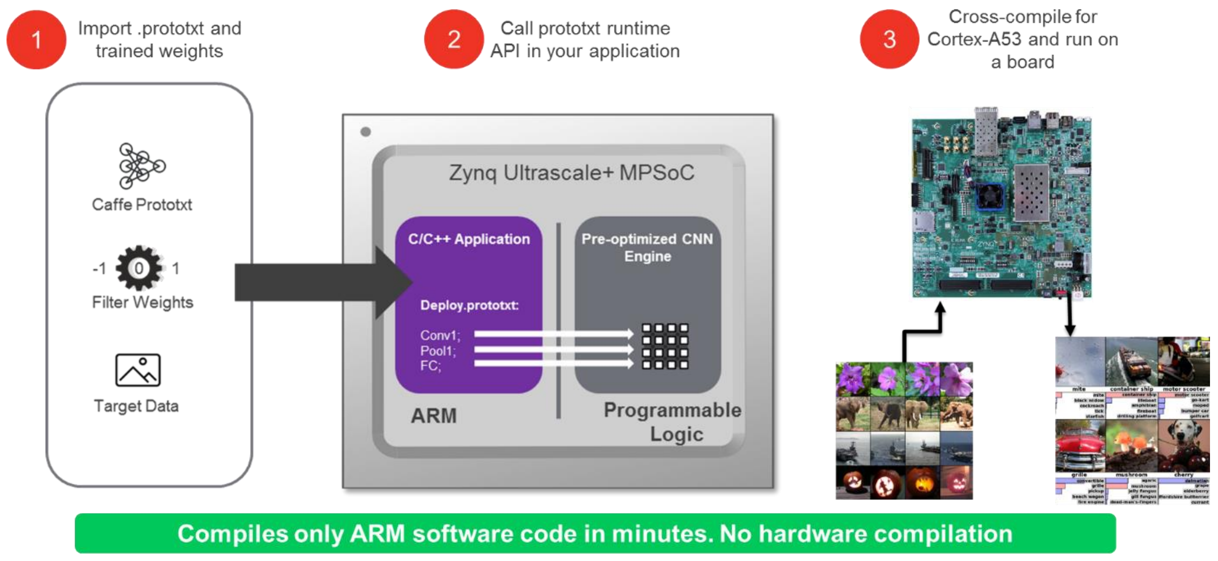
Figure 3, Benefits of the programmable logic implementation

Implementing both the image processing algorithm and the machine learning network within a heterogeneous SoC can be achieved with ease using the reVISION™ stack from Xilinx. reVISION provides support for both traditional image processing applications and machine learning applications based around the SDSoC™ tool. Within reVISION support is provided for both the OpenVX and Caffe Frameworks. To support the OpenVX framework, the core image processing functions can be accelerated into the programmable logic to create the image processing pipeline. While the machine learning inference environment provides support for hardware optimised libraries in the programmable logic to implement the inference engine which performs the machine learning implementation.



reVISION Stack

reVISION provides integration with Caffe as such implementing machine learning inference engines is as easy as providing a prototxt file and the trained weights, and the framework handles the rest. This prototxt file is then used to configure the C/C++ scheduler running on the processing system to accelerate the neural network inference on the hardware optimised libraries within the programmable logic. The programmable logic is used to implement the inference engine and contains such functions as Conv, ReLu and Pooling etc.


1
Import .prototxt and trained weights
2
Call prototxt runtime API in your application
3
Cross-compile for Cortex-A53 and run on a board
Caffe Prototxt
-1 0 1
Filter Weights
Target Data
Zynq Ultrascale+ MPSoC
C/C++ Application
Deploy.prototxt:
Conv1;
Pool1;
FC;
Pre-optimized CNN Engine
ARM
Programmable Logic
Compiles only ARM software code in minutes. No hardware compilation

Caffe Flow Integration

The number representation systems used within machine learning inference engine implementations also play a significant role in its performance. Machine learning applications are increasingly using more efficient reduced precision fixed point number systems, such as INT8 representation. The use of fixed point reduced precision number systems comes without a significant loss in accuracy when compared to a traditional floating point 32 (FP32) approach. As fixed point mathematics are also considerably easier to implement than floating point this move to INT8 provides for more efficient faster solutions in some implementations. This use of fixed point number systems is ideal for implementation within a programmable logic solution, reVISION provides the ability to work with INT8 representations in the PL. These INT8 representations enable the use of dedicated DSP blocks within the PL. The architecture of these DSP blocks enables a maximum of two concurrent INT8 Multiply Accumulate operations to be performed when using the same kernel weights. This provides not only a high-performance implementation but also one which provides a reduced power dissipation. The flexible nature of programmable logic also enables easy implementation of further reduced precision fixed point number representation systems as they are adopted.

	FP-32	FIXED-16 (INT16)	FIXED-8 (INT8)	Difference vs FP32
VGG-16	86.6%	86.6%	86.4%	(0.2%)
GoogLeNet	88.6%	88.5%	85.7%	(2.9%)
SqueezeNet	81.4%	81.4%	80.3%	(1.1%)

Accuracy of networks with different weight representations.

Source: https://arxiv.org/pdf/1510.00149v5.pdf

Real World Performance
Within the real world, the reVISION stack can provide a significant benefit. One example of an application which deploys machine learning within an embedded vision application would be a vehicle collision avoidance system. Targeting a Xilinx UltraScale+ MPSoC and developing the application within reVISION using SDSoC to accelerate functions to the programmable logic as required to optimise performance, provides a significant increase in responsiveness. When comparing the reVISION MPSoC response time against a GPU based approach when both implement a GoogLeNet solution, the difference is impressive. The reVISION design can identify a potential collision event and engage the vehicle brakes within 2.7ms (with batch size of 1) while the GPU based approach takes between 49ms and 320ms (with large batch size) depending upon its implementation. The large batch size is needed for GPU architecture to get to reasonable throughput with significant sacrifice in response time while Zynq can achieve high performance even at batch size of 1 with lowest latency. This difference in reaction time can be the difference between avoiding a collision or not.

Wrapping it up
Machine learning will continue to be a significant driving factor in many applications, especially vision guided robotics or ‘cobots’ as they are increasingly called. Heterogeneous SoC, which combine processor cores with programmable logic enable the creation of very efficient, responsive and reconfigurable solutions. The provision of stacks like reVISION, open up for the first time the benefits of programmable logic to a wider developer community and reduce the development time of the solution.