How to Develop High-Performance Deep Neural Network Object Detection/Recognition Applications for FPGA-based Edge Devices

By FARHAD FALLAH, ALDEC

NOVEMBER 7, 2019

Machine learning is the process of using algorithms to parse data, learn from it, and then make a decision or prediction. Instead of preparing program codes to accomplish a task, the machine is "trained" using large volumes of data and algorithms to perform the task on its own.

Machine learning is being revolutionized using neural network (NN) algorithms, which are digital models of the biological neurons found in our brains. These models contain layers which are connected like a brain's neurons. Many applications benefit from machine learning, including image classification/recognition, big data pattern detection, ADAS, fraud detection, food quality assurance, and financial forecasting.

As algorithms for machine learning, neural networks include a wide range of topologies and sizes consisting of multiple layers; the first layer (the "input layer"), middle layers (the "hidden layers") and the last layer (the "output layer"). Hidden layers perform a variety of dedicated tasks on the input and pass it to the next layer until, at the output layer, a prediction is generated.

Some neural networks are relatively simple and have only two or three layers of neurons, while so-called deep neural networks (DNNs) might comprise up to 1000 layers. Determining the right topology and size of the NN for a specific task requires experimentation and comparison against similar networks.

Designing a high-performance machine learning application requires network optimization, which is typically done using pruning and quantizing techniques, and computation acceleration, which is performed using ASICs or FPGAs.

In this article, we will discuss how DNNs work, why FPGAs are becoming popular for DNN inference and consider the tools you need to start designing and implementing a deep learning-based application using FPGAs [1].

Design Flow for Developing a DNN Application

Designing a DNN applica teps are choosing the right network, training th w data to the trained model for prediction (infe eps for an application to recognize cats.

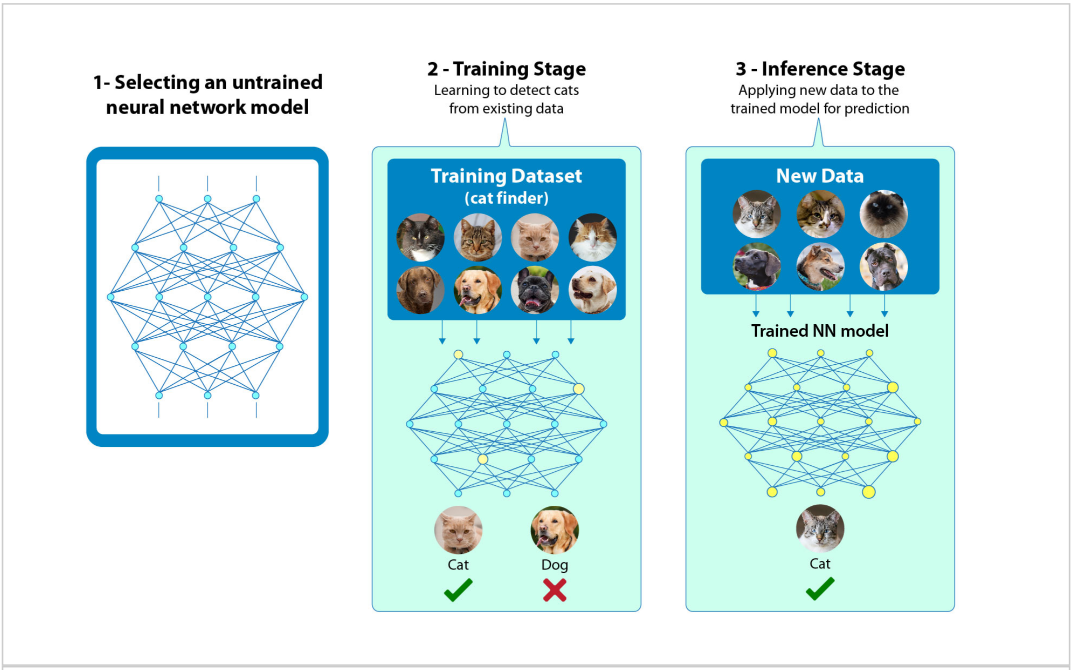

1- Selecting an untrained neural network model
2 - Training Stage
Learning to detect cats from existing data
3 - Inference Stage
Applying new data to the trained model for prediction
Training Dataset (cat finder)
New Data
Trained NN model
Cat
Dog
Cat

Figure 1. The three steps of recognizing a cat.

As mentioned, there are multiple layers in a DNN model, and each layer has a specific task. In deep learning, each layer is designed to extract features at different levels. For example, in an edge detection neural network, the first middle layer detects features such as edges and curves. The output of the first middle layer is then fed to the second layer, which is responsible for detecting higher level features, such as semicircles or squares. The third middle layer assembles the output of the other layers to create familiar objects and the last layer detects the object.

In another example, if we set out to recognize a stop sign, the trained system would include layers for detecting the octagonal shape, the color, and the letters "S," "T," "O," and "P" in that order and in isolation. The output layer would be responsible for determining if it is a stop sign.

DNN Learning Models

There are four main learning models:

- **Supervised:** In this model, all the training data is labeled. The NN classifies the input data into different labels learned from the training dataset.
- **Unsupervised:** In unsupervised learning, a deep learning model is handed a dataset without explicit instructions on what to do with it. The training dataset is a collection of examples without a specific desired outcome or correct answer. The NN then attempts to automatically find structure in the data by extracting useful features and analyzing its structure.
- **Semi-Supervised:** This consists of a training dataset with both labeled and unlabeled data. This method is particularly useful when extracting relevant

features from the dat [illegible] es is a time-intensive task for experts.

- **Reinforcement:** This is [illegible] ork to get the results and improve the performance. It is an iterative process: the more rounds of feedback, the better the network becomes. This technique is especially useful for training robots, which make a series of decisions in tasks like steering an autonomous vehicle or managing inventory in a warehouse.

Training vs Inference

In training, the untrained neural network model learns a new capability from the existing data. Once the trained model is ready, it is fed new data and the performance of the system is measured. The ratio of detecting the image correctly is called inference.

In the example given in Figure 1 (recognizing a cat), after inputting the training dataset, the DNN starts tuning the weights to find cats; where a weight is a measure of the strength of the connection between each neuron.

If the result is wrong, the error will be propagated back to the network's layer to modify the weights. This process happens again and again until it gets the correct weighting, which results in getting a correct answer every time.

How to Achieve A High-Performance DNN Application

Using DNN for classification requires a big dataset, which increases the accuracy. However, a drawback is that it produces many parameters for the model, which increases the compute cost and requires high memory bandwidth.

There are two main ways to optimize a DNN application. The first is network optimization through pruning redundant connections and quantizing the weights and fusing the neural networks to narrow down the network size.

- **Pruning:** This is a form of DNN compression. It reduces the number of synaptic connections to other neurons so that the overall amount of data is reduced. Typically, weights close to zero are removed. This can help eliminate the redundant connections with minor accuracy drops for tasks such as classification [2].

- **Quantization:** The is done to bring the neural network to a reasonable size while also achieving high-performance accuracy. This is especially important for edge applications, where the memory size and number of computations are necessarily limited. In such applications, to get better performance the model parameters are held in the local memory to avoid time-consuming transfers using PCIe or other interconnection interfaces. In this method, the process of approximating a neural network that uses floating-point numbers (FTP32) by a neural network of low-bit width numbers (INT8) is performed. This dramatically reduces both the memory requirement and computational cost of using neural networks. By quantizing the model, we lose precision and accuracy slightly. However, for most applications there is no need for a 32-bit floating point.

The second way to optim on acceleration, using ASICs or FPGAs. Of the benefits for machine learning applications. The

- **Power Efficiency:** FPGAs provide a flexible and customizable architecture, which enable the usage of only the compute resources that we need. Having low-power systems for DNN is critical in many applications, such as ADAS.

- **Reconfigurability:** FPGAs are considered raw programmable hardware compared to ASICs. This feature makes them easy to use and reduces the time to market significantly. To catch up with daily-evolving machine learning algorithms, having the capability to reprogram the system is extremely beneficial rather than waiting for a long fabrication time of SoCs and ASICs.

- **Low Latency:** Block RAM inside the FPGA provides at least 50 times faster data transfer compared to the fastest off-chip memories. This is a game-changer for machine learning applications, for which low latency is essential.

- **Performance Portability:** You can get all the benefits of the next generation of FPGA devices without any code modification or regression testing.

- **Flexibility:** FPGAs are raw hardware and can be configured for any architecture. There is no fixed architecture or data paths to tie you down. This flexibility enables FPGAs to do massive parallel processing, since the data path could be reconfigured at any time. The flexibility also brings any-to-any I/O connection capability. This enables FPGAs to connect to any device, network, or storage devices without the need for a host CPU.

- **Functional Safety:**: FPGAs users can implement any safety feature in the hardware. Depending on the application, encoding could be done with high efficiency. FPGAs are widely used in avionics, automation, and security, which are the proof of functional safety in these devices that machine learning algorithms could benefit from it.

- **Cost Efficiency:** FPGAs are reconfigurable and the time to market for an application is pretty low. ASICs are very costly, and the fabrication time takes 6 to 12 months, if no errors show up. This is an advantage for machine learning applications, since the cost is very important and NN algorithms are evolving daily.

Modern FPGAs typically offer a rich set of DSP and BRAM resources within their fabric that can be used for NN processing. However, compared to the depth and layer size of DNNs, these resources are no longer enough for a full and direct mapping; certainly not in the way it was often done in previous generations of neural network accelerators. Even using devices like the Zynq MPSoC (where even the largest device is limited to 2k DSP slices and a total BRAM size of less than 10 MB) a complete mapping with all neurons and weights directly onto the FPGA is not possible.

So, how can we use the power efficiency, re-programmability, low latency, and other features of FPGAs for deep learning?

New NN algorithms and architectural modification are required to enable the inference of DNNs on platforms with limited memory resources such as FPGAs.

A modern DNN divides t ks to be processed by FPGAs. Since the on-chi gh for storing all the required weights for a ore the weights and parameters for the curre an external memory (which could be a DDR memory).

However, transferring data back and forth between the FPGA and memory is going to increase the latency up to 50 times. The first thing that springs to mind is to reduce the memory data. In addition to the network optimization discussed above (pruning and quantization), there are:

- **Weight Encoding:** In the FPGA, the encoding format can be chosen with no obligation. There might be some accuracy loss, however this would be negligible compared to the latency caused by data transferring and the complexity of its processing. Weight encoding created Binary Neural Networks (BNN) where the weights are reduced to only one bit. This method shrinks the amount of data for transferring and storing, as well as the computation complexity. However, this method only results in small reductions for hardware multipliers with a fixed input width.
- **Batch processing:** In this method, we reuse the weights already on the chip for multiple inputs using a pipelining method. It also reduces the amount of data for transferring from off-chip memory to the FPGA [5].

Design and Implementation of DNN Applications on FPGAs

Let's dive into implementing a DNN on an FPGA. In doing so we'll take advantage of the most appropriate commercially available solutions to fast-track the development of an application.

For instance, Aldec has an embedded development board called the TySOM-3A-ZU19EG (https://www.aldec.com/en/products/emulation/tysom_boards/zynq_ultrascale_mps
Along with a wide range of peripherals, it carries the largest FPGA in the Xilinx Zynq UltraScale+ MPSoC family, a device that has over a million logic cells and includes a quad-core Arm Cortex-A53 processor running up to 1.5GHz.

Importantly, for our purposes, this mammoth MPSoC also supports Xilinx's deep learning processing unit (DPU), which the company created for machine learning developers.

The DPU is a programmable engine dedicated to convolutional neural network (CNN) processing. It is designed to accelerate the computing workloads of DNN algorithms used in computer vision applications such as image/video classification and object tracking/detectio.

There is a specific instruction set for the DPU, which enables it to work efficiently with many CNNs. Like a regular processor, a DPU fetches, decodes, and executes instructions stored in DDR memory. This unit supports multiple CNNs such as VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, FPN, etc. [3].

The DPU IP can be integ   nable logic (PL) of the selected Zynq®-7000 Sc   C devices with direct connections to the proce:

To create the instructions for DPU, Xilinx provides a Deep Neural Network Development Kit (DNNDK) toolkit. Xilinx states:

The DNNDK is designed as an integrated framework, which aims to simplify and accelerate deep learning application development and deployment on the Deep Learning Processor Unit (DPU). DNNDK is an optimizing inference engine, and it makes the computing power of DPU become easily accessible. It offers the best of simplicity and productivity to develop deep learning applications, covers the phases of neural network model compression, programming, compilation, and runtime enablement [4].

The DNNDK framework comprises the following units:

- **DECENT:** Performs pruning and quantization to satisfy low latency and high throughput
- **DNNC:** Maps the neural network algorithm to the DPU instructions
- **DNNAS**: Assembles DPU instructions into ELF binary code
- **N^{2}Cube:** Acts as the loader for the DNNDK applications and handles resource allocation and DPU scheduling. Its core components include DPU driver, DPU loader, tracer, and programming APIs for application development.
- **Profiler:** Consists of DPU tracer and DSight. D tracer gathers the raw profiling data while running NNs on the DPU. DSight uses this data to generate the visualized charts for performance analysis.
- **Dexplorer:** Provides running-mode configuration, status checking, and code signature checking for the DPU.
- **DDump:** Dumps the info inside the DPU ELF, hybrid executable, or DPU shared library. It accelerates debug and analysis for users.

These fit into the flow shown in Figure 2.

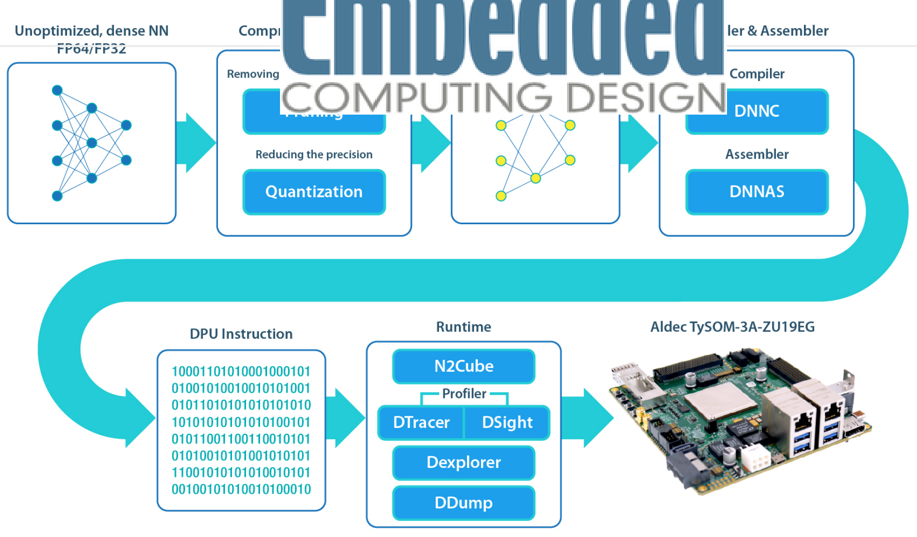


Figure 2. The above Deep Neural Network Development Kit (DNNK) framework makes the design process of an FPGA-based machine learning project much easier for developers.

Using DNNDK makes the process of designing an FPGA-based machine learning project much easier for developers; In addition, platforms like Aldec's TySOM-3A-ZU19EG board are also there to provide an invaluable kick-start. For instance, Aldec has prepared some examples – including gesture detection, pedestrian detection, segmentation, and traffic detection – that target the board, meaning developers are not starting with a blank sheet.

Let us consider one demo that was showcased at Arm TechCon earlier this year. It was a traffic-detection demonstration built using a TySOM-3A-ZU19EG and an FMC-ADAS daughter card that provides interfaces and peripherals for 5x high-speed data (HSD) cameras, RADAR, LIDAR, and ultrasonic sensors – sensory inputs for most ADAS applications.

Figure 3 shows the architecture of the demo. There are two DPUs implemented in the FPGA that are connected to the processing unit over AXI HP ports to perform deep learning inferencing tasks such as image classification, object detection, and semantic segmentation. The DPUs require instructions to implement a neural network, which are prepared by DNNC and DNNAS tools. They also need access to memory locations for input videos as well as output data.

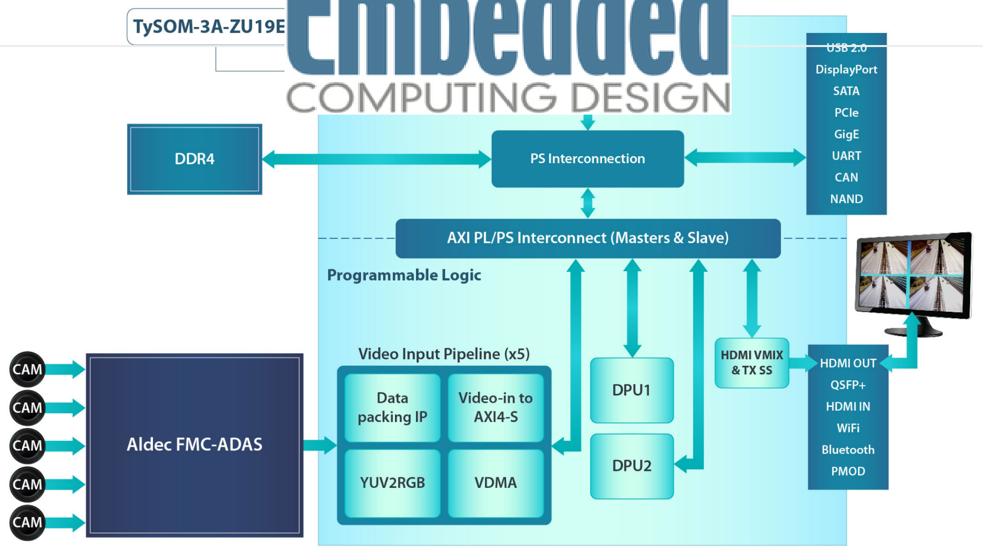


Figure 3. The traffic-detection demo has 5x video input pipelines that are used for data packing, AXI4 to AXI stream data transferring, color space conversion (YUV2RGB), and sending video to memory.

An application is run on the application processing unit (APU) to control the system by managing interrupts and perform data transfer between units. The connection between the DPU and the user application is through a DPU API and Linux driver. There are functions to read a new image/video to the DPU, run the processing, and send the output back to the user application.

Developing and training the model is done using Caffe outside of the FPGA, whereas optimization and compilation is done using DECENT and DNNC units provided as a part of the DNNDK toolkit (figure 2). In this design, the SSD object detection CNN is used for background, pedestrian, and vehicle detection.

In terms of performance, 45 fps was achieved using four input channels, demonstrating the high-performance deep learning application using the TySOM-3A-ZU19EG and the DNNDK toolkit.

Farhad Fallah is an Applications Engineer at Aldec, Inc.

References:

[1] Guo, Kaiyuan, et al. "A survey of fpga-based neural network accelerator"

[2] FPGA-based Accelerators of Deep Learning Networks for Learning and Classification: A Review

[3] DPU for convolutional neural network "Xilinx.com"

[4] DNNDK user guide "Xilinx.com"

[5] Efficient deep neural network acceleration through FPGA-based batch processing

Embedded
COMPUTING DESIGN

NEXT RECOMMENDATION
(https://www.embedded-computing.com/guest-blogs/gaining-an-edge-with-ai)

Gaining an Edge with AI
AI research now offers the possibility of using thousands of hours of powerful computing resource...

0 Comments Embedded Computing Design 1 Login

Recommend Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd

Recommended

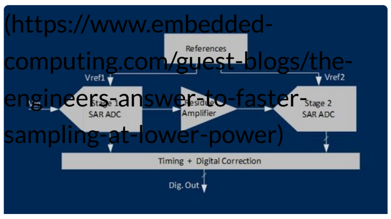
(https://www.embedded-computing.com/guest-blogs/gaining-an-edge-with-ai)

Gaining an Edge with AI

AI research now offers the possibility of using thousands of hours of powerful computing resources to train and refine a neural network that will then be able to run on cheap, low-power devices.

Read Article

(https://www.embedded-computing.com/guest-blogs/the-engineers-answer-to-faster-sampling-at-lower-power)



The Engineer's Answer to Faster Sampling at Lower Power

We live in an analog world, despite the dominance of digital technology. Moving between domains inevitable.

Read Article

(https://www.embedded-computing.com/home-page/sensiml-launches-fre version-of-its-analytics-to

SensiML launches fr trial version of its Analytics Toolkit

SensiML Corporation, a deve of AI tools for building IoT endpoints, announced the re Analytics Toolkit and introdu of its Data Depot datase.

Read Article

Return to Home (https://www.embedded-computing.com/)