A Test Drive of the NVIDIA Jetson TX1 Developer Kit for Deep Learning and Computer Vision Applications

By the staff of

June 2016

OVERVIEW

The Jetson TX1 module is NVIDIA's latest processor system-on-module for embedded applications, based on the Tegra X1 chip. The Jetson TX1 Developer Kit is a low-cost, feature-rich development kit based on the Jetson TX1 module.

BDTI, a technology analysis firm, used the Jetson TX1 Developer Kit to develop a deep-learning-based computer vision application—a camera that recognizes objects—leveraging the popular OpenCV and Caffe software packages. BDTI also implemented a classical computer vision algorithm on the Jetson TX1 kit, leveraging software packages provided by NVIDIA. Based on BDTI's experience developing the deep-learning-based application and the classical computer vision algorithm, this report presents an independent evaluation of the ease of use of the Jetson TX1 kit for new users. This report is intended for developers and managers interested in an independent perspective on the Jetson TX1 kit.

Setting up and using the Jetson TX1 Developer Kit was quick and easy. Within a few days, an engineer with no prior OpenCV or Caffe experience was able to create a real-time implementation of a smart camera application on the Jetson TX1 board, making use of GPU-accelerated versions of OpenCV and Caffe to obtain real-time performance without having to optimize code himself. A computer vision expert was able to quickly further optimize the deep-learning application and was able to leverage NVIDIA libraries and SDKs to quickly implement a classical computer vision algorithm.

Contents

1. Introduction

This report is intended to provide an understanding of what it's like to get started using the NVIDIA Jetson TX1 Developer Kit for developing embedded applications—especially applications incorporating deep learning and computer vision.

BDTI, a technology analysis firm, used the Jetson TX1 Developer Kit to develop a deep-learning-based computer vision application. BDTI also implemented a classical computer vision algorithm on the Jetson TX1, leveraging software packages provided by NVIDIA. Based on BDTI's experience developing the deep learning application and the computer vision algorithm, this report presents an independent evaluation of the ease of use of the Jetson TX1 Developer Kit for new users.

This brief report is intended for developers and managers interested in an independent perspective on the Jetson TX1 kit.

2. BDTI

BDTI is a technology analysis and software development firm specializing in embedded computer vision and deep learning applications. Over the past 24 years, BDTI has performed in-depth, hands-on evaluations of more than 100 CPUs, DSPs, GPUs FPGAs and associated development kits and tools. For more information about BDTI, visit www.BDTI.com. For questions about this report, email us at info@BDTI.com.

3. The Tegra X1 and Jetson TX1

The Tegra X1 is NVIDIA's latest system-on-chip processor for embedded applications. The Jetson TX1 is NVIDIA's system-on-module based on the Tegra X1 chip. The Tegra X1 CPU subsystem comprises four ARM Cortex-A57 cores. (Four lower-performance Cortex-A53 cores are also included for lower-power operation in mobile devices, but are not enabled on the Jetson TX1 module). The Tegra X1 also features a 256-core GPU based on NVIDIA's "Maxwell" architecture. For compute-intensive applications, the GPU is the most interesting feature of this chip. Using NVIDIA's CUDA technology, the GPU can be programmed to perform a wide range of parallel computation tasks in addition to handling 3D graphics.

The Jetson TX1 Developer Kit is a low-cost development kit based on the Jetson TX1 module. It's the successor to the Jetson TK1 kit, which was based on NVIDIA's previous-generation Tegra K1 chip.

The Jetson TX1 Developer Kit is intended to enable embedded system and application developers to evaluate the Tegra X1 processor and to quickly create embedded applications and systems. The Jetson TX1 kit is particularly aimed at developers working in computer vision, deep learning, robotics and related fields.

The centerpiece of the Jetson TX1 Developer Kit is the Jetson TX1 development board, shown in Figure 1. In addition to the Jetson TX1 processor system-on-module, the board is equipped with a wide range of I/O interfaces, including USB, HDMI, Ethernet and a general-purpose I/O header. In addition, the board includes WiFi and a camera. The board comes

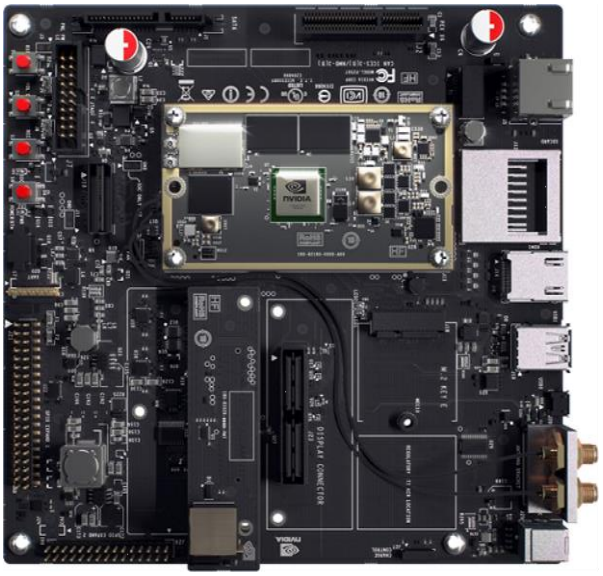

Figure 1 The Jetson TX1 development board

preloaded with Ubuntu Linux 14.04 LTS and drivers supporting the platform.

4. BDTI's Evaluation Methodology

Since NVIDIA's CUDA technology is widely used in computer vision and deep learning, we expect that many users of the Jetson TX1 Developer Kit will be developing embedded computer vision applications using deep learning. Many of these developers will have little or no experience with embedded processors. They will want to leverage existing application software and libraries developed for servers or workstations.

In evaluating the Jetson TX1 Developer Kit, BDTI followed a process that we believe is typical of how such developers will use the kit. We began by defining a representative real-time, deep-learning-based computer vision application that relies on open source software. (The application is described in Section 6.)

We followed what we expect to be the path of a typical user of the kit: We began by bringing up the board and configuring it, and then proceeded with developing and debugging our application.

In addition to the deep learning application, BDTI evaluated the Jetson TX1 Developer Kit's support for classical computer vision algorithms. BDTI implemented a simple computer vision algorithm (described in Section 7) using two different NVIDIA-provided APIs: OpenCV4Tegra and VisionWorks. With each API we created an independent implementation of the algorithm, following what we expect to be the path of a typical algorithm developer or implementer.

5. Computer Vision, Deep Learning and CUDA

"Computer vision" refers to the automated extraction of meaning from images and video. Computer vision has been an active field of research for decades, but until recently has had few major commercial applications. However, with the advent of high-performance, low-cost, energy efficient processors, in recent years it has become feasible to deploy computer vision in a wide range of applications spanning mobile devices, embedded systems, PCs and the cloud [1]. We use the term "embedded vision" to refer to this new wave of widely deployed, practical computer vision applications.

Traditionally, computer vision applications have relied on a diverse range of special-purpose algorithms painstakingly designed to recognize specific types of objects (such as faces) and features (such as shapes). Recently, however, convolutional neural networks (CNNs) have been shown to be superior to traditional algorithms for a range of image classification tasks. In contrast to traditional vision algorithms, CNNs are generalized recognition algorithms that are trained through examples to recognize specific classes of objects [2][3]. The term "deep neural network" (or "deep learning") is applied to neural networks with more than two layers of neurons—which includes virtually all neural networks used for computer vision tasks.

CNNs tend to be very computationally demanding and contain extensive parallelism. This makes them natural candidates for acceleration on parallel processors. With its CUDA technology (which encompasses the GPU architecture and application programming interface), NVIDIA pioneered the use of GPUs as general-purpose parallel processors. In research environments, high-performance NVIDIA GPUs are often used to accelerate both the training phase and the recognition phase of CNNs. For example, Caffe [4] is a popular open source deep learning framework developed at U.C. Berkeley, which is often used with NVIDIA GPUs on PCs and servers.

6. BDTI's Deep Learning Application

As an example of a typical deep-learning-based computer vision application, BDTI devised a smart camera for home monitoring applications. This device recognizes specific classes of objects in or around the home and alerts the owner. Such a device could be used to alert a home owner to the presence of their pet inside the home, for example, or to the presence of people outside the home.

Typical of such applications, BDTI made extensive use of open source software. In our application, OpenCV is used for scaling video frames from the camera to the appropriate size. Resized video frames are fed into a deep neural network that has been trained to recognize a large number of object types. The neural network (a variation on the AlexNet design [5] , shown in Figure 2) is implemented using the Caffe open source framework. For purposes of our evaluation, the application uses OpenCV to annotate video frames with its best object

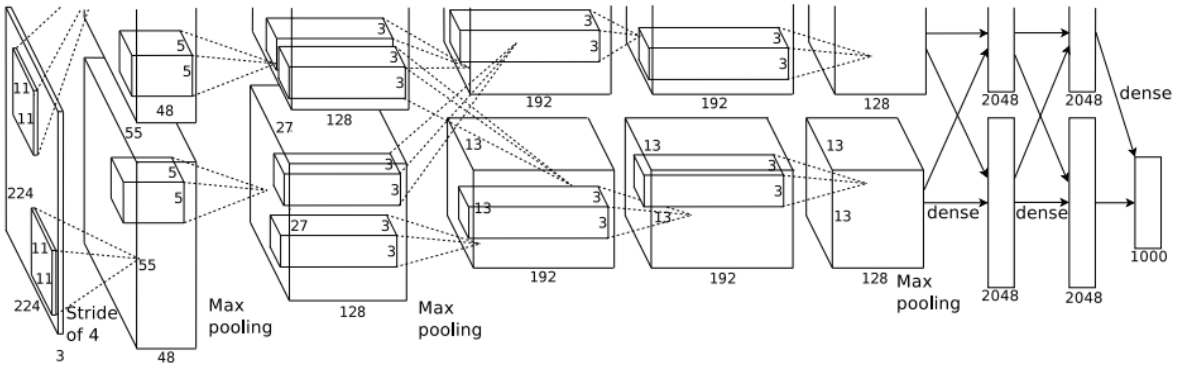

Figure 2 High-level block diagram of the AlexNet convolutional neural network (Source: Krizhevsky et al. [5])

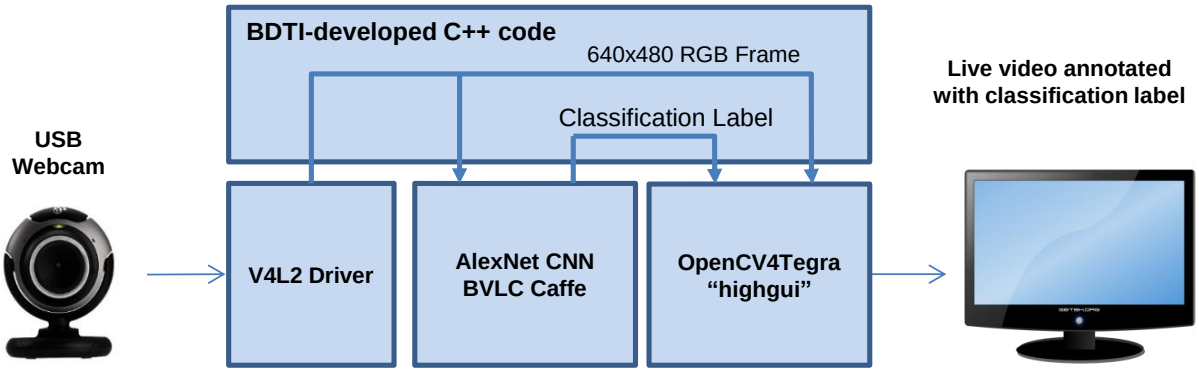

Figure 3 Software architecture of BDTI's deep learning application

recognition guesses, and outputs video to the monitor connected to the Jetson TX1 board in real time. In an actual product, these outputs could be used to generate text message alerts, for example. The software architecture of BDTI's deep learning application is shown in Figure 3.

7. BDTI's Computer Vision Algorithm

Background subtraction is a classical computer vision task that is prevalent in video surveillance. BDTI implemented a simple yet effective background subtraction algorithm described in [6] and [7] . A diagram of this background subtraction algorithm is shown in Figure 4 and example input and output are shown in Figure 5.

8. OpenCV and OpenCV4Tegra

OpenCV is a popular open source library of computer vision and related functions [www.opencv.org]. OpenCV comprises a comprehensive set of over 2,500 functions ranging from simple building-block functions such as matrix arithmetic functions to substantial computer vision modules such as object detection and image stabilization. OpenCV enables developers to quickly implement and test sophisticated computer vision algorithms. While all OpenCV functions can execute on a CPU alone, OpenCV also includes CUDA and OpenCL implementations of many of its functions, thus allowing the developer to utilize a GPU in addition to the CPU. To use these GPU Compute facilities in OpenCV 2.4.x, one needs to explicitly invoke certain functions from specific modules, whereas in OpenCV 3.x the OpenCL optimizations are more seamlessly integrated.

NVIDIA's OpenCV4Tegra is a closed-source port of OpenCV that is optimized specifically for the Tegra architecture, containing a mix of OpenGL and CUDA optimizations (OpenGL acceleration being a legacy of OpenCV4Tegra's Android heritage, which never offered CUDA support). While "plain vanilla" OpenCV includes some ARM NEON optimizations, OpenCV4Tegra goes above and beyond with Tegra specific optimizations in 50+ functions (see

[9] for a complete list). These Tegra optimizations are used by default, without requiring the user to explicitly call GPU-enabled versions of OpenCV functions. OpenCV4Tegra provides excellent compatibility with OpenCV, promising seamless porting of code developed with the open source OpenCV library.

9. OpenVX and VisionWorks

OpenVX is an open standard from the Khronos Group aimed at enabling low power computer vision applications in mobile and embedded devices [8] . OpenVX consists of a software framework and a library of common computer vision building blocks. The OpenVX framework allows developers to describe their computer vision algorithms in the form of a dataflow graph. The framework can then execute the algorithm with dataflow optimized for the device architecture. For example, the OpenVX framework can automatically apply image tiling techniques to greatly reduce bandwidth to off-chip memory, improving speed and reducing power consumption. Because the graph-based programming approach allows OpenVX to optimize dataflow, OpenVX promises better efficiency (higher speed and/or lower power consumption) compared to the more familiar programming model of function libraries such as OpenCV.

NVIDIA's VisionWorks is a software development package for computer vision and image processing. VisionWorks supports NVIDIA's Tegra TK1 and TX1, and NVIDIA's Kepler and Maxwell generations of discrete GPUs. At its core VisionWorks is a full-fledged implementation of the OpenVX standard. However, NVIDIA provides significant additional functionality above and beyond the core OpenVX 1.0.1 standard, including:

- A large number of custom OpenVX extensions that significantly expand the number of basic building nodes, including (but not limited to) specialized color conversion, stereo matching, FAST keypoint extraction, and Hough shape detectors.
- NVXIO library for providing sophisticated rendering tools and wrapping of hardware codecs for ingest of compressed video (emanating either from media content on disk or streamed from USB or IP cameras).
- A Structure from Motion library, allowing

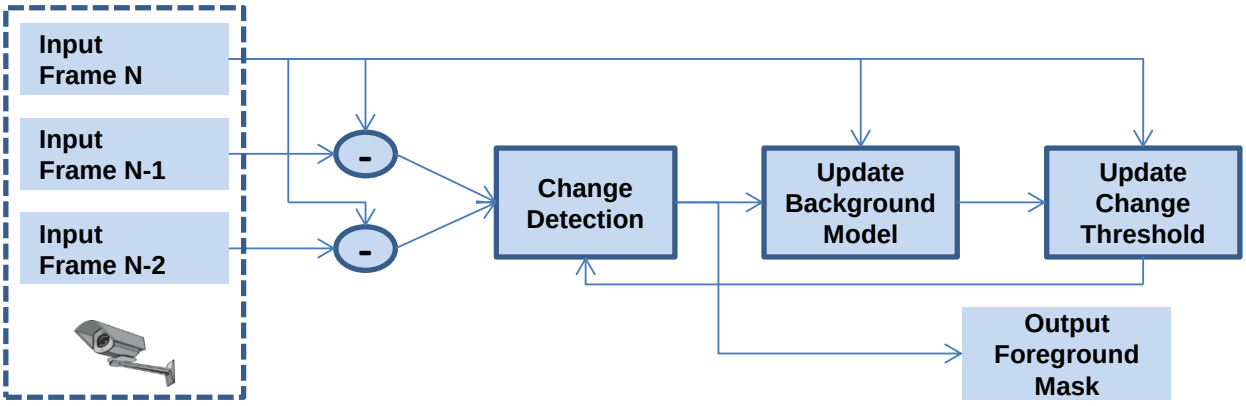

Figure 4 High-level block diagram of the background subtraction algorithm

Figure 5 Example input and output of background subtraction

developers to experiment with 3D reconstruction.

Lastly, the package includes a set of non-trivial vision sample applications to demonstrate effective usage of VisionWorks. Figure 6, taken from the VisionWorks Toolkit Reference Documentation, illustrates how VisionWorks fits into the Tegra tool flow.

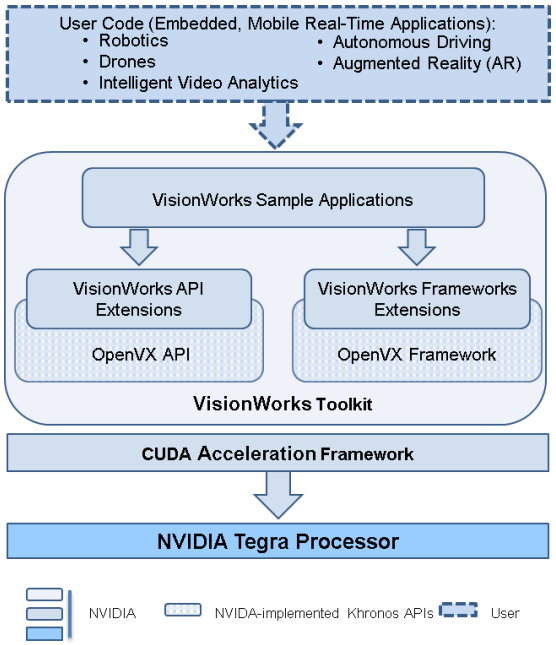

Figure 6 VisionWorks in the Tegra tool flow

10. OpenCV4Tegra vs. VisionWorks

As explained above, OpenCV4Tegra and VisionWorks are based on OpenCV and OpenVX, respectively. These complementary APIs are intended to address different facets of computer vision application development: OpenCV is a broad library covering a vast array of computer vision, image processing, and other related functions. OpenCV aims to allow developers to quickly prototype a computer vision application. In contrast, OpenVX aims to enable deployment of computer vision applications with optimal speed and power consumption as described in Section 9 above. OpenVX provides a deliberately constrained set of computer vision and image processing functions.

OpenCV4Tegra is an easier starting point than VisionWorks for most developers due to its simpler programming model and breadth of support of OpenCV functions. Some developers will find that the Tegra-specific optimizations provide sufficient performance for their applications. VisionWorks promises superior performance and power consumption, at the cost of a longer learning curve, less intuitive programming model, and narrower range of functions.

11. Our Jetson TX1 Development Experience

Jetson TX1 Setup

To begin experimenting with the Jetson TX1 Developer Kit all that is required after opening the box is connecting power, keyboard, mouse and monitor. A USB3 hub is recommended so that a keyboard, mouse, and USB webcam can be connected simultaneously to the board. We also recommend running the Jetson JetPack installer utility to generate an updated system image before beginning work with the board.

With keyboard, mouse and monitor connected, the Jetson TX1 board appears and behaves very much like an Ubuntu PC. This is not typical of embedded development boards, and will be especially welcomed by developers lacking extensive experience with embedded systems.

Typically, significant effort is required to generate a system image for an embedded development board (often requiring compiling the image from source code) and to install ("flash") the image onto the board. NVIDIA's Jetson JetPack installer simplifies this process for the Jetson TX1 board.

Jetson JetPack runs on a host system running Ubuntu 12.04 or 14.04, and communicates with the Jetson TX1 board via USB. We installed JetPack on an Ubuntu 14.04 Virtual Machine, and it took about 30 minutes to install the OS and requested packages on the board, with occasional user input required. Compared to typical embedded development boards, Jetson JetPack dramatically reduces the complexity of getting the Jetson TX1 board loaded with the operating system and libraries, and also simplifies the process of updating the OS. The trade-off for this automation is reduced visibility into the process of assembling the system image and flashing the board. We were impressed by how complete and trouble-free installation, execution and testing were using Jetson Jetpack. We were left with a fully configured Jetson TX1 board, ready for development.

BDTI's Deep Learning Application Development

As mentioned earlier, BDTI's home monitoring camera application makes use of OpenCV functions for video frame acquisition and scaling. For our OpenCV functions on the Jetson TX1 board we used NVIDIA's OpenCV4Tegra. OpenCV4Tegra was installed by Jetpack, and was simple to use, as it provides the same API as the generic OpenCV library.

Jetson Jetpack also installed cuDNN, a GPU-accelerated library of primitives for deep neural networks, which is used by Caffe and many other deep learning frameworks to accelerate CNN execution on NVIDIA GPUs. In addition, Jetson Jetpack installed a number of example applications demonstrating GPU performance, the Firefox web browser and a few typical Ubuntu desktop applications.

The one difficulty we encountered with the Jetson TX1 platform itself was that while the on-board camera worked with GStreamer, we were unable to get the OpenCV4Tegra VideoCapture facility to work with GStreamer. Rather than spend time troubleshooting this, we instead connected a USB webcam, which was automatically detected and set up by the operating system. We verified correct operation of the USB webcam using the Linux "cheese" utility.

Installation of Caffe was straightforward; in fact, it was easier to get Caffe up and running with GPU acceleration compared to getting the same software running properly on a Ubuntu PC with a Titan X discrete GPU card. The biggest challenge in developing our application was getting the USB webcam working in our C/C++ application. While GStreamer worked, we were unable to easily get the USB webcam running with OpenCV's VideoCapture module. We instead ended up using lower-level V4L2 (Video4Linux2) code to stream frames into our application. We then instantiated a cv::Mat data structure which was passed on to Caffe and displayed to the screen using OpenCV's "highgui" module.

The complexity of setting up the environment and developing our application on the Jetson TX1 was comparable to what we experienced on our Linux workstation. In fact, we simultaneously performed the same application development steps using our desktop Ubuntu 14.04 LTS system. Getting our application up and running on the Jetson TX1 kit was quicker than doing the same on our desktop system.

Our deep-learning-based home monitoring camera application performs live object recognition at frame rate (30 frames per second), with data coming in from a USB webcam. The CNN classification (implemented via the Caffe framework) clocked in approximately 12.5 ms per frame, which provides ample headroom to allow for the kinds of frame pre-processing and other vision tasks encountered in typical full-blown vision applications. Note that the 12.5 ms classification time is based on classifying one image at a time. By classifying multiple frames at a time, per-frame classification time can be reduced, at the cost of increased latency.

BDTI's Computer Vision Algorithm Development

We implemented our background subtraction algorithm in two different ways: first using VisionWorks and again using OpenCV4Tegra, as described above.

VisionWorks has a "polished" feel. First, installation is quite straightforward, simply a matter of checking the correct boxes during the Jetson Jetpack installation process. Building the library components and code samples proceeded without problems, and all of the samples ran "out of the box."

Given that VisionWorks is built upon a foundation of OpenVX, it makes sense that the code samples leverage the framework. Reviewing the provided code samples, we appreciated that (in contrast to most OpenVX examples found on the web) these are bona-fide, non-trivial end-to-end computer vision pipelines. Too often developers wishing to learn OpenVX encounter basic edge detection or simplistic image filtering examples that are too rudimentary to illustrate the capabilities and nuances of the API. In contrast, the NVIDIA samples perform real vision tasks, leveraging OpenVX, and most of the custom OpenVX extensions that NVIDIA provides as part of the VisionWorks suite. For example, NVIDIA includes a feature tracker, stereo matcher and shape detector samples.

NVIDIA also provides some simpler samples to demonstrate specific functionality such as interoperability with various other libraries (e.g. CUDA, OpenGL, and OpenCV), video playback, and camera capture.

We faced several complications in implementing our background subtraction pipeline with OpenVX. The algorithm operates on

grayscale (monochrome) video, which is common in video surveillance, yet ingest of grayscale video does not seem to be natively supported under NVXIO. Import of uncompressed single-channel AVI video failed, as did TIFF grayscale image sequences (NVXIO returned RGB frames). As a result, we ended up working with RGB video input. Fortunately, OpenVX provides a color conversion node that allowed us to extract monochrome data from RGB video.

Our background subtraction vision pipeline accesses a current frame and two delayed frames. We used an OpenVX delay object to incorporate a three-frame ring buffer into the frame processing loop. Unfortunately delay objects are not well documented in the OpenVX specification, and it took some trial and error to correctly utilize the delay object.

In addition, our algorithm uses some pixel-wise arithmetic that is somewhat awkward to implement in the form of an OpenVX graph, and might have required us to implement a custom OpenVX user node. After some experimentation, we chose to make direct calls to OpenVX kernels instead of utilizing OpenVX's data flow oriented graph construction. This is not recommended practice, since the OpenVX framework can provide much higher performance when executing a computer vision pipeline defined as a graph. But this choice allowed us to bypass the subtleties of the OpenVX specification and quickly get our code working. We used NVXIO functions to render the output.

BDTI also developed the same background subtraction algorithm using OpenCV4Tegra, without utilizing the VisionWorks framework. In the OpenCV4Tegra implementation of the algorithm, all rendering was performed using the "highgui" OpenCV module, and in contrast to our VisionWorks implementation of the algorithm we were able to ingest grayscale video contained in an AVI file.

OpenVX can in theory deliver better performance than OpenCV4Tegra due to its ability to optimize the data flow of user-defined graphs. However, OpenVX's graph-based programming model is not always intuitive to programmers, especially those accustomed to the much more straight-forward approach of OpenCV. Because we did not use OpenVX's graph construction capabilities, we were not able to validate the performance advantage of OpenVX.

12. Conclusions

Among embedded development kits, the Jetson TX1 Developer Kit stands out in four respects. First, developing applications on the Jetson TX1 board feels more like developing on a PC than developing on a typical embedded development board. Second, the Jetson JetPack installer makes it easy to install a custom-configured or updated system image on the board. Third, "out of the box" support for CUDA and a comprehensive set of associated software packages (including cuDNN, OpenCV4Tegra, and VisionWorks) enable developers to seamlessly leverage GPU compute to accelerate their applications without having to delve into the complexities of GPU programming—and facilitates re-use of existing PC and server code. Lastly, Jetson TX1 performance on computer vision and deep learning applications is impressive; we were easily able to achieve 30 frames per second with our deep learning demo application—without doing any optimization. (And, considering the 12.5 ms classification time of our neural network, it's likely that the application would run significant faster than 30 FPS if used with a camera supporting higher frame rates.)

Overall, setting up and using the Jetson TX1 board was quick and easy. Within a few days, an engineer with no prior OpenCV or Caffe experience was able to create an implementation of a smart camera application on the Jetson TX1 board, making use of GPU accelerated versions of OpenCV (via OpenCV4Tegra) and Caffe (via cuDNN) to obtain real-time performance without having to optimize code himself.

13. References

[1] J Bier. *What Can You Do With Embedded Vision?* Embedded Vision Summit presentation, April 2013. http://bit.ly/1QlRGp9

[2] Y LeCun, L Bottou, Y Bengio, and P Haffner. *Gradient-Based Learning Applied to Document Recognition*, Proceedings of the IEEE, Nov. 1998.

[3] Y LeCun. *Convolutional Neural Networks*, Embedded Vision Summit presentation, May 2014. http://bit.ly/1NAE7Bh

[4] Y Jia, E Shelhamer, J Donahue, S Karayev, J Long, R Girshick. *Caffe: Convolutional architecture for fast feature embedding*, Proceedings of the ACM International Conference on Multimedia, 2014.

[5] A Krizhevsky, I Sutskever, G E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems 25 (NIPS 2012).

[6] R. Collins, A. Lipton, T. Kanade, H. Fujiyoshi, Y. Tsin et al., *A system for video surveillance and monitoring:* VSAM final report, technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, 2000.

[7] Wen-Mei W. Hwu, *GPU Computing Gems (Emerald Edition),* Morgan Kaufmann, © 2011, pp.548-561.

[8] B. Dipert, A. Shoham, P. Desai, V. Eruhimov, S.H. Lin, *OpenVX Enables Portable, Efficient Vision Software,* www.embedded-vision.com, Feb. 2016. http://bit.ly/1Va11mZ

[9] http://docs.nvidia.com/gameworks/index.html#technologies/mobile/opencv_known_issues.htm%3FTocPath%3DTechnologiesa