

Neural Network Benchmarking: See the Forest for the Trees

When evaluating a neural net, consider the entire pipeline to get a realistic picture of performance

So, you've decided to plunge into the world of artificial intelligence (AI) in order to make your systems smarter. A natural, but not entirely obvious, first question to ask is whether or not your newly-AI-enabled systems will operate fast enough to keep up with incoming data. You can find benchmark data to assess performance of the selected neural-net (NN) in isolation, but that will only give you a part of the answer. That approach may leave you disappointed to find that your system runs slower and less efficiently than those benchmarks suggested it would. Why would that be?

The issue is that NN benchmarks do not tell the whole story. It is not uncommon that the neural net is not the rate-limiting component in a data-processing pipeline. If that's true, then the extra cost or energy burned by specifying an unnecessarily faster neural net is wasted. For example, a vision neural net may have a full pipeline that operates at one tenth the speed of the neural net alone. Consideration of the entire pipeline at the onset prevents learning this painful lesson the hard way.

Application Pipelines

Neural nets form one part of a processing pipeline. The job of the neural net may be to classify objects in an image, recognize speech or otherwise generate information that will lead to a decision. The inputs to a network will generally be an image, a video, or a soundstream, and that input will typically originate in a sensor like a camera or a microphone. In most implementations, though, raw sensor output is not usually suitable for direct use by the neural net.

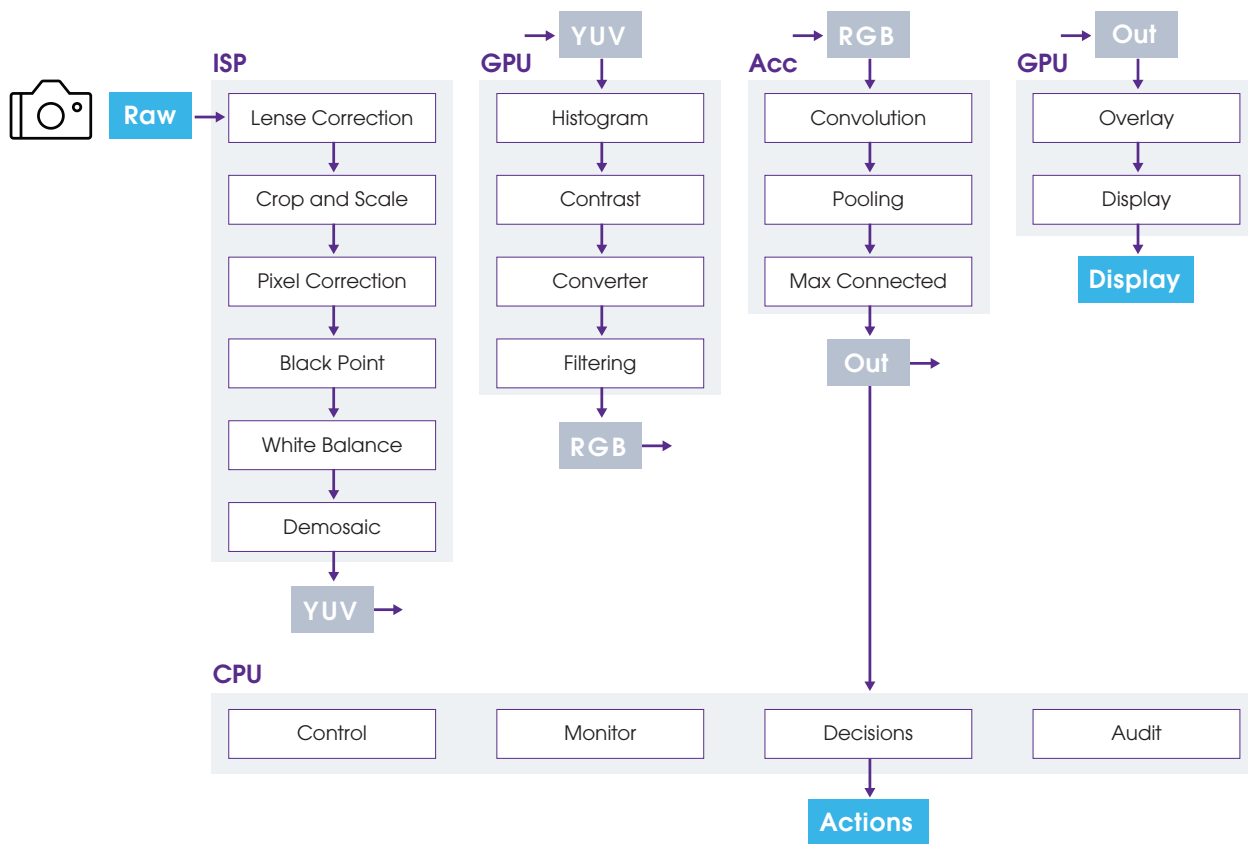
- The original signal may be noisy.
- Aspects of the original signal may need to be enhanced to make neural-net processing more effective.
- Even with a clean signal, additional processing may be needed – like determining the difference between two successive video frames rather than using the frames themselves.

The full application pipeline contains all of these pre-processing functions as well as the neural net. The pre-processing may very well be the slowest part of the pipeline.

Image Processing as an Example

A typical image processing pipeline uses an image sensor processor (ISP) to convert the raw image into a YUV representation, and then uses a GPU to convert that into RGB. The AI accelerator can then operate on the RGB data, delivering a decision to the CPU for action and a final image to the GPU for display.

This pipeline is typically managed by the CPU. Because it uses multiple components, it is unlikely that a CPU will dispatch a task to the start of the pipeline and get the final result when finished. Instead, the CPU will dispatch tasks to each component in turn, arranging the data movements as needed.




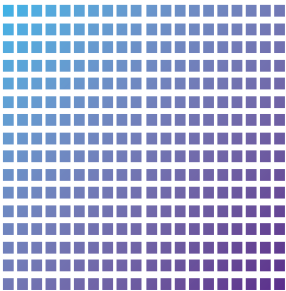
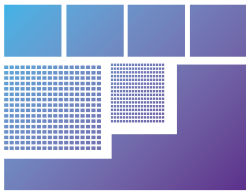


In this pipeline, performance is affected by the following considerations:

- How fast are the individual components? This is where the usual benchmarking occurs, but it's but one consideration.
- How much is performance slowed by the need to move data around for each component?
- How much does CPU management slow the pipeline down?

Platform Choices

The performance of a particular neural net will depend partly on the structure of the network itself. But the fundamental processing capabilities are determined by the choice of the underlying processing unit. The primary computing operation used is the multiply-accumulate, or MAC, function, which can be implemented in different ways with different processing rates. A big tradeoff must be made against flexibility or programmability, which, according to conventional wisdom, comes at the expense of speed.

- CPUs and DSPs are fully software programmable. But because one, or at most a few, units must do the millions of calculations, overall performance will be slow.
- Graphics processors (GPUs) can be programmed to do one task over and over very quickly.
- Field-programmable gate arrays (FPGAs) are fully hardware and software programmable. They can implement multiple functions concurrently. Their programmability makes them very useful for exploring hardware acceleration of new complex functionality, but the result will not be optimized. In addition, each application will need a custom software stack and a lengthy development time due to the hardware-based programmability they support.
- Graph processors and hybrid processors offer varying degrees of programmability, with results that are better optimized. In addition, a single software stack can serve all applications. Blaize® GSP architecture is fully software programmable, offering maximum flexibility and speed of development and deployment.
- Fixed-function devices are not programmable, but they can achieve the highest performance. At this stage of neural net evolution, fixed-function devices make sense only for well-understood applications that will be used in very high volumes, as they cannot be modified easily once deployed.

CPU	GPU	FPGA	Fixed Function	Blaize Hybrid
				
Any task Unoptimized	One task Many times	Multiple tasks Exploration	A fixed task Many times	Multiple tasks Optimized
100% Programmability	Pervasive Programmability (Brute force ecosystem)	Custom Programmability (Each App a new stack)	Limited Programmability	Good Programmability (Single SW stack)

BENCHMARKS, BENCHMARKS, AND BENCHMARKS

With neural nets, as with most electronic subsystems that perform calculations, benchmarks help establish how well a particular NN does at a particular job. But NNs are relatively new, so, while benchmarks exist, many more are in development, and it is a dynamic scene — making it potentially difficult to pick the right benchmarks.

That task is made more difficult by the fact that there are at least three different types of benchmarks that tell different stories.

- There are benchmarks for measuring the processing speed of a specific network. This is critical for streaming applications like video and natural-language processing. It is also useful, from a productivity standpoint, for static tasks like image recognition or handwriting analysis. Units of such a benchmark will establish some level of work accomplished per unit time, like frames per second for video.
- There are benchmarks for measuring the quality of a specific model on a specific network. These benchmarks consist of pools of samples — video clips, photos, samples of handwriting — and, by using a recognized collection, you can establish an accuracy rating that can be compared against alternatives. These ratings specify what percentage of the samples were correctly recognized — or, conversely, the error rate.
- There are benchmarks for measuring how long it takes to train a specific network. Some benchmarks focus on the turn-around time of a single training batch, others measure the total time over all batches used to reach a specified accuracy level.

Most benchmarks are downloadable for free from sites like Github.

So which one of these is best? It's easy to turn to the NN benchmarks alone to make that determination. You will generally want to use the most flexible implementation that can meet the speed, power, and cost requirements of the application. But if that speed is dominated by blocks other than the neural net, then the NN benchmarks are not helpful on their own.

ResNet is an oft-used vision neural network, and performance typically lands in the range of hundreds of images per second when using Graph processors or GPUs. But a pipeline such as the one above can typically operate at around 50 frames per second (fps). That is an order of magnitude slower than the pipeline as a whole can run, making attempts to optimize neural net speed alone an academic exercise.

In addition, programmability can allow various components to be collapsed into a single unit. The GSP, for example, can do far more than just hosting the neural network. Other functions in the pipeline can be included, saving time moving data and requiring less management by the CPU. In the ideal case, the entire pipeline goes into a single device, and the CPU has but one dispatching task, receiving the final result with no further intervention. This can dramatically improve performance.

Look at the Whole Pipeline

As a result, looking only at the neural-net speed will set unrealistic expectations for the final performance of the subsystem. Unfortunately, however, there is no single universal benchmark that can be applied to all applications. Each application has too many variables, and there are far too many applications to be captured by a single benchmark.

So what's the best way to proceed? Questions to ask in addition to the component benchmarks are,

- How will data be moved between components? How will that affect performance?
- How much CPU intervention will there be? How will that affect performance?
- Are there ways to merge functions into a single programmable unit? How much will that increase performance?

The best way to get good answers to these questions is to work with the suppliers providing the hardware platforms under evaluation to gain insight into the implementation and optimization of applications on the platforms. This will take more time than a simple benchmarking review, but it is the only way to pick the best neural-net implementation for your specific processing pipeline.

