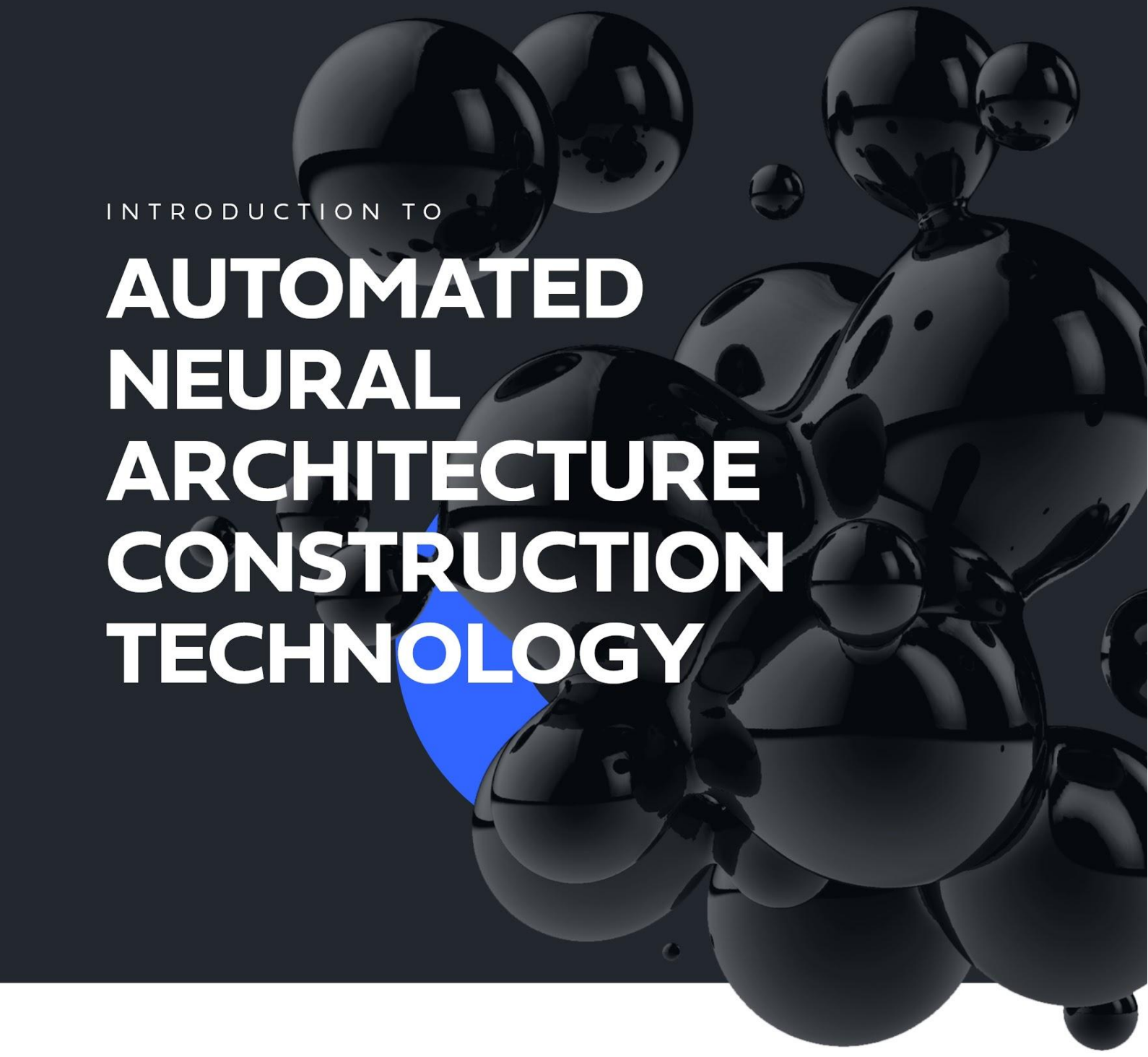
deci

Technical White Paper



INTRODUCTION TO

AUTOMATED NEURAL ARCHITECTURE CONSTRUCTION TECHNOLOGY

Accelerate Deep Neural Network Inference on Any Hardware While Preserving Accuracy

by Prof. Ran El-Yaniv
Chief Scientist and Co-founder at Deci

Technical White Paper

Introduction to Automated Neural Architecture Construction Technology

Accelerate Deep Neural Network Inference on Any Hardware While Preserving Accuracy

Executive Summary

Deep neural networks (DNNs) are transforming entire industries by providing state-of-the-art performance to existing machine learning tasks, along with the means to create new and exciting AI applications. The effective deployment and operation of DNNs in commercial applications depends on high performance, in terms of both accuracy and efficiency. Automated Neural Architecture Construction (AutoNAC), made available by Deci AI, is a seamless procedure that provides a substantial performance boost to existing deep-neural solutions. This acceleration enables dramatic reductions in inference latency and cost-to-serve savings, and can sometimes be accompanied by improvements in accuracy. AutoNAC optimizes deep models to more effectively use their hardware platform, be it CPU, GPU, FPGA, or special purpose ASIC accelerators. The AutoNAC accelerator is a data- and hardware-dependent algorithmic solution that is complementary to other known compression techniques such as pruning and quantization. The AutoNAC pipeline takes as input a user-trained deep neural network, a dataset, and access to an inference platform. It then redesigns the user's neural network to derive an optimized architecture whose latency is substantially lower, without compromising accuracy. For example, we describe below an actual client case where 4.6X speedup on a GPU was achieved, starting from a state-of-the-art architecture. In addition, we overview our 2020 MLPerf submission with Intel where AutoNAC optimized latency for Resnet 50 on a Xeon CPU by a 11.9x factor. These speedups can lead to the immediate enablement of real-time applications, alongside considerable reductions in operating costs for cloud deployments. AutoNAC is also useful for accelerating models over edge-devices to achieve real-time responsiveness and improve energy consumption.

The New Electricity Requires Faster Wiring

Powering Effective Deep Learning for Everything from Autonomous Cars to Smart Stores

Recent years have witnessed a technological revolution that is transforming entire industries and fueling the creation of new ones. The forces driving this intense progress are the deep neural networks and powerful computing that are providing new technology for superior modeling and prediction. For example, deep learning models are enabling unprecedented machine vision capabilities. Unavailable just a few years ago, these capabilities are providing accurate perception functionalities essential for many low-level visual recognition tasks such as object classification, detection and tracking, pixel-level semantic segmentation, depth estimation, pose estimation, and even the recognition of visual relationships in images and videos. Exceptional performance in these tasks has made possible powerful applications such as autonomous driving, video analytics, security control and monitoring, smart home and city, automated shopping (e.g., Amazon Go), medical diagnostics, conversational AI applications, and more. Humans are currently performing many tedious and difficult tasks that often require substantial expertise to be done correctly; many of these same tasks are well on their way to being carried out by machines equipped with deep artificial intelligence (AI). Stanford professor and Coursera co-founder Andrew Ng pointed out that deep learning (DL) is the new electricity: "Just as electricity transformed almost everything 100 years ago, today I actually have a hard time thinking of an industry that I don't think AI will transform in the next several years".[1]

When creating any commercial machine learning model, high prediction quality is a must; this is typically correlated directly with customer satisfaction, sales, and subsequently, profitability

Today, there are thousands of technology ventures that have plans to rely on or are already using this new electricity as a major ingredient in their solutions. The vast majority of these companies are startups at various stages that are investing substantial resources in building deep neural models. Creating and using these models is a complex process that requires considerable expertise and involves the following general phases: model design, training, quality testing, deployment, service and periodic fitting, and revisions. When creating any commercial machine learning model, high prediction quality is a must; this is typically correlated directly with customer satisfaction, sales, and subsequently, profitability.

[1] https://www.gsb.stanford.edu/insights/andrew-ng-why-ai-new-electricity

Deep Learning Relies on Costly Hardware, Training Sets, and Expertise

The deployment of an accurate deep neural network can be costly and time consuming—especially if the task at hand is non-standard, involves very large training sets, or includes strict constraints on the model size (e.g., inference must be performed on a small device with limited memory). Once the model's training is completed successfully, an application is created and put into operation. This is where performance speed and low latency become essential. Suppose, for instance, you need to perform a certain image-processing task on a stream of customer images. Each of the images is fed to the DL model, which generates the required response. The model's computation for each input instance, known as the *inference*, takes time (latency) and consumes energy. Latency and energy consumption are tightly related and both depend heavily on the model's architectural complexity. That said, the impact of high latency can sometimes be more critical than high operating costs. For example, in the case of imaging applications for autonomous cars, if the latency isn't low enough to allow real-time processing, it can disqualify the entire application. Thus, generating an accurate prediction with a sufficiently small latency is critical to successful deployment.

The impact of high latency can sometimes be more critical than high operating costs

Both DL training and inference rely on customized hardware/software frameworks to reach the necessary performance requirements. In today's marketplace, graphical processing units (GPUs) generally serve as the workhorse behind accelerated DL computing, both for training and production. These GPUs can accelerate DL performance by distributing the underlying computation over thousands of small computing cores. A number of other DL hardware accelerators also exist or are currently being manufactured. The most mature among these is Google's tensor processing unit (TPU). The TPU is a family of ASIC chips designed for neural network training and inference, and has been available in the Google Cloud Platform (GCP) for several years. Among the emerging accelerator manufacturers, we can find Intel (with their Goya, Habana, Movidius, and Mobileye chips), AWS, Cerebras, Graphcore, Groq, Gyrfalcon, Horizon Robotics, Mythic, Xilinx, Nvidia, and AMD.

There are several known "compression" approaches aimed at speeding up inference. Among the more prominent compression techniques are *weight pruning* and *quantization*. Using weight pruning, redundant network weights are nullified. With quantization, or binarization, floating-point dynamic ranges of weights and activations (e.g., 32-bit representations) are compressed to a discrete range of a few bits, or one bit in binarization. These methods, which are discussed below, can substantially speed up inference time, but often compromise the accuracy.

The ideal accelerator should be capable of speeding up inference by cutting down latency, without sacrificing accuracy. This article introduces a novel acceleration method, called *Automated Neural Architecture Construction* (AutoNAC). AutoNAC is designed to boost the performance of a given trained model. The acceleration uses a constrained optimization process to construct a new model composed of a collection of smaller related models, whose overall functionality closely approximates the given model. This optimization process leverages information embedded in the original model to reduce the latency, while preserving the accuracy of the original model.

The AutoNAC accelerator is a data-dependent and hardware-dependent algorithmic solution that is complementary to other known techniques, such as graph compilation and quantization. The AutoNAC pipeline takes as input a user-trained deep neural network, a dataset, and access to an inference platform. It then redesigns the user's neural network to derive an optimized architecture whose latency is typically two to ten times better—without compromising accuracy. These speedups can lead to the immediate enablement of real-time applications, alongside considerable reductions in operating costs for cloud deployments.

A Growing Zoo of Deeper Architectures and the Inference Barrier

Accuracy and Performance Tend to be Inversely Related

In 2011, the first utilization of massively parallel GPU acceleration was introduced to train convolutional neural networks (Ciresan et al., 2011). Shortly thereafter, the famous AlexNet (after being trained on a GPU) unveiled the enormous advantage of neural networks over the conventional methods of that time, as demonstrated on the ImageNet classification competition (Krizhevsky et al., 2012). These works were among the precursors to the deep learning revolution. In the subsequent years, several neural architecture families were identified as being most suitable for specific tasks. For example, convolutional neural networks (CNNs) were discovered to excel in image processing tasks, and long short-term memory (LSTM) networks (and later, transformers) were found best for sequential problems including NLP. Within just a few short years, dozens of architecture variants were introduced. For example, in the CNN family, hundreds of handcrafted architectural designs were invented, introducing seminal ideas such as residual links, dense feature connections, filter attention, and group convolutions. This rapid architectural evolution also demonstrated that larger, deeper, and generally over-parameterized models prevail.

For example, consider Figure 1 in which we see top-1 ImageNet accuracy and the total floating point operations per second (FLOPs) for a forward pass of many (already trained) CNN architectures running on a TITAN-Xp GPU. The diagram compares the performance of 44 architecture variants, among which are famous winners of the annual ImageNet competition such as AlexNet, ResNet, and Inception. Interestingly, with the exception of a few architectures such as the NASNets family, most of these architectures were hand-crafted by deep architecture researchers. It is evident that the top performing models tend to consume more operations. While AlexNet is the most efficient in terms of FLOPs, it also achieves the worst accuracy. At the other extreme, we see NASNet-A-Large with top accuracy performance and the worst efficiency for FLOPs. Similar diagrams can be drawn for other image processing tasks, such as semantic segmentation, alongside additional domains such as NLP and other sequential prediction tasks.

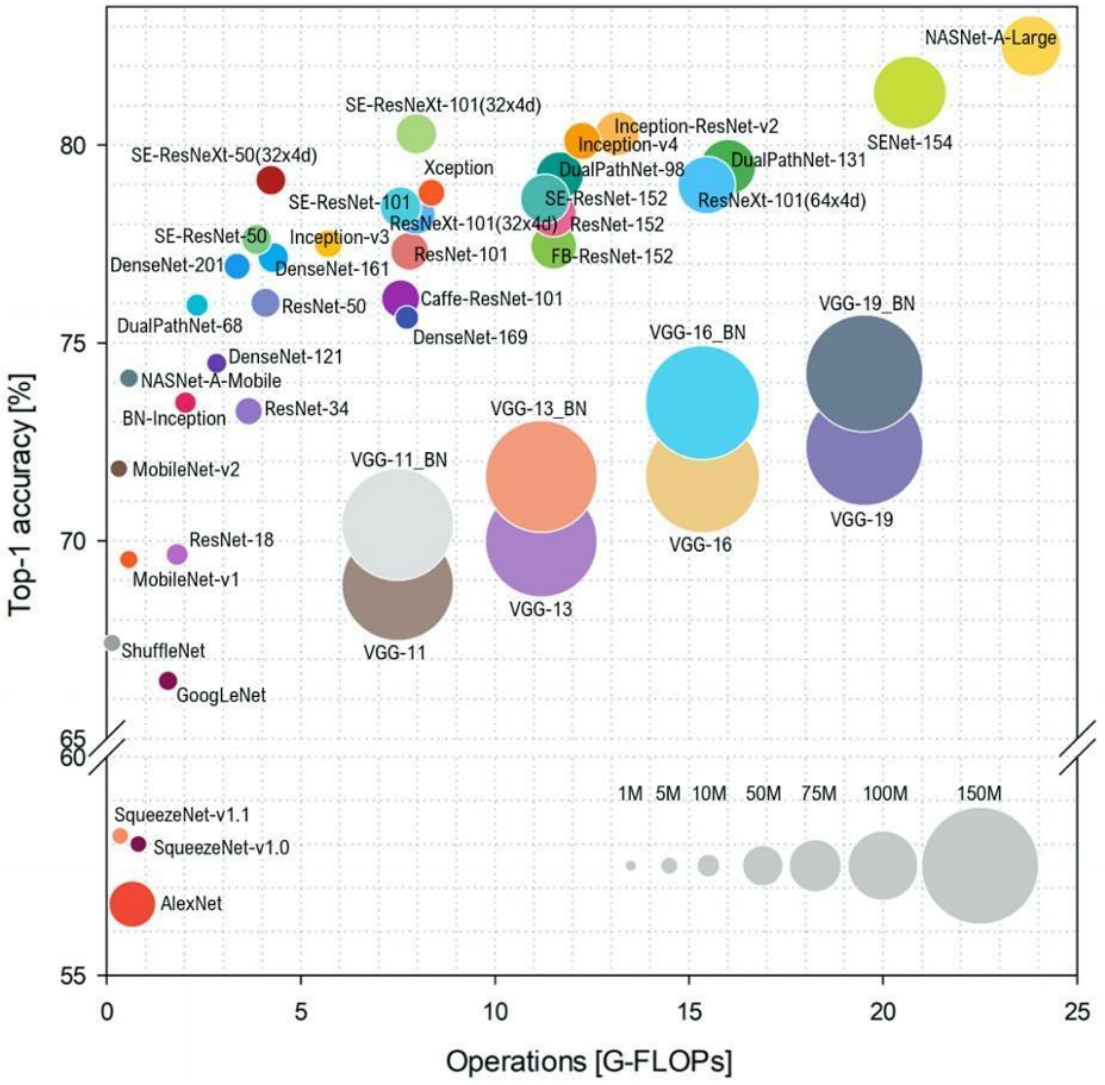


Figure 1. ImageNet top-1 accuracy vs. inference computational complexity (FLOPs) of various architectures. Source: Bianco et al. Benchmark Analysis of Representative Deep Neural Network Architectures, IEEE. Reprinted with permission.

More Complex Challenges Lead to Even Larger Models

When considering accuracy versus inference time, we see that the more accurate architectures tend to incur higher inference latency (Figure 2). Importantly, we can observe that computational complexity (total inference FLOPs) is not an accurate proxy for latency. For example, SENet-154 is more efficient than NASNet-A-Large in terms of FLOPs (Figure 1), but its latency is substantially larger (Figure 2).

This natural evolution of hand-crafted architectures clearly shows that larger and more complex models can be created

When considering accuracy versus inference time, we see that the more accurate architectures tend to incur higher inference latency

to achieve better accuracy. In the meantime, more complex challenges are being posed that require yet larger models. For example, when considering tasks involving both text and images, or when considering finer resolution tasks such as the recognition of visual relationships and semantic image understanding, we can expect that substantially larger models will be required to achieve adequate performance. And, of course, the gigantic OpenAI's GPT-3 model, with its 175 billion parameters stands out as a precursor to even larger models that will be required by the industry in just a few years.

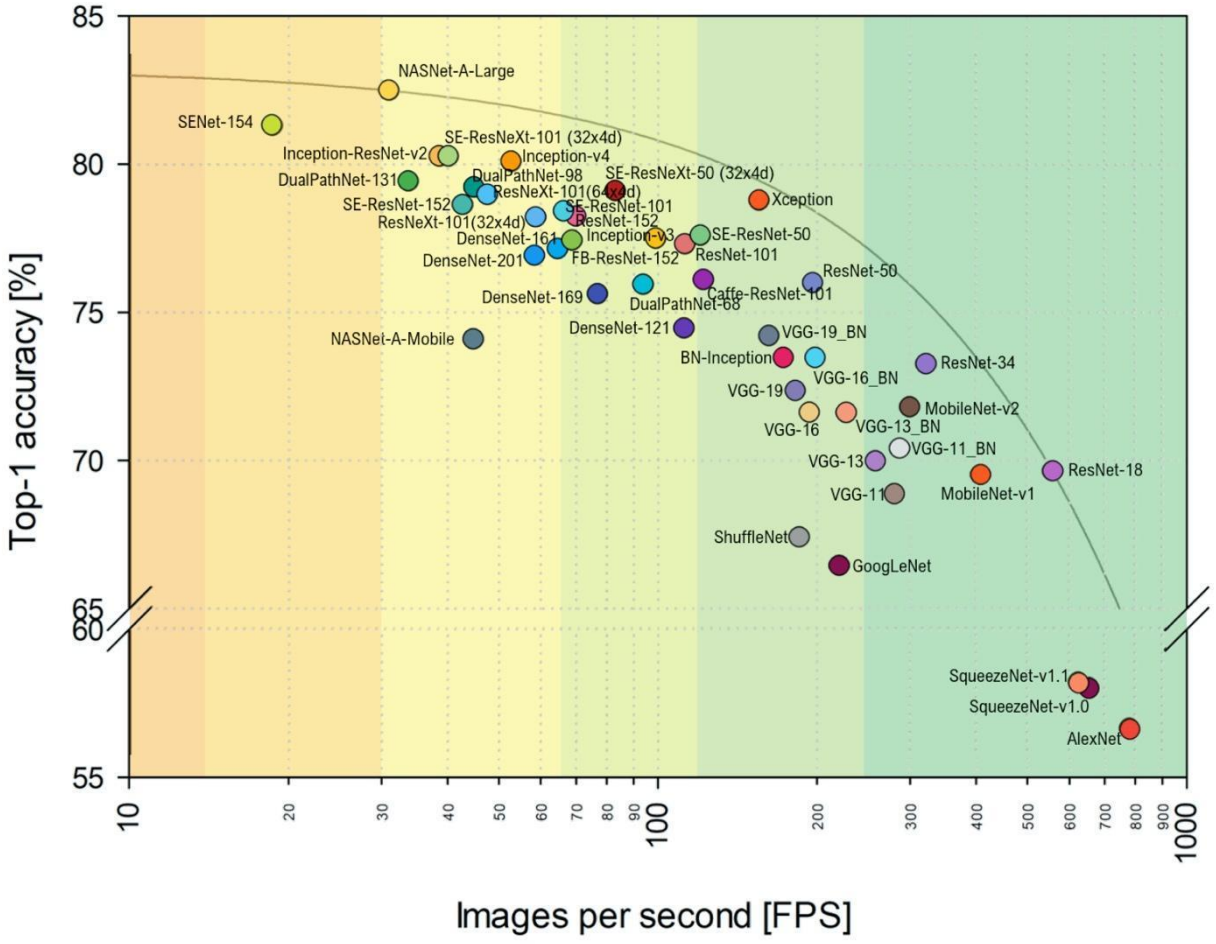


Figure 2. ImageNet Top-1 accuracy vs. throughput (images per second) of various architectures. Source: Bianco et al. Benchmark Analysis of Representative Deep Neural Network Architectures, IEEE. Reprinted with permission.

The excessively large (and accurate) models are critical for academic research, and for extending and showcasing the potential power of deep models. But, when considering model deployment for commercial purposes, these large models, with their expensive inference latency, pose a conceptual or economical barrier that must be overcome.

The Deep Inference Stack

Hardware and Software Working Together

Effectively computing a forward pass on a trained neural network relies on the operation of both hardware and software components that must work in concert. The software components represent abstraction levels of computation and optimization; these abstractions range from primitive hardware operations to high-level neural computation operations such as convolution and pooling, ending with data-dependent model restructuring. The diagram in Figure 3 illustrates this stack of hardware and software components. While we emphasize the role of the top level containing model optimization and adaptation methods, to achieve top inference performance it is essential to harmonize between all levels, and we now review the components of this "inference stack" , starting from the bottom layer, to provide the context in which these methods coexist.

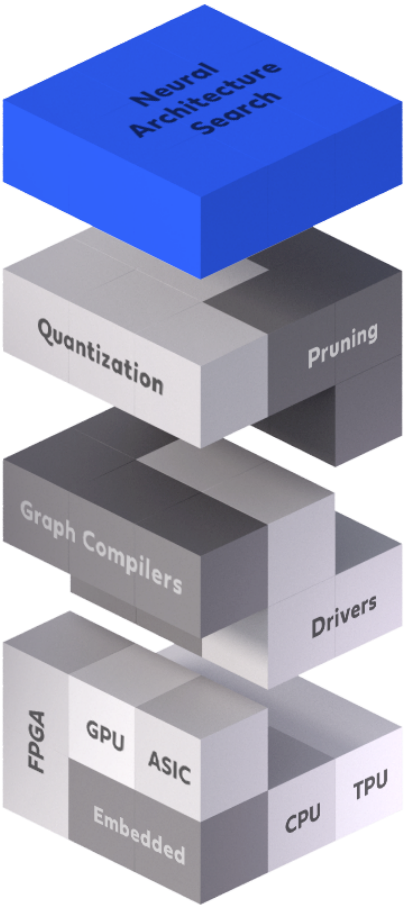


Figure 3. The deep learning inference stack.

Bottom of the Stack: Hardware Devices and Low Level Libraries

At the base of the pyramid, we have hardware devices performing the computation itself. The commercial hardware devices currently available include several types of CPUs, GPUs, FPGAs, and a few specialized ASIC accelerators such as Google's TPU. At the next level we find three software components: low-level *libraries* that can operate the hardware devices, computation graph compilers, and deep learning frameworks. The low-level libraries (e.g., cuDNN and MKL-DNN) are typically optimized for specific hardware devices; they provide highly-tuned implementations for standard neural layers such as convolution, pooling, and activation. On top of these libraries there are *graph compilers*, such as TVM, Tensor-RT, XLA, and Glow. The purpose of graph compilers is to optimize the processing of a forward, or backward pass over the computation graph. The compilers perform optimizations at several levels.

Fusion is a common higher-level manipulation to overcome the naive and extremely inefficient approach that iterates over the graph nodes and executes them one by one. In *vertical* fusion, several consecutive neural layers are collapsed to a single node. For example, convolution, biasing, and activation nodes can be collapsed to a single node, which uses one library kernel to implement the three operations efficiently. In *horizontal* fusion, multiple nodes that have identical operations are collapsed to a single node, which feeds their (identical) output to some other nodes.

Lower-level compiler optimizations strive to prevent cache misses, which can decelerate the computation by orders of magnitude. This objective can be achieved by performing explicit scheduling through the memory hierarchy, rather than relying on the oblivious functionality of the cache hierarchy. To perform well, compilers must also account for the hyperparameters in the neural layers, such as padding, strides, and tensor dimensions, and adjust the computation for specific hardware devices.

On top of the graph compilers, there are the *frontends* or deep learning *frameworks* such as Tensorflow, Pytorch, MXNet, and Caffe. These frameworks serve deep learning engineers, offering them high-level programming libraries within modern programming languages like Python. The libraries provide an abstract interface that allows engineers to easily construct, train, and validate deep models.

The Next Level: Algorithmic Inference Removes Redundancies (But Also Accuracy)

The inference components described so far essentially perform data-independent optimizations, except for the frontends, which serve as programming interfaces. The next levels of the inference stack include acceleration methods for algorithmic inference. In contrast to methods in the lower levels, these perform data-dependent optimizations. The first of the upper levels contains known compression techniques, which typically rely on

pruning and quantization (described in the next section). Because they are data-dependent, these compression optimizers can remove network redundancies by further adapting neural models to specific tasks. However, as discussed below, compression-based methods tend to reduce the accuracy of large models. At the uppermost level, we have inference optimizers, such as the Deci AI AutoNAC (described below). The optimizers at this level completely restructure given neural models by relying on complementary data-dependent statistical principles that don't compromise accuracy.

While the structure and types of components in the inference stack are likely to endure, the specific components themselves are rapidly evolving

While the structure and types of components in the inference stack are likely to endure, the specific components themselves are rapidly evolving. For example, a number of new ASIC inference accelerators have already been released (e.g., Intel's Goya) but aren't yet widely commercially available for consumers. It is likely that in just a few short years we'll be seeing a wide variety of ASIC inference accelerators available for both data centers and edge devices. Moreover, sub-levels and even complete levels of the inference software stack will likely be merged. For example, the Tensor RT graph compiler already contains full post-training quantization.

Aiming for Accuracy Increases Deep Net Bloat

The rapid development of deep neural models has been generating a growing industrial appetite to solve even more complex prediction tasks with higher accuracy. At the same time, we are seeing continuous growth in the size and computational complexity of deep net architectures (including FLOPs and latency). It is often much easier for deep learning engineers to achieve better accuracy with larger overparameterized neural models. These sizable models make it simpler to automatically create better features and representations. This deep learning reality stands in sharp contrast to the conventional wisdom in classical machine learning, where larger models are always worse beyond the sweet spot of the generalization-complexity tradeoff curve.

Excessively Large Neural Models Cost Too Much

In reality, however, it is often prohibitively expensive and even impossible to deploy excessively large neural models. They consume enormous amounts of energy and incur excessive latency at inference time. We need methods that can reduce the size of large neural networks, thus shrinking their latency and energy parameters, while preserving high-accuracy performance.

Several known techniques are aimed at reducing network size or creating smaller networks from scratch. *Pruning* and *quantization* are the main network reduction or "compression" techniques. *Neural architecture search (NAS)* is the primary technique for creating improved architectures. The following sections briefly summarize these methods.

Weight Pruning: Great for Simple Networks, Challenged for Deeper Ones

The idea behind weight pruning is quite straightforward. Given a model, one prunes the weights that are relatively less important. A simple heuristic to achieve this is to zero the small magnitude weights, retrain the pruned model (fine-tuning), and repeat these two steps over several iterations. This technique can compress the weights of certain networks such as AlexNet by an order of magnitude, without forfeiting accuracy (Han et al., 2015). Other more sophisticated techniques overcome the need for fine-tuning, such as the Alternating Direction Method of Multipliers (Zhang et al. ,2018) and end-to-end pruning, which uses a specialized global sparse momentum stochastic gradient descent (SGD) optimization technique (Ding et al., 2019).

Weight pruning techniques can dramatically compress shallow and simple networks such as LeNet-5 and AlexNet, without forfeiting accuracy. But, when applied to deeper, more contemporary architectures, they have trouble preserving accuracy. More importantly, weight pruning is equivalent to removing connections between neurons; this makes the weight tensors sparser in an irregular manner, but not necessarily amenable to faster computation. For example (and contrary to various marketing claims), when considering GPUs and CPUs, it is nearly impossible with present techniques to speedup matrix multiplication (which is the main computational routine required to perform inference) beyond a certain negligible factor, based on sporadic weight sparsification. In short, little or no latency speedup can be expected from weight pruning without the support of dedicated hardware that can exploit irregular sparse tensor manipulations. At present, such hardware devices are not commercially available.

Filter Pruning: Remove Unimportant Filters

This inability to exploit irregular tensor sparsity motivated the idea of filter pruning (or network slimming) in the context of convolutional networks (CNNs): instead of removing sporadic connections from the network, remove unimportant filters from convolutional

layers to slim down the network with minimal performance drop. In this sense, filter pruning is quite similar to the classical problem of feature subset selection, whose combinatorial nature circumvents optimal solutions. The main difference is that CNN filters are trained, as opposed to feature vectors, which are typically fixed; this fact can be exploited to collapse them. Simple filter pruning methods rank the filter importance using some criterion, prune the less essential ones, and then fine-tune. The filtering criteria can include channel contribution variance, information derived from the next layer, and sparsity inducing Lasso-regression to minimize layer reconstruction error, to name just a few. As opposed to importance-based channel pruning, more advanced techniques use a specialized stochastic gradient descent (SGD) that encourages channel merging, and are based on reinforcement learning or meta-learning (Liu et al., 2019).

Quantization: More Efficient Operations and Memory Access

In network quantization (Hubara et al., 2017), the objective is to substitute floating point weights and/or activations with low precision compact representations. Quantization can lead to substantially smaller networks and allows for more efficient computation, both in terms of tensor arithmetic operations and memory access. The extreme case of quantization is known as binarization (Courbariaux et al. 2016, Rastegari wt al. 2016), where both weights and activations are constrained to be binary numbers. The challenge in constructing a quantized network is that the low bitwidth weights and activations result in information loss; this loss distorts the network representation and circumvents the ordinary differentiation required for training.

Conventional quantization methods use the same number of bits for all layers. However, different layers can have different optimal redundancy profiles, which can potentially be exploited (Wang et al., 2019). Several contemporary hardware inference accelerators already allow for mixed precision quantization for inference, whereby different bit widths can be defined and applied to each layer individually. For example, the Apple A12 Bionic chip supports some mixed precision, and Nvidia's Ampere technology for GPUs supports 1,2,4,8, and 16-bit bit-width operations, which can be applied per layer. Most future acceleration hardware units are expected to enable such mixed precision.

Neural Architecture Search (NAS): Outstanding Results, Challenging to Implement

NAS is a technique to automate neural architecture engineering and is currently one of the hottest topics in deep learning research. The general idea is straightforward: from a space of allowable architectures, select the best architecture. The selection algorithm relies on a search strategy, which in turn depends on an objective evaluation scheme. Efficiently implementing NAS is extremely challenging, but we're seeing some outstanding results even with brute force implementations. For example, NAS has been used to engineer some of the best performing low-complexity (FLOPS) architectures to-date, such as EfficientNet (Tan and Le 2019). That said, these impressive achievements were achieved by employing huge computational resources in the order of tens of thousands of GPU hours.

> ***The general idea of NAS is straightforward: from a space of allowable architectures, select an appropriate architecture***

The NAS search space determines what type of architecture can be discovered by the NAS algorithm. This space is defined by specifying the overall structure (skeleton) of the network, the type of units or blocks that define the layers, as well as the allowable connectivity between layers. Search strategies that have been considered include neuro-evolutionary methods (Elsken et al., 2019), Bayesian approaches (Mendoza et al., 2016), and reinforcement learning (Zoph and Le, 2016). Interestingly, some recent evidence suggests that evolutionary techniques perform just as well as reinforcement learning (Real et al., 2019). Moreover, the evolutionary methods tend to have better "anytime performance" and settle on smaller models. While earlier NAS techniques were based on discrete search spaces, a continuous formulation of the architecture search space has introduced differentiable search methods, which opened the way for gradient-based optimization (Liu et al., 2019).

During a NAS search, many architectures must be evaluated for their validation set performance. Training each architecture from scratch on the entire training set typically leads to computational demands in the order of thousands of GPU days. Many ideas have been considered to reduce evaluation time, such as: low-fidelity performance estimation (early exit after a few epochs, training over a subset of the data, downscaled models or data), weight inheritance, weight sharing, learning-curve extrapolation, and network morphism (Jin et al., 2019). One of the popular approaches is single-shot training, where the search space consists of sub-architectures belonging to a single super-architecture

whose trained weights are shared among all sub-models (Xie et al., 2019). Different single-shot NAS methods differ in how the single-shot model is trained. ENAS (Pham et al., 2018) is a prominent example of a single-shot algorithm that achieves a 1000X speedup of the search relative to previous techniques. One of the prominent results in this venue is the “once for all” technique (Cai et al., 2019).

In short, by relying on huge computational resources, NAS techniques have already outperformed manually designed architectures on standard tasks such as image classification. However, for the most part, faster NAS techniques have only shown promising results over the smaller benchmark datasets (e.g., Cifar-10, MNIST, or reduced versions of ImageNet). Presently, improving NAS efficiency is a prerequisite for making it commercially available to users who don’t have Google-scale computational resources.

Introducing AutoNAC - Automated Neural Architecture Construction

Boost Deep Learning Model Performance Without Sacrificing Accuracy

AutoNAC is a data- and hardware-dependent architecture optimization algorithm. Applying AutoNAC is a seamless process in which the user provides a trained model, training and test datasets, and access to the hardware platform over which the model should be deployed. AutoNAC then automatically computes a new low-latency (high throughput, or low power consumption) model that preserves the accuracy of the original model.

The AutoNAC objective is to derive an optimal architecture a^* that solves the following constrained optimization problem:

$$a^* = argmin_{a \in A} Lat_H(a, D)$$

$$\text{subject to } Acc(a, D) \geq Acc(a_0, D),$$

where a^* is an optimal solution, a_0 is the original baseline model provided by the user, A is the architecture search space (constructed automatically), D is the user dataset, $Lat_H(a, D)$ is the mean inference latency of model a over dataset D computed over hardware platform H, and $Acc(a, D)$ is the accuracy of model a over dataset D (or any required performance measure). The AutoNAC optimization process is depicted in Figure 5. Unlike standard NAS techniques, AutoNAC starts the search procedure from a relatively good initial point by heavily utilizing the baseline model including several of its already trained layers. AutoNAC is applied on a discrete space of architectures that is set in accordance with the allowable neural operations that are supported in the target hardware. The (proprietary) search algorithm itself relies on prediction models to determine effective optimization steps. This algorithm results in very fast convergence times that are often lower than known NAS techniques by orders of magnitude. In addition, one of the main advantages of AutoNAC is its ability to consider all levels of the inference stack and optimize the baseline architecture while preserving accuracy and taking into consideration the target hardware, the (hardware-dependent) compilation, and quantization.

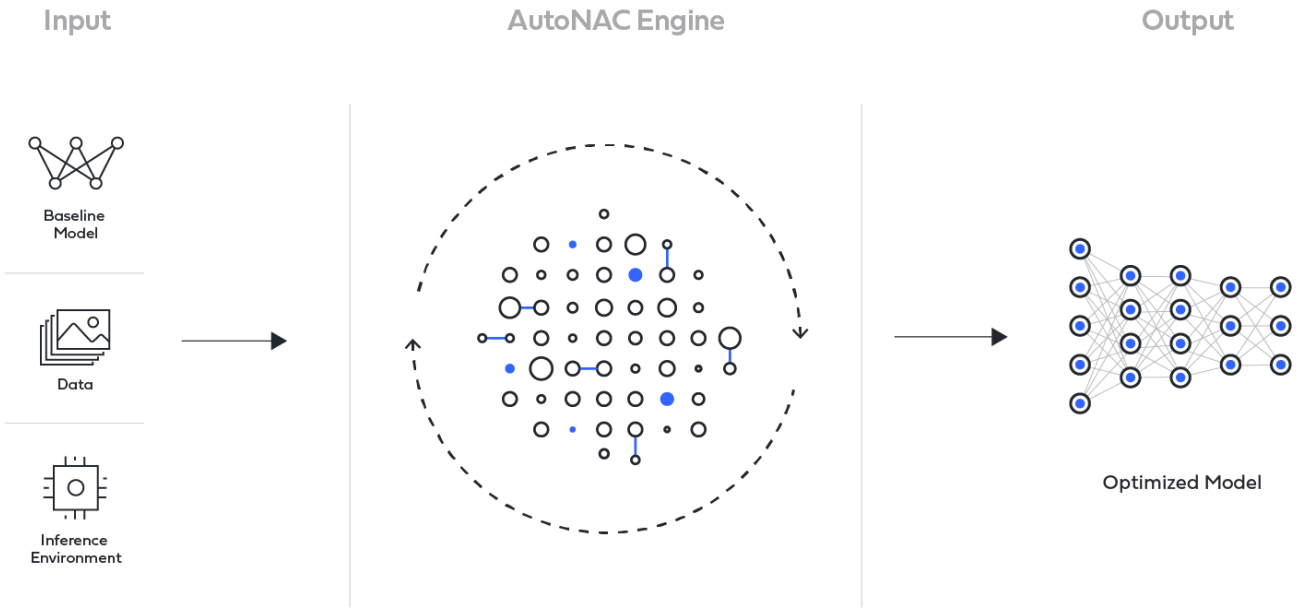


Figure 5. The AutoNAC engine redesigns DL models to squeeze the most out of the hardware and leverage the hidden structures of the data.

AutoNAC: Client Case Study

Making Real-Time Deep Learning on GPU in the Cloud Possible

AutoNAC is already being used to benefit clients in commercial settings. In this section, we highlight the case of a well-established video technology provider. One of their products includes streaming video footage that is processed through a deep learning-based pipeline with near real-time latency. The inference of this pipeline is executed on large commercial clouds on the Nvidia K-80 GPU machine, which was selected as best value for money by the client.

The client was experiencing unsatisfactory latency/throughput in one of the deep learning models in their pipeline, which was developed using the TensorFlow framework. The model is a feature extraction CNN model whose architecture is proprietary.

AutoNAC is already being used to benefit clients in commercial settings

The client's highly-experienced data science team tried several methods to solve this problem, but quickly concluded that none would achieve the combination of lower latency and high accuracy at their cost-to-serve target.

The alternatives they considered, along with the results, include:

- **Reducing input video resolution** - reduced the model's accuracy because less information was available.

- **Pruning and quantization** - compromised the model's accuracy.
- **Increase hardware spec** - didn't meet their cost-to-serve target due to the high price of more powerful hardware.
- **Moving from cloud inference to on-prem inference** - didn't solve the inherent computation latency/throughput, which was still unacceptable.
- **Changing the model to a state-of-the-art architecture** - although very time consuming, switching to a state-of-the-art (SoTA) architecture did improve the model's accuracy but still fell short of the latency/throughput target because the architecture was not optimized for their hardware and their specific problem.The client used Deci's Automated Neural Architecture Construction (AutoNAC) engine to get the job done. Because AutoNAC is complementary to other optimization efforts, the data science team selected the SoTA-based model, which was the best model they reached, as a candidate for AutoNAC.

The entire optimization process was done automatically, with no human intervention. The client could choose to carry out the process on-prem or in their virtual private cloud, thus avoiding any data security or privacy issues. At the end of the optimization process, the new AutoNAC model was plugged seamlessly into their CI/CD process.

After using Deci's AutoNAC, the client was able to use the model for real-time inference in production. The outcome was a new AutoNAC model that had improved the latency/throughput by 4.6 times (Figure 6), without sacrificing the original model's accuracy. Moreover, the newer model's jump in efficiency unlocked cost-saving opportunities such as switching to a cheaper GPU and even a simple CPU machine.

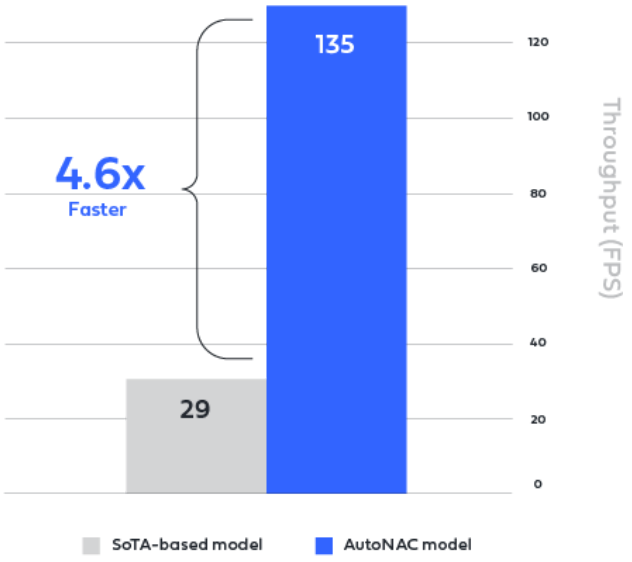


Figure 6. Client case where 4.6X speedup was achieved, while preserving model's accuracy, starting from a state-of-the-art architecture.

Deci+Intel's MLPerf submission for CPUs - 12x Speedup

Can Deep Learning inference on CPUs be competitive with GPUs?

Established by AI leaders from academia, industry, and research labs, MLPerf is a non-profit organization that provides fair and standardized benchmarks for measuring the training and inference performance of machine learning hardware and software.

Relying on AutoNAC, Deci submitted in collaboration with Intel several deep learning models to the Open division of the MLPerf v0.7 inference benchmark (Sept. 2020). This submission, which benefited from both Intel's OpenVINO compiler and Deci's AutoNAC, aimed to cut the inference latency and boost throughput of the ResNet-50 architecture on three CPU types.

In accordance with MLPerf rules, the goal was to maximally reduce the latency, or increase the throughput, while staying within 1% of ResNet-50. Table 1 and 2 display the results for latency and throughput scenarios on the three CPUs tested. As shown, the optimized models (which preserve accuracy within 1%) improve latency between 5.16x and 11.8x when compared to vanilla ResNet-50.

	Latency (ms)			
Hardware	**ResNet-50 OpenVINO 32-bit**	**ResNet-50 OpenVINO 8-bit**	**Deci**	**Deci's Boost**
1.4GHz 8th-generation Intel quad core i5 MacBook Pro 2019	83	83	7	**11.8x**
1 Intel Cascade Lake Core	76	21	6.45	**3.3 - 11.8x**
8 Intel Cascade Lake Cores	11	5.5	2.13	**2.6 - 5.16x**

Table 1. Results for latency scenario. Three hardware types were tested. ResNet-50 vanilla is ResNet-50 compiled with OpenVINO to 32-bit. The "ResNet-50 OpenVINO 8-bit" columns show the same hardware with compilation to 8-bit, and the columns labeled "Deci" show the AutoNAC-optimized mode with 8-bit compilation.

	Throughput (FPS)			
Hardware	**ResNet-50 OpenVINO 32-bit**	**ResNet-50 OpenVINO 8-bit**	**Deci**	**Deci's Boost**
1.4GHz 8th-generation Intel quad core i5 MacBook Pro 2019	30	30	207	**6.9x**
1 Intel Cascade Lake Core	14	50	154	**3.1 - 11x**
8 Intel Cascade Lake Cores	110	410	1092	**2.7 - 9.9x**

Table 2. Results for throughput scenario. Three hardware types were tested. ResNet-50 vanilla is ResNet-50 compiled with OpenVINO to 32-bit. The "ResNet-50 OpenVINO 8-bit" columns show the same hardware with compilation to 8-bit, and the columns labeled "Deci" show the AutoNAC-optimized mode with 8-bit compilation.

In addition, MLPerf results allow us to compare the Deci performance to other submissions. To compare apples to apples, we normalized each submission with the number of cores it used. Using the throughput results of Tables 2 Figure 7 depicts Deci's results compared to several other submissions. This draws a very optimistic picture, with our throughput per core 3x higher than other submissions.

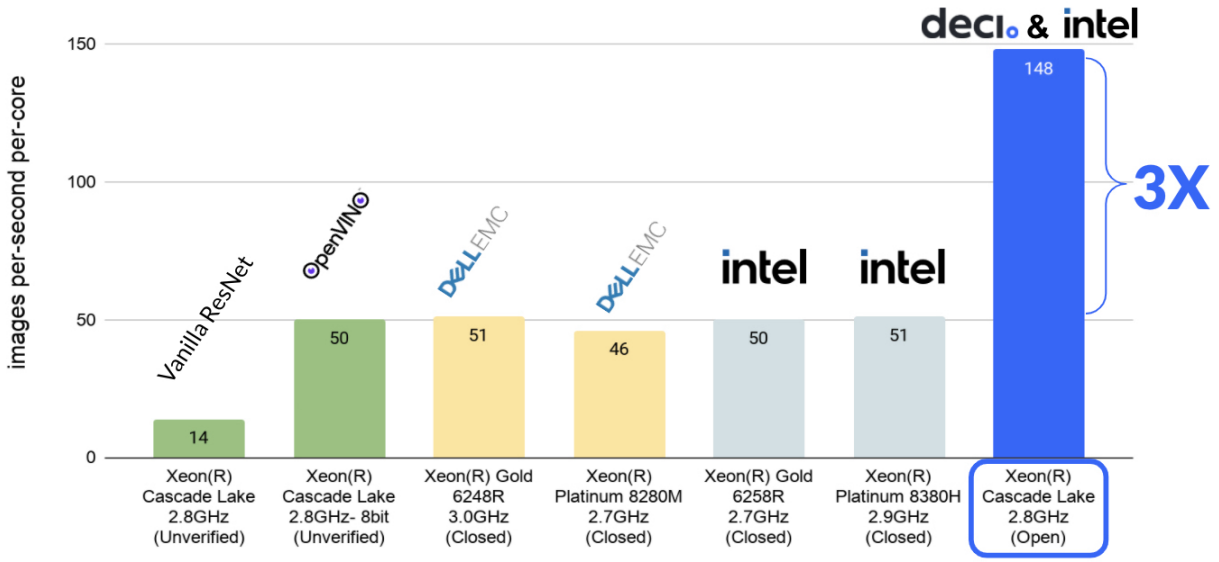


Figure 7. Client case where 4.6X speedup was achieved, while preserving model's accuracy, starting from a state-of-the-art architecture.

Conclusion: Overcoming the Latency/ Throughput Challenge in Deep Learning

Several challenges must be overcome to successfully deploy deep learning models and operate them at scale. Beyond the obvious accuracy performance requirements, these models must be efficient enough to reduce operating costs or allow for real-time latency/throughput. Despite the continual improvement of inference compute power, the efficiency barriers are expected to increase as models become more accurate and more sophisticated. These new models will be required to tackle the burgeoning needs for solving more demanding applications. Moreover, the increasing need for complex architectures that can handle multi-modal data signals will inevitably yield larger and computationally-heavier neural models. The relative advantage of deep learning operations will thus remain critically dependent on the efficiency of their solutions. To remain competitive, businesses will ultimately need to rely on deep networks that can employ unique and expensive deep modeling expertise. The AutoNAC optimizer, offered by Deci AI, is an effective and affordable solution that can enable real-time application on any hardware and dramatically improve the infrastructure costs of deep learning commercial operations. The AutoNAC engine relies on solid machine learning principles and in-depth knowledge of deep nets. Under the hood, AutoNAC contains the distilled knowledge and technical wisdom of many research years, which seamlessly allows it to optimize most neural models. AutoNAC is a power multiplier and will improve a model's efficiency even after it has been optimized using conventional methods such as pruning and quantization techniques—even if the model is already executing on the fastest inference hardware platforms.

References

Kenigsfield, Gal S., and Ran El-Yaniv. "TranstextNet: Transducing Text for Recognizing Unseen Visual Relationships." Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2021.

Ciresan, Dan Claudiu, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. "Flexible, high performance convolutional neural networks for image classification." In Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in Neural Information Processing Systems, pp. 1097-1105. 2012.

Bianco, S., Cadene, R., Celona, L., & Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. IEEE Access, 6, 64270-64277.

Nakkiran, Preetum, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. "Deep double descent: Where bigger models and more data hurt." arXiv preprint arXiv:1912.02292 (2019)

Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal. "Reconciling modern machine learning and the bias-variance trade-off." arXiv preprint arXiv:1812.11118 (2018).

Courbariaux, Matthieu, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Hubara, Itay, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Quantized neural networks: Training neural networks with low precision weights and activations." The Journal of Machine Learning Research 18, no. 1 (2017): 6869-6898.

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016, October). Xnor-net: Imagenet classification using binary convolutional neural networks. In European conference on computer vision (pp. 525-542). Springer, Cham.

Zoph, Barret, and Quoc V. Le. "Neural architecture search with reinforcement learning." arXiv preprint arXiv:1611.01578 (2016).

Real, Esteban, Alok Aggarwal, Yanping Huang, and Quoc V. Le. "Aging evolution for image classifier architecture search." In AAAI Conference on Artificial Intelligence. 2019.

Liu, Zhuang, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. "Rethinking the value of network pruning." International Conference on Learning Representations, (2019).

Jin, Haifeng, Qingquan Song, and Xia Hu. "Efficient neural architecture search with network morphism." arXiv preprint arXiv:1806.10282, 2018.

Xie, Sirui, Hehui Zheng, Chunxiao Liu, and Liang Lin. "SNAS: stochastic neural architecture search." International Conference on Learning Representations, 2019.

Pham, Hieu, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. "Efficient neural architecture search via parameter sharing." International Conference on Machine Learning, 2018.

Tan, Mingxing, and Quoc V. Le. "EfficientNet: Rethinking model scaling for convolutional neural networks." arXiv preprint arXiv:1905.11946, 2019.

Mendoza, Hector, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. "Towards automatically-tuned neural networks." In Workshop on Automatic Machine Learning, pp. 58-65, 2016.

Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter. "Efficient multi-objective neural architecture search via lamarckian evolution." arXiv preprint arXiv:1804.09081, 2018.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." International Conference on Learning Representations, 2019.
Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2019). Once-for-all: Train one network and specialize it for efficient deployment. arXiv preprint arXiv:1908.09791.

About The Author

Ran El-Yaniv is a co-founder and the chief scientist at Deci AI, and a full professor of computer science at the Technion. Prior to founding Deci, Ran was a visiting Staff Research Scientist at Google. Ran's research activities span numerous topics in machine learning, deep learning, and AI. He received a PhD in computer science from the University of Toronto, and completed his post-doctoral studies at the MIT Computer Science Laboratory and the Center for Rationality at the Hebrew University. Ran has been serving as area chair in premier machine learning and AI conferences including NeurIPS, ICML, and IJCAI. He is a co-author of the book Online Computation and Competitive Analysis (Cambridge University Press), an associate editor at the Journal of Artificial Intelligence Research, and a member of the editorial board at the Journal of Machine Learning Research. Ran received the 2016 Yanai Prize for Excellence in Academic Education and the Best Research paper in ISMIR 2020.

Yonatan Geifman, PhD, and Jonathan Elial also contributed to this white paper.

About Deci

Deci, which means "tenth" (decimus in Latin), is ushering in a new AI paradigm by using AI to build and operate AI models. Deci's deep learning platform enables data scientists to transform their AI models into production-grade solutions on any hardware, crafting the next generation of AI for enterprises across the board. Deci's proprietary AutoNAC (Automated Neural Architecture Construction) technology autonomously redesigns an enterprise's deep learning models to squeeze the maximum utilization out of its hardware. Founded in 2019 and based in Tel Aviv, Deci's team of deep learning experts are dedicated to eliminating production-related bottlenecks across the AI lifecycle to allow developers and engineers the time to do what they do best - create innovative AI solutions for our world's complex problems.

For more information visit us at www.deci.ai