SOAR

STATE-OF-THE-ART REPORT (SOAR)
SEPTEMBER 2022

CSIAC-BCO-2022-233

COMMODITY DEEP LEARNING TECHNOLOGIES SUPPORTING AUTONOMY ON SMALL, INEXPENSIVE PLATFORMS

By Christiaan Gribble

Contract Number: FA8075-14-D-0001
Published By: CSIAC

This Page Intentionally Left Blank

SOAR

STATE-OF-THE-ART REPORT (SOAR)
SEPTEMBER 2022

COMMODITY DEEP LEARNING TECHNOLOGIES SUPPORTING AUTONOMY ON SMALL, INEXPENSIVE PLATFORMS

CHRISTIAAN GRIBBLE

ABOUT CSIAC

The Cybersecurity & Information Systems Information Analysis Center (CSIAC) is a U.S. Department of Defense (DoD) IAC sponsored by the Defense Technical Information Center (DTIC). CSIAC is operated by SURVICE Engineering Company under contract FA8075-21-D-0001 and is one of the three next-generation IACs transforming the DoD IAC program: CSIAC, Defense Systems Information Analysis Center (DSIAC), and Homeland Defense & Security Information Analysis Center (HDIAC).

CSIAC serves as the U.S. national clearinghouse for worldwide scientific and technical information in four technical focus areas: cybersecurity; knowledge management and information sharing; modeling and simulation; and software data and analysis. As such, CSIAC collects, analyzes, synthesizes, and disseminates related technical information and data for each of these focus areas. These efforts facilitate a collaboration between scientists and engineers in the cybersecurity and information systems community while promoting improved productivity by fully leveraging this same community's respective knowledge base. CSIAC also uses information obtained to generate scientific and technical products, including databases, technology assessments, training materials, and various technical reports.

State-of-the-art reports (SOARs)—one of CSIAC's information products—provide in-depth analysis of current technologies, evaluate and synthesize the latest technical information available, and provide a comprehensive assessment of technologies related to CSIAC's technical focus areas. Specific topic areas are established from collaboration with the greater cybersecurity and information systems community and vetted with DTIC to ensure the value-added contributions to Warfighter needs.

CSIAC's mailing address:

CSIAC
4695 Millennium Drive
Belcamp, MD 21017-1505
Telephone: (443) 360-4600

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE September 2022	**2. REPORT TYPE** State-of-the-Art Report	**3. DATES COVERED**
4. TITLE AND SUBTITLE Commodity Deep Learning Technologies Supporting Autonomy on Small, Inexpensive Platforms		**5a. CONTRACT NUMBER** FA8075-14-D-0001 **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Christiaan Gribble		**5d. PROJECT NUMBER** **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADRESS(ES) Cybersecurity & Information Systems Information Analysis Center (CSIAC) SURVICE Engineering Company 4695 Millennium Drive Belcamp, MD 21017-1505		**8. PERFORMING ORGANIZATION REPORT NUMBER** CSIAC-BCO-2022-233
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Technical Information Center (DTIC) 8725 John J. Kingman Road Fort Belvoir, VA 22060		**10. SPONSOR/MONITOR'S ACRONYM(S)** DTIC **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION/ AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report reviews state-of-the-art artificial intelligence/machine learning (AI/ML) hardware and software technologies supporting autonomy on small, inexpensive platforms. It focuses on commodity hardware components and widely available software ecosystems for deep learning, the subset of AI/ML that uses multilayered neural networks to deliver best-in-class performance and accuracy for the low-level tasks that drive higher-level applications of autonomy.

15. SUBJECT TERMS
artificial intelligence, machine learning, deep learning, autonomy

16. SECURITY CLASSIFICATION OF: U			**17. LIMITATION OF ABSTRACT** UU	**18. NUMBER OF PAGES** 42	**19a. NAME OF RESPONSIBLE PERSON** Vincent "Ted" Welsh
a. REPORT UNCLASSIFIED	**b. ABSTRACT** UNCLASSIFIED	**c. THIS PAGE** UNCLASSIFIED			**19b. TELEPHONE NUMBER** *(include area code)* 443-360-4600

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

ON THE COVER:
(Source: 123rf.com)

THE AUTHOR

CHRISTIAAN GRIBBLE

Christiaan Gribble is currently a principal member of technical staff, silicon design engineer at Advanced Micro Devices, Inc. He was previously the director of high-performance computing (HPC) and a principal research scientist within the Applied Technology Operation (ATO) at SURVICE Engineering Company, where he led ATO's HPC-related research and development initiatives and provided technical oversight for the embedded systems and software engineering teams within ATO. His research explores the synthesis of interactive visualization and HPC. He was also associate professor in the Department of Computer Science at Grove City College. Mr. Gribble holds a Ph.D. in computer science from the University of Utah.

ABSTRACT

This report reviews state-of-the-art artificial intelligence/machine learning (AI/ML) hardware and software technologies supporting autonomy on small, inexpensive platforms. It focuses on commodity hardware components and widely available software ecosystems for deep learning, the subset of AI/ML that uses multilayered neural networks to deliver best-in-class performance and accuracy for the low-level tasks that drive higher-level applications of autonomy.

ACKNOWLEDGMENTS

The author thanks Mr. Mark Butkiewicz (SURVICE Engineering), Dr. Shawn Recker (LufCo), and Mr. Jonathan Worobey (Meta Reality Labs Research) for the many conversations and technical discussions that contributed to the information presented in this report. The author also thanks the CSIAC contributors Mr. Phil Payne and Mr. Ryan Fowler and the external subject matter expert reviewers Mr. Marek Ososinski (Tando Ltd.) and Mr. Jonathan Worobey for their helpful and insightful comments regarding early drafts of this state-of-the-art report.

CONTENTS

	ABOUT CSIAC	**iv**
	THE AUTHOR	**vi**
	ABSTRACT	**vii**
	ACKNOWLEDGMENTS	**viii**
SECTION 1	**INTRODUCTION**	**1-1**
SECTION 2	**BACKGROUND**	**2-1**
SECTION 3	**HARDWARE**	**3-1**
3.1	Core Operations	3-1
3.2	Parallel Computing	3-2
3.3	Commodity Hardware Platforms	3-3
3.4	Special-Purpose Hardware	3-4
3.5	Hardware Considerations	3-5
SECTION 4	**SOFTWARE**	**4-1**
4.1	Programming Model	4-1
4.2	DL Programming Libraries	4-2
4.3	DL Frameworks	4-3
4.4	Software Considerations	4-6
SECTION 5	**APPLICATIONS**	**5-1**
5.1	Autonomous Systems	5-1
5.2	Convolutional Neural Networks	5-1
5.3	Modern Computer Vision	5-3
5.4	Example DoD Use Cases	5-5
5.5	Application Considerations	5-5
SECTION 6	**CHALLENGES**	**6-1**
SECTION 7	**SUMMARY**	**7-1**
SECTION 8	**REFERENCES**	**8-1**
SECTION 9	**BIBLIOGRAPHY**	**9-1**

CONTENTS, *continued*

FIGURES

Figure 1-1 DL for Next-Generation Autonomous Platforms 1-1
Figure 1-2 AI, ML, and DL 1-2
Figure 2-1 DL via DNNs 2-1
Figure 2-2 Timeline of AI/ML Developments 2-2
Figure 2-3 Phases of DL 2-3
Figure 3-1 Practical DL via Parallel Computing 3-1
Figure 3-2 Parallelism for Large-Scale Distributed Training 3-3
Figure 3-3 Commodity Hardware for Scalable DL 3-4
Figure 3-4 Special-Purpose Hardware for DL 3-5
Figure 3-5 Possible Tradeoffs When Considering Modern Processors for DL Applications 3-6
Figure 4-1 SIMD Processor Architectures 4-1
Figure 4-2 Hardware-Optimized DL Libraries, APIs, SDKs, and Frameworks 4-3
Figure 5-1 Pooling Layers in a CNN 5-2
Figure 5-2 Image Classification on ImageNet 5-3
Figure 5-3 Example Autonomous System Architecture 5-3
Figure 5-4 Computer Vision Techniques for Machine Perception 5-4
Figure 5-5 SURVICE Engineering's TRV-150 Tactical Resupply UAS 5-5

SECTION 01 INTRODUCTION

From self-driving cars and warehouse inventory robots to delivery drones and micro-aerial vehicles (MAVs) small enough to land on a person's hand, autonomous mobile platforms of various scales are transforming industries across the public and private sectors. The popular Roborace competition [1], for example, demonstrates the promise of artificial intelligence/machine learning (AI/ML) capabilities in revolutionizing mainstream pastimes, such as auto racing, while unmanned aerial systems (UASs), such as the tactical resupply vehicle (TRV) [2] depicted in Figure 1-1, provide game-changing capabilities to the Warfighter for logistics resupply. These platforms are powered by deep learning (DL), a class of AI/ML algorithms that solve the representation learning problem by building complex representations from simpler concepts [3].

Broadly, AI is a branch of computer science that seeks to replicate or simulate human intelligence in a machine. AI systems are powered by algorithms that exhibit intelligence through decision-making. ML is a subset of AI and uses statistical techniques

Figure 1-1: DL for Next-Generation Autonomous Platforms *(Source: SURVICE Engineering Company).*

to enable an AI system to learn—the system gets better at tasks over time, without having to be specifically programmed to do so. Similarly, DL is a subset of ML in which learning algorithms attempt to mimic the human brain using multilayered algorithmic structures called neural networks. The relationship between AI, ML, and DL is depicted in Figure 1-2.

Deep neural networks (DNNs) are one approach to DL in which a network composed of artificial neurons takes several inputs and produces an output. These neurons are grouped together into layers such that one layer is connected to both the preceding and subsequent layers. Flexible DNN architectures enable a diverse range of applications, from computer vision and speech recognition to medical imaging and combat support.

With the mapping of DNNs to modern massively parallel computing architectures [4], DNNs now achieve breakthrough performance in modern computer vision tasks, including image classification, object detection, and image segmentation—tasks that form the basis of autonomous mobile platforms. In fact, DNN performance in these tasks rivals or even surpasses human capabilities [5]. The neuron layers, or convolutional layers, in so-called convolution neural networks (CNNs) drive performance in these tasks. The convolutional layers automatically learn important visual features from vast collections of training data by optimizing convolutional operations applied to these data—features that were previously extracted by complex algorithms laboriously handcrafted and explicitly programmed by computer vision experts.

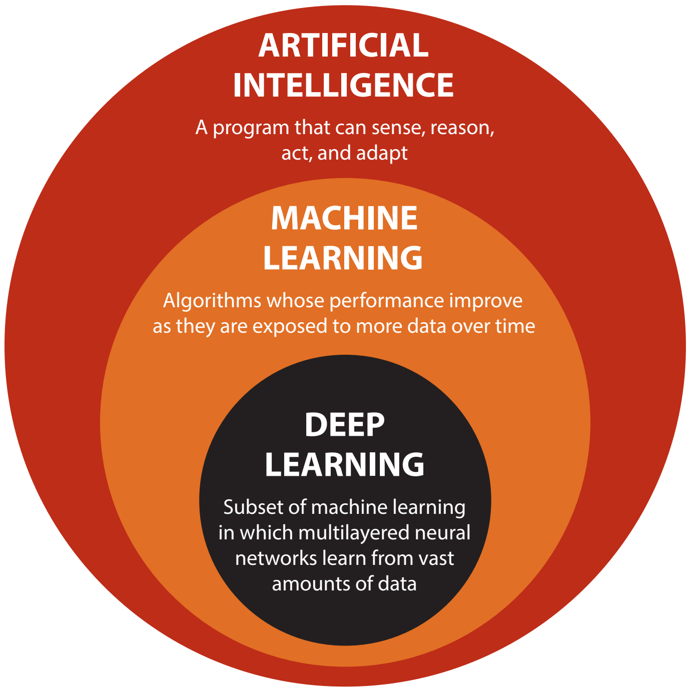

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Figure 1-2: AI, ML, and DL *(Source: SURVICE Engineering Company).*

High-performance DNNs, massively parallel computing architectures, and hardware-optimized software components now combine with real-world training data to solve problems in autonomy for small, inexpensive platforms. This report reviews state-of-the-art AI/ML hardware and software technologies supporting autonomy on these platforms, focusing on commodity hardware components and widely available software ecosystems for DL. Together, these technologies deliver best-in-class performance and accuracy for the low-level tasks that drive higher-level applications of autonomy.

SECTION 02 BACKGROUND

DL is a class of AI/ML that uses DNNs, or multilayered artificial neural networks (ANNs), to deliver state-of-the-art performance in complex tasks, such as image classification and speech recognition, among others.

ANNs are statistical models that adapt (via self-programming) by using learning algorithms to build complex representations from simpler concepts. These networks date back to the early 1940s, when mathematicians McCulloch and Pitts built a simple algorithm-based system to emulate human brain function [6]. They used a combination of mathematics and algorithms, or what they called "threshold logic units," to encode logical propositions.

As depicted in Figure 2-1, modern ANNs are composed of artificial neurons, or nodes, arranged in several layers that operate in parallel. Each neuron has one or more inputs and produces a single output, which is then forwarded to one or more nodes in the next layer. The input layer is analogous to dendrites in the human brain. The hidden layer, comparable to the cell body, sits between the input layer and the output layer, which itself is analogous to synaptic outputs in the human brain. The hidden layer ingests inputs based on synaptic weight, the amplitude or strength of a connection between nodes. These weighted inputs generate an output through a transfer or activation function to the output layer.

In 1958, Rosenblatt introduced the perceptron—a groundbreaking algorithm designed to perform

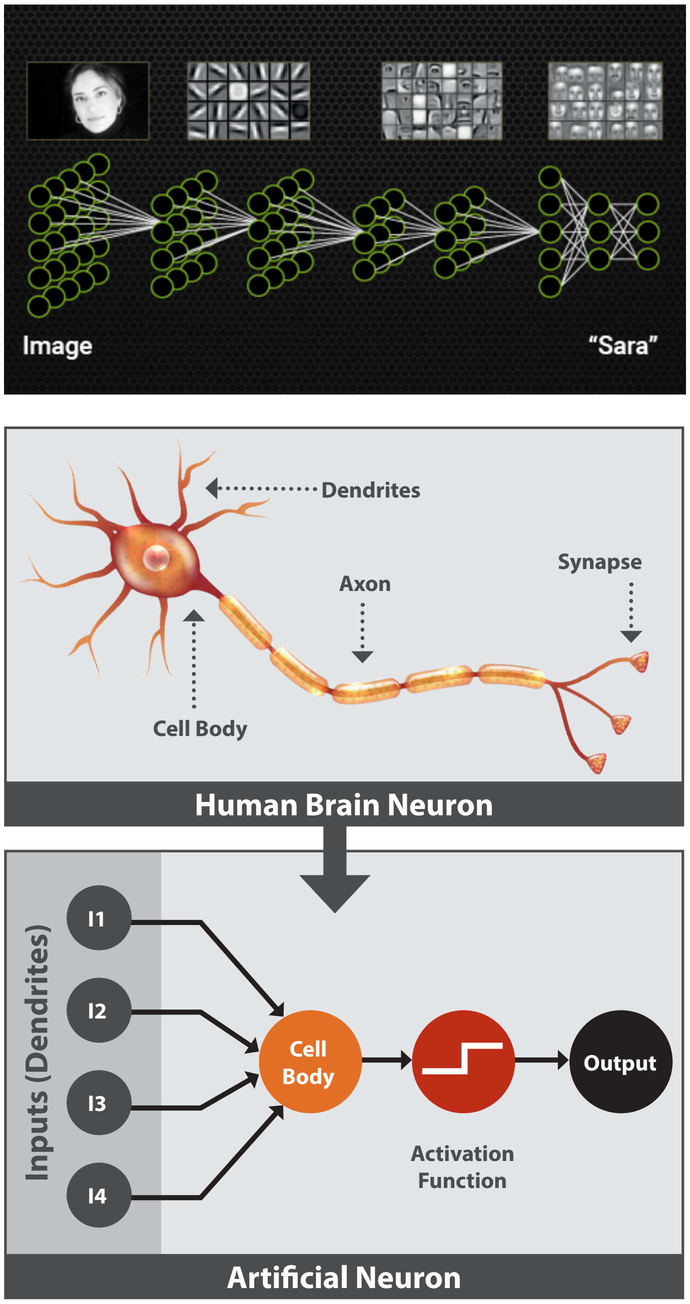


Figure 2-1: DL via DNNs *(Source: NVIDIA and SURVICE Engineering).*

complex recognition tasks [7]. The perceptron was designed for image recognition, and, though originally conceived as machine, the first implementation was a software program. Initially promising, practitioners quickly proved that perceptrons could not be trained to recognize many pattern classes. Nevertheless, mathematical and algorithmic progress continued throughout the 1960s, 70s, and 80s, with notable advances, including development of the basis of back propagation [8, 9], polynomial activation [10], the first "convolutional" neural network [11], and the first practical demonstration of back propagation [12]. However, throughout this time, lack of computing power sufficient to process large amounts of data hindered application of these developments in practical settings.

Not until the 2000s, when massively parallel computing hardware and large repositories of real-world training data became commonplace, did practitioners have the necessary components to realize practical applications. In fact, by 2011, modern graphics processing units (GPUs) offering hundreds or even thousands of processing elements made it possible to train CNNs without tedious layer-by-layer pretraining [13]. With increased computing power, the significant advantages of DL in terms of speed and efficiency became obvious. The timeline of historical AI/ML developments is summarized in Figure 2-2.

Unlike traditional AI/ML techniques, DL automatically learns representations from data (images, video, text, etc.) and does not require explicitly programmed rules or significant domain knowledge from human experts. Instead, DNN models learn directly from real-world data in a process called training—the first of two phases necessary to utilize DL models. The second phase, called inference, is the use of a trained DNN model to make predictions against previously unseen data. These phases are depicted in Figure 2-3. Loosely speaking, research demonstrates that more training data leads to lower estimation variance and, thus, better predictive performance; that is,

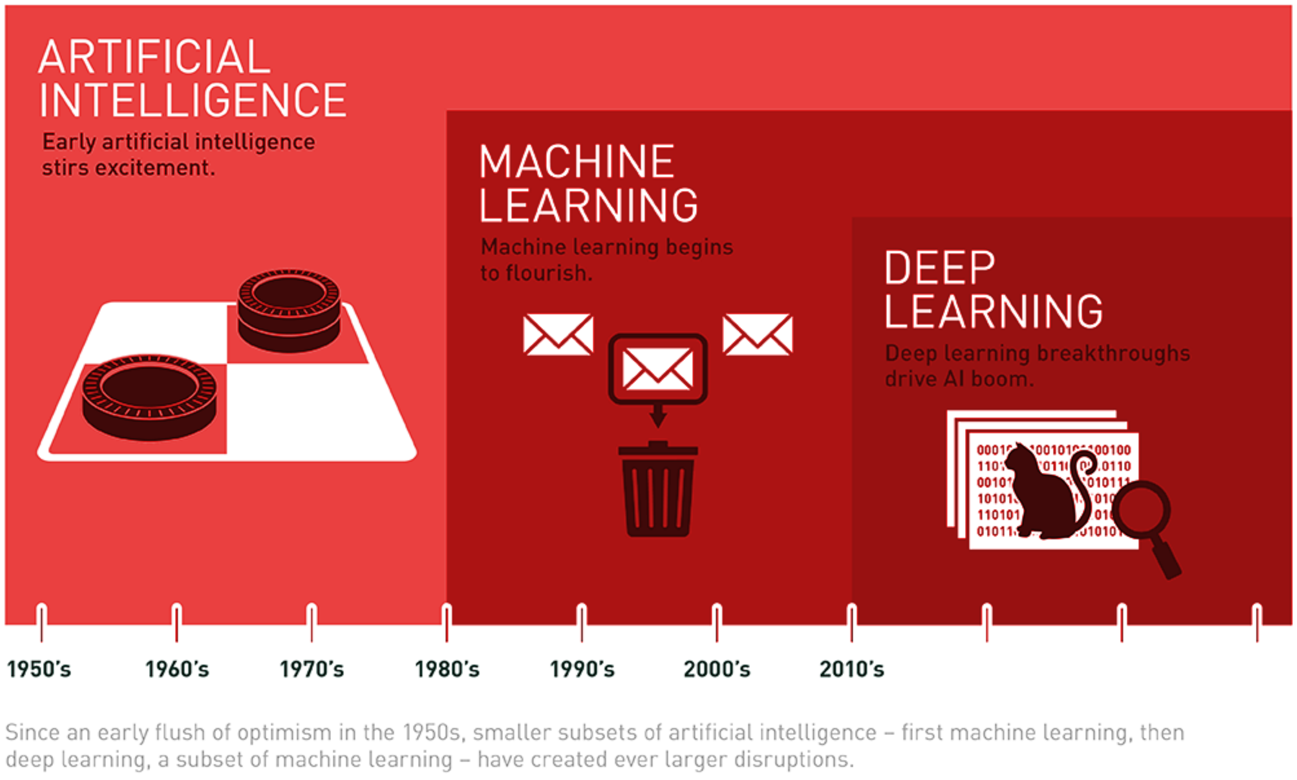

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Figure 2-2: Timeline of AI/ML Developments *(Source: NVIDIA)*.

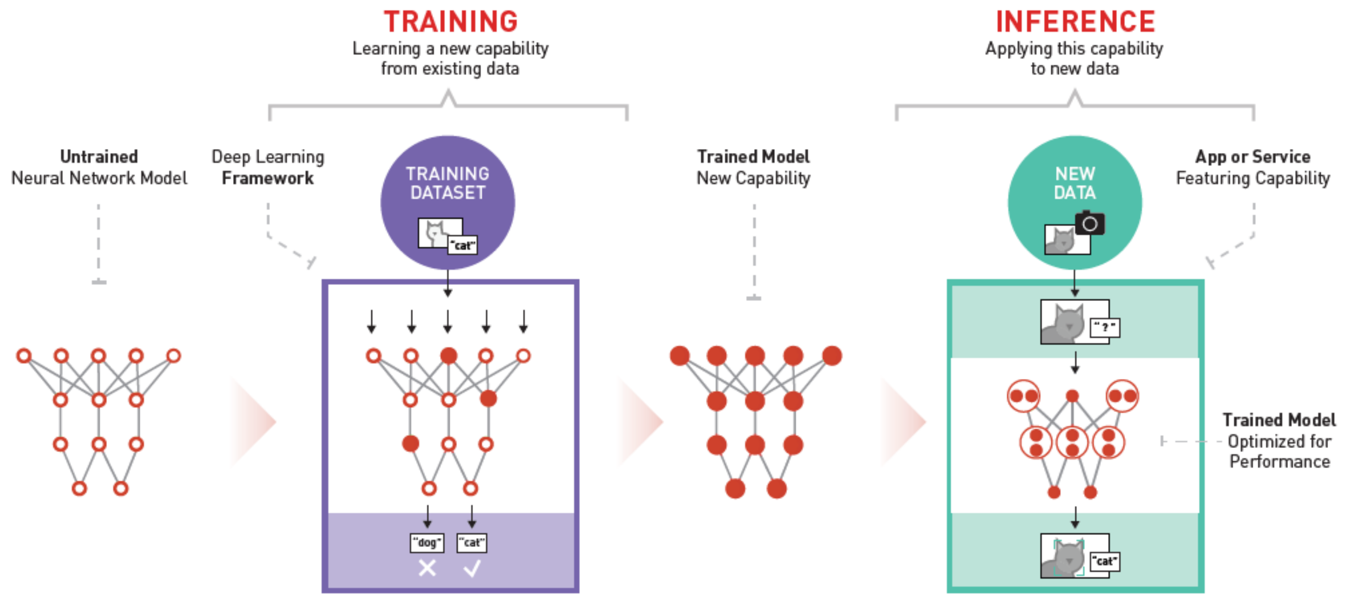

TRAINING
Learning a new capability from existing data
INFERENCE
Applying this capability to new data
Untrained Neural Network Model
Deep Learning Framework
TRAINING DATASET
"cat"
"dog"
"cat"
Trained Model New Capability
NEW DATA
"?"
App or Service Featuring Capability
Trained Model Optimized for Performance
"cat"

Figure 2-3: Phases of DL (Left: DNN Training; Right: DNN Inference) *(Source: NVIDIA).*

more data increases the probability that the DNN encounters useful information during training, which can be advantageous for inference [14].

With hardware-accelerated DL frameworks, the time required to train DL models is reduced from days or weeks to just hours or days. When these trained models are ready for deployment, hardware-accelerated inference platforms (from the smallest embedded systems and mobile devices to desktop personal computers [PCs], workstations, and the cloud) deliver high-performance, low-latency inference for the most computationally intensive DNNs across all scales.

AlexNet [4], a CNN architecture that won several international computer vision competitions during 2011 and 2012, is the most well-known example of the impact massively parallel computing architectures (modern GPUs, in particular) have had on ML with DNNs. The depth of AlexNet (i.e., the number of convolutional layers) was certainly critical to its record-breaking performance in these competitions. However, because deeper networks impose more computation, training AlexNet only became feasible with the use of contemporary GPUs [15].

In particular, the AlexNet architecture consists of five convolutional layers and three fully connected layers. However, depth alone is not the only novel aspect of AlexNet; additional breakthrough features include:

- **Rectified Linear Units (ReLU) Nonlinearity.** AlexNet uses ReLU instead of the sigmoid or hyperbolic tangent (tanh) functions, which were standard at the time. ReLU was first shown to enable training supervised DNNs in 2011 [16], and it allows fast and effective training of DNN architectures on large, complex datasets.
- **Multiple GPUs.** AlexNet allows parallel training by distributing the model across two GPUs, or so-called model-parallel training. Parallel training increases the size of models that can be trained and reduces the time required to train any particular model.
- **Overlapping Pooling.** Prior to AlexNet, typical CNNs pooled outputs of neighboring neuron groups without overlapping. However, by introducing overlapped pooling, AlexNet reduced both top-1 and top-5 error rates [17].

AlexNet is a large model with 60 million parameters and 650,000 neurons, which increases the model's capacity to learn critical features for the task at hand. However, training a large model can also lead to overfitting—a problem in which the model effectively memorizes features in training samples and is unable to generalize these features to new, previously unseen data. AlexNet combats overfitting using the following:

- **Data Augmentation.** Label-preserving transformations introduced additional variation in the training data. Specifically, the number of training samples was increased by more than 2000x using image translations and (horizontal) reflections, and image channel intensities were modified using principal component analysis to create more training data.
- **Dropout.** Here, neurons are disabled with a predetermined probability so that every iteration uses a different subset of the model's parameters. Dropout is just one approach to regularization—any method or process designed to prevent overfitting by reducing interdependent learning among neurons. Dropout increases the training time required for convergence but forces neurons to learn features more robustly.

AlexNet achieved a top-5 error of 15.3% in the 2012 ImageNet Large Scale Visual Recognition Challenge [18], or nearly 11 percentage points better than its closest competitor. With the success of AlexNet, the practical application of DL began in earnest.

Later, Google DeepMind's AlphaGo became the first computer program to defeat a professional human Go player and the first to defeat a Go world champion [19]. AlphaGo combines advanced tree search with DNNs. One neural network (the policy network) selects the next move to play, and a second neural network (the value network) predicts the winner of the game. AlphaGo was first exposed to amateur Go games to develop an understanding of reasonable human play, followed by play against different versions of itself thousands of times—each time learning from its mistakes. This process, known as reinforcement learning, rewards desired behaviors, punishes undesired ones, or both. In general, a reinforcement learning agent (in this case, the AlphaGo program) perceives and interprets its environment, acts with that environment, and learns through trial and error. AlphaGo eventually defeated Go world champions in different global arenas and has arguably become the greatest Go player of all time.

DNNs have found similar success in natural language processing (NLP), or algorithms that represent and analyze human language. NLP-based systems enable a wide range of applications, including virtual-assistant technologies, machine translation, dialogue generation, and others. Much like computer vision, NLP techniques were traditionally based on shallow ML models and time-consuming, hand-crafted features. More recently, however, DNNs have achieved superior results on various language-related tasks compared to traditional ML models. For example, early work by Collobert et al. [20] demonstrated a simple DL framework that outperformed most contemporary state-of-the-art approaches in several NLP tasks, and numerous algorithms based on DL have been proposed to solve difficult NLP tasks since then [21].

DL finds successful application across a wide range of other domains, including medical image analysis [22], wireless communication [23], robotics [24], and more [25, 26]. Interested readers are encouraged to consult the many available resources to learn more about the application of DNNs to problems in science, engineering, and medicine.

SECTION 03 HARDWARE

Widely available massively parallel computing architectures have revolutionized DL, making it practical to apply complex DNN models to real-world problems. For example, consider DNN training, as outlined in Figure 3-1. The DNN ingests data at its input layer, multiplies these inputs by the synaptic weights (or more simply, weights) in its hidden layers, and then outputs a prediction. Weights are adjusted throughout training to extract meaningful patterns and thus make better predictions.

3.1 CORE OPERATIONS

Typically, the goal of training is to reduce prediction error or loss—the difference between predicted outputs and reference data. Reference data is constant; therefore, training must change prediction values by updating synaptic weights to reduce error.

Backpropagation is a commonly used mechanism to update weights via gradient descent—an iterative first-order optimization algorithm that finds a local minimum or maximum value of a function. In training, backpropagation computes the gradient of the loss function with respect to each weight one layer at a time and iterates backward from the last layer to avoid redundant calculation of intermediate terms. This approach makes the application of gradient methods feasible for training deep multilayer neural networks.

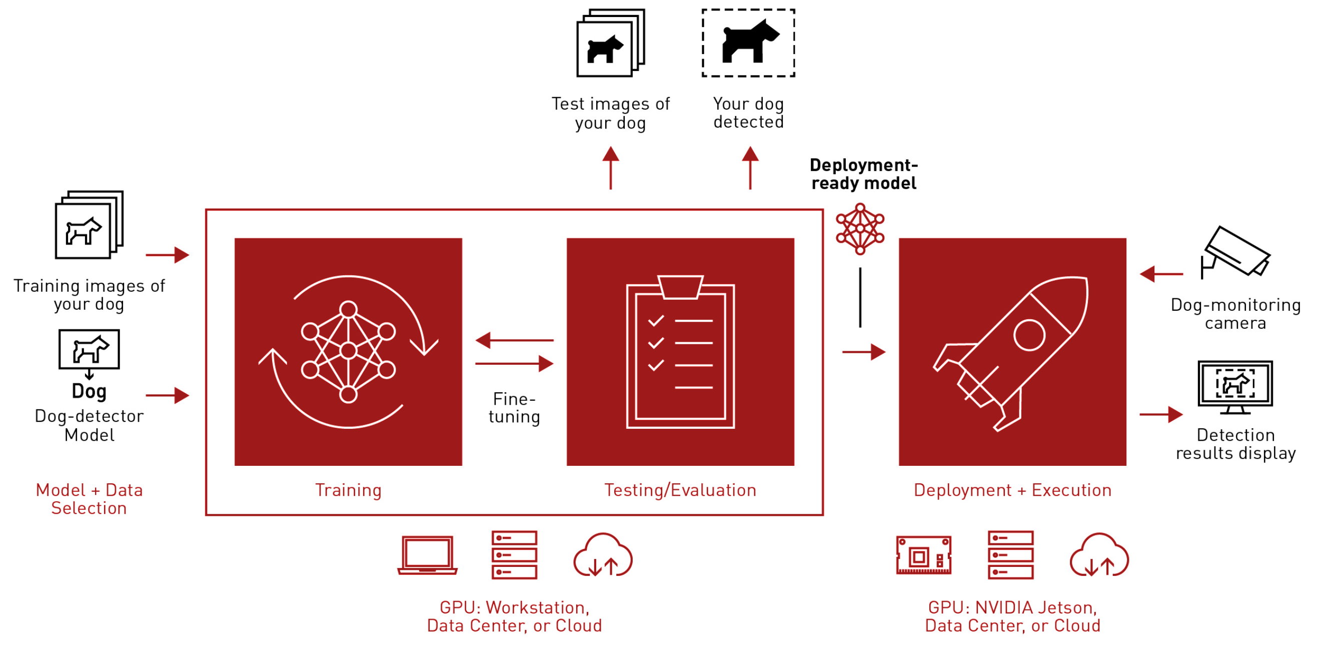


Figure 3-1: Practical DL via Parallel Computing *(Source: NVIDIA).*

Backpropagation refers only to the algorithm for computing gradients, not how the gradient is used; however, the term is often used loosely to refer to the entire learning algorithm. Backpropagation generalizes the gradient computation in the delta rule—the gradient descent learning rule for updating weights in a single-layer version of backpropagation. It is, in turn, generalized by automatic differentiation—a set of techniques for evaluating the derivative of a function. Interested readers are encouraged to consult the many available resources to learn more about DNN training; for example, Goodfellow et al. [3] provide a modern overview of backpropagation and differentiation algorithms in DL.

DNN inference, also outlined in Figure 2-3, proceeds in much the same way—the network ingests data at its input layer, multiplies these inputs by its weights, and then outputs a prediction. In contrast to the training phase, however, DNN weights are not adjusted during inference and instead use the values determined by training and are fixed at the time of deployment.

In both phases, the core operation is tensor multiplication. Generally, tensors are mathematical objects that describe relationships between other mathematical objects that are themselves linked in some way. In DNN training and inference, tensors are simply two-dimensional matrices, so tensor multiplication is essentially matrix multiplication—an array of inputs multiplied by an array of weights. Modern DNN models comprise potentially billions of such weights, however, so computations for even a single training iteration could require hours, days, or even weeks.

3.2 PARALLEL COMPUTING

Fortunately, massively parallel architectures execute multiple, simultaneous computations across tens, hundreds, or even thousands of processing cores, each of which uses fewer resources than a more traditional central processing unit (CPU). These architectures implement parallel computing—an approach in which a complex task (e.g., DNN training) is divided into smaller, independent computations that can be executed simultaneously. The results of these independent computations are then combined to form the result of the original task. The number of computations into which a task is divided depends, in part, on the number of available processing cores. Typical CPUs have 4, 8, or 16 cores, while GPUs have hundreds or thousands.

In fact, DL tasks are embarrassingly parallel, meaning little or no effort is necessary to divide a task into smaller, independent computations. Recall that, during both training and inference, a DNN ingests data at its input layer, multiplies these inputs by weights in its hidden layers, and outputs a prediction. Here, the most common functions are basic linear algebra operations, such as matrix multiplication and addition. At the same time, computations for each node in a layer are independent of every other node in that layer; therefore, these computations can be executed simultaneously. Massively parallel computing architectures are thus well matched to both training and inference.

Modern GPUs are the most prevalent massively parallel computing architectures at present. GPUs are specialized processors with dedicated memory originally designed to accelerate the operations required for graphics rendering—the process of generating images from three-dimensional models or scene descriptions via a computer. Rendering is itself an embarrassingly parallel problem; consequently, GPUs have evolved to execute hundreds or thousands of these operations in parallel to support increasingly faster rendering times. Moreover, GPUs began to expose programmable elements in their otherwise fixed-function rendering pipelines, giving software developers low-level control over many aspects of computation. Together, these GPU features (unprecedented levels of parallel hardware under explicit programmer control) paved the way for accelerated DL, as demonstrated by AlexNet.

Although not the first CNN to exploit GPUs for accelerated DL, AlexNet is often seen as the catalyst behind the GPU-accelerated DL revolution [15].

The parallelism inherent to DNN training extends to even larger scales via distributed training, wherein either computation or data is distributed across many independent processors (e.g., multiple CPUs in a cluster or multiple GPUs in a workstation). As illustrated in Figure 3-2, distributed training usually proceeds in one of two ways: model parallel or data parallel.

In model-parallel training, each layer's parameters and the corresponding computations in the DNN (i.e., the model itself) are distributed across the processors, and each one ingests the same data. With data-parallel training, training data is distributed across the processors and each one executes the entire DNN model on its data subset. Model-parallel training permits DNNs with increasingly more parameters to improve model performance, while data-parallel training permits training with progressively more data to complete within practical timeframes. Distributed training is thus essential to robust, large-scale DL solutions.

3.3 COMMODITY HARDWARE PLATFORMS

Major processor manufacturers, including AMD, ARM, Intel, and NVIDIA, offer scalable, accelerated solutions for DL training, typically comprising their respective processor technologies and a DL software stack optimized for that platform. From multi-GPU workstations to purpose-built DL systems of various scales, these products (as well as those offered by top cloud service providers, original equipment manufacturer partners, and various resellers) provide easy access to the latest massively parallel computing architectures that enable fast, efficient, and economical DNN design, training, and optimization. Some of these commodity hardware platforms are depicted in Figure 3-3.

Whereas these platforms are typically designed for DL training in the laboratory, in the data center, or in the cloud, other platforms are designed for DL inference in the field. For example, NVIDIA Jetson is a DL platform designed for autonomous machines and other in-situ applications running at the edge. The Jetson platform includes Jetson modules (small, low size, weight, and power [SwaP] and

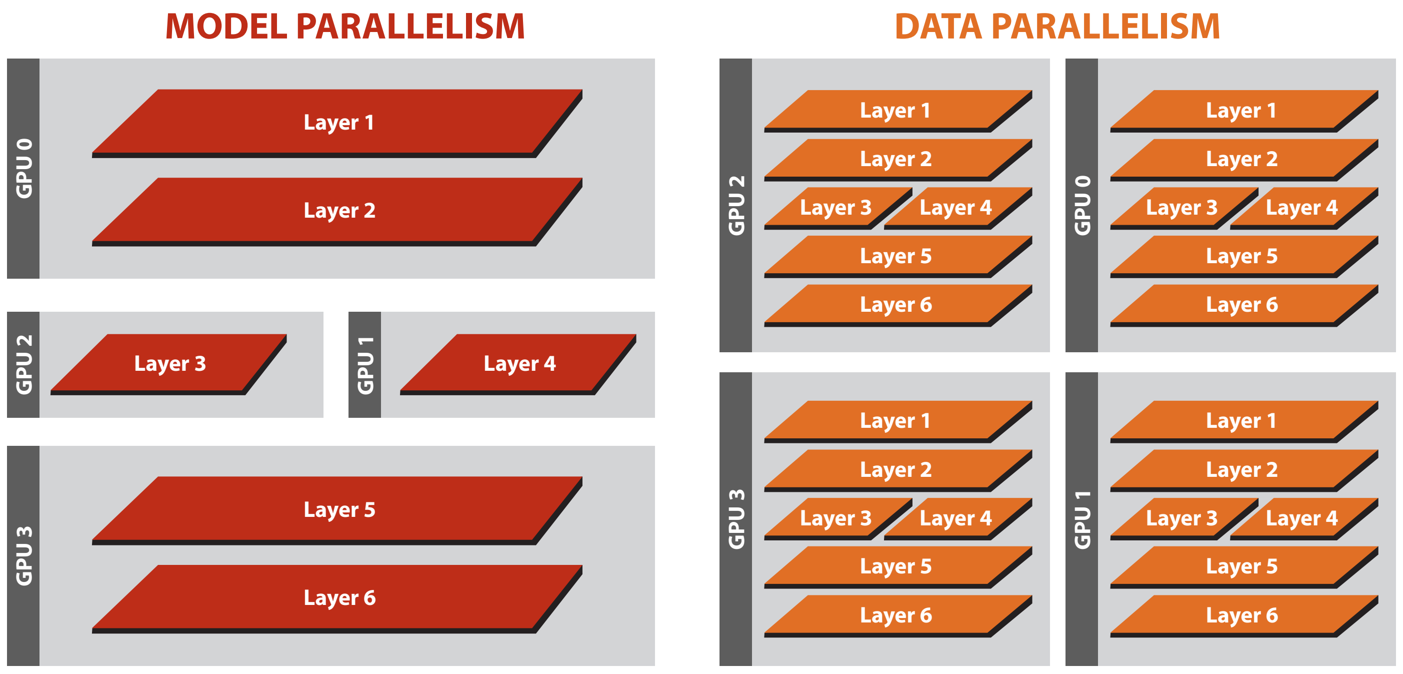


Figure 3-2: Parallelism for Large-Scale Distributed Training (Left: Model Parallelism; Right: Data Parallelism) *(Source: Jordi Torres).*

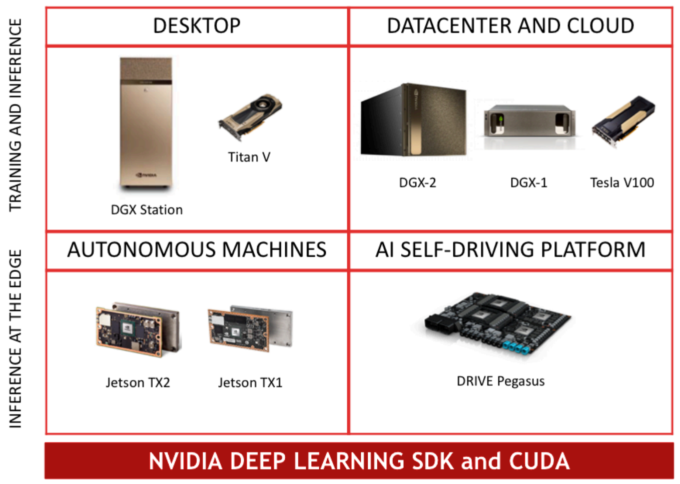


Figure 3-3: Commodity Hardware for Scalable DL *(Source: NVIDIA).*

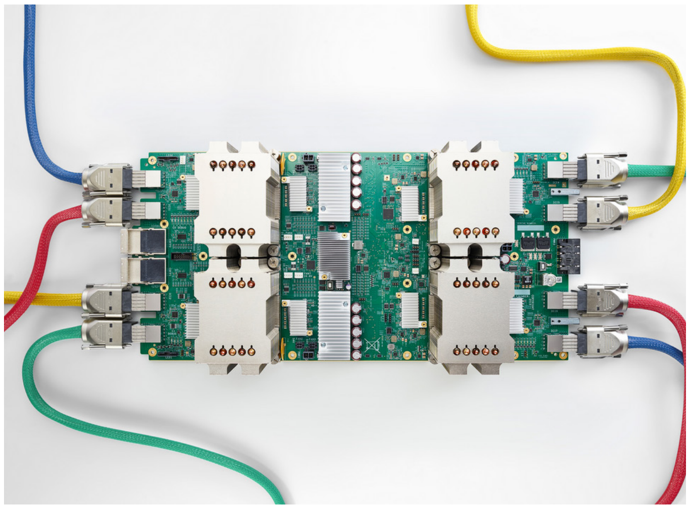
high-performance system-on-module computers, complete with a CPU, GPU, memory, power management, and high-speed input/output [I/O] interfaces); the NVIDIA JetPack software development kit (SDK); and an entire ecosystem of compatible sensors, SDKs, services, and products to support application development. Jetson also executes the same DL software and workflows used across other NVIDIA DL platforms, which enables these applications to scale up or down as required by deployment targets. Together, these features make NVIDIA Jetson an ideal and unique platform for low-cost, low-SwaP, and high-performance DL inference on next-generation autonomous machines.

3.4 SPECIAL-PURPOSE HARDWARE

Just as GPUs originally evolved to enable fast rendering times with specialized hardware designed for rendering operations, these architectures are now evolving to enable increasingly faster DNN training and inference operations with specialized hardware. For example, the recently announced NVIDIA Hopper GPU architecture includes NVIDIA's latest Tensor Cores—special-purpose hardware designed to accelerate the tensor operations underlying DNN training and inference. The AMD CDNA2 GPU architecture includes similarly special-purpose hardware called Matrix Cores. These specialized compute units provide high-performance reduced- and mixed-precision operations, while direct support in native DL software frameworks via hardware-optimized libraries provides automatic implementation, reduces training times, and maintains accuracy. These same features (high-performance reduced- and mixed-precision operations and direct support in native DL software frameworks) provide low latency at high throughput and maximize utilization to deploy inference reliably across scales.

Whereas NVIDIA Tensor Cores and AMD Matrix Cores are specialized GPU components designed to accelerate core DL operations, the Google Tensor Processing Unit (TPU) and Intel Neural Compute Stick 2 (NCS2) are entire devices designed to accelerate DNN operations. The Google TPU, introduced in 2016, is a custom, application-specific integrated circuit built specifically for ML and tailored for the Google TensorFlow ML software platform. Google TPUs are available commercially via Google Cloud, where they can be connected to virtual machines and mixed with other types of hardware for DNN training. Likewise, the Intel NCS2 is a dedicated hardware accelerator for DNN inference. Packaged as a plug-and-play Universal Serial Bus thumb stick, this special-purpose device exploits Intel's Neural Compute Engine and 16 programmable cores to accelerate DNN inference. These devices are depicted in Figure 3-4. Unlike GPUs, which support compute-intensive applications beyond DL, these purpose-built devices are specifically designed to accelerate DNN operations and are not intended to support applications in other domains.

Similarly, vision processing units (VPUs) are a type of system-on-chip designed to acquire and process visual data. VPUs typically target mobile applications and are optimized for small size and power efficiency. For example, Intel's Movidius Myriad X VPU [27] can interface with an image sensor, preprocess captured image data, and pass results through a pretrained DNN to compute



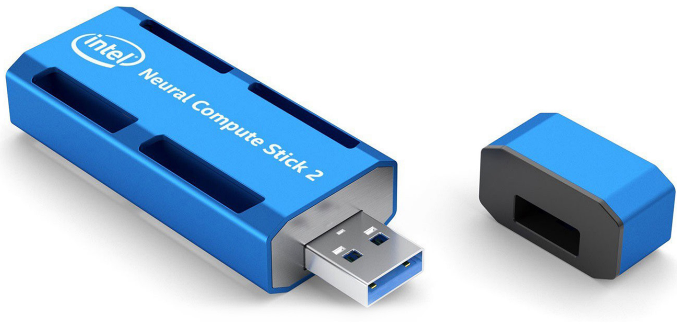


Figure 3-4: Special-Purpose Hardware for DL (Top: Google TPU; Bottom: Intel's NCS2) *(Source: Google and Intel).*

predictions, all in a low-cost, low-SwaP design that balances computational performance and power efficiency.

Typical VPUs are not specifically designed for DNN operations; however, these processors excel at convolutions and other parallel operations due to single-instruction, multiple-data (SIMD) vector units. Intel's VPUs, for example, combine traditional CPU cores and SIMD units to accelerate the highly branching logic typical of DNNs and other computer vision algorithms.

Moreover, power-efficient designs make VPUs particularly well suited for embedded applications, such as those running on handheld or mobile devices that require long battery life. Though less powerful than GPUs, VPUs are quite small; for example, Teledyne FLIR's Firefly DL camera [28] offers on-camera, DNN-based decision-making without a host computer system, and it is designed specifically for low-SwaP-embedded applications, including mobile platforms.

Field-programmable gate arrays (FPGAs) offer an interesting middle ground between fully programmable processors and purpose-built, application-specific devices. An FPGA is a hardware circuit with reprogrammable logic gates that enables users to create or program a custom circuit while the chip is deployed, not just during the design or fabrication phases. This programmability contrasts with standard processors in which circuits are hard wired and cannot be reprogrammed. FPGAs with thousands of memory units enable circuits to implement a massively parallel computing model, much like GPUs. Moreover, FPGAs are particularly well suited for embedded applications due to lower power requirements than either CPUs or GPUs.

Programming FPGA circuits typically requires significant expertise, however, and though some work has been done in this area (e.g., the DL Accelerator Unit [29]), implementing FPGAs for practical DL applications is relatively untested. Lack of support and minimal community knowledge also suggest that FPGAs are not yet widely accessible as a commodity DL technology.

3.5 HARDWARE CONSIDERATIONS

Modern CPUs, GPUs, DL-specific devices, VPUs, and even FPGAs form the foundation for high-performance DL applications. These processors execute and accelerate the core operations required to design, train, and deploy complex DNN models, typically through widely accessible DL software stacks featuring platform-specific libraries, flexible application programming interfaces (APIs), and end-to-end frameworks optimized for the underlying hardware architecture.

Each of these architectures offers certain advantages and suffers from certain disadvantages; therefore, hardware selection is a key consideration in planning an end-to-end DL system. Potentially, significant differences in architecture between CPUs, GPUs, VPUs, and FPGAs make performance comparisons in terms of floating-point operations per second of little practical value, however. Comparing published inference time is a useful starting point, but inference time alone may be misleading. For example, processor A may process a single frame faster than processor B, but processor B could process multiple frames in parallel and yield greater throughput. As outlined in Figure 3-5, other considerations, including, cost, power consumption, physical size, and software support, may be a factor as well.

Testing is the only sure-fire method for comparing available architectures. Prior to selecting the hardware for a DL-enabled system, practitioners should conduct various tests to determine the accuracy, performance, and efficiency requirements necessary to satisfy application constraints. These parameters will determine the characteristics of the DL hardware onto which the required DNN can be successfully deployed.

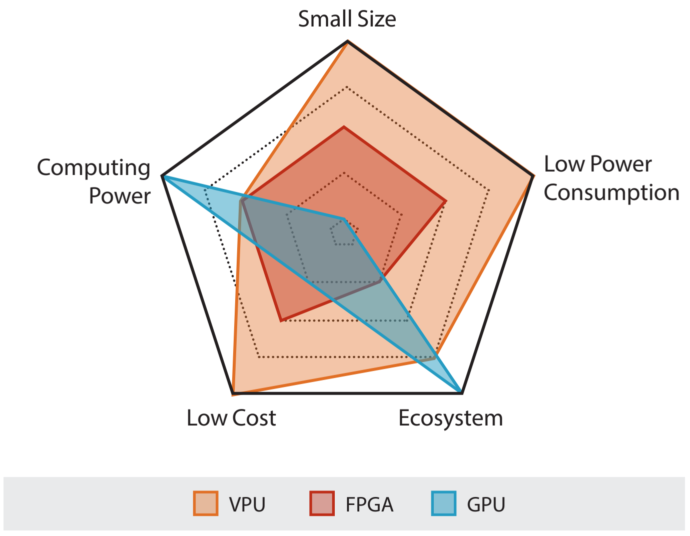


Figure 3-5: Possible Tradeoffs When Considering Modern Processors for DL Applications *(Source: SURVICE Engineering Company).*

The specific hardware components highlighted in this report represent only a small subset of the current hardware technologies supporting DL applications. Interested readers are encouraged to consult the many available resources to learn more about these and other hardware technologies enabling modern, high-performance DL.

SECTION 04 SOFTWARE

Massively parallel computing architectures provide significant performance gains when algorithms are designed to fully exploit their underlying hardware configuration. While several implementations of these architectures are available, many share a common computational model: data-level parallelism. The SIMD model is one approach to data-level parallelism in which a single instruction stream executes across multiple data streams simultaneously. To exploit the benefits of data-level parallelism, programmers must understand and use these SIMD units carefully and correctly, which is often a difficult and time-consuming task.

4.1 PROGRAMMING MODEL

As illustrated in Figure 4-1, SIMD architectures impose the constraint that all SIMD units associated with a single control unit execute the same instruction across a group of data elements. Most available SIMD architectures support wide memory-fetch operations that fill an entire SIMD vector unit in a single fetch. These memory operations take several orders of magnitude longer to execute than a SIMD arithmetic operation; therefore, arranging data elements in a manner that minimizes memory fetches per arithmetic operation in turn maximizes computational throughput, leading to higher performing code. The parallel programming models for SIMD units underlying modern processor architectures thus dictate that computations be arranged differently than with traditional programming models.

Nearly all computing devices (from traditional workstation- and desktop-class computers to low-power tablets, smartphones, and other embedded systems) now include some form of SIMD unit. Although not specifically designed for DNN operations, these units accelerate the highly parallel algorithms supporting modern DL techniques, and thus form the basis of low-cost,

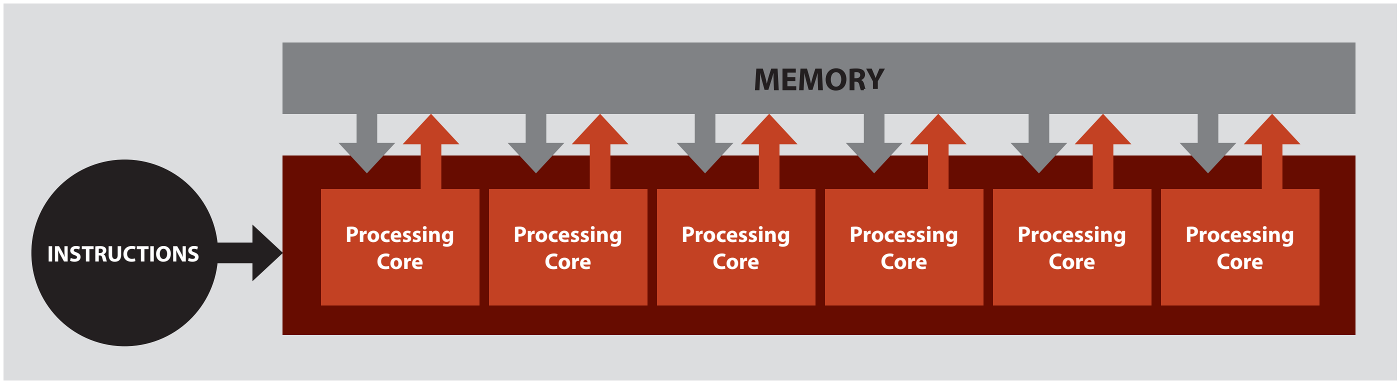


Figure 4-1: SIMD Processor Architectures *(Source: SURVICE Engineering Company).*

low-SwaP hardware architectures supporting applications in edge computing, including autonomous platforms.

Despite the ubiquitous nature of these units, SIMD processors require algorithms and data structures radically different from those used in traditional software [30, 31], making it difficult for existing approaches to realize the performance promised by massively parallel computing. As a result, software applications across many domains do not leverage SIMD units effectively, due in part to the challenges of developing efficient algorithms for these processors. However, with careful consideration of the low-level architectural features, the SIMD units of modern processors offer potentially significant increases in runtime performance across the full range of computing platforms.

Modern parallel programming paradigms and compiler technologies help to address some difficulties imposed by massively parallel computing architectures. For example, NVIDIA's Compute Unified Device Architecture (CUDA) or Intel's Single Program, Multiple Data (SPMD) Program compiler (ispc) offer variants of the C/C++ programming language with extensions for SPMD programming. In this model, a parallel program appears to be a regular serial program; however, the compiler and language runtime implement an execution model that runs several program instances in parallel on the underlying SIMD hardware. When implemented correctly, SPMD programs frequently achieve performance improvements of about 2×–16× on modern CPUs and 10×–100× on modern GPUs—all without the difficulty of code using SIMD intrinsics. These programming tools also support parallelization across the full range of processor architectures so that programs achieve performance portability by scaling with both core count and SIMD unit width—all without device-specific code paths.

Rarely are practitioners required to develop DL applications for massively parallel computing architectures at this low level, however. Instead, popular DL libraries, APIs, SDKs, software frameworks, and product ecosystems leverage these technologies to exploit the computational power afforded by modern massively parallel computing architectures and expose DL functionality and workflows (from core operations to end-to-end DL solutions) to increase productivity, promote flexibility, and realize scalability for modern DL applications.

4.2 DL PROGRAMMING LIBRARIES

Processor manufacturers offer several hardware-optimized software libraries for applications requiring highly customized DL components, as highlighted in Figure 4-2. For example, NVIDIA CUDA-X provides core libraries delivering high performance for customized DL applications across various deployment targets (from resource-constrained embedded devices to big iron machines in high-performance computing [HPC] centers). CUDA-X includes highly optimized libraries for common math, communication, and DL tasks, including GPU-accelerated DNN primitives (cuDNN), high-performance inference (TensorRT), and real-time streaming analytics and multisensor processing (DeepStream SDK), among others. The CUDA-X libraries are themselves implemented using CUDA and exploit the specialized hardware components of NVIDIA GPUs, including NVIDIA Tensor Cores.

Similarly, AMD offers ROCm—an open software platform that enables developers to execute applications on CPUs, GPUs, or both. ROCm supports a broad range of both AMD and non-AMD GPUs and is open to enable support for third-party GPU and FPGA devices. Moreover, ROCm encourages a write-once, run-anywhere (WORA) development model, allowing developers to write, test, and debug software applications in a device-independent manner. These applications can then be deployed across all supported systems at all supported scales. Similar to CUDA-X, ROCm includes highly optimized libraries for common math, communication, and DL tasks, including accelerated DNN primitives (MIOpen) and the OpenVX computer vision API (MIVisionX), among

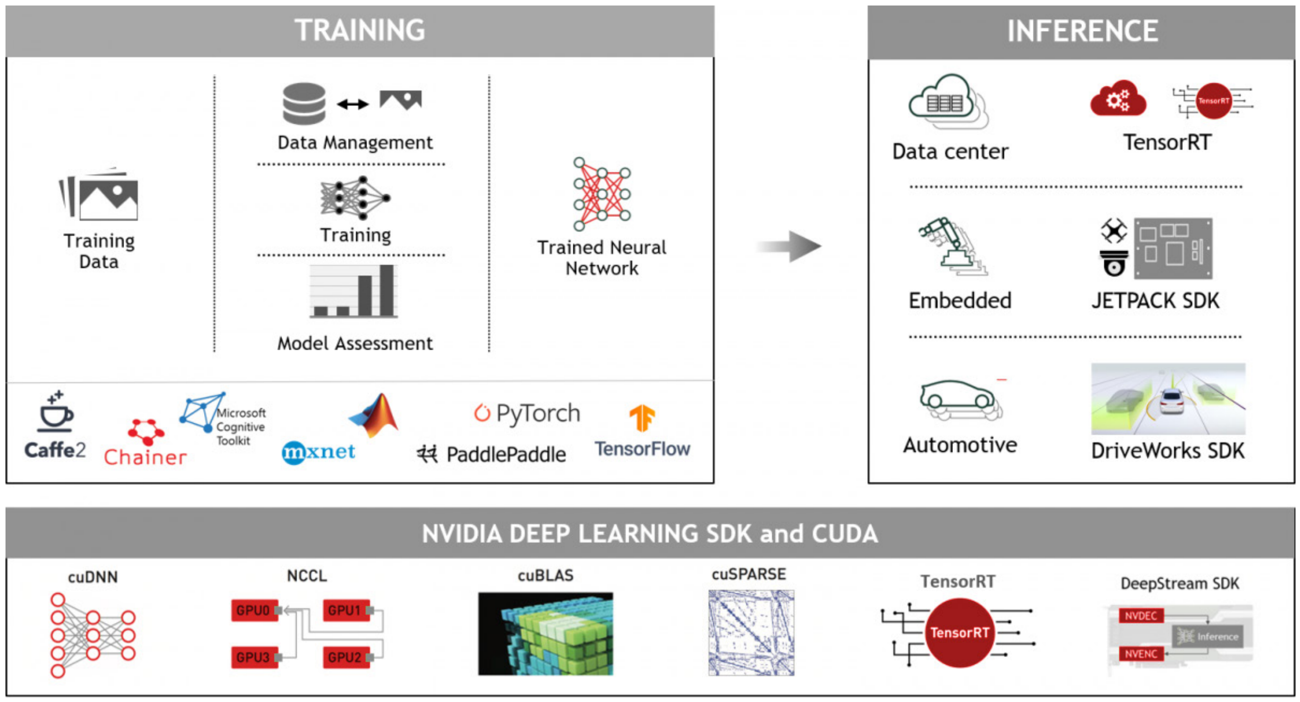


Figure 4-2: Hardware-Optimized DL Libraries, APIs, SDKs, and Frameworks *(Source: NVIDIA).*

others. Where possible, these libraries exploit the specialized hardware components of AMD GPUs, including AMD Matrix Cores.

Intel offers the oneAPI DL Framework Developer Toolkit (DLFD Kit)—a suite of libraries for building and optimizing DL applications targeting Intel CPUs, GPUs, or FPGAs. The DLFD Kit includes the oneAPI DNN Library (oneDNN) and the oneAPI Collective Communications Library (oneCCL). oneDNN enables developers to create fast DNNs using performance-optimized building blocks and improves programmer productivity by exposing the same API across deployment targets. oneCCL enables developers to train larger and deeper DNN models more quickly using optimized communication patterns to distribute training across multiple nodes. It is implemented using lower-level communication middleware, including the Message Passing Interface (MPI), to support several different distributed-system interconnects, including InfiniBand and Ethernet. The DLFD Kit is part of Intel's oneAPI initiative—an open, cross-industry, standards-based, unified, multiarchitecture, multivendor programming model that delivers a common developer experience across accelerator architectures [32].

4.3 DL FRAMEWORKS

Whereas these library collections provide low-level control for highly customized DL application development, processor manufacturers also provide higher-level frameworks that simplify common DL tasks, including DNN model development, training, and inference. For example, the NVIDIA Train, Adapt, and Optimize (TAO) Toolkit implements transfer learning—a training approach in which features learned by an existing model are extracted or transferred to a new model. Transfer learning is particularly useful when training data for a new application domain is scarce or insufficient but a large pool of data exists for a similar domain. For example, developers can refine an existing object detection model trained with electro-optical (EO) imagery using limited quantities of infrared (IR) training data to detect objects of interest in IR imagery. NVIDIA TAO hides the underlying complexity of transfer learning by exploiting NVIDIA's pretrained DNN models, enabling practitioners to fine-tune these models for new domains with significantly less training data compared to training a DNN model from scratch.

The NVIDIA DL GPU Training System (DIGITS) is another high-level framework that hides the underlying complexity of DNN model development and training, specifically for computer vision tasks, such as image classification, object detection, and image segmentation. DIGITS streamlines data management, performance monitoring, and results visualization, and even scales training across multi-GPU systems in an interactive, browser-based graphical user interface (GUI). With DIGITS, practitioners focus on designing and training effective DNNs rather than programming and debugging.

The Intel oneAPI AI Analytics Toolkit (AI Kit) similarly streamlines DL tasks for applications running on Intel processors. AI Kit components are implemented using oneAPI libraries, including oneDNN, to exploit low-level hardware features for optimal performance. Similar to NVIDIA TAO, Intel AI Kit provides access to pretrained models that have been optimized for Intel processors, and, once again, enables practitioners to focus on designing and training effective DNNs (here using a scriptable Python interface) rather than low-level programming and debugging.

Efficient DNN model training and DL application development workflows are critical to any DL solution's long-term performance and maintainability. NVIDIA TAO, NVIDIA Digits, and Intel AI Kit provide hardware-optimized components and streamlined workflows for model training and application development. However, DL inference performance is as important, perhaps even more important, particularly for deployments targeting autonomous mobile platforms. In these scenarios, split-second decisions based not only on accurate, correct, and robust DNN predictions but also fast predictions can mean the difference between mission success and mission failure.

Recall that during both training and inference, DNNs ingest data at their input layers, multiply these inputs by synaptic weights in their hidden layers, and finally output predictions. Whereas DNN weights are updated to reflect and improve prediction accuracy during training, these weights are not adjusted during inference and are instead fixed at the time of deployment. In this case, the simplest inference methods simply execute the DNN model in framework but disable synaptic weight updates.

This approach is far from optimal, however, particularly for real-time, latency-sensitive production deployments on small, inexpensive autonomous platforms. In this context, approaches in which DNN computations are offloaded to the cloud attempt to address both latency and energy consumption—cloud-based processing potentially provides sufficient computation and storage resources for DNN operations, anytime and anywhere [33]. These approaches trade onboard processing latency and energy requirements for latency in data transfer and communications bandwidth; however, potentially significant amounts of data must be transferred to the cloud over wireless networks, typically over long distances.

To reduce total latency, data transfer latency must also be reduced. Computation and storage resources could be deployed at the edge of mobile network, as in so-called edge clouds (e.g., cloud-based resources integrated in network base stations) [34]. Resources in edge clouds are still limited, but model-parallel inference suggests partial offloading as a viable alternative. In this case, mobile devices process subsets of DNN layers using a combination of onboard and edge-cloud resources [35].

Even so, an autonomous platform's mobility will itself impose challenges for continuity of service in any cloud-based processing infrastructure, edge-based or otherwise [36]. For example, frequent connection failures seriously affect the edge devices' quality of service (QoS); therefore, mobile scenarios require stable partition offloading schemes to ensure service continuity [37], which is itself a difficult problem and an area of active research.

Ideally, real-time, latency-sensitive production deployments on autonomous platforms would execute standalone, optimized DNN inference. Toward this end, processor manufacturers provide tools to minimize resource demands and maximize inference performance in these scenarios.

NVIDIA TensorRT is an SDK for high-performance DL inference that includes an inference optimizer and runtime designed to optimize trained models for the highest throughput and lowest latency while maintaining prediction accuracy. Optimizations include model quantization, layer and tensor fusion, optimized kernel selection, and multistream execution, among others. These optimizations improve latency, throughput, and runtime efficiency across NVIDIA GPU architectures, application-specific SDKs, and DL problem domains.

TensorRT is part of NVIDIA's DL Inference Platform, which combines the specialized hardware features of its GPU architectures with an optimized end-to-end DL software stack for easy model deployment across application domains. The inference platform also includes NVIDIA Triton Inference Server—an open-source software platform designed to simplify production DNN model deployment. Using NVIDIA Triton Inference Server, practitioners can deploy models trained using popular DL frameworks from local storage or the cloud to any CPU- or GPU-based infrastructure, including autonomous mobile platforms.

Similarly, Intel's distribution of the OpenVINO toolkit provides a comprehensive, open-source solution for optimizing and deploying DL inference across domains, including computer vision. Using OpenVINO, practitioners can again deploy models trained with popular frameworks that have been optimized to reduce resource demands and increase efficiency across Intel CPUs, GPUs, and computer vision accelerators. DNN models optimized with OpenVINO thus maximize inference performance for deployment targets ranging from tablets, smartphones, and embedded processors at the edge to desktop PCs, high-performance workstations, and enterprise servers in large data centers and the cloud.

Other DL frameworks also use hardware-accelerated libraries and APIs to deliver scalable, high-performance DNN model development, training, and deployment. For example, TensorFlow is an end-to-end, open-source ML platform created by Google and popular among DL practitioners, especially for DNN training. A flexible collection of tools and libraries, as well as extensive community resources, support model development, training, and deployment across hardware targets, including Google TPU and at-the-edge or embedded processors. In fact, TensorFlow includes TensorFlow Lite—a library supporting model deployment on mobile platforms and edge devices. The TensorFlow API targets Python but also provides limited support for APIs in C/C++, Java, and several other languages. Only the Python API is guaranteed to be stable, however [38]. Beyond extensive language support, a wide range of libraries, extensions, models, datasets, and tools that integrate with or are built on top of TensorFlow support and accelerate common DL tasks and workflows.

PyTorch is also an open-source ML framework supporting DL application design, training, and deployment. Created by Facebook's AI Research Lab, PyTorch is focused largely on computer vision and NLP tasks. Similar to TensorFlow, the primary PyTorch API is Python (with limited support for C++), and several other libraries and tools integrate with PyTorch as well. Although PyTorch has experimental (beta) support for mobile devices, it is optimized for cloud-computing platforms; therefore, some features may not be available on embedded or mobile platforms.

Similar to TensorFlow and PyTorch, MXNet is an open-source DL framework, sponsored by The Apache Software Foundation, that supports both fast prototyping and production deployment. The MXNet Python API called Gluon enables practitioners to switch between imperative mode for dynamic, flexible execution; symbolic mode for fast, optimized execution; and model deployment using different language bindings, including C++. MXNet also enables scalable, distributed training and performance optimization with multi-CPU or multi-GPU support. Integration with Python, as well as support for other languages including C++ and Java, enables a smooth transition from Python-based training to optimized deployment for production. Additionally, a collection of libraries and tools extends MXNet to enable applications in computer vision, NLP, and several other domains.

4.4 SOFTWARE CONSIDERATIONS

The DL frameworks provided by hardware manufactures, third-party software vendors, and the open-source community build directly on low-level, platform-specific libraries and APIs to enable DL application development. These frameworks offer building blocks for designing, training, and deploying models, typically through a high-level programming interface or GUIs and other user-friendly interaction modalities.

Each of these frameworks provides functionality necessary to develop an end-to-end DL system. Differences in support for the various underlying processor architectures, including degree of optimization for any particular architecture, suggest that vendor-specific frameworks provide better performance for DNN training and inference operations. However, as with the hardware architectures themselves, other considerations, including actual DL functionality, cost, licensing, open- vs. closed-source implementation, and level of vendor or community support, may become factors as well.

The specific software libraries and frameworks highlighted in this report represent only a small subset of the currently available DL ecosystems. Interested readers are encouraged to consult the many available resources to learn more about these and other DL libraries, APIs, SDKs, and frameworks.

SECTION 05 APPLICATIONS

Developments in DL hardware and software are exceptionally fast paced, accelerated not only by major processor manufacturers but also by DL projects across academia, industry, and the wider open-source community. The array of algorithms, libraries, APIs, SDKs, and frameworks is extensive and growing rapidly, and the domains in which DL finds successful application is equally extensive and growing at least as rapidly. Perhaps among these, nowhere is DL more revolutionary than for autonomous platforms.

5.1 AUTONOMOUS SYSTEMS

In his 2017 report, "Artificial Intelligence and Autonomy: Opportunities and Challenges," Andrew Ilachinski characterizes an autonomous system as:

> [A] system that can independently compose and select among alternative courses of action to accomplish goals based on its knowledge and understanding of the world; of itself; and of the local, dynamic context. Unlike automated systems, autonomous systems must be able to respond to situations that are not pre-programmed or anticipated prior to their deployment. In short, autonomous systems are inherently, and irreducibly, artificially intelligent robots [39].

Of particular importance in this characterization is a system's ability to respond to situations that are not preprogrammed. Recall that, historically, AI systems require explicitly programmed algorithms to handle each anticipated situation and response, which necessarily limits the scope of potential scenarios to make the explicit logic handling each scenario tractable. In contrast, DL approaches avoid explicit programming and instead expose an AI system to a wide variety of scenarios with known correct actions during training. The system learns correct behaviors (even for scenarios not explicitly encountered in training) and performs appropriately during deployment.

5.2 CONVOLUTIONAL NEURAL NETWORKS

With modern DL techniques, an autonomous system can learn and then execute correct responses based on visual inputs. Recall that massively parallel computing architectures (and modern GPUs, in particular) have transformed AI/ML with DNNs, leading to significant advances in supervised representation learning tasks. For example, in object detection, machine accuracy now rivals human capabilities [5]. Recognition accuracy is accomplished by augmenting traditional DNNs operating on image data with convolutional layers that learn complex visual features (i.e., with CNNs).

CNNs capture patterns in multidimensional spaces quite efficiently, making CNNs especially well suited for image-based data (though they are used to process other types of data as well). Specifically, convolutional layers in CNNs apply convolution kernels of a fixed size to each pixel in the input layer. A convolution kernel is a filter or matrix that encodes the weights applied to a pixel's nearest neighbors. Each filter has different values and extracts different features from the input image. These weights are learned by the network during training. The output of each convolutional layer is a set of images, so-called convolutional images or feature maps, to which the kernel has been applied. The number and size of the feature maps vary according to the kernel size and network topology, but together the convolutional layers detect a hierarchy of visual patterns. For example, lower layers produce feature maps for vertical and horizontal edges, corners, and other simple patterns, while deeper layers detect more complex patterns, such as grids, circles, and other geometric shapes. Layer by layer, the network detects complicated objects, such as cars, houses, trees, and people, by building complex representations from simpler features.

Most CNNs use pooling layers to gradually reduce the size of their feature maps while retaining the most important features. For example, max-pooling retains the maximum value in a patch of pixels. Using a pooling layer of size 2, max-pooling extracts 2×2-pixel patches from the feature maps of the preceding layer and retains the highest value, as illustrated in Figure 5-1. This operation halves the size of the maps but retains the most relevant features. Pooling layers enable CNNs to generalize their capabilities and reduce sensitivity to the location of objects within images.

Ultimately, this process is applied repeatedly, layer by layer, until feature maps comprise a single pixel. The collection of single-pixel outputs is encoded as a one-dimensional matrix that represents, numerically, the prominent image features and serves as input to a fully connected DNN that performs the target task (e.g., image classification or object detection).

As illustrated in Figure 5-2, modern CNNs achieve superhuman performance in image classification [18]. With high accuracy in common computer vision tasks, CNNs are now powering the next generation of autonomous mobile platforms. For example, Bojarski et al. [40] demonstrate a CNN controlling a car on various ground surfaces. Likewise, TrailNet enables MAVs to navigate dense forest trails [41], demonstrating autonomous flights up to 1 km.

These and other autonomous systems require accurate, robust, and high-performance computer

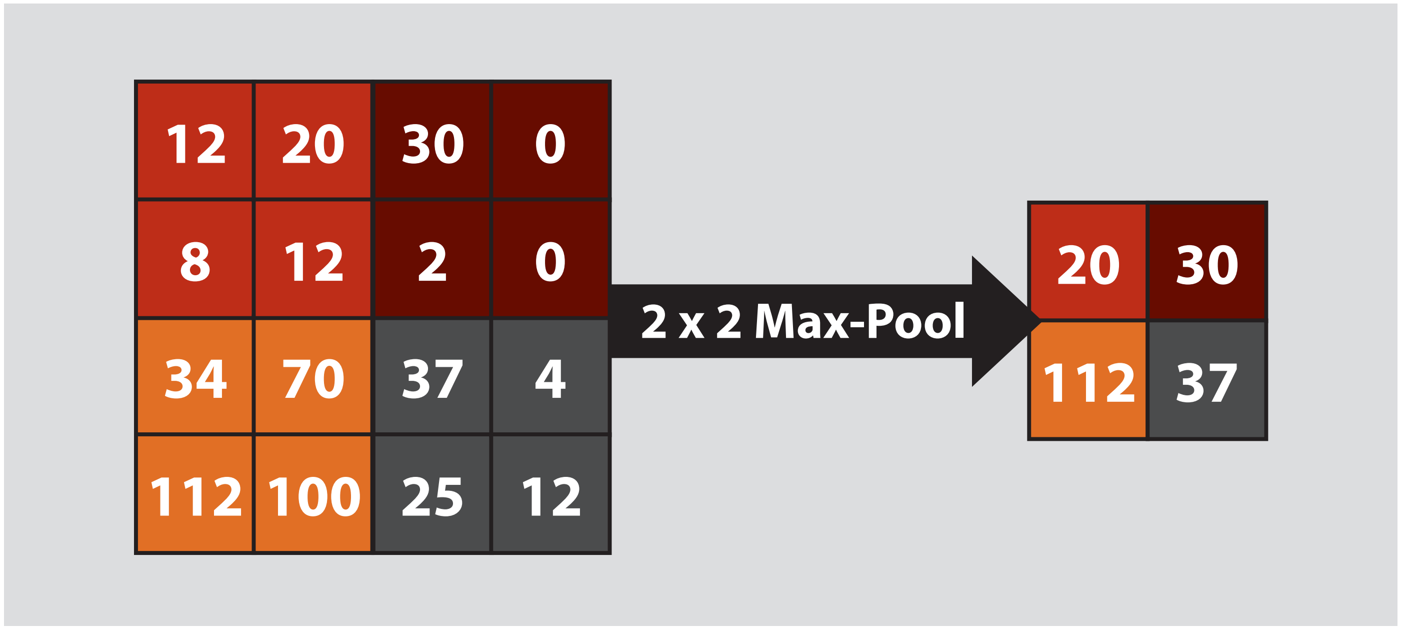


Figure 5-1: Pooling Layers in a CNN *(Source: SURVICE Engineering Company).*

vision techniques that support a broad range of tasks, including autonomous operation. An example autonomous platform architecture is depicted in Figure 5-3. Here, machines sense and perceive their environment with computer vision to extract meaningful information from the real world, ultimately classifying, detecting, locating, or tracking objects within that environment [42]. When combined with hardware and software components for decision and action, these technologies realize a fully autonomous system—one that achieves a set of goals in a changing environment by gathering information about its environment without human control or intervention [43].

5.3 MODERN COMPUTER VISION

As outlined in Figure 5-4, core computer vision techniques for machine perception include

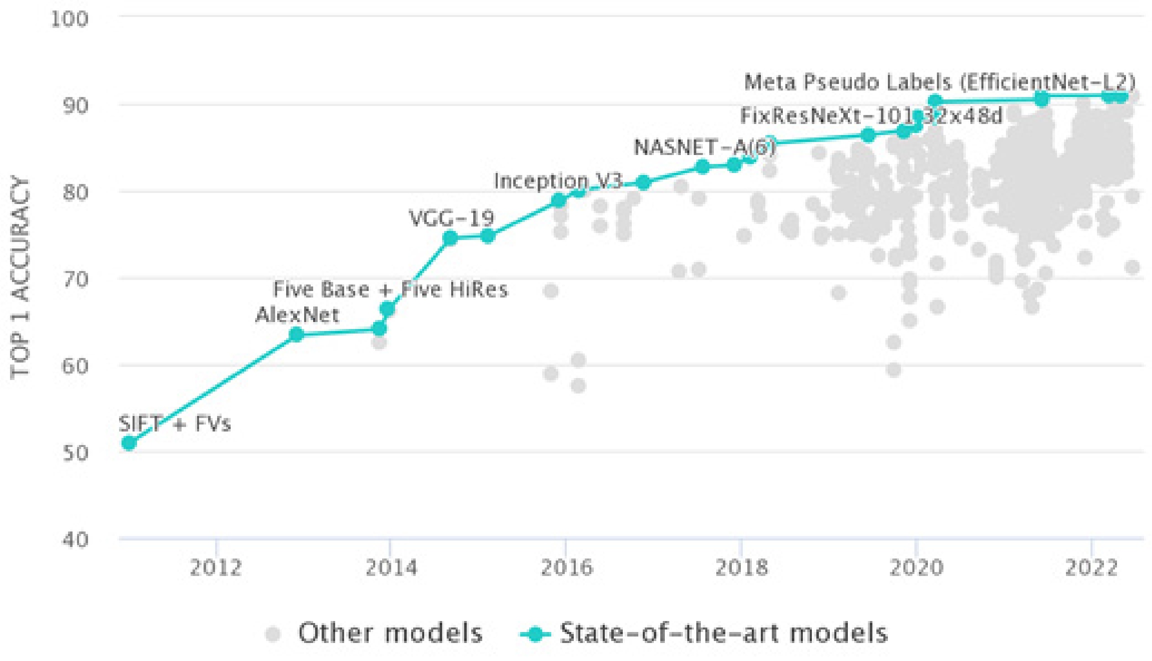


Figure 5-2: Image Classification on ImageNet *(Source: SURVICE Engineering Company).*

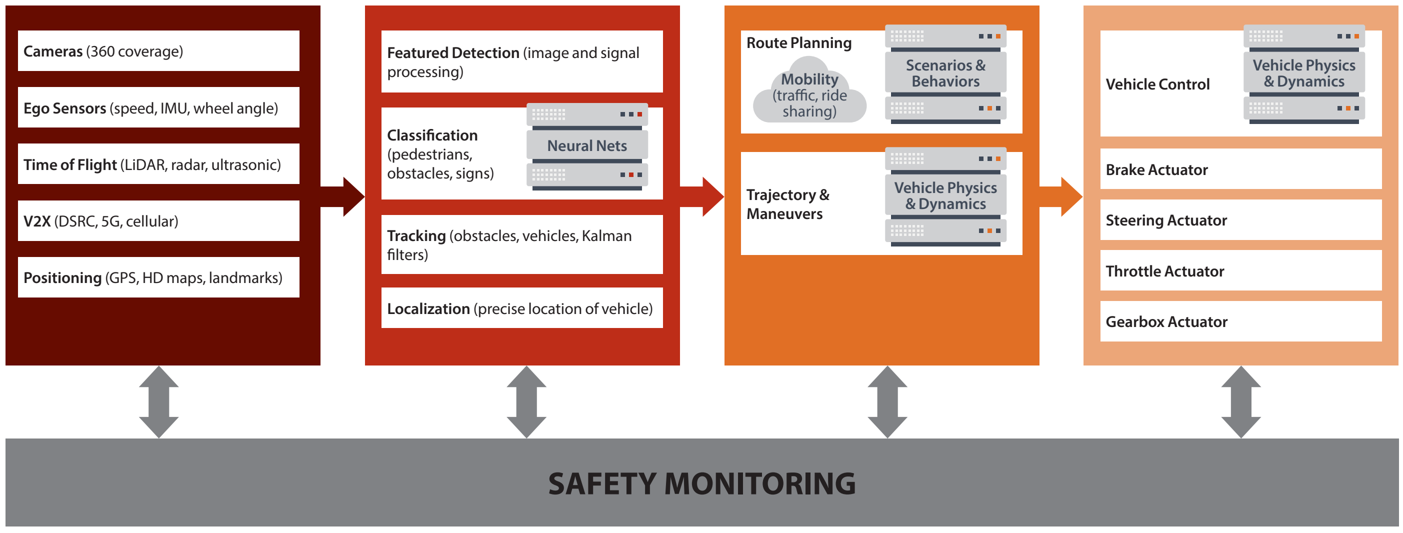

Cameras (360 coverage)
Ego Sensors (speed, IMU, wheel angle)
Time of Flight (LiDAR, radar, ultrasonic)
V2X (DSRC, 5G, cellular)
Positioning (GPS, HD maps, landmarks)
Featured Detection (image and signal processing)
Classification (pedestrians, obstacles, signs)
Neural Nets
Tracking (obstacles, vehicles, Kalman filters)
Localization (precise location of vehicle)
Route Planning
Mobility (traffic, ride sharing)
Scenarios & Behaviors
Trajectory & Maneuvers
Vehicle Physics & Dynamics
Vehicle Control
Vehicle Physics & Dynamics
Brake Actuator
Steering Actuator
Throttle Actuator
Gearbox Actuator
SAFETY MONITORING

Figure 5-3: Example Autonomous System Architecture *(Source: BlackBerry QNX).*

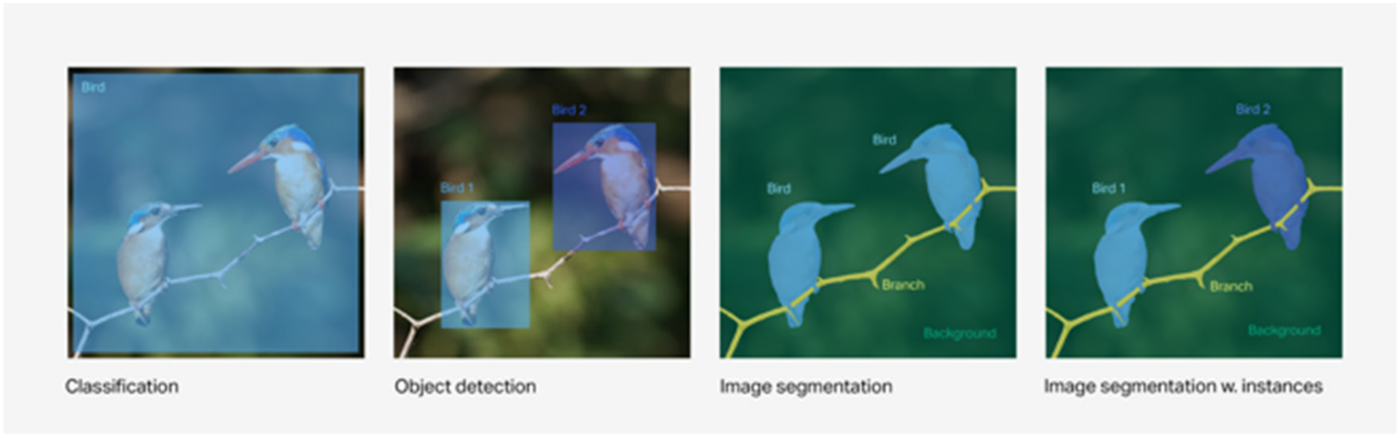


Figure 5-4: Computer Vision Techniques for Machine Perception (From Left to Right: Classification, Object Detection, Semantic Segmentation, and Instance Segmentation) *(Source: SURVICE Engineering Company).*

image classification, object detection, and image segmentation. Image classification determines or classifies objects in an image or video frame and predicts an image label. Classification models are trained using large datasets comprising thousands of ground-truth images and object labels relevant to the anticipated deployment scenarios. As with all supervised DL techniques, prediction accuracy depends in part on the training data used—increasingly more diverse, training data tends to result in higher-accuracy predictions.

Object detection extends classification not only to identify but also to localize objects within an image or video frame. Detection models typically output rectangular bounding boxes around detected objects that indicate or bound their locations within an image. Similar to image classifiers, object detectors are trained to identify and locate objects relevant to the anticipated deployment scenarios; moreover, prediction accuracy depends, in part, on the size and diversity of the training dataset.

Finally, image segmentation extends classification to locate objects precisely by assigning a label to every pixel in an image or video frame—segmentation implements per-pixel classification. In semantic segmentation, pixels with the same label compose the same type of object or share similar characteristics, such as color, texture, or material. Here, multiple objects of the same class make up a single semantic entity. In contrast, instance segmentation treats multiple objects of the same class as distinct instances (e.g., individual cars on a busy roadway) and each object is assigned a unique instance label. Panoptic segmentation combines the concepts of both semantic and instance segmentation by assigning two labels to every pixel: a semantic label and an instance identifier. Pixels thus belong to the same class and instance (i.e., the same object) only when their respective semantic and instance labels agree.

Segmentation models are the most widely applicable and versatile models because they provide the highest information content and they are common in applications that require high precision, such as medical imaging, remote sensing, or autonomous navigation. However, they are also the most expensive to develop, train, and execute because they require ground-truth training data with per-pixel labels and generating these datasets is a laborious and often tedious process. Although flexible, segmentations models may be unnecessarily complex or expensive for machine perception tasks in which simpler image classification or object detection models will suffice or for application domains that do not necessarily require precise, per-pixel labels.

5.4 EXAMPLE DoD USE CASES

Together with the commodity hardware and software components accelerating DL, modern computer vision techniques provide the basis for autonomous platforms in several applications across the U.S. Department of Defense (DoD). For example, the U.S. Navy and Marine Corps Small Tactical UAS Program Office (PMA-263) envisions rapid-deploy autonomous systems that are modular, open-architecture aerial platforms equipped with advanced sensors and payloads to resupply Warfighters on the front line. The PMA-263 Tactical Resupply Unmanned Aircraft System (TRUAS) effort seeks a UAS capable of transporting at least 60 lb of cargo in various configurations commonly found in U.S. Marine resupply operations [44]. Vision-based autonomous navigation powered by DL will enable small, inexpensive aerial platforms, such as SURVICE Engineering's TRV-150 Tactical Resupply UAS depicted in Figure 5-5, to support assured logistics resupply, even in GPS-denied regions, to satisfy the TRUAS program requirements.

Figure 5-5: SURVICE Engineering's TRV-150 Tactical Resupply UAS *(Source: U.S. Navy).*

Similarly, the Artificial Intelligence for Maneuver and Mobility Essential Research Program endeavors to reduce soldier distractions on the battlefield through the integration of autonomous systems in U.S. Army vehicles, including the construction of a robotic combat vehicle that operates independently of the main vehicle. The recent advances in commodity DL technologies enable narrow AI, or the ability to complete very specific tasks consistently, which is a first step necessary to realize autonomous teammates for soldiers. Continued development of these technologies will enable future combat vehicles that fully sense and perceive their environment and thereby realize an autonomous system that is able to analyze complex, adversarial environments and develop possible courses of action [45].

These technologies also serve as proof of concept for applications at the local, state, and federal levels. For example, vision-based autonomous navigation will allow law enforcement to track fleeing individuals through dense urban environments. Likewise, autonomous drones will track and capture unauthorized UASs using similar DL methods, while, prior to infiltration by manned forces, intelligent MAVs will navigate and map dense urban environments, potentially even building interiors.

5.5 APPLICATION CONSIDERATIONS

The size and complexity of practical DNNs is growing rapidly as researchers seek increasingly higher levels of accuracy in computer vision tasks. Large and complex DNNs require significant resources (i.e., computation, storage, and energy) to repeatedly execute inference operations. Small, inexpensive autonomous platforms are inherently resource limited; therefore machine-perception algorithms with lower computational cost (i.e., smaller, less complex DNNs) typically perform best on such platforms. For example, practitioners might consider solutions based on image classification or object detection for real-time systems operating at the tactical edge, while solutions based on semantic, instance, or panoptic segmentation, which enable potentially more detailed analyses, are a better match for systems operating in resource-rich environments.

As with the underlying hardware and software components, DNN-based computer vision techniques provide certain advantages and suffer from certain disadvantages; therefore, matching these techniques both to the problem at hand and to the target platform constraints is a critical step in planning an end-to-end DL system. Here, testing is the only sure-fire method for ensuring an algorithm satisfies application constraints—first, in solving the problem, and second, in executing efficiently using available resources. Practitioners should once again conduct various tests to determine the accuracy, performance, and efficiency requirements necessary to satisfy application constraints. These parameters will determine the characteristics of the machine vision techniques appropriate to the task at hand.

Modern computer vision techniques provide a foundation for a wide range of tracking and navigation applications that are otherwise out of reach without modern, high-performance DL. These techniques deliver best-in-class performance and accuracy for the low-level visual perception tasks that drive higher-level applications of autonomy.

The specific techniques highlighted in this report represent only a small subset of the DL technologies enabling autonomy on small, inexpensive platforms, however. Interested readers are encouraged to consult the many available resources to learn more about these and other applications of DL in autonomy.

SECTION 06 CHALLENGES

Despite success across several application domains, DL is not without challenges. For example, a DNN's effectiveness is, in part, dependent on both the quantity and quality of the data with which it is trained. This data shapes a model's ability to learn critical features and, as a result, must be carefully and deliberately curated for DL applications.

Typical DNNs require sufficient data (sometimes more than 10 million samples) to not only identify features on their own but to do so reliably. In his paper "Deep Learning: A Critical Appraisal" [46], Gary Marcus notes that many experts consider humans to be far more efficient in learning complex rules than DL systems [47–49]. He writes, "Deep learning currently lacks a mechanism for learning abstractions through explicit, verbal definition, and works best when there are thousands, millions or even billions of training examples…In problems where data are limited, deep learning often is not an ideal solution."

At the same time, training data must provide sufficient variation in the critical features to prevent overfitting. Recall that overfitting is the problem in which a model effectively memorizes features in training samples but is unable to generalize these features to new, previously unseen, data (i.e., prediction works well for the training set but not for samples gathered during inference in a real-world environment). However, data quality is typically a task-specific measure; therefore, curating high-quality data often requires domain-specific knowledge.

In contrast, sources of erroneous data are common: errors in data collection; erroneous, irrelevant, or incomplete measurements; incorrect or irrelevant content; or even statistical outliers and duplicate samples. DL algorithms are similarly vulnerable to adversarial samples—inputs crafted by adversaries with the intent of causing prediction failure [50, 51]. Often, erroneous or adversarial samples are easily detected and ignored by human learners, but DL models are highly sensitive to the training data they ingest, as erroneous or adversarial samples can induce potentially catastrophic errors in DNN prediction.

DL practitioners also encounter a lack of transparency, or the so-called black box problem, where even DL experts do not yet fully understand exactly why DNNs make certain predictions. Whereas predictions made by rule-based software can be traced to previous decision blocks, DL models operate differently, effectively sifting through millions of samples to discover patterns and correlations among them—relationships that often remain hidden, even from human experts. Current DL systems have millions or even billions of parameters identifiable to model developers not in terms of well-known, well-understood software development constructs, such as structured programming control blocks, variables, and so forth, but only in terms of their location within incredibly complex networks of artificial neurons and synaptic weights [46].

At the same time, lack of transparency complicates practitioners' understanding of why DNNs fail. Accurate, correct, and robust predictions are as important as prediction performance and even more important in mission-critical scenarios, where split-second decisions based not just on fast predictions but on accurate, correct, and robust predictions can mean the difference between success and failure. In this context, Marcus observes, "The transparency issue, as yet unsolved, is a potential liability when using deep learning for problem domains … in which human users might like to understand how a given system made a given decision" [46].

The specific challenges discussed in this report are just a few of the many-facing contemporary DL techniques, and the DL community is working hard to uncover, understand, and overcome these challenges. Interested readers are encouraged to consult the many available resources to learn about these and other challenges facing modern DL techniques.

07 SUMMARY

DL combines recent developments in high-performance DNNs, massively parallel computing architectures, and hardware-optimized software components with large collections of real-world training data to support autonomy for small, inexpensive platforms.

- **DNNs.** Significant advances in computational performance have ignited a resurgence in AI/ML applications, particularly those based on DL. DL is a class of AI/ML algorithms that solve the representation learning problem by building complex representations from simpler concepts [3]. Modern DL approaches exploit DNNs—multilayered neural networks composed of artificial neurons taking several inputs and producing a single output—to support a diverse range of applications (from computer vision and speech recognition to medical imaging and combat support).
- **Massively Parallel Computing.** Massively parallel computing architectures (modern GPUs, in particular) offer a compelling platform to satisfy the demands of compute intensive applications, including AI/ML applications. These architectures boast tens, hundreds, and thousands of processing cores that provide a massively parallel computational environment at a fraction of the cost of traditional HPC systems. With the mapping of DNNs to modern GPUs [4], DNNs now achieve breakthrough performance in the modern computer vision tasks that form the basis of autonomous mobile platforms.
- **Hardware-Optimized Software.** Massively parallel computing architectures typically dictate that computations be arranged differently than with traditional processors. While modern parallel programming paradigms and compiler technologies address some of the difficulties, rarely are practitioners required to develop DL applications at a low level. Instead, popular DL libraries, APIs, SDKs, and frameworks exploit these architectures effectively, while exposing higher-level DL functionality and workflows that increase productivity; promote flexibility; and enable modern, high-performance, and scalable DL applications.

These state-of-the-art AI/ML hardware and software technologies enable fast, accurate, and robust DL applications and, together, deliver best-in-class performance and accuracy for the low-level tasks that drive higher-level applications of autonomy.

This Page Intentionally Left Blank

REFERENCES

1. Roborace. "Roborace." https://roborace.com/, accessed on 16 July 2022.
2. SURVICE Engineering Company, LLC. "Tactical Resupply Vehicle (TRV)." https://survice.com/who-we-are/focus-areas/tactical-resupply-vehicle-trv, accessed on 16 July 2022.
3. Goodfellow, I., Y. Bengio, and A. Courville. *Deep Learning*. Cambridge: MIT Press, 2016.
4. Krizhevsky, A., I. Sutskever, and G. E. Hinton. "ImageNet Classification With Deep Convolutional Neural Networks." *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.
5. He, K., X. Zhang, S. Ren, and J. Sun. "Delving Deep Into Rectifiers: Surpassing Human Level Performance on ImageNet Classification." arXiv:1502.01852v1, https://arxiv.org/abs/1502.01852, accessed on 23 August 2022.
6. McCulloch, W. S., and W. Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.
7. Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.
8. Kelley, H. J. "Gradient Theory of Optimal Flight Paths." *ARS Journal*, vol. 30, no. 10, pp. 947–954, 1960.
9. Dreyfus, S. "The Numerical Solution of Variational Problems." *Journal of Mathematical Analysis and Applications*, vol. 5, no. 1, pp. 30–45, 1962.
10. Ivakhenko, A., and V. G. Lapa. *Cybernetic Predicting Devices*. New York: CCM Information Corporation, 1965.
11. Fukushima, K. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position." *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1979.
12. LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition." *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.
13. Foote, K. D. "A Brief History of Deep Learning." https://www.dataversity.net/brief-history-deep-learning/, accessed on 16 July 2022.
14. Chawla, V. "Is More Data Always Better for Building Analytics Models?" https://analyticsindiamag.com/is-more-data-always-better-for-building-analytics-models/, accessed on 16 July 2022.
15. Wikipedia. "AlexNet." https://en.wikipedia.org/wiki/AlexNet, accessed on 16 July 2022.
16. Glorot, X. "Deep Sparse Rectifier Neural Networks." *Proceedings of Machine Learning Research*, vol. 15, pp. 315–323, 2011.
17. Wei, J. "AlexNet: The Architecture That Changed CNNs." https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951, accessed on 17 July 2022.
18. Stanford Vision Lab. "ImageNet Large Scale Visual Recognition Challenge." https://www.image-net.org/challenges/LSVRC/, accessed on 17 July 2022.
19. DeepMind. "AlphaGo." https://www.deepmind.com/research/highlighted-research/alphago, accessed on 16 July 2022.
20. Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. "Natural Language Processing (Almost) From Scratch." *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.
21. Otter, D. W., J. R. Medina, and J. K. Kalita. "A Survey of the Usages of Deep Learning for Natural Language Processing." *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2020.
22. Litjens, G., T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sanchez. "A Survey on Deep Learning in Medical Image Analysis." *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.
23. Jagannath, A., J. Jagannath, and T. Melodia. "Redefining Wireless Communication for 6G: Signal Processing Meets Deep Learning." arXiv:2004.10715v5, https://arxiv.org/abs/2004.10715, accessed on 23 August 2022.
24. Polydoros, A. S., and L. Nalpantidis. "Survey of Model-Based Reinforcement Learning: Applications on Robotics." *Journal of Intelligent and Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.
25. Pouyanfar, S., S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar. "A Survey on Deep Learning: Algorithms, Techniques, and Applications." *ACM Computing Surveys*, vol. 51, no. 5, 2019.
26. Dong, S., P. Wang, and K. Abbas. "A Survey on Deep Learning and its Applications." *Computer Science Review*, vol. 40, 2019.

REFERENCES, *continued*

27. Intel Corporation. "Intel Movidius Myriad X VPU Product Brief." https://www.intel.com/content/www/us/en/products/docs/processors/movidius-vpu/myriad-x-product-brief.html, accessed on 17 July 2022.

28. Teledyne Flir, LLC. "Firefly DL|Teledyne Flir." https://www.flir.com/products/firefly-dl/?vertical=machine%20vision&segment=iis, accessed on 17 July 2022.

29. Whang, C., Q. Yu, X. Li, Y. Xie, and X. Zhou. "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 6, pp. 513–517, 2016.

30. Boyd, C. "Data-Parallel Computing." *ACM Queue*, March/April 2008.

31. Fatahalian, K., and M. Houston. "GPUs: A Closer Look." *ACM Queue*, March/April 2008.

32. oneAPI. "oneAPI Programming Model|oneAPI." https://www.oneapi.io/, accessed on 16 July 2022.

33. Alameddine, H. A., S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi. "Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing." *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3,pp. 668–682, 2019.

34. Hu, Y.-C., M. Patel, D. Sabella, N. Sprecher, and V. Young. "Mobile Edge Computing: A Key Technology Towards 5G." *ETSI White Paper*, no. 11, pp. 1–16, 2015.

35. Tian, X., J. Zhu, T. Xu, and Y. Li. "Mobility-Included DNN Partition Offloading From Mobile Devices to Edge Clouds." *Sensors*, vol. 21, no. 1, pp. 1–16, 2021.

36. Shaikh, A., and M. J. Kaur. "Comprehensive Survey of Massive MIMO for 5G Communications." *Proceedings of Advances in Science and Engineering Technology International Conference*, pp. 1–5, 2019.

37. Nadembega, A., A. S. Hafid, and R. Brisebois. "Mobility Prediction Model-Based Service Migration Procedure for Follow Me Cloud to Support QoS and QoE." *Proceedings of IEEE International Conference on Communications*, 2016.

38. Google, LLC. "API Documentation|TensorFlow Core v2.9.0." https://www.tensorflow.org/api_docs, accessed on 16 July 2022.

39. Ilachinski, A. "Artificial Intelligence and Autonomy: Opportunities and Challenges." DIS 2017-U-016388, Center for Naval Analyses, Arlington, VA, October 2017.

40. Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. "End to End Learning for Self-Driving Cars." arXiv:1604.07316v1, https://arxiv.org/abs/1604.07316, accessed on 23 August 2022.

41. Smolyanskiy, N., A. Kamenev, J. Smith, and S. Birchfield. "Toward Low-Flying Autonomous MAV Trail Navigation Using Deep Neural Networks." arXiv:1705.02550v3, https://arxiv.org/abs/1705.02550, accessed on 23 August 2022.

42. Brown, R. "Role of Computer Vision in AI for Developing Robotics, Drones and Self Driving Cars." https://becominghuman.ai/role-of-computer-vision-in-ai-for-developing-robotics-drones-self-driving-cars-9c92b89d57c, accessed on 16 July 2022.

43. BlackBerry Limited. "Autonomous Systems|Ultimate Guides|BlackBerry QNX." https://blackberry.qnx.com/en/ultimate-guides/autonomous-systems, accessed on 16 July 2022.

44. NAVAIR News. "Navy UAS Demo Displays Potential for Future Cargo Resupply." https://www.navair.navy.mil/news/Navy-UAS-demo-displays-potential-future-cargo-resupply/Tue-11092021-0729, accessed on 16 July 2022.

45. U.S. Army. "Army Researchers Augment Combat Vehicles With AI." https://www.army.mil/article/236733/army_researchers_augment_combat_vehicles_with_ai, accessed on 16 July 2022.

46. Marcus, G. "Deep Learning: A Critical Appraisal." arXiv:1801.00631v1, https://arxiv.org/abs/1801.00631, accessed on 23 August 2022.

47. Lake, B. M., R. Salakhutdinov, and J. B. Tenenbaum. "Human-Level Concept Learning Through Probabilistic Program Induction." *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

48. Lake, B. M., T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. "Building Machines That Learn and Think Like People." *Behavioral and Brain Sciences*, vol. 40, 2017.

49. Marcus, G. F., S. Pinker, M. Ullman, M. Hollander, T. J. Rosen, and F. Xu. "Overregularization in Language Acquisition." *Monographs of the Society for Research in Child Development*, vol. 57, no. 4, pp. 1–182, 1992.

50. Goodfellow, I. J., J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples." arXiv:1412.6572v3, https://arxiv.org/abs/1412.6572v3, accessed on 23 August 2022.

51. Yuan, X., P. He, Q. Zhu, and X. Li. "Adversarial Examples: Attacks and Defenses for Deep Learning." *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

BIBLIOGRAPHY

Interested readers are encouraged to consult the many available resources to learn more about the commodity hardware and software technologies highlighted in this state-of-the-art report (SOAR). The scholarly articles, product websites, and news stories cited throughout this SOAR provide many details for a thorough study of these topics, but links are provided for some of the most useful online resources to serve as a springboard for those interested in getting started quickly.

HARDWARE

Advanced Micro Devices, Inc. "Machine Learning (ML)|Deep Learning (DL)|AMD." https://www.amd.com/en/technologies/deep-machine-learning.

Ben-Zvi, N. "A 2022-Ready Deep Learning Hardware Guide." https://towardsdatascience.com/another-deep-learning-hardware-guide-73a4c35d3e86.

Intel Corporation. "Intel Artificial Intelligence (AI) and Deep Learning Solutions." https://www.intel.com/content/www/us/en/artificial-intelligence/overview.html.

LeCun, Y. "Deep Learning Hardware: Past, Present, and Future." IEEE International Solid State Circuits Conference, http://www.cit.ctu.edu.vn/~dtnghi/rech/p2017/lecun-isscc-19.pdf, February 2019.

NVIDIA Corporation. "Deep Learning|NVIDIA Developer," https://developer.nvidia.com/deep-learning.

SOFTWARE

Advanced Micro Devices, Inc. "ROCm|Machine Learning|AMD," https://www.amd.com/en/graphics/servers-solutions-rocm-ml.

Exxact Corporation. "A Breakdown of Deep Learning Frameworks," https://www.exxactcorp.com/blog/Deep-Learning/a-breakdown-of-deep-learning-frameworks.

Intel Corporation. "AI Tools," https://www.intel.com/content/www/us/en/developer/topic-technology/artificial-intelligence/tools.html.

MXNet, https://mxnet.apache.org/versions/1.9.0/.

NVIDIA Corporation. "Deep Learning Software," https://developer.nvidia.com/deep-learning-software.

PyTorch, https://pytorch.org/.

TensorFlow, https://www.tensorflow.org/.

COMMODITY DEEP LEARNING TECHNOLOGIES SUPPORTING AUTONOMY ON SMALL, INEXPENSIVE PLATFORMS

By Christiaan Gribble

CSIAC-BCO-2022-233