General Purpose Computing on Graphics Processing Units for Accelerated Deep Learning in Neural Networks

Conor Helmick

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2022

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial
fulfillment of the requirements for graduation from the
Honors Program of Liberty University.

____
Daniel O'Malley, Ph.D.
Thesis Chair

____
Melesa Poole, Ph.D.
Committee Member

____
David Schweitzer, Ph.D.
Assistant Honors Director

____
Date

Abstract

Graphics processing units (GPUs) contain a significant number of cores relative to central processing units (CPUs), allowing them to handle high levels of parallelization in multithreading. A general-purpose GPU (GPGPU) is a GPU that has its threads and memory repurposed on a software level to leverage the multithreading made possible by the GPU's hardware, and thus is an extremely strong platform for intense computing – there is no hardware difference between GPUs and GPGPUs. Deep learning is one such example of intense computing that is best implemented on a GPGPU, as its hardware structure of a grid of blocks, each containing processing threads, can handle the immense number of necessary calculations in parallel. A convolutional neural network (CNN) created for financial data analysis shows this advantage in the runtime of the training and testing of a neural network.

General Purpose Computing on Graphics Processing Units for Accelerated Deep Learning in Neural Networks

Introduction

According to Amdahl's Law, computing speeds and capabilities are plateauing with respect to the processing cores used in systems (Pandey & Badal, 2019). Due to a power wall, the processing cores normally found in a CPU have been optimized to a point where the development of their architecture is no longer making significant changes to computing speeds – which means that alternatives must be explored to enhance computing and surpass its current limits (Breß et al., 2019). On conventional systems, computing is handled by a central processing unit (CPU); it handles a large portion of computation as it is responsible for a variety of functions that range from calculations to operating system processes to directing other computer parts in how to execute a task. This unit can handle different general computer functions at the same time by splitting them up across multiple processor cores, which the grand majority of functioning and useful computer-driven machines have. It follows, then, that a computer with more CPU cores would effectively be able to compute more all at once than one with fewer cores, and it would be able to split complex tasks into more simple subtasks across the cores (Geer, 2005).

Due to size and power usage, a CPU can only handle so many processors before their power drain outweighs their computing advantage (Calore et al., 2020). Therefore, the problem revisited is that there are not enough cores on a CPU to handle tasks of enough volume or complexity to improve computing. A CPU, though, is not the only structure within the computer that has processor cores constantly processing subtasks. A system's Graphics Processing Unit

(GPU), as suggested by the similar structural label, also takes care of processing. Rather than central or general processing, it handles the processing of computer graphics and displays. This Domain-Specific Architecture, in handling a specific task set, does not require large and demanding cores, solving the problem that limits the number of processing cores that are found on the CPU. However, GPUs can experience coordination overhead, which may cause a slowdown when multiple distinct grids must communicate (Kale et al., 2020). GPUs are capable of having thousands more cores than CPUs, and indeed already do have many more cores (Nvidia Corporation, 2021). If the GPU could be harnessed in a way that takes advantage of its numerous cores to process general computing tasks, then computing and its capabilities would be open to all new possibilities and efficiencies. So, the solution to the current CPU core problem and the limitations described by Amdahl's Law is to develop General Purpose Graphics Processing Units, or GPGPUs. Of course, Amdahl's Law also states that parallelization provides no speedup in the case of serial processes – the limit of speedup for processing is determined by how fast serial processes can be handled (Pandey & Badal, 2019). Therefore, GPGPUs provide speedup by optimizing aspects of processing which can be parallelized to a large scale.

Given the significant difference between running processes on GPGPUs and CPUs, there are uses for a GPGPU that are not available to CPUs. One such application, and perhaps the one with the most potential, is the creation of large neural networks that can handle extremely large data sets efficiently (Fonseca & Cabral, 2017). A GPGPU can achieve this by spreading the functions and calculations for neurons and synapses of a neural network across its different cores, or threads, and running them next to each other. Neural networks are the foundation of deep learning, which allows artificial intelligence to recognize patterns and continually perfect its own algorithm without any human interference (Schmidhuber, 2015).

Processing

A computer has multiple components responsible for power, data storage, and function, each of these being actual hardware for the machine. The component that handles actual computation and general function within the computer is the central processor; while modern machines tend to have multiple processors, the CPU is the processor that is in charge of computer function and communication between other computer components (Ambaka, 2021).

Processing units use their unique architecture to maximize processing efficiency and their ability to process multiple tasks. The core(s) on processing units can be used to split processes into threads – smaller sub-parts of a process that can be run next to each other rather than sequentially. Building on that concept, processors further increase their efficiency by multithreading, which splits processes into threads and then runs those across the processing cores (Gao et al., 2020). Some applications have grown to be so intense on volume of calculations and/or complexity that multithreading is practically essential for them to run, and multithreading capability must be considered in evaluating the overall effectiveness of any processing component (Tino et al., 2020). Threading takes place when a thread or threads of a process get assigned to individual processing cores – each core taking up to two threads at a time on a CPU (Intel Corporation, 2019). Each core can handle the threads designated to it while the other cores do the same, integrating their threads into one process. A GPU threads in a similar manner, although the way data travels and is split is slightly different. All of the threads in a warp, which is a group of threads on the processor, are responsible for the same instruction type, but each thread can still handle distinct data (Gao et al., 2020). Warps of threads make up blocks on a GPU, and the blocks make up the grid of the GPU (Nvidia Corporation, 2022). Distributing different instructions with large amounts of data to the blocks on the grid allows many aspects of a process to be split into many different threads to be processed in parallel all at once. Depending on how it is used, each processor has unique advantages and limitations to its use in computing.

CPU

The CPU is perhaps the most critical part of the computer, directing its activities and telling the other parts of the computer what to do. In reality, a modern CPU has multiple cores that each take care of the responsibilities of individual processors; one central processing unit holds multiple processing cores that can handle instructions and normal computation. The Intel CPU with the most cores currently is the Xeon Platinum 8280, which has 28 processing cores (Intel Corporation, 2019). Intel and AMD have also manufactured dual, quad, and octa processors, which physically have more total cores, but this number is simply a result of multiple units being implemented side-by-side, and often at increased heat and power costs (Georgis et al., 2016). Intel architecture is used as a general example of CPU architecture as Intel is the leading manufacturer of this component.

Despite being separate from other processors in a computer, such as the GPU, a CPU can be responsible for scheduling and memory management of other processors being used for a task (Villegas et al., 2019). For example, it can schedule the image processing done by a GPU, as well as allocate memory for it to do so. In fact, in high performance computing and supercomputer architecture, it is practically necessary that GPUs be repurposed as GPGPUs to act as an accelerator for computing, as CPUs are not built for multithreading at the same degree as GPUs (Tian et al., 2021). Even if it is not the component primarily responsible for processing a certain task, it is a head of operations that still must communicate with the other necessary components (Ambaka, 2021). Its processing cores consist of registers, data buses, arithmetic logic units (ALUs), and control units: registers hold data so that other data can be moved through the processor; data buses transfer data in the form of electrical signals through the processor; ALUs handle math operations and comparisons; and control units receive commands and then tell the CPU how to execute them using instructions (Taştan et al., 2020).

Advantages of CPU Processing

The CPU's most noticeable advantages are found in the fact that it is created for general processing – it is made to handle all of the general computing needs and demands of a machine. Since it needs to be ready to handle any request that the system sends it, it is a necessity for the CPU to have a robust design and low latency (Syberfeldt & Ekblom, 2017). Even in cases where another processor is needed to process a function or data, the CPU is required to direct that processor and allocate memory for the process. In fact, GPU and GPGPU processing usually begin with figuring out how to receive data efficiently from memory that it shares with the CPU in either a PCI express bus or RAM (Dong et al., 2021). That being said, perhaps the greatest advantage of CPU computing is its management of memory. In using a much larger local cache than other GPUs, the CPU spends a relatively small amount of its processing time with memory reads and writes to memory structures external to the processor, such as RAM (Syberfeldt & Ekblom, 2017). Memory and data management between a CPU and GPU are typically handled by a PCI-Express bus, but the CPU also has access to RAM that shares memory with the GPU, allowing it to use RAM to send large amounts of data to the GPU or other components that might otherwise get backed up within the PCI-Express bus (Rivera-Alvarado & Torres-Rojas, 2020). Additionally, RAM can have its own clock speed and a dual data rate (DDR), meaning that multiple reads and writes can occur to RAM in one clock cycle of the CPU (Srikar et al., 2020)

A CPU also has a generally higher clock speed than GPUs; as an example, the aforementioned Xeon Platinum 8280 has a clock speed of 2.7 GHz (Intel Corporation, 2019). Higher clock speed, alongside DDR means that the processing cores can process sequential data and tasks much faster, which is greatly beneficial for an architecture that is constantly receiving input from various operating system components (Srikar et al., 2020). With less parallel cores running at the same time than a GPU, a CPU can also maintain a high clock rate without lowering the accuracy of its calculations.

Limitations of CPU Processing

The CPU is designed for general processing, which defines the control of the system and its functions/calculations, as well as scheduling for the computer's resources; for a system to operate, this component is necessary (T. Wang et al., 2019). On the opposite side of this general purpose, then, is the lack of its ability to do exceptionally well at certain, specified tasks. The GPU was designed and implemented for this very reason, as while a CPU technically has the ability to process and render a graphic, a GPU is built in a way that does it much better and faster using much more parallelization in multithreading, which renders videos and images much more effectively (Georgis et al., 2016). The same is true for neural networks – a CPU can support a neural network, but it can only handle one that is so big before the time and power costs render it unable to support the structure of the network or the amount of data that must be processed for it to learn effectively (Kulkarni et al., 2021). While a CPU handles general tasks and memory accesses very well, even with a high clock rate it is unable to handle specific tasks that require a large amount to be done all at the same time.

GPU

The graphics processing unit is still a processor, but, unlike the CPU, it is made for a specialized purpose, making it a domain-specific architecture. For this reason, the cores on a GPU do not need to be nearly as complex as the cores on a CPU, which must be designed with a variety of instructions in consideration (Georgis et al., 2016). A core that is less complex does not necessarily mean that the hardware is completely different – GPU cores still have components such as registers for fast, core-local memory, units for accepting instructions, and a multi-level cache memory system similar to that of multi-core CPU’s (Syberfeldt & Ekblom, 2017). The GPU having less complex cores also means that it gets many more cores than a CPU can have on a single unit.

Advantages of GPU Processing

The number of cores that a GPU can have is higher than the number of cores that is found on any CPU, in fact it can be a few orders of magnitude greater (Gelado & Garland, 2019). While Intel is the leading manufacturer of CPUs, Nvidia is by a large margin the leading manufacturer of GPUs, so Nvidia architecture will be used for architecture examples and application (Peddie & Dow, 2021). The Nvidia GeForce GTX Titan Z GPU has an extremely high number of processing cores, with 5760 CUDA cores (Nvidia Corporation, 2021). The GPU is able to use this extreme number of cores to handle all of the aspects of processing an image all at once at a high speed, whether that be shading, pixel color, or transparency.

Limitations of GPU Processing

The limitations of a GPU stem from what ultimately provides its advantages as well – its intended purpose of only processing display graphics. Since the CPU still essentially runs the computer, operating system management and input/output stream data get to the GPU from the CPU, so processing speeds and capabilities are greatly limited by the need for this interaction (Dong et al., 2021). More specifically, instructions and commands can get backed up when the CPU tries to send them all to the GPU at once – this is the case for instructions or processing that must be done serially. Additionally, since so many cores are used in effective GPU processing, there is often considerable overhead of both power and memory between caches when a large number of threads are created (Vieira et al., 2021). In this way, the CPU is superior to the GPU as in doing its own processing there are less memory accesses and data transfers needed to execute tasks.

Local to the GPU, memory is a problem as well. A series of caches, as well as Dynamic Random-Access Memory (DRAM), is responsible for memory on the GPU during processing, but there is a threshold of available memory that cannot be surpassed during processing; a GPU cannot process more memory than it can hold or move between caches and RAM in the case of shared memory (Syberfeldt & Ekblom, 2017). Due to the organization of a GPU, which is a grid of blocks of threads, a lot of writes to the caches can be redundant or unnecessary, as threads on different blocks do not typically share their most immediately accessible cache (Ibrahim et al., 2020). There are shared memory caches local to blocks, and there is global memory between blocks on the grid as a whole; in some cases, the processing done on one block replicates processing on equivalent input data on another block, and this replication can lower processing optimization when there is no need for two equal data points to both be processed.

GPGPU

A GPGPU is not too different from a GPU, and it is essentially of the exact same hardware structure (B. Hu & Rossbach, 2020). The only difference is that it removes the domain-specific nature of the GPU and instead, by a software implementation, allocates a GPU's resources towards general processing (Kenyon et al., 2019).

Advantages of GPGPU Processing

A GPGPU, in being at the hardware level a GPU, is loaded with processing cores for optimized graphics display and processing. The first researcher-recognized capability provided by this design outside of processing resource-intensive graphics was matrix multiplication, and it remains that this ability is the root of its superiority to other processors in certain implementations (Du et al., 2012). Matrix multiplication, in practical implementation, is defined by the multiplication of one multidimensional array of numerical data by another, which contains a number of individual calculations with polynomial growth as the size and dimensions of the input data grow (Alman & Williams, 2021). A CPU can handle this kind of computation, but it cannot do so as efficiently as a GPGPU, as it has a very limited number of cores to run calculations in parallel on a large scale (Gelado & Garland, 2019). That being said, some CPUs do have vector registers, which can store a vector of data in a single register allowing vector calculations to be made on one vector of data all at once; Intel develops processors with these registers and then defines architectures for those processors that allow for vector instructions, such as Advanced Vector Extensions 3 for Xeon line processors (Amiri & Shahbahrami, 2020).

A GPGPU, on the other hand, can disperse these calculations across thousands of cores to split the work into threads and run a massive amount of input data in parallel. Not only that, but

GPGPUs are now being built with tensor cores – cores that are designed with the purpose of matrix and tensor multiplication in mind (Sioutas et al., 2020). These tensor cores further enhance the ability of a GPGPU to process data for the purpose of deep learning, as tensors can define matrices of higher dimensionality (Yan et al., 2020).

Limitations of GPGPU Processing

GPGPUs share in large part the weaknesses of GPUs, outside of the weakness that they are only built and programmed for graphics processing. An additional weakness to CPUs that they share with normal GPUs, though, is their clock speed. GPUs have a lot of cores that are meant to do many parallel calculations, so it is necessary that the clock speed be slower for parallel threads and processes to be executed accurately (Harakannanavar et al., 2021). For instance, the clock speed of the Nvidia GeForce GTX Titan Z GPU is 705 MHz, which is not even half of the 2.7 GHz speed of the Intel Xeon Platinum 8280 CPU (Nvidia Corporation, 2021). When used for more general-purpose tasks, a high clock speed is beneficial for a considerable increase in the diversity and amount of input to the processing unit, for which a GPU is not inherently built. In other words, central processing involves taking many different processes and calculations that often must be executed serially rather than in parallel, which requires a high clock speed. Nevertheless, the parallel processing power of the GPGPU effectively negates this weakness.

Deep Learning

Deep learning is a derivative of artificial intelligence (AI) that uses a complex system of interconnected nodes, or a neural network, to map inputs to outputs (Schmidhuber, 2015). Deep learning itself refers to the computer or program’s ability to recognize patterns between certain inputs and their outputs simply by being exposed to real-world or theoretical examples. It should be noted that deep learning, in implementation, needs a huge amount of data in order to develop a successful model (Erickson, 2021).

Deep Learning and Machine Learning

Machine learning and deep learning are not two distinct fields; rather, deep learning is derivative of machine learning. This fact can be seen in some of the characteristics of machine learning; it is algorithmic, data driven, has a utility that is rooted in prediction accuracy of an instance, and is designed to work without human interference. Additionally, it shares the key function of being able to learn, which implies that given more data or experience, it can adjust its design or algorithms to better predict future cases (Jakhar & Kaur, 2020).

Their difference, then, is in their software implementations and the structures and algorithms which they use to learn. The neural network structure of any deep learning system will also define a machine learning structure, while any other machine learning algorithms are outside the scope of deep learning, as deep learning is always built on layered neural networks. Deep learning, though, does require much more data than other machine learning methods do in order to learn, so there are scenarios where deep learning is either inefficient or implausible for a problem (Fonseca & Cabral, 2017).

Potential Applications of Deep Learning in Artificial Intelligence

Deep learning has strong applications in cases where there is some data set involved that has material to serve as both input and expected output for training. For example, the medical field has been using deep learning to look at imaging for years, whether it be x-rays, CT-scans, or another image type, and then using the gathered data to detect what even trained surgeons or other equipment cannot identify. While this capability in medical imaging is in no way new, a particularly relevant example of a deep learning machine's ability to evaluate imaging is one developed for radiologists trying to detect COVID-19 (Vaid et al., 2020). The model looks at x-ray imaging of the chest, and then collects numerical data on various aspects of the lungs, such as the contour of a commonly affected area in the lungs, to see if there are any unusual features that would not be found in the lungs of a healthy patient.

The need for a lot of data is not a total negative for deep learning, and it certainly is not a reason to abandon the approach altogether. In the case of big data, there really is no other category of machine learning that compares to deep learning. Big data itself is characterized by massive data sets; these data sets are also ever changing and ever increasing in volume, and that complexity yields itself towards the deep learning model. Deep learning is the key to handling big data and leveraging artificial intelligence for its analysis in a time efficient manner (Fonseca & Cabral, 2017).

Neural Networks

Deep learning is rooted in neural networks, a software schema that has the intent of functioning like a human brain by learning from experiences, which take the form of some set of training data (Asghar et al., 2021). A neural network takes its current state and exposes itself to input data, compares the output it produces with an expected output, and then adjusts itself to better produce an output identical to what is expected in future iterations. It is important to note, though, that if a neural network trains on the same exact data for too long and is allowed to over adjust itself to that data, then it will memorize specifically how to handle only that data set and not be prepared for different or unique data in the future; this concept is called overfitting. Some common methods to prevent overfitting are dropout, which drops random nodes on different iterations of training, and data augmentation, which includes tactics such as removing parts of the input on different iterations (Rice et al., 2020).

Neural networks can be further divided into their own categories and purposes; there is not necessarily one design that applies the best to all deep learning problems. Two of the most general classifications of artificial neural networks are feedforward neural networks, or FNNs, and recurrent neural networks, or RNNs (Schmidhuber, 2015). Feedforward indicates that data is handled by one component and then immediately sent to the next; once one layer finishes with data, it sends that data on through the network and essentially forgets about it. Recurrent, on the other hand, means that data might pass through one layer more than once. This design is useful for when it is desired that the algorithm remembers past inputs and their paths when processing newer input, such as in the case of learning facial recognition (Li & Zhuang, 2020). There is also a type of neural network that is purposed for problems that cannot be solved with a conventionally layered neural network; graph neural networks (GNNs) are neural networks for problems represented with complex, unlayered graphs (Wu et al., 2021).

RNNs can group what would be multiple layers all into one with its recurring nature, meaning that FNNs generally require more processing power to support a neural network running in parallel. However, by keeping layers separate and nonreusable, a feedforward neural network can maintain a level of complexity that must be abandoned by recurrent neural networks to be simplified into cyclical architecture. In terms of processing unit support of the network, an RNN has a structure much better fit for a CPU’s number of processing cores, as less cores are needed (Li & Zhuang, 2020). The RNN needs less cores because less nodes are needed to be supported by threads; the same node can be used for the recurring execution of the same function. Still, a GPGPU is ideal for supporting both, and it is especially effective for the heavy hardware needs of a feedforward neural network.

Neural Network Structure

The structure of a neural network is also the basis of the name of deep learning – while only the inputs and outputs are viewed by one training a deep learning machine, there are often many hidden layers and nodes between that initial input and result(s). The neural network, then, is multiple layers deep of self-adjusting nodes and edges, and the deeper the network is, the more processors it needs to efficiently handle big data. In the case of gigantic data sets, such as data needed to monitor and predict values of stocks, time efficiency is vital to having beneficial AI.

The foundation of a neural network is each of its nodes, which is referred to as a neuron in neural network structure. A layer refers to neurons that line up in proximity from the input neurons or previous. In visual representation of the network, these neurons are grouped in a line with each other to distinguish how deep in the network they are located. In GNNs, though, neurons have interdependencies and edges that cannot be represented logically by layers (Wu et al., 2021). Neurons handle calculations based on the neurons and edges found before them, reading both as numerical data and doing a summation of some kind to send a value out to receiving neurons.

Connecting each of the neurons is a weighted edge, or a synapse. In a feedforward neural network, each neuron has synaptic connections sending their output to following neurons, as well as synaptic connections receiving the output of previous neurons. A fully connected neural network is a configuration where each neuron is connected to every neuron in the layer before it and every neuron in the layer that follows, and it is better than most other configurations in classification tasks (Sahlani et al., 2020). This design is typical of neural networks and allows for more complexity in processing data and interconnecting the meaning of the different inputs, but it is not a requirement for neural networks, nor is it impossible for neurons in nonadjacent layers to have synaptic connections (Schmidhuber, 2015). Another configuration that does not involve full connectivity is a convolutional neural network (CNN). CNNs use a process called convolution to greatly reduce the volume of inputs being processed; this advantage is crucial in image processing when learning by individual pixels (Basha et al., 2020). Convolution takes an input matrix and multiplies a smaller matrix over itself in different locations to create a new matrix that is the size of the smaller matrix. The common factor in all neural network designs is that they use matrix multiplication to process data, which is greatly sped up by parallelization. GPU architecture, then, is ideal for supporting neural networks due to its ability to run the same calculations on thousands of parallel threads at once (Nvidia Corporation, 2021).

Since neurons are defined by calculations and often summations, they rarely see any need for change, as if a calculation method itself is constantly changing, then the network would need to be repetitively retrained and become ineffective. Therefore, the learning component of deep learning happens across the weights of synapses. Synaptic weight is typically defined within a preset range, depending on what activation function, or function that decides whether a neuron fires, is used and how it is leveraged (Parisi et al., 2022). These weights are most often integrated into data passing through neurons through a process of multiplication – with the idea of a final summation happening when all common synapses meet at a neuron. It is clear, then, that a synaptic weight farther from 0 means that the synapse has a greater strength within the network

(Lu et al., 2018). Synaptic weights are often randomized before learning rather than being hard-coded (Lopez-Bernal et al., 2021). Random starting weights keep a neural network from having a symmetrical pattern where every single neuron gets the same exact input weight, and it also keeps every neuron from getting an empty signal like they would if the weights were initialized as zero (Chanda et al., 2021). However, after the neural network has been exposed to significant amounts of data, most of the synapses are adjusted to heavy weights or weights close to zero; what makes a weight heavy varies between activation functions, but generally heavy weights are those that are farther from 0, thus creating a greater impact on the signal moving to the next neuron. The learning process is seen in the continual adjustment of synaptic weights.

Inputs and outputs, no matter what the structural basis is for the neural network, define the beginning and end points of the network. The number of input neurons of a neural network is equal to the number of features of interest in data, while the number of output neurons is equal to the number of results, or classes in classification, that can be produced by the network (Ghiasi-Shirazi, 2018). The foundation of many data processing neural networks is a neuron which is used to activate and produce a signal or output, called a perceptron (Bi & Zhou, 2019). For any explanation or discussion of neural network structure or function, assume a perceptron-based architecture where each neuron either activates or stays off.

Neural Network Function

Neural networks are created in such a way that, by continually being exposed to data and compared to expected outputs, they will eventually adjust themselves enough to provide a solution to a problem. In taking a set of inputs, connecting them with various neurons of activation functions, and then feeding those to an output, a composite of functions is mapped and approximated between inputs and outputs by neural networks (Amerineni, 2020). The functions represented by deep neural networks are often Borel-measurable, meaning that they are continuous for all real numbers within set bounds (Teng & Todo, 2019). By continually adjusting over a cost function, which is based on the error of the network, a neural network gets closer and closer to perfectly predicting outputs (Rice et al., 2020). Regarding the perceptron building blocks of a neural network, a neuron's evaluation of the data passing through its preceding synaptic paths and neurons will determine whether or not it fires, or continues to push an impulse through the network, mimicking the activity of neurons in human brains (Asghar et al., 2021). A good representation of this impulse is turning a wire or bit to the *on* state in a circuit, except depending on the evaluation of values, the state can be represented by values other than 1.

Matrix multiplication is the base calculation of effective neural networks that handle large amounts of data, and also the largest driver behind the utility of GPGPUs in processing neural networks. In using this calculation style, multiple inputs for neurons can be run against multiple synaptic weights all at once, progressing through a feedforward neural network with a polynomial complexity, which is a function of the number of neurons multiplied by the sample inputs provided (Nguyen et al., 2021). GPGPUs can use each block of threads to run the same type of calculation for multiple pieces of data all at once, with each thread handling an individual calculation. This method of parallelizing matrix multiplication makes the GPGPU much stronger than the CPU even when the CPU is leveraging vector registers; the GPGPU's advantage is strengthened further when using tensor cores to handle matrix multiplication (Zachariadis et al., 2020). By leveraging matrix multiplication, neural networks are able to analyze and predict data better and quicker than humans or other machine learning strategies, as they recognize patterns and relationships that cannot be simply observed or deduced by hand (Xu et al., 2020).

Handling Synaptic Weights

Synaptic weight adjustment is the heart of deep learning; while neurons help to recreate the structure of the human brain, synapses attempt to duplicate its function. It is important to note that some weights are also accompanied by biases, which are typically implemented by giving a previous neuron passing data to the current neuron a hard value to affect the sum that it gathers from other inputs. Adding neurons with bias gives a neuron one more value to help it find when to correctly activate, and they can also help the neuron by forcing an activation to happen more or less often (Ozen & Orailoglu, 2019). Neurons with a bias sometimes pass data to following neurons through a synapse of a constant weight so that the value of the bias is not lost or problematically changed by the synapse, but biases are still changed in learning.

Implementing synaptic weights is not an overly difficult task in theory – a designer for the neural network just needs to initialize the values of the weights and determine an algorithm for their correction. Weights for each group of synapses between layers can be stored in arrays; grouping together these weights makes it easy to include them all at once in complex calculations and to see how they affect the neurons into which they feed. The weights, then, are randomized and stored in a matrix which will be edited later to fix the synapses (Lopez-Bernal et al., 2021).

To better understand how synaptic weights are updated for learning following their random initialization, it needs to be seen how they affect the output of individual perceptrons. The summation to determine whether or not the neuron fires is a function of its inputs and input synaptic weights, as shown by Equation 1, where *N* is the number of input nodes, *n* is the value coming from an input node or previous perceptron, *w* is the weight of the synapse paired with its respective node, and *b* is any biases added by additional neurons (Schmidhuber, 2015). If the resulting value meets or exceeds a predetermined threshold, then the neuron fires and sends a value down the synapse(s) leaving itself (Parisi et al., 2022). Matrix multiplication is leveraged to multiply all neuron inputs and weights in a single calculation type, minimizing the number of cores needed to evaluate data and optimizing the potential size of a neural network. This minimization is a result of a GPGPU's ability to run this single calculation type on a block of threads handling multiple neurons, making it a very effective hardware component for supporting a high degree of parallelism (Harakannanavar et al., 2021).

$$\text{neuron}_{\text{current}} = \left(\sum_{i=1}^{N} n_i w_i\right) + b \tag{1}$$

The summation itself provides a value that is checked against an activation function, the most popular of which is the Rectified Linear Unit, or ReLU (Parisi et al., 2022). The ReLU function is relatively easy to understand and implement, as it simply compares the output with 0 and, if the value is greater, it outputs the value as it is. Minimization of error is simple with ReLU as the function's derivative is either 1 or 0, with an undefined derivative when the input is 0 that is usually just hardcoded to be either 1 or 0 – this also provides itself as a solution to vanishing gradient when backpropagation reaches earlier layers (You et al., 2020).

Given the desired complexity of the system or the training approach, a function is chosen to determine how significant the error between actual and expected output is for neurons; these functions are called loss functions when looking at individual neurons and synapses and otherwise called cost functions when looking at the network as a whole. The choosing of a loss function is based on what kind of purpose the neural network is serving, as a regression problem might use one such as the mean squared error loss function, while a classification is better suited with cross entropy (Ozyildirim & Kiran, 2021). While cost/loss functions may vary, they are the most important part of backpropagation – the process in which the neural network looks at its outputs and moves opposite the direction of the feedforward process to correct synaptic weights (Pratama & Kang, 2020).

An easy cost function to visualize is a simple quadratic function of weight, where the absolute minimum is where loss, or output error, is at a minimum. Every time the error is checked, the loss function is run to find where on the curve the loss for a given weight is, and backpropagation takes place using the process of gradient descent (Ozyildirim & Kiran, 2021). Gradient descent is defined by determining the positivity of the tangent line to a point on the loss curve, and then making stepwise progression by changing the weight towards the absolute minimum where the tangent line is zero. The backpropagation starts by adjusting the weights at the back or later layers of the network, and it works its way to the front using gradient descent to make adjustments. This tactic can cause issues though, as a function with multiple minima and plateauing cost values can cause the network to train to a fake value of lowest error, as the gradient is still extremely close to zero at these points (Wilson et al., 2017). The step size is identified as a learning rate, and it is iterated until the weight produces a value with minimum loss, or where the error in output is practically none. Determining step size is crucial, as too large of a learning rate will cause the adjustments to likely overlook the absolute minimum and train the network inaccurately, while too small of a learning rate can cause overfitting or the settling of gradient descent into a local minimum (Ozyildirim & Kiran, 2021). This concept is why deep learning is crucial to understanding topics that were previously too complex or convoluted to understand – one can look at an input or set of inputs and see what ultimately results, but the recognition of patterns that link the two might be too much of a precise task for a human to undertake.

Exploration of AI Implementation

GPGPUs hosting neural networks for deep learning are now evidently one of the strongest achievable forms of machine learning, but there are other machine learning implementations that, while perhaps inferior, provide effective and powerful solutions to modern problems (Georgis et al., 2016). Outside of quantum computing, though, it is difficult to find a more powerful architecture-to-computing pair that will be possible with any sort of improvements to modern hardware alone (Chen, 2020). Leveraging currently available resources such as GPU hardware, then, is the next step in advanced computing.

Common Implementation Methods

In terms of neural networks, implementation can be done with virtually any Turing-complete programming language, or one that can make conditional decisions, do mathematic data manipulations, and manipulate memory, since it is ultimately a composite of mathematic functions (Michaelson, 2020). Implementations using these languages that do not make use of a GPGPU's architecture are hardly fit for the processing of big data, especially should it be used in a field such as business analytics where data is consistently changing, as the GPGPU's capability for multithreading is unmatched among processors (Harakannanavar et al., 2021). Nevertheless, calculations such as matrix multiplication and derivations of critical functions are possible using solely the CPU; they simply lack speed and efficiency when compared to other machine learning strategies and neural networks with a GPGPU implementation, as they cannot compete with its parallelization capabilities in training a neural network (Chaves et al., 2019).

Image classification is one function of deep neural networks that has alternative current implementations (Abd Elaziz et al., 2021). An immediately apparent fact of the shallow current machine learning methods is that as an algorithm is written for learning, the features of interest in classifying the images must be identified by the designer, whereas in deep learning the propagation across multiple layers can teach the network which features are significant to classification; current shallow machine learning methods are slightly more human dependent when getting started, although they do not need as much training data as a deep neural network. Shallow machine learning refers to algorithms or structures that require relatively small amounts of manipulations made between data input and output as compared to those that deep learning structures are capable of (Z. Wang et al., 2020).

P. Wang et al. (2021) performed a comparative study of the support vector machine learning algorithm (SVM) and convolutional neural networks (CNNs), which are neural networks with a structure and function ideal for the analysis of image features. SVM handles classification by drawing a vector between different classifications – inputs that are analyzed and produce output on one side of the vector get identified as one classification, and outputs on the other side of that barrier are identified as the other classification. Sometimes that vector is linear, and its function is easily distinguished, but, in other cases, it is not linear, and more dimensions must be added to the space in which the vector is drawn so that there can be a linear separation of data. Finding and representing functions for a solution space of higher dimensionalities, though, becomes increasingly complex and inefficient; this weakness is referred to as the Curse of

Dimensionality (Alaka & Kasamani, 2021). A commonly approached solution is the kernel trick, which uses vector multiplication of data in the original dimensional space to simulate the creation of added dimensions without the need for finding and implementing a function for that space (Kozak, 2019). Nonlinear functions can still be defined in the algorithm but consume more power and time scaling with their complexity. With small data sets, SVM outperforms CNNs in terms of speed, 1.02 minutes to 2.01 minutes, and is slightly more accurate, 86% to 83%, while with large data sets, CNNs greatly outperform SVM, in both a speed of 23.2 minutes to SVM's 27.6 minutes, and in accuracy, beating out SVM 98% to 88% (P. Wang et al., 2021).

GPGPU Implementation and CUDA

Neural network implementation is a resource-heavy and data-heavy practice, but, given the advantageous hardware for multithreading of a strong GPU, tensor cores for efficient matrix multiplication, and the potential for a core-distributed neuron design, it is well applied to complex problems that other machine learning algorithms might struggle with. For instance, classification, which is evaluated by correctness of the network's predictions given input features, is extremely strong when implemented in a neural network (Krizhevsky et al., 2012). Considering current hardware and the limitations to CPU processors and transistors being reached, the parallelism achieved by a GPGPU is really the only plausible way to handle these computations. Moore's Law states that the growth and improvement of transistors placed on chips is slowing due to power drain, size, and heat, meaning that the amount of data and operations that a CPU can hold at once is also slowing in growth (Ajayan et al., 2021).

Nvidia Corporation has developed and released Compute Unified Device Architecture (CUDA), which is a framework allowing for the coding against an API and libraries that let the

GPU's hardware be used for purposes outside of intensive graphics processing (Park et al., 2012). Coding with CUDA represents the C++ coding structure, and its function calls are similar to those made in the C language; while this means that with the proper installations that CUDA code can be written in any C++ IDE, there are also platforms such as TensorFlow and PyTorch that can be used to leverage CUDA coding conventions (Nvidia Corporation, 2022). This software level manipulation is what makes the GPU a GPGPU (Kenyon et al., 2019).

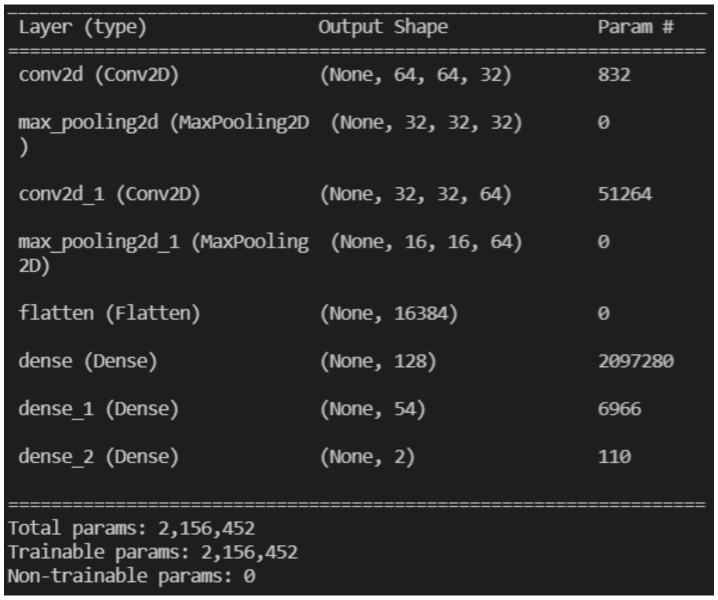
The structure and function of deep neural networks are optimized when paired with the capabilities offered by CUDA. Matrices are still set for inputs and synaptic weights, and algorithms such as the loss function are implemented in a mathematically identical way. Backpropagation and the feedforward of data are executed as expected for each perceptron in the neural network, but calculations are done on a massively parallel scale. By using the CPU as a driver of sort for processes, CUDA uses a processing hierarchy to maximize parallelism of tasks (Nvidia Corporation, 2022). This varies from normal GPU function in the fact that CUDA is used to repurpose the hardware for calculations and processes outside of rendering images. GPGPU cores are set up in blocks where all of the threads on those blocks are running the same kernel instruction or calculation type in parallel. So, CUDA is used to allocate memory for input data, then to copy data from the CPU into that memory. Computations are done in parallel as described by the neural networks structure and activation/loss functions, and then the results of each block are brought back together and copied back to the CPU for output (Noruzi et al., 2019). CUDA is the foundation for coding that allows a GPU to be transformed into a GPGPU for uniquely optimized processing of deep learning tasks.

Convolutional Neural Network Acceleration

In a software implementation of a convolutional neural network, which has its structure shown in Figure 1, financial time series data was analyzed for real estate pricing using wavelet transforms; wavelets are particularly useful for recognizing patterns and signal analysis (Moumene & Ouelaa, 2022). The data used consists of real estate sale prices from 2001 to 2019 in the state of Connecticut that exceed $2000 (data.ct.gov, 2021). Both the code and the data are available at https://github.com/cshelmick/real-estate-volatility-CNN. The purpose of the analysis was to create images that could be run through a convolutional neural network as input images to train and test the CNN and determine moments of market volatility. Volatility in this regard is synonymous to standard deviation, as it measures the statistical difference between an actual value and what it normally is expected to be, so an average of the absolute values of standard deviations for random time series was compared to a bound of 0.3 to classify whether the housing market was relatively volatile during a given period (X. Hu, 2019).

Figure 1

CNN Model


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        832

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        51264

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 128)               2097280

 dense_1 (Dense)             (None, 54)                6966

 dense_2 (Dense)             (None, 2)                 110

=================================================================
Total params: 2,156,452
Trainable params: 2,156,452
Non-trainable params: 0
```

Wavelet transforms are ideal transforms for evaluating data without evident repeating patterns of amplitude and frequency, making it superior to a transform such as the Fourier transform, which is better suited for data of repeated and easily predictable amplitude and frequency (Moumene & Ouelaa, 2022). The wavelet transform accomplishes this analysis by taking a mother wavelet that suits the shape of the data well and adjusting its size, or scale, to compare itself to different pieces of data in time within a data set. The mother wavelet chosen for this financial analysis was the Morlet wavelet, as it is consistently one of the most effective mother wavelets in analyses of financial data (Martínez & Cervantes, 2021). The continuous wavelet transform (CWT) used for this particular analysis is one that performs analysis at as many possible scales and locations as it can, as opposed to the discrete wavelet transform (DWT), which has predetermined limitations for those values (Silik et al., 2021).

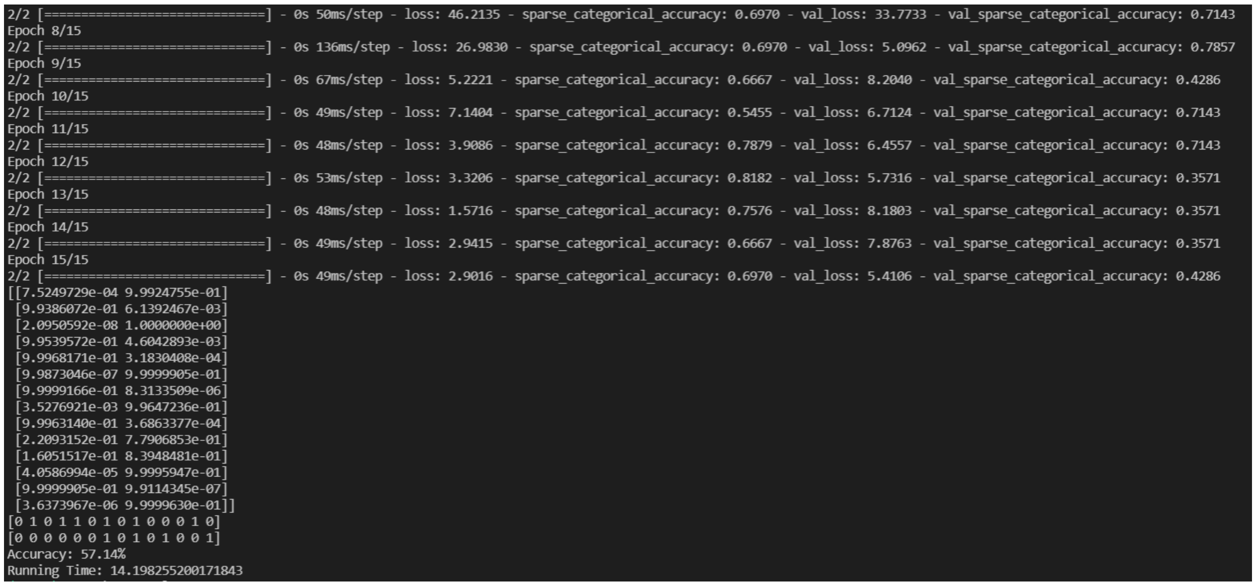
First, the data, in csv format, was processed by the KNIME application to convert the necessary data into a format that could be used with Python, which was then passed to the application using `genfromtext`(). Then, using the `test_train_split`() function, the data was split into training and testing sets for the CNN with respect to both independent and dependent variable arrays of a time index and standard deviation of sales price, respectively. Afterwards, the keras library was used to construct and train the neural network. Lastly, with the CNN constructed, the testing sets were used to evaluate the model.

The data structures and algorithms used for handling input data and construction of the CNN were greatly influenced by the code shared by Sebastian Feike in his 2020 article *Multiple Time Series Classification by Using Continuous Wavelet Transformation*. Feike created a demonstration that analyzed the Human Activity Recognition (HAR) dataset, which consists of smartphone-collected input signals based on human movement and six output activities to identify based on that movement.

The CNN was run using both a CPU, the Intel Core i7-8750H, and a GPGPU, the Nvidia Tesla V100 PCIe 32 GB. Figure 2 displays the running time of the training and evaluation process as 14.2 seconds for the CNN run on the CPU. On the other hand, as shown by Figures 3 and 4, the GPU implementation takes full advantage of the GPU's memory resources, and moments later its processing power, in order to build, train, and evaluate the CNN within 6.0 seconds. There is a clear improvement in the usage of the GPU implementation; despite a relatively small input data set, it is more than twice as fast as the CPU implementation. Just as the accuracy of the neural network would increase as the size of the data set grew, so would the processing efficiency advantage provided by the GPU implementation.

Figure 2

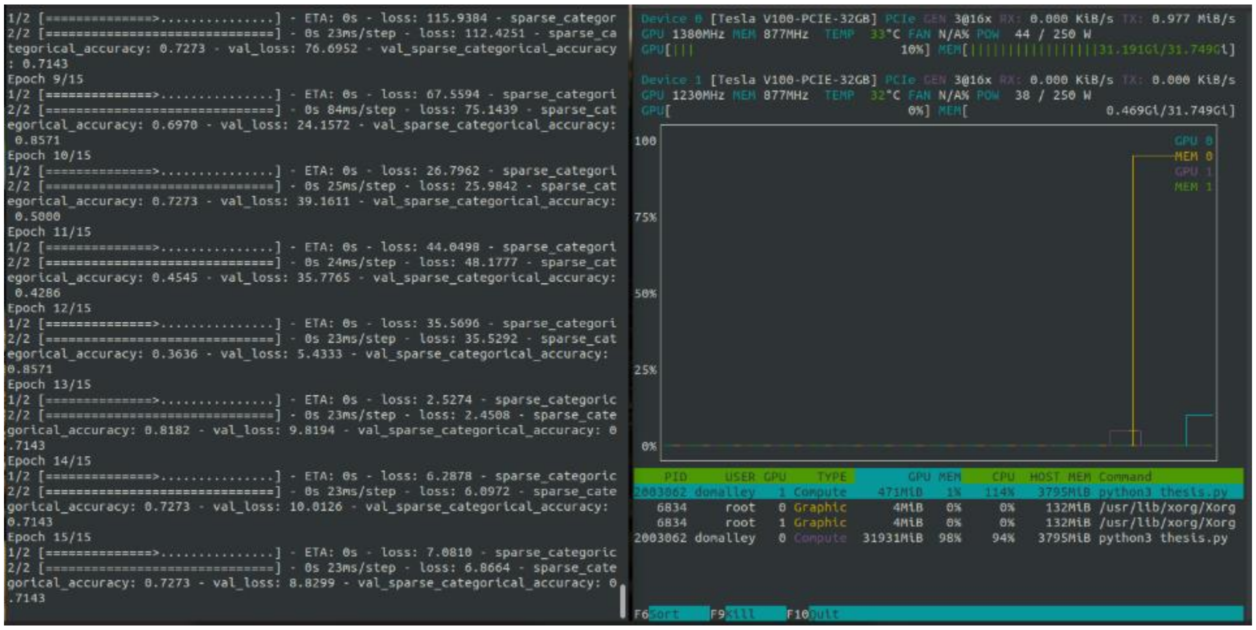
Running Time Following CNN Training and Evaluation on CPU



```
2/2 [==============================] - 0s 50ms/step - loss: 46.2135 - sparse_categorical_accuracy: 0.6970 - val_loss: 33.7733 - val_sparse_categorical_accuracy: 0.7143
Epoch 8/15
2/2 [==============================] - 0s 136ms/step - loss: 26.9830 - sparse_categorical_accuracy: 0.6970 - val_loss: 5.0962 - val_sparse_categorical_accuracy: 0.7857
Epoch 9/15
2/2 [==============================] - 0s 67ms/step - loss: 5.2221 - sparse_categorical_accuracy: 0.6667 - val_loss: 8.2040 - val_sparse_categorical_accuracy: 0.4286
Epoch 10/15
2/2 [==============================] - 0s 49ms/step - loss: 7.1404 - sparse_categorical_accuracy: 0.5455 - val_loss: 6.7124 - val_sparse_categorical_accuracy: 0.7143
Epoch 11/15
2/2 [==============================] - 0s 48ms/step - loss: 3.9086 - sparse_categorical_accuracy: 0.7879 - val_loss: 6.4557 - val_sparse_categorical_accuracy: 0.7143
Epoch 12/15
2/2 [==============================] - 0s 53ms/step - loss: 3.3206 - sparse_categorical_accuracy: 0.8182 - val_loss: 5.7316 - val_sparse_categorical_accuracy: 0.3571
Epoch 13/15
2/2 [==============================] - 0s 48ms/step - loss: 1.5716 - sparse_categorical_accuracy: 0.7576 - val_loss: 8.1803 - val_sparse_categorical_accuracy: 0.3571
Epoch 14/15
2/2 [==============================] - 0s 49ms/step - loss: 2.9415 - sparse_categorical_accuracy: 0.6667 - val_loss: 7.8763 - val_sparse_categorical_accuracy: 0.3571
Epoch 15/15
2/2 [==============================] - 0s 49ms/step - loss: 2.9016 - sparse_categorical_accuracy: 0.6970 - val_loss: 5.4106 - val_sparse_categorical_accuracy: 0.4286
[[7.5249729e-04 9.9924755e-01]
 [9.9386072e-01 6.1392467e-03]
 [2.0950592e-08 1.0000000e+00]
 [9.9539572e-01 4.6042893e-03]
 [9.9968171e-01 3.1830408e-04]
 [9.9873046e-07 9.9999905e-01]
 [9.9999166e-01 8.3133509e-06]
 [3.5276921e-03 9.9647236e-01]
 [9.9963140e-01 3.6863377e-04]
 [2.2093152e-01 7.7906853e-01]
 [1.6051517e-01 8.3948481e-01]
 [4.0586994e-05 9.9995947e-01]
 [9.9999905e-01 9.9114345e-07]
 [3.6373967e-06 9.9999630e-01]]
[0 1 0 1 1 0 1 0 1 0 0 0 1 0]
[0 0 0 0 0 0 1 0 1 0 1 0 0 1]
Accuracy: 57.14%
Running Time: 14.198255200171843
```

Figure 3

Full Memory Usage of GPU During CNN Runtime

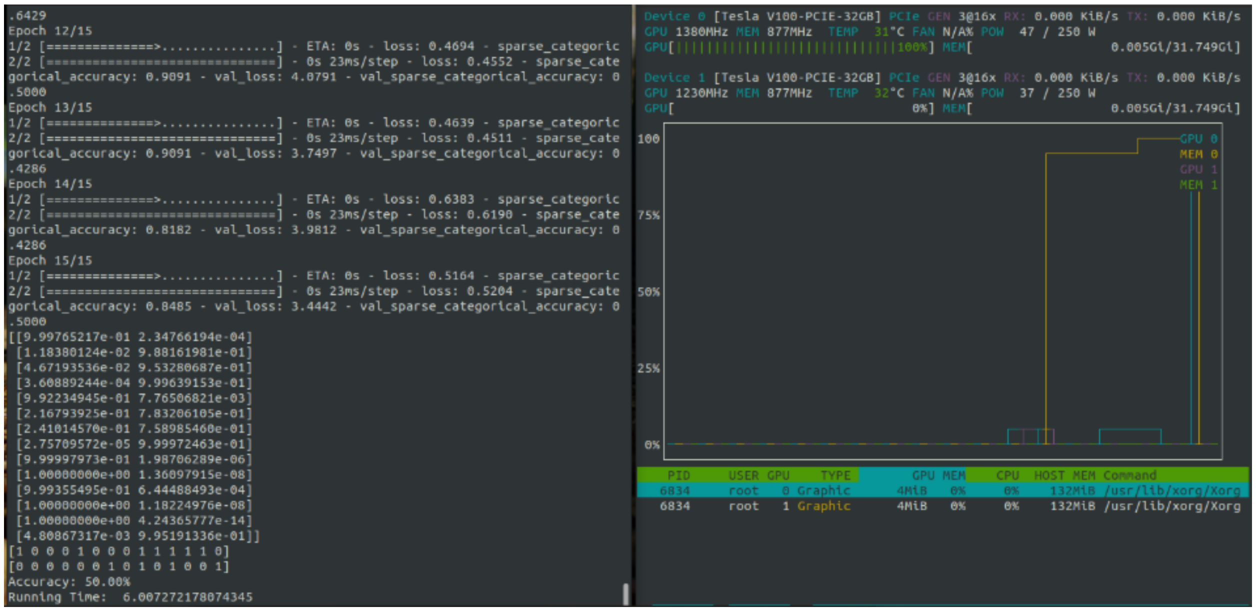


```
1/2 [==============>...............] - ETA: 0s - loss: 115.9384 - sparse_categor
2/2 [==============================] - 0s 23ms/step - loss: 112.4251 - sparse_ca
tegorical_accuracy: 0.7273 - val_loss: 76.6952 - val_sparse_categorical_accuracy
: 0.7143
Epoch 9/15
1/2 [==============>...............] - ETA: 0s - loss: 67.5594 - sparse_categori
2/2 [==============================] - 0s 84ms/step - loss: 75.1439 - sparse_cat
egorical_accuracy: 0.6970 - val_loss: 24.1572 - val_sparse_categorical_accuracy:
 0.8571
Epoch 10/15
1/2 [==============>...............] - ETA: 0s - loss: 26.7962 - sparse_categori
2/2 [==============================] - 0s 25ms/step - loss: 25.9842 - sparse_cat
egorical_accuracy: 0.7273 - val_loss: 39.1611 - val_sparse_categorical_accuracy:
 0.5000
Epoch 11/15
1/2 [==============>...............] - ETA: 0s - loss: 44.0498 - sparse_categori
2/2 [==============================] - 0s 24ms/step - loss: 48.1777 - sparse_cat
egorical_accuracy: 0.4545 - val_loss: 35.7765 - val_sparse_categorical_accuracy:
 0.4286
Epoch 12/15
1/2 [==============>...............] - ETA: 0s - loss: 35.5696 - sparse_categori
2/2 [==============================] - 0s 23ms/step - loss: 35.5292 - sparse_cat
egorical_accuracy: 0.3636 - val_loss: 5.4333 - val_sparse_categorical_accuracy:
0.8571
Epoch 13/15
1/2 [==============>...............] - ETA: 0s - loss: 2.5274 - sparse_categoric
2/2 [==============================] - 0s 23ms/step - loss: 2.4508 - sparse_cate
gorical_accuracy: 0.8182 - val_loss: 9.8194 - val_sparse_categorical_accuracy: 0
.7143
Epoch 14/15
1/2 [==============>...............] - ETA: 0s - loss: 6.2878 - sparse_categoric
2/2 [==============================] - 0s 23ms/step - loss: 6.0972 - sparse_cate
gorical_accuracy: 0.7273 - val_loss: 10.0126 - val_sparse_categorical_accuracy:
0.7143
Epoch 15/15
1/2 [==============>...............] - ETA: 0s - loss: 7.0810 - sparse_categoric
2/2 [==============================] - 0s 23ms/step - loss: 6.8664 - sparse_cate
gorical_accuracy: 0.7273 - val_loss: 8.8299 - val_sparse_categorical_accuracy: 0
.7143
```

```
Device 0 [Tesla V100-PCIE-32GB] PCIe GEN 3@16x RX: 0.000 KiB/s TX: 0.977 MiB/s
GPU 1380MHz MEM 877MHz  TEMP  33°C FAN N/A% POW  44 / 250 W
GPU[|||                     10%] MEM[||||||||||||||||||31.191Gi/31.749Gi]

Device 1 [Tesla V100-PCIE-32GB] PCIe GEN 3@16x RX: 0.000 KiB/s TX: 0.000 KiB/s
GPU 1230MHz MEM 877MHz  TEMP  32°C FAN N/A% POW  38 / 250 W
GPU[                         0%] MEM[                 0.469Gi/31.749Gi]

    PID     USER GPU    TYPE        GPU MEM      CPU  HOST MEM Command
2003062 domalley   1 Compute    471MiB   1%    114%   3795MiB python3 thesis.py
   6834     root   0 Graphic      4MiB   0%      0%    132MiB /usr/lib/xorg/Xorg
   6834     root   1 Graphic      4MiB   0%      0%    132MiB /usr/lib/xorg/Xorg
2003062 domalley   0 Compute  31931MiB  98%     94%   3795MiB python3 thesis.py

F6Sort    F9Kill    F10Quit
```

Figure 4

Running Time Following CNN Training and Evaluation on GPU



```
.6429
Epoch 12/15
1/2 [==============>...............] - ETA: 0s - loss: 0.4694 - sparse_categoric
2/2 [==============================] - 0s 23ms/step - loss: 0.4552 - sparse_cate
gorical_accuracy: 0.9091 - val_loss: 4.0791 - val_sparse_categorical_accuracy: 0
.5000
Epoch 13/15
1/2 [==============>...............] - ETA: 0s - loss: 0.4639 - sparse_categoric
2/2 [==============================] - 0s 23ms/step - loss: 0.4511 - sparse_cate
gorical_accuracy: 0.9091 - val_loss: 3.7497 - val_sparse_categorical_accuracy: 0
.4286
Epoch 14/15
1/2 [==============>...............] - ETA: 0s - loss: 0.6383 - sparse_categoric
2/2 [==============================] - 0s 23ms/step - loss: 0.6190 - sparse_cate
gorical_accuracy: 0.8182 - val_loss: 3.9812 - val_sparse_categorical_accuracy: 0
.4286
Epoch 15/15
1/2 [==============>...............] - ETA: 0s - loss: 0.5164 - sparse_categoric
2/2 [==============================] - 0s 23ms/step - loss: 0.5204 - sparse_cate
gorical_accuracy: 0.8485 - val_loss: 3.4442 - val_sparse_categorical_accuracy: 0
.5000
[[9.99765217e-01 2.34766194e-04]
 [1.18380124e-02 9.88161981e-01]
 [4.67193536e-02 9.53280687e-01]
 [3.60889244e-04 9.99639153e-01]
 [9.92234945e-01 7.76506821e-03]
 [2.16793925e-01 7.83206105e-01]
 [2.41014570e-01 7.58985460e-01]
 [2.75709572e-05 9.99972463e-01]
 [9.99997973e-01 1.98706289e-06]
 [1.00000000e+00 1.36097915e-08]
 [9.99355495e-01 6.44488493e-04]
 [1.00000000e+00 1.18224976e-08]
 [1.00000000e+00 4.24365777e-14]
 [4.80867317e-03 9.95191336e-01]]
[1 0 0 0 1 0 0 0 1 1 1 1 1 0]
[0 0 0 0 0 0 1 0 1 0 1 0 0 1]
Accuracy: 50.00%
Running Time:  6.007272178074345
```

```
Device 0 [Tesla V100-PCIE-32GB] PCIe GEN 3@16x RX: 0.000 KiB/s TX: 0.000 KiB/s
GPU 1380MHz MEM 877MHz  TEMP  31°C FAN N/A% POW  47 / 250 W
GPU[||||||||||||||||||||||||||||100%] MEM[                0.005Gi/31.749Gi]

Device 1 [Tesla V100-PCIE-32GB] PCIe GEN 3@16x RX: 0.000 KiB/s TX: 0.000 KiB/s
GPU 1230MHz MEM 877MHz  TEMP  32°C FAN N/A% POW  37 / 250 W
GPU[                         0%] MEM[                 0.005Gi/31.749Gi]

    PID     USER GPU    TYPE        GPU MEM      CPU  HOST MEM Command
   6834     root   0 Graphic      4MiB   0%      0%    132MiB /usr/lib/xorg/Xorg
   6834     root   1 Graphic      4MiB   0%      0%    132MiB /usr/lib/xorg/Xorg
```

Conclusion

Deep learning implemented through deep neural networks is the next large step for computing to take towards significant improvements in analyzing and classifying big data. As it is still relatively young and has room for growth, deep learning will become stronger as more effective activation functions, loss/cost functions, and general backpropagation methods are theorized, researched, and developed. Even in its current state, deep learning proves to be far superior in complex analysis of real-world data sets, such as mapping functions between inputs and outputs to find trends for big data samples and classification of otherwise difficult to interpret inputs (Rajawat & Jain, 2020).

The numerous layers of depth in these neural networks demand immense resources in terms of time, space, and energy to function effectively – resources that are not efficiently available in modern CPU processing. While complexity of running data through a neural network is polynomial, adding layers for depth requires large amounts of neurons and synapses to be making calculations in parallel so that processing is efficient and so that output is not outdated by the time the neural network is able to produce it; a CPU implementation of a neural network attempting to handle big data can take days to process data that needs to be processed in much less time to have any sort of significance (Nguyen, 2021). The use of a general-purpose GPU is the key to achieving this processing power, as the hardware designed for intense graphics processing possesses potentially thousands of processing cores and tensor cores that can be repurposed for the components of a neural network. GPGPU implementation of these neural networks is already being experimented with and implemented across multiple professional fields. Deep learning modeled in GPGPUs displays advances in computing efficiency that can no longer be brought about by improving the hardware design of processing cores.

References

Abd Elaziz, M., Dahou, A., Abualigah, L., Yu, L., Alshinwan, M., Khasawneh, A. M., & Lu, S. (2021). Advanced metaheuristic optimization techniques in applications of deep neural networks: A review. *Neural Computing and Applications*, *33*(21), 14079–14099. https://doi.org/10.1007/s00521-021-05960-5

Ajayan, J., Nirmal, D., Tayal, S., Bhattacharya, S., Arivazhagan, L., Fletcher, A. S. A., Murugapandiyan, P., & Ajitha, D. (2021). Nanosheet field effect transistors-A next generation device to keep Moore's law alive: An intensive study. *Microelectronics Journal*, *114*, 105141. https://doi.org/10.1016/j.mejo.2021.105141

Alaka, B., & Kasamani, B. S. (2021). Redefining data dimensionality through dynamic linkages in data-space continuum. *Advances in Intelligent Systems and Computing*, 427–436. https://doi.org/10.1007/978-981-33-4299-6_35

Alman, J., & Williams, V. V. (2021). A refined laser method and faster matrix multiplication. *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, 522–539. https://doi.org/10.1137/1.9781611976465.32

Ambaka, P. (2021, August 10). *Best CPU for GTX 1080*. Computer Station Nation. https://computerstationnation.com/best-cpu-for-gtx-1080/

Amerineni, R. (2020). *Brain-inspired machine learning classification models* (dissertation).

Amiri, H., & Shahbahrami, A. (2020). SIMD programming using Intel vector extensions. *Journal of Parallel and Distributed Computing*, *135,* 83–100. https://doi.org/10.1016/j.jpdc.2019.09.012

Asghar, M. S., Arslan, S., & Kim, H. (2021). A low-power spiking neural network chip based on a compact LIF neuron and binary exponential charge injector synapse circuits. *Sensors*, *21*(13), 4462. https://doi.org/10.3390/s21134462

Basha, S. H. S., Dubey, S. R., Pulabaigari, V., & Mukherjee, S. (2020). Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*, *378*, 112–119. https://doi.org/10.1016/j.neucom.2019.10.008

Bi, Z., & Zhou, C. (2019). Understanding the computational difficulty of a binary-weight perceptron and the advantage of input sparseness. *Journal of Physics A: Mathematical and Theoretical*, *53*(3). https://doi.org/10.1088/1751-8121/ab2682

Breß, S., Funke, H., Zeuch, S., Rabl, T., & Markl, V. (2019). An overview of hawk: A hardware-tailored code generator for the heterogeneous many core age. *Gesellschaft Für Informatik*, 87–90. https://doi.org/10.18420/btw2019-ws-07

Calore, E., Gabbana, A., Schifano, S. F., & Tripiccione, R. (2020). ThunderX2 performance and energy-efficiency for HPC workloads. *Computation*, *8*(1), 20. https://doi.org/10.3390/computation8010020

Chanda, D., Al-Badi, A.-E.-, & Islam, A. K. M. N. (2021). Performance analysis of initialization algorithms of deep neural network based coordinated beamforming system for mmwave. *2021 5th International Conference on Electrical Engineering and Information & Communication Technology*. https://doi.org/10.1109/iceeict53905.2021.9667939

Chaves, D., Fidalgo, E., Alegre, E., Janez-Martino, F., & Velasco-Mata, J. (2019). CPU vs GPU performance of deep learning based face detectors using resized images in forensic applications. *9th International Conference on Imaging for Crime Detection and Prevention*. https://doi.org/10.1049/cp.2019.1174

Chen, Y. (2020). 2020: Looking forward to the next decade [from the editor]. *IEEE Circuits and Systems Magazine*, *20*(1), 3. https://doi.org/10.1109/mcas.2019.2962260

data.ct.gov. (2021, November 29). *Real estate sales 2001-2019 GL*. Real Estate Sales 2001-2019 GL - CKAN. https://catalog.data.gov/dataset/real-estate-sales-2001-2018

Dong, J., Zheng, D., Yang, L. F., & Karypis, G. (2021). Global neighbor sampling for mixed CPU-GPU training on giant graphs. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. https://doi.org/10.1145/3447548.3467437

Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., & Dongarra, J. (2012). From CUDA to OpenCL: towards a performance-portable solution for multi-platform GPU programming. *Parallel Computing*, *38*(8), 391–407. https://doi.org/10.1016/j.parco.2011.10.002

Erickson, B. (2021). Basic artificial intelligence techniques. *Radiologic Clinics of North America*, *59*(6), 933–940. https://doi.org/10.1016/j.rcl.2021.06.004

Feike, S. (2020, February 10). *Multiple time series classification by using continuous wavelet transformation*. https://towardsdatascience.com/multiple-time-series-classification-by-using-continuous-wavelet-transformation-d29df97c0442

Fonseca, A., & Cabral, B. (2017). Prototyping a GPGPU neural network for deep-learning big data analysis. *Big Data Research*, *8*, 50–56. https://doi.org/10.1016/j.bdr.2017.01.005

Gao, L., Xu, Y., Wang, R., Luan, Z., Yu, Z., & Qian, D. (2020). Thread-level locking for SIMT architectures. *IEEE Transactions on Parallel and Distributed Systems*, *31*(5), 1121–1136. https://doi.org/10.1109/tpds.2019.2955705

Geer, D. (2005). Chip makers turn to multicore processors. *Computer*, *38*(5), 11–13. https://doi.org/10.1109/mc.2005.160

Gelado, I., & Garland, M. (2019). Throughput-oriented GPU memory allocation. *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*. https://doi.org/10.1145/3293883.3295727

Georgis, G., Lentaris, G., & Reisis, D. (2016). Acceleration techniques and evaluation on multi-core CPU, GPU and FPGA for image processing and super-resolution. *Journal of Real-Time Image Processing*, *16*(4), 1207–1234. https://doi.org/10.1007/s11554-016-0619-6

Ghiasi-Shirazi, K. (2018). Competitive cross-entropy loss: A study on training single-layer neural networks for solving nonlinearly separable classification problems. *Neural Processing Letters*, *50*(2), 1115–1122. https://doi.org/10.1007/s11063-018-9906-5

Harakannanavar, S. S., S, G. K., Ramachandran, S., S, T. G., & C, R. A. (2021). Performance analysis of CPU & GPU for real time image/video. *2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology*. https://doi.org/10.1109/rteict52294.2021.9573554

Hu, B., & Rossbach, C. J. (2020). Altis: modernizing GPGPU benchmarks. *2020 IEEE International Symposium on Performance Analysis of Systems and Software*. https://doi.org/10.1109/ispass48437.2020.00011

Hu, X. (2019). Volatility targeting strategy in S&P500. *Proceedings of the 10th International Conference on E-Education, E-Business, E-Management and E-Learning - IC4E '19*, 393–396. https://doi.org/10.1145/3306500.3306578

Ibrahim, M. A., Kayiran, O., Eckert, Y., Loh, G. H., & Jog, A. (2020). Analyzing and leveraging shared L1 caches in GPUs. *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 161–173. https://doi.org/10.1145/3410463.3414623

Intel Corporation. (2019, April 2). *Intel® Xeon® Platinum 8280 processor (38.5m Cache, 2.70 GHz)*. Intel. https://www.intel.com/content/www/us/en/products/sku/192478/intel-xeon-platinum-8280-processor-38-5m-cache-2-70-ghz/specifications.html?wapkw=xeon+platinum+8280

Jakhar, D., & Kaur, I. (2020). Artificial intelligence, machine learning and deep learning: definitions and differences. *Clinical and Experimental Dermatology*, *45*(1), 131–132. https://doi.org/10.1111/ced.14029

Kale, V., Lu, W., Curtis, A., Malik, A. M., Chapman, B., & Hernandez, O. (2020). Toward supporting multi-gpu targets via Taskloop and user-defined schedules. *OpenMP: Portable Multi-Level Parallelism on Modern Systems*, 295–309. https://doi.org/10.1007/978-3-030-58144-2_19

Kenyon, C., Volkema, G., & Khanna, G. (2019). Overcoming limitations of GPGPU-computing in scientific applications. *2019 IEEE High Performance Extreme Computing Conference*. https://doi.org/10.1109/hpec.2019.8916330

Kozak, S. (2019). Kernel trick for the cross section. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3307895

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Proceedings of the 25th International Conference on*

Neural Information Processing Systems, *1*, 1097–1105. https://doi.org/10.5555/2999134.2999257

Kulkarni, S. R., Parsa, M., Mitchell, J. P., & Schuman, C. D. (2021). Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing*, *447*, 145–160. https://doi.org/10.1016/j.neucom.2021.03.028

Li, B., & Zhuang, X. (2020). Multiscale computation on feedforward neural network and recurrent neural network. *Frontiers of Structural and Civil Engineering*, *14*(6), 1285–1298. https://doi.org/10.1007/s11709-020-0691-7

Lopez-Bernal, D., Balderas, D., Ponce, P., & Molina, A. (2021). Education 4.0: teaching the basics of KNN, LDA and simple perceptron algorithms for binary classification problems. *Future Internet*, *13*(8), 193. https://doi.org/10.3390/fi13080193

Lu, L., Jia, Y., Kirunda, J. B., Xu, Y., Ge, M., Pei, Q., & Yang, L. (2018). Effects of noise and synaptic weight on propagation of subthreshold excitatory postsynaptic current signal in a feed-forward neural network. *Nonlinear Dynamics*, *95*(2), 1673–1686. https://doi.org/10.1007/s11071-018-4652-9

Martínez, M., & Cervantes, P. (2021). Testing the resilience of CSR stocks during the COVID-19 crisis: A transcontinental analysis. *Mathematics*, *9*(5), 514. https://doi.org/10.3390/math9050514

Michaelson, G. (2020). Programming paradigms, Turing completeness and computational thinking. *The Art, Science, and Engineering of Programming*, *4*(3). https://doi.org/10.22152/programming-journal.org/2020/4/4

Moumene, I., & Ouelaa, N. (2022). Gears and bearings combined faults detection using optimized wavelet packet transform and pattern recognition neural networks. *The International Journal of Advanced Manufacturing Technology*. https://doi.org/10.1007/s00170-022-08792-2

Nguyen, T. G., Phan, T. V., Hoang, D. T., Nguyen, T. N., & So-In, C. (2021). Federated deep reinforcement learning for traffic monitoring in SDN-based IOT Networks. *IEEE Transactions on Cognitive Communications and Networking*, *7*(4), 1048–1065. https://doi.org/10.1109/tccn.2021.3102971

Noruzi, A., Mahlouji, M., & Shahidinejad, A. (2019). Iris recognition in unconstrained environment on graphic processing units with CUDA. *Artificial Intelligence Review*, *53*(5), 3705–3729. https://doi.org/10.1007/s10462-019-09776-7

Nvidia Corporation. (2021). *GeForce GTX Titan Z graphics card*. Nvidia. https://www.nvidia.com/en-us/geforce/graphics-cards/gtx-titan-z/specifications/

Nvidia Corporation. (2022, January 12). *CUDA toolkit documentation*. Nvidia Documentation Center. https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html

Ozen, E., & Orailoglu, A. (2019). Sanity-check: Boosting the reliability of safety-critical deep neural network applications. *2019 IEEE 28th Asian Test Symposium*. https://doi.org/10.1109/ats47505.2019.000-8

Ozyildirim, B. M., & Kiran, M. (2021). Levenberg–Marquardt multi-classification using hinge loss function. *Neural Networks, 143,* 564–571. https://doi.org/10.1016/j.neunet.2021.07.010

Pandey, R., & Badal, N. (2019). Understanding the role of parallel programming in multi-core processor based systems. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3350311

Parisi, L., Neagu, D., Ma, R., & Campean, F. (2022). Quantum ReLU activation for convolutional neural networks to improve diagnosis of Parkinson's disease and COVID-19. *Expert Systems with Applications*, *187,* 115892. https://doi.org/10.1016/j.eswa.2021.115892

Park, S. I., Cao, Y., Watson, L. T., & Quek, F. (2012). Performance analysis of a novel GPU computation-to-core mapping scheme for robust facet image modeling. *Journal of Real-Time Image Processing*, *10*(3), 485–500. https://doi.org/10.1007/s11554-012-0272-7

Peddie, J., & Dow, R. (2021, August 26). *Q2 GPU shipments soar year-to-year*. Jon Peddie Research. https://www.jonpeddie.com/press-releases/gpu-shipments-soar-in-q2-year-over-year

Pratama, K., & Kang, D.-K. (2020). Trainable activation function with differentiable negative side and adaptable rectified point. *Applied Intelligence*, *51*(3), 1784–1801. https://doi.org/10.1007/s10489-020-01885-z

Rajawat, A. S., & Jain, S. (2020). Fusion deep learning based on back propagation neural network for Personalization. *2nd International Conference on Data, Engineering and Applications*. https://doi.org/10.1109/idea49133.2020.9170693

Rice, L., Wong, E., & Kolter, Z. (2020). Overfitting in adversarially robust deep learning. *Proceedings of the 37th International Conference on Machine Learning*, *119,* 8093–8104.

Rivera-Alvarado, E., & Torres-Rojas, F. J. (2020). APU performance evaluation for accelerating computationally expensive workloads. *Electronic Notes in Theoretical Computer Science*, *349*, 103–118. https://doi.org/10.1016/j.entcs.2020.02.015

Sahlani, H., Hourali, M., & Minaei-Bidgoli, B. (2020). Coreference resolution using semantic features and fully connected neural network in the Persian language. *International Journal of Computational Intelligence Systems*, *13*(1), 1002. https://doi.org/10.2991/ijcis.d.200706.002

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Silik, A., Noori, M., Altabey, W. A., Ghiasi, R., & Wu, Z. (2021). Comparative analysis of wavelet transform for time-frequency analysis and transient localization in structural health monitoring. *Structural Durability & Health Monitoring*, *15*(1), 1–22. https://doi.org/10.32604/sdhm.2021.012751

Sioutas, S., Stuijk, S., Basten, T., Somers, L., & Corporaal, H. (2020). Programming tensor cores from an image processing DSL. *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*, 36–41. https://doi.org/10.1145/3378678.3391880

Srikar, M. S., Kumar, H. R., Kumar, S. A., Veena, N., Vasanthe, Rana, S. S., Chippalkatti, V., Niranjan, R. K., & Sarada, N. (2020). Challenges in design of very high-speed data acquisition system for high-altitude application. *Lecture Notes in Mechanical Engineering,* 451–460. https://doi.org/10.1007/978-981-15-1724-2_43

Syberfeldt, A., & Ekblom, T. (2017). A comparative evaluation of the GPU vs the CPU for parallelization of evolutionary algorithms through multiple independent runs. *International Journal of Computer Science and Information Technology*, *9*(3), 1–14. https://doi.org/10.5121/ijcsit.2017.9301

Taştan, İ., Karaca, M., & Yurdakul, A. (2020). Approximate CPU design for IOT end-devices with learning capabilities. *Electronics*, *9*(1), 125. https://doi.org/10.3390/electronics9010125

Teng, F., & Todo, Y. (2019). Dendritic neuron model and its capability of approximation. *2019 6th International Conference on Systems and Informatics (ICSAI)*. https://doi.org/10.1109/icsai48974.2019.9010147

Tian, J., Rivera, C., Di, S., Chen, J., Liang, X., Tao, D., & Cappello, F. (2021). Revisiting Huffman coding: toward extreme performance on modern GPU architectures. *2021 IEEE International Parallel and Distributed Processing Symposium*. https://doi.org/10.1109/ipdps49936.2021.00097

Tino, A., Collange, C., & Seznec, A. (2020). SIMT-X: extending single-instruction multi-threading to out-of-order cores. *ACM Transactions on Architecture and Code Optimization*, *17*(2), 1–23. https://doi.org/10.1145/3392032

Vaid, S., Kalantar, R., & Bhandari, M. (2020). Deep learning COVID-19 detection bias: Accuracy through artificial intelligence. *International Orthopaedics*, *44*(8), 1539–1542. https://doi.org/10.1007/s00264-020-04609-7

Vieira, J., Roma, N., Falcao, G., & Tomás, P. (2021). A compute cache system for signal processing applications. *Journal of Signal Processing Systems*, *93*(10), 1173–1186. https://doi.org/10.1007/s11265-020-01626-y

Villegas, A., Navarro, A., Asenjo, R., & Plata, O. (2019). Toward a software transactional memory for heterogeneous CPU–GPU processors. *The Journal of Supercomputing*, *75*(8), 4177–4192. https://doi.org/10.1007/s11227-018-2347-0

Wang, P., Fan, E., & Wang, P. (2021). Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. *Pattern Recognition Letters*, *141*, 61–67. https://doi.org/10.1016/j.patrec.2020.07.042

Wang, T., Xie, Z., & Gao, B. (2019). Design of distributed heterogeneous general signal processing platform architecture. *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*. https://doi.org/10.1109/icsidp47821.2019.9173426

Wang, Z., Hong, T., & Piette, M. A. (2020). Building thermal load prediction through shallow machine learning and Deep Learning. *Applied Energy*, *263*, 114683. https://doi.org/10.1016/j.apenergy.2020.114683

Wilson, A. C., Roclofs, R., Stern, M., Srebro N., Recht, B. (2017). *The marginal value of adaptive gradient methods in machine learning* [Paper]. 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA. https://arxiv.org/abs/1705.08292v2

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24. https://doi.org/10.1109/tnnls.2020.2978386

Xu, X., Tan, M., Corcoran, B., Wu, J., Nguyen, T. G., Boes, A., Chu, S. T., Little, B. E., Morandotti, R., Mitchell, A., Hicks, D. G., & Moss, D. J. (2020). Photonic perceptron based on a Kerr microcomb for high-speed, scalable, optical neural networks. *Laser & Photonics Reviews*, *14*(10), 2000070. https://doi.org/10.1002/lpor.202000070

Yan, D., Wang, W., & Chu, X. (2020). Demystifying tensor cores to optimize half-precision matrix multiply. *2020 IEEE International Parallel and Distributed Processing Symposium*. https://doi.org/10.1109/ipdps47924.2020.00071

You, W., Shen, C., Wang, D., Chen, L., Jiang, X., & Zhu, Z. (2020). An intelligent deep feature learning method with improved activation functions for machine fault diagnosis. *IEEE Access*, *8*, 1975–1985. https://doi.org/10.1109/access.2019.2962734

Zachariadis, O., Satpute, N., Gómez-Luna, J., & Olivares, J. (2020). Accelerating sparse matrix–matrix multiplication with GPU tensor cores. *Computers & Electrical Engineering*, *88*, 106848. https://doi.org/10.1016/j.compeleceng.2020.106848