Perfect Storm: DSAs Embrace *Deep Learning* for GPU-based Computer Vision

Marcelo Pias, Silvia Botelho, Paulo Drews-Jr
Intelligent Robotics and Automation Group (NAUTEC)
Computing Science Centre (C3)
Federal University of Rio Grande (FURG)
mpias, silviacb, paulodrews@furg.br

***Abstract*—This paper explores Domain Specific Deep Learning Architectures for GPU Computer Vision through a "brain storming" approach on selected hands-on topics in the area. We intend to discuss Deep Neural Networks (DNNs) to image classification problems through tools, frameworks and data pipelines commonly used to train and deploy DNNs in GPUs and Domain Specific Architectures (DSAs).**

I. Introduction

Artificial Intelligence systems need the capacity to acquire their knowledge through the extraction of patterns in the raw data. Such an ability, defined as *machine learning (ML)*, has enabled the resolution of problems that appeared too subjective including, for instance, filtering out SPAM messages from the user's mailboxes using the *naive Bayes* algorithm [9].

The performance of ML algorithms relies on the representation of data in much higher-level data features. A standard approach is to manually perform the feature engineering which often calls for close involvement of a domain expert. Manual feature discovery is a complex task and depends on the expert and designer intuition. To devise a predictive model capable of assessing cardiopathies risks, for instance, the model designer might use a set of patient-related data including blood pressure, heart rate, family history, physical activity levels and many others. The goal is then to discover a set of features that is capable of maximising the discriminatory power of the original patient data. Expert 1 might choose the mean of the heart rate variable within a 5-sec time window. Expert 2 could well select the mean, maximum and minimum for the same variable and time window. Different groups of experts are likely to choose different *features*. This methodology is used mainly in methods of the so-called *conventional machine learning* (*e.g.* SVM, naive Bayes, Logistic Regression, Random Forest and many others).

A novel approach to feature engineering is to discover the underlying data representation automatically. In other words, the machine now accomplishes the process of feature discovery. This new approach, called *representational learning*, has achieved improved performance results given that the model training encompasses feature discovery as part of a loss function minimisation procedure.

Figure 1 contrasts conventional machine learning with representational learning. Boxes in grey are elements that can learn from data [9].

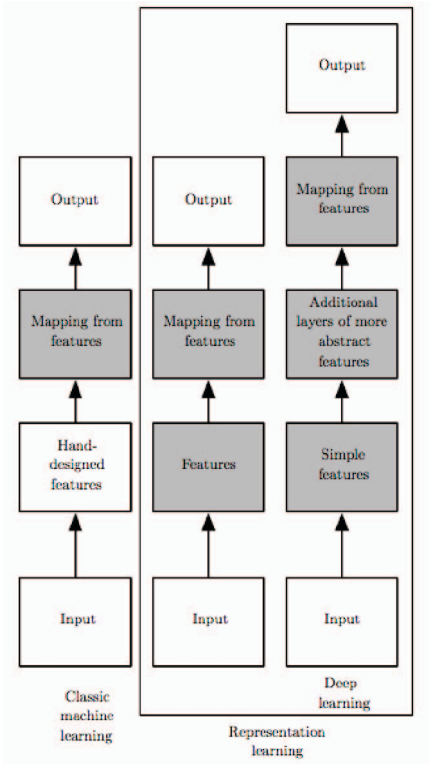

Fig. 1. Machine Learning: conventional versus representational learning [9].

Although this paper presents a few selected topics, it is not meant to be a comprehensive survey on deep learning. On the contrary, this paper is intended to offer a companion material to the tutorial **NVIDIA Deep Learning for Computer Vision** to be presented at SIBGRAPI 2019[1]. The NIVIDA-based course is structured into four sets of hands-on exercises organised in GPU-based tasks as follows:

- *GPU hands-on 1*: training a DNN for image classification; exploiting three key elements: deep neural networks, GPU vector parallel processing and big data.
- *GPU hands-on 2*: model performance assessment, fine-

[1]32nd Brazilian Conference on Graphics, Patterns and Images (SIBGRAPI 2019), October 28th to 31st in Rio de Janeiro.

2474-0705/19/$31.00 ©2019 IEEE
DOI 10.1109/SIBGRAPI-T.2019.00007

tuning of hyperparameters, dataset preparation, network architecture optimisations.
- *GPU hands-on 3*: deploying DNN models within real-world applications, real-time inference, datacenter versus edge computing.

For an in-depth discussion on deep learning, the readers are encouraged to consult the Ian Goodfellow *et al.* textbook [9] and the recently published literature surveys [41] [40].

This paper has the following structure. Sections I-IV provides the background information to briefly discuss a few deep neural network architectures. Tools, frameworks and data pipelines for image classification are discussed in Section V-VI. Section VII elaborates on the Domain Specific Architectures (DSAs) that target deep neural network applications. Section VII draws conclusions on the topic.

Deep Learning

Deep learning makes use of automated learning of data representation as a means for the machine to suitably capture complex concepts from simpler ones. In Deep Learning, artificial neural networks (ANNs) are stacked in multiple layers - hence the term *deep*. Such deep neural networks are trained using highly parallel systems capable of auto-detecting patterns in the original data.

The problem space of image classification has seen relevant progress in the past few years due to emerging deep learning techniques. Convolutional Neural Networks (CNNs) are considered high performing architectures for image classification warranted by the significant decreases in error rates achieved in the ILSVRC challenge[2]. Such neural networks, when applied to image classification, are trained using a large image dataset. The obtained low error rate suggests that similar results could be achieved in other application domains.

II. Deep Learning and New Processor Architectures

A recent article in the MIT Technology Review (March 2019)[3] describes *deep learning* pervasive throughout a growing number of applications such as processing photos in Facebook, personalised Google ads and self-driving cars that automatically discover the context in public roads. This article also mentions that the recipients of the prestigious *ACM Turing Award* in 2018, Geoffrey Hinton, Yann LeCun, and Yoshua Bengio, persevere developing neural networks whereas a significant fraction of the I.A research community focused on symbolic approaches such as rules manually coded. The *breaktrough* happened in 2012 when the Convolutional Neural Network AlexNet [8] scored the first place in the LSVRC Imagenet competition with a substantial difference in the computed error compared to the runner-up entries. The AlexNet original paper discusses three key performance factors: (i) the depth of the neural network, (ii) the usage of highly paralleled vector processors (e.g. general purpose GPUs) and (iii) a broad base of datasets made available for their CNN training. The term *big bang* of the machine learning have been coined because of these coupled factors.

A further front led by David Patterson and John Hennessy, also recipients of a Turing Award (2017), brings a forward-looking perspective whereby the design of modern computer systems leverages domain-specific architectures (DSAs) and open instruction set architectures (RISC-V[4] [1]). Some proposals for deep learning specific processors have started to appear in leading computer architecture conferences [10]–[12].

Disjoint as it might look, this train of thoughts intersect to make a coherent whole. New advances in new deep learning algorithms and techniques capitalise on novel architectures for parallel processing. The delivery of performance is not only expected from HPC datacenters but also from A.I edge computing. Edge-based applications include self-driving cars, autonomous vessels, point-of-care clinical image analysis, IoT sensors and many others.

This paper takes advantage of Hinton et al. and Patterson et al. ideas to bring a hands-on approach to deep learning for computer vision. The idea is to present and discuss tools, frameworks and data pipelines commonly used for training and validation of a deep neural network (DNN). This document complements a tutorial material to cover: (i) steps to build and deploy deep neural network models for image classification, (ii) accuracy and performance improvements, and (iii) discussion of recent results in the intersection of *Deep Learning* and new *Domain-Specific Architectures (DSAs)*. These issues are related to a broader understanding of deep learning in GPU parallel processors:

- To implement a *data pipeline* of commonly used deep learning tasks;
- To experiment with datasets, fine-tuning of DNN training parameters, improve the capacity and performance of a neural network;
- To integrate and deploy neural networks with the primary aim to solve a real-world problem.

The sections that follow present a brief description of the main topics covered in the tutorial.

III. Elements of the Storm

Three events happened at nearly the same time that enabled disruptive advances in machine learning. We named the combination as the *Perfect Storm* because these events have given a huge shake to the industry. Perfect means the right point in time to enable applications for speech recognition (e.g. Google Assistant & Apple Siri), recommender systems for personalised e-commerce, image classification and object recognition in social media, and natural language processing (NLP). The combination of these events is also called the *Big Bang* in machine learning. The events are as follows:

1) **Algorithms:** improvements in the solvers and techniques such as Convolutional Neural Networks (CNN)

[2]http://image-net.org/challenges/LSVRC/

[3]https://www.technologyreview.com/f/613233/the-pioneers-of-deep-learning-win-the-turing-award

[4]https://riscv.org/

made the use of DNNs more tractable. Significant progress started with the proposed AlexNet in 2012 [8].
2) **Big Data:** growth of data volume has provided the data needed to build deep networks. Around 2.5 quintillion bytes of data are produced each day. With the introduction of social media, there has been a proliferation of data primarily image and text-based.
3) **GPU:** this graphics system has been providing the computational power intended to solve DNNs in reasonable amounts of time. GPUs empower Data Scientists to build better models faster. A data scientist can run more "trial and errors" on GPUs which can be 2X to 10X faster than CPUs.

A. Data Representational Learning

A key point for the traditional machine learning techniques is the representation of data. Inadequate data representation can lead to an unsuccessful handling of these techniques. Feature engineering thus has become an important area of machine learning research. Several techniques to extract raw data features have been proposed in the literature over the last 20 years. Such approaches accounted for the need of domain-specific knowledge making it difficult to have generic solutions, and in many cases, significant human efforts are mandatory still.

In the computer vision context, the usage of data representation resources is a critical strategy in obtaining high-quality information from a high dimensional raw image data. Several methods have been proposed to tackle this issue, including Invariant Scale Resource Transformation (SIFT) [4], Oriented Gradient Histogram (HOG) [5] and Word Bag (BoW) [6]. Such feature-based methods have become popular and performing well in pre-processing machine vision algorithms and other domains such as speech recognition and natural language processing.

Deep Neural Networks (DNNs) bring a new scheme to raw data pre-processing. DNNs perform the extraction of essential data features with none human intervention [23]. These algorithms include a layered data representation architecture in which high-level information can be extracted from the last network layers, while low-level information is obtained from the lower layers.

The architecture of DNNs is inspired by the process of the primary sensory areas of the human brain. These areas automatically extract the representation of data from different scenes. The input image sensitises the eyeball, and it travels through different layers of neurons until their objects are correctly classified. The series of processing layers are mimicked in the architecture of DNNs.

B. Data is King

DNN-based methods do not require explicit feature extraction algorithms to obtain relevant data characteristics automatically. There is a shift in focus to the creation of training datasets that encompass the diversity of the data samples. It is safe to state that DNN-based algorithms crucially depend on large-scale datasets in addition to the recent advances in new digital processors.

The recent years have seen a steady increase in dataset availability. ImageNet [25] is a large set of annotated images, used to train the most popular networks (AlexNet [43], GoogleNet [47], VGGNet and ResNet). CIFAR10/100 [26] is another important dataset explored in classification tasks. On the other hand, datasets such as PASCAL VOC and Microsoft COCO have been used in segmentation tasks. YouTube-8M [24] is a relevant dataset for event detection, classification of video, and many other applications.

IV. Neural Network Architectures

Recent research results [11] indicate that 95% of Google datacentres inference demand for deep neural network applications employ the following network architectures: (i) Multi-Layer Perceptrons (MLP): 29%, (ii) Convolutional Neural Networks (CNN): 5% and (iii) Recurrent Neural Networks (RNN): 29%. This document attempts to bridge the gap between the algorithms and domain-specific architectures (DSAs). This paper focuses on a set of popular network architectures including CNNs [33] which represent the beginning of feasible deep neural networks [11]:

1) Multi-Layer Perceptrons (MLP): each new layer is a set of nonlinear functions of the weighted sum of all outputs (fully connected) from a prior one, which reuses the weights.
2) Convolutional Neural Networks (CNN): each inner layer is a set of nonlinear functions of weighted sums of spatially nearby subsets of outputs from the prior layer, which also reuses the weights.
3) Recurrent Neural Networks (RNN): each subsequent layer is a collection of nonlinear functions of weighted sums of outputs and the previous state. The most popular RNN is the Long Short-Term Memory (LSTM) which decides what data should be forgotten and what should be passed on as a state to the next layer. The weights are reused across multiple time steps.

A. Classic CNN architectures

Multi-Layer Perceptron (MLP) is a classic neural network. It functions by receiving input data that propagates through connected processing elements based on values of weight parameters. MLPs employ fully connected (FC) neurons so that learnt weights build a suitable mapping function from the network input to its output. The MLP bioinspiration trace back to how the network of the brain's neurons establish synaptic connections.

Deep neural networks have their genesis in convolutional neural networks (CNNs) [33] [34] where the inner structure resembles the mammalian visual cortex with a complex sequence of cells [27]. CNNs are capable of taking a two-dimensional input data structure (e.g. an image). Local connections and shared weights convolve over this 2-D data structure.

Using shared instead of individual weights between highly connected neurons results in a network with fewer weights

to learn, which makes the network faster and easier to train. This operation is similar to cells of the visual cortex. Such neurons operate over small parts of a scene rather than the entire image resulting in local filters with the ability to extract spatial correlations from the data.

Although CNNs share similarities with MLPs in the learning process, the convolution layers bring more efficiency through data summarization going from low to high-level data representations.

A CNN is composed of layers through which information will flow. The input data will go through several *convolutional layers* followed by *pooling layers* (subsampling) and, in the final stages, through *fully connected layers*.

Convolutional Layer: the image classification task has benefited tremendously from a convolutional neural network. Suppose an image I of dimension $n \times n$ and r colour channels (*e.g.* $r = 3$ for an RGB image). This image is propagated through convolutional layers formed by k core filters of size $m \times m \times p$. Notice that m must be smaller than the input image, but p may be smaller or the same size as r. Local filters share equal W_k weights and are convoluted over the input image, generating feature maps (h_k). Each feature map has size $n - m - 1$. Similarly to MLPs, a convolutional layer calculates the inner product between the weights and their inputs. The difference is that the input is now a portion of the original data entry. An activation or non-linear function is applied to the inner product producing the convolutional layer output, h_k.

Pooling Layer: subsampling layers reduce the dimensionality of feature maps. This operation not only speeds up the model training but also control the network to avoid overfitting. Functions such as mean, maximum and minimum can be applied to a $q \times q$ region for all feature maps where q is the filter size.

Fully Connected Layer: the last-stage layers are usually fully connected ones as also used in MLPs. Such layers rely on previous computed lower-level features to generate high-level abstractions from the data. A final layer in the network can still be used to create a ranking where each rank score is a probability associated with a specific classification category for a given data input instance. Softmax and SVM are function examples of this type of layer [28].

B. Generative adversarial networks (GAN)

First introduced in [32], GANs have become well known for their ability to synthesise realistic images. This technique learns how to generate new data with the same statistics as the training set. Specifically, a GAN is a framework for training generative parametric models. The core idea is to build two adversary networks: i) a generator *G* capable of generating data such as text, images, or even music and, ii) a discriminator *D* which informs whether a given input represents real data. Both *G* and *D* are trained simultaneously while the generator attempts to produce convincing fakes to fool the discriminator. The latter's job is to catch such fake data. Once the training is accomplished, the expected outcome is that the generator is so specialised in producing fake data that the discriminator is unable to classify which data is fake or real. For example, a GAN trained on dog images can generate new images that look at least superficially authentic to human observers, keeping many realistic characteristics.

C. LSTM

Recurrent Neural Networks (RNN) [29] is another approach widely used in applications that depend on sequential data. The main feature of this type of network is to capture the information present in data series. For example, a scene in a video to be analysed requires the capturing of contextual information. Unlike traditional neural networks, an RNN uses as its input a sequence of data that propagates through short-term memory units coupled with an input layer x, a hidden layer (state), and an output layer y.

Long Short-term memory (LSTM) [30] networks provide memory blocks in their recurring connections. Each memory block incorporates cells that are stored in temporary network states. Also, LSTMs include structures that direct data propagation by improving information flow control.

V. Deep Learning Principle

(a) Cat 1 (b) Cat 2 (c) Cat 3

(d) Dog 1 ("Luna") (e) Dog 2 ("Bono") (f) Dog 3 ("Flor")

Fig. 2. Samples of the Imagenet dataset [25]: Cats and German Shepherd Dogs.

The way the machine learns patterns through deep learning has similarities to human learning. Figure 2 shows six images of cats and carefully named dogs. Suppose that goal is to teach a potential learner to recognise German Shepherd dogs. The learning method for a human could be to present each ImageNet[5] sample image in Figure 2 and ask the learner to say from a scale of 1 to 10 how confident the learner is that that image is a German Shepherd. This confidence rating is then used to adjust the overall error, which is the distance between the learner guess and the actual animal in the image. For instance, a 10-points confidence of a human learner on a dog image yields a zero error for such an instance. As any good learner, the more the human sees the images, the better they get in providing the confidence scales. At the end of the

[5]http://www.image-net.org/

learning process, the overall error should be minimum for a good learner. In machine learning, a similar task called image classification, trains a neural network to be able to separate images into groups or classes. In this case, the two classes of interest are:

1) *Dogs*
2) *Not Dogs*

Although the power of deep learning lies in operating over massive datasets, the small dataset presented in Figure 2 can still be useful to exemplify the learning principles. Such a dataset contains six labelled images of German Shepherd dogs and cats, three images each respectively. A neural network designed specifically for image classification can be used. For instance, AlexNet was architected after the human visual cortex and offer capabilities to application developers to focus on the model training process instead of understanding the structure of the brain. Frameworks such as Caffe, Theano and Tensorflow can be used to train this specific neural network for the dog recognition. Tools that provide intuitive user bring together software libraries and frameworks in a visual development environment (e.g. NVIDIA DIGITS and Jupyter Notebooks). The user manages the deep learning workflow with no need for error-prone coding or even command-line input.

Figure 3 shows a simplified version of a deep neural network. In this example, the inputs at the bottom of the diagram represent each pixel value of an image instance. Ideally, this network should be trained using the full ImageNet dataset.

On the top, the network generates an output that in this example is the probability the image contains a dog. On the left side, the output that **should** have been generated is presented as labelled data instances. In this case, if the image was a dog, the output should have been 100%.

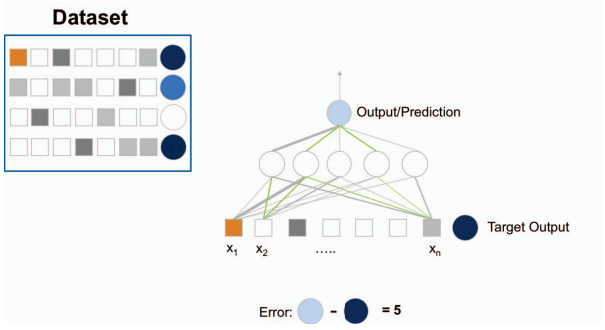

Fig. 3. Learning Step 1.

The learning process feeds each image through the network to generate a prediction which at first will be wrong. In this case, the guess is 5 units away from what it should have been (error = 5). As a result, the network is changed slightly to adapt to the current error calculation.

The *weights* represent the values which have changed. Weights are the multipliers associated with each operation.

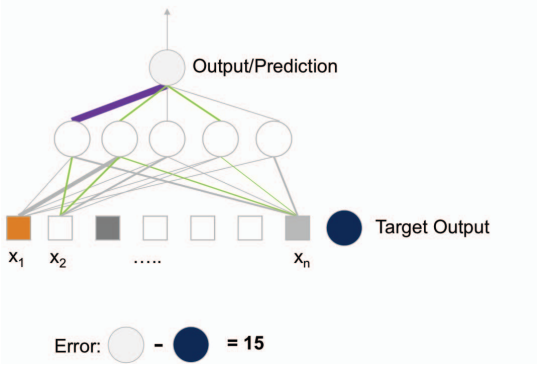

Fig. 4. Learning Step 2.

Figure 4 shows a slightly bigger weight represented as a thicker line at the top left side. This weight adjustment has made the error to increase with the new output (error = 15 units). The change made to the weight during the Step 1 could have been smaller.

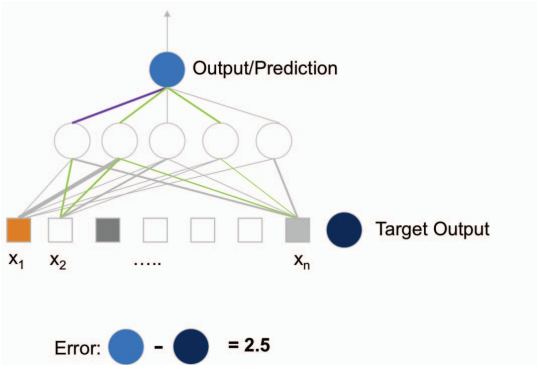

Fig. 5. Learning Step 3.

The goal is to adjust each weight by a small amount in the gradient direction that decreases the overall error (Figure 5. It is important to note that this is one of hundreds of thousands or millions of weights in a deep neural network. Although it has been presented in a very simplified manner, this process of iterative error function minimisation is at the core of the Stochastic Gradient Descent method [9].

VI. Deep Learning Pipeline in Practical Terms

A. Model Training

Deep Neural Networks are flexible algorithms inspired by the human brain that allow practitioners to use training strategies inspired by human learning. Figure 6 presents the training process of deep neural networks in a practical situation. The input data, e.g. an image, is passed through the network and an error (or loss) is computed. This step is called *forward propagation*. The *loss* information is passed back through the network to adjust each weight value by a small amount. Such adjustments are made towards the direction that reduces the

loss. This is called *backward propagation*. This process is repeated for a large number of iterations with thousands of pieces of data. The end-goal is to *learn* a function that takes an input and generates an output that is close as possible to the target labelled data (Figures 3-5).

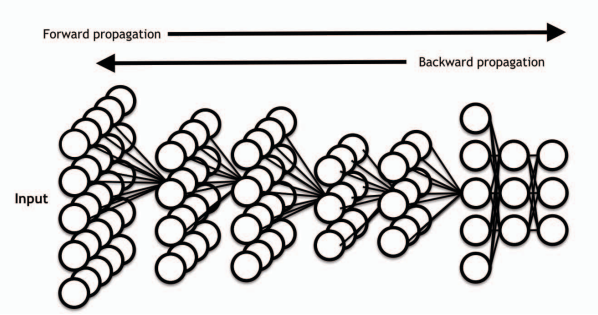

Fig. 6. Training a Deep Neural Network. The **forward propagation** yields an inferred label for each training image. The **loss** function is used to calculate the difference between known label and predicted label for each image. The **weights** are adjusted during the backward propagation step. This process is repeated over and over until it converges to an acceptable minimum error.

It is important to note that although trained DNN models are complex internally, at their core, they are simply functions where for each distinct input they will generate a specific output (Figure 7). In image classification problems, the input and output images follow specific rules in terms of data format and image size.

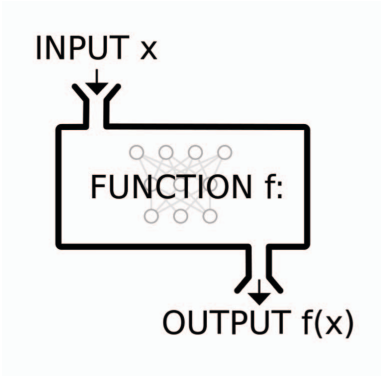

Fig. 7. Expected Inputs and Useful Outputs.

B. Model Validation

The data that a neural network is exposed to during training is the only guidance it has about the world. When a human learns how to identify a German Shepherd Dog, they begin with an understanding of not only dogs but even of features that might differentiate between dogs (colour, size, etc.) A neural network still interprets data as data, so has to discover which features are useful.

Successfully training a neural network to perform well on new data requires enough data to demonstrate the diversity of the environment where the network should be active. Fitting an untrained neural network to a small dataset of sample images such as the cats and dogs in Figure 2 will lead to highly specialised networks that have overfit, i.e. it is only useful on the exact images that it was exposed to during training. In case the network receives an unseen dog image with different characteristics (e.g. dog is facing a different direction, a different amount of light, or even with a different age), the network has no indication that those still fall under the category of a dog. Essentially, instead of "learning" the difference between dogs, the network has "memorised" which image in the training dataset belongs to which class.

The issue of overfitting can be assessed during the model validation process. The validation dataset is typically used to assess performance on new data using an approach that a human might find difficult to follow. Validation data is fed through the network to generate an output. In this case, the network does not learn anything from the data. Differently from the model training, the loss is reported, but the model itself is unchanged (Figure 8). The same validation dataset can be used to assess the model performance on new data over and over again while continuing to treat it as new data.

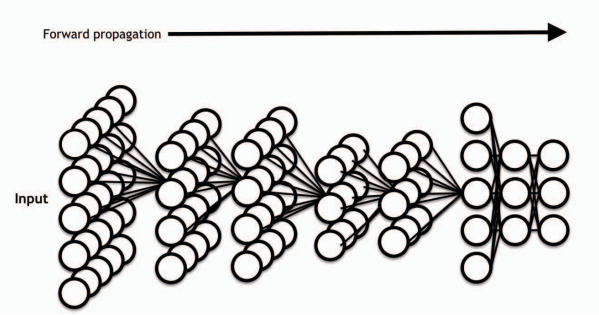

Fig. 8. Model Validation: Forward Propagation only.

Figure 9 shows a typical plot for model performance evaluation. This one has been generated using DIGITS[6] - NVIDIA visual tool for deep learning workflow.

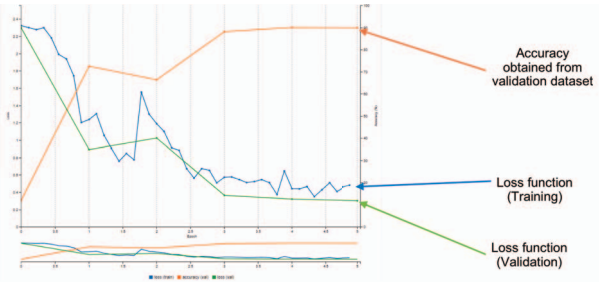

Fig. 9. Performance Evaluation.

The blue curve is the *loss or error* of the network while it is learning from the data. The green curve is the *loss* of the network on data that it is **NOT** used for learning. Datasets

[6]https://developer.nvidia.com/digits

are usually split as fractions, say 75% and 25%, for training and validation respectively. The validation part is set aside to be used to check whether the network is learning about the differences between ALL dogs and cats, and not just the ones that are used for training. The orange curve is the accuracy which measures how often the prediction made from the validation set is correct. Other performance metrics which characterise the individual classes could be calculated (e.g. precision and recall).

C. Deployment, Tools and Frameworks

The inference is the process of making decisions based on what has been learnt. The power of a DNN, say image classification, is that under a certain performance threshold, it can classify unlabelled images. Figure 10 shows the online inference whereby the trained model is exploited in a real-world application to make predictions from new unseen data. The best-trained model is then used in an application running in a production environment (e.g. datacentre, a vehicle, or a user smartphone). For some applications, such as autonomous vehicles, the inference is carried out in real-time and therefore, high throughput and low response time are critical.

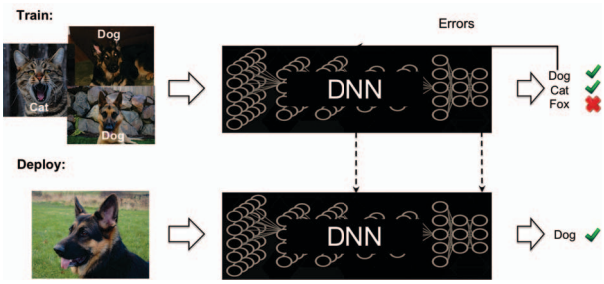

Fig. 10. Model Training and Deployment.

Figure 10 shows an example of a DNN training and deployment. Since the network is adequately trained, it can be used to deploy functionalities in several real-world applications. This process is used when unseen images are inputted into the network, and the correct class is expected as the outcome of the classification inference.

There are two distinct tasks of a deep learning workflow worthwhile mentioning: (i) model training and validation, (ii) and model deployment in real-world scenarios. Often the model is part of a more prominent application software. A model can identify whether dogs or cats are present in an image. A complete software could use this model to build an application to prevent a cat from coming into the house through the main entrance door.

Deployment involves system-related skills which merge deep learning with traditional programming. The traditional programming builds the easy-to-use software solution for the users. The application is created using the trained neural network model.

The model deployment needs to overcome several practical issues including data pre/post-processing, model weights and network architecture specs of software and potentially hardware environments (e.g. GPU edge-computing) - to cite a few. The deployment can also be regarded as the process of packaging a few components, including the developed models, databases and scripts in a container-like package. The end-goal of the deployment is to provide online inference services. Model training (or readjustment) is also possible, and it is getting more traction in recent years with new application-specific architectures (DSAs). Such components can be packaged in a container format (e.g. Docker or Kubernetes [7]. This model-based container for a real-world application is also called a *data-product*.

The regular training and inference cycle is depicted in Figure 11. The deployment tackles application questions such as: (i) how an image classifier can be deployed to an iPhone app that can tell the user whether or not a plant is poisonous? How can a pedestrian detection model be deployed to a self-driving vehicle in order to adjust the cars speed? How can a speech recognition tool be deployed to a piece of hardware that can authorise customer purchases?

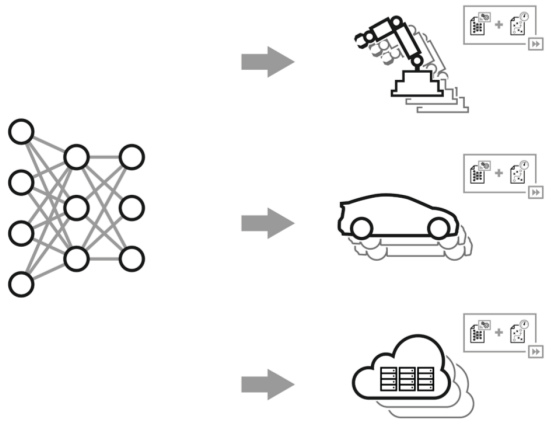

Fig. 11. How to Deploy a Model.

VII. Domain Specific Architectures (DSAs)

This section explores the deep learning problem space of where DSAs can bring significant benefits in terms of performance in the training, validation and real-time inference within the application domain. Performance, in this case, has a broad definition that comprises model deployment performance (application time response, processing throughput) and overall model quality (e.g. accuracy). In deployed applications, a balance should be reached between performance and model output accuracy while functionality transfers from resourceful data centres to edge computing devices (e.g. DNNs running on a self-navigating vessel, or users' smartphones).

Moore's law led to an expected performance delivery for modern CPUs whereby the number of transistors on a silicon chip doubles roughly every two years. Such a performance

[7]https://www.docker.com/ and https://kubernetes.io/

delivery has given a signal of stagnation over the past ten years. Dennard scaling law states that as transistors get smaller, their power density stays constant in proportion with the silicon area. Dennard law has shown signs of breakdown as of 2016. Currently, high temperatures of transistors in large scale integration pose a real system challenge. Cooling down the transistors takes more energy than the amount of energy that already passes through the transistors. This scenario has limited the performance delivery in terms clock frequency, where for instance, it remains at around 3 GHz. This law collapse has been referred to as the "Moore's law hit a wall".

However, Moore's law in the last decades enabled many computer system design innovation. Most new system design address the question on how bottlenecks at higher levels of abstraction can be detected and overcome in delivering the main low-level characteristic of large scale transistor integration. Pursuit of parallelism in all levels in a modern computer architecture started with deep memory hierarchy and instruction-level parallelism with deep pipelines. These examples of inner CPU system optimisations helped to deliver performance to application up to the point when other forms of parallelism had to be introduced. Performance and low power consumption started to be delivered through multiprocessor units where the potential of a capped CPU clock frequency could now be leveraged in a parallel multicore architecture.

Deep neural networks (DNNs) need intensive processing and data transfer capabilities for such tasks as building, testing and deploying a model to the real-world applications. Such issues have been satisfactorily tackled using general-purpose graphical processing units (GPGPUs) where the nature of parallel arithmetic operations in GPUs matches the DNNs development workflow. A few companies target deep learning through their existing GPU offering. For instance, NVIDIA and AMD have created a custom system design for deep learning data pipelines (e.g. Tesla V100 and Radeon Instinct, respectively).

A. Beyond CPUs and GPUs

DNN model training for large datasets is a complex optimisation problem that seeks minimise an objective loss function. It often takes a significant number of operations for algorithms such as the stochastic gradient descent (SGD) to converge into a global minimum solution. High-performance GPUs can speed up the training task with a cut in the training time from months to days for some workloads. The search for improved performance has led to several recent proposals on domain-specific architectures (DSAs) that embrace deep neural networks [10]–[12]. The argument here is that for DNN workloads a factor of 100 in improvements (no. operations per instruction) is highly desirable in accomplishing faster model training and inference. A possible direction is to implement the necessary functionality in ASICs and FPGAs accelerators leading to a higher performance/watt ratio. However, the R&D costs cannot be efficiently amortised over large volumes in case of ASIC specific design, and FPGAs are less efficient than ASICs [44].

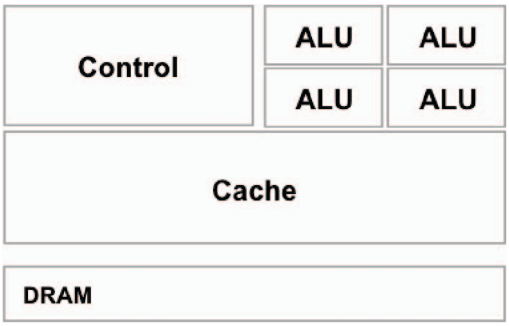

Fig. 12. CPU Design: Latency Oriented [48]

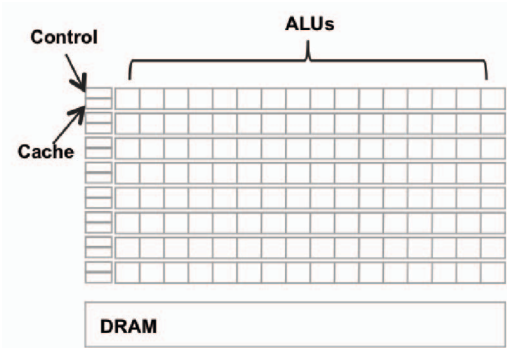

Fig. 13. GPU Design: Throughput Oriented [48]

CPUs are designed to reduce operation latency with powerful arithmetic logic units (ALUs) (Figure 12). Large and fast cache memory convert remote memory access to short-latency cache accesses. Sophisticated control is used to drive a complex system with techniques such as prediction for reduced branch latency. CPUs are optimised for sequential parts of code where latency is of prime importance. CPUs can be 10X+ faster than GPUs for sequential code. In contrast, GPUs is designed for high throughput operations with small caches to boost memory throughput (Figure 13). The control is simplified with no branch prediction. Many ALUs in GPUs lead to long latency, but they are heavily pipelined for high throughput. GPUs systems require a massive number of threads to tolerate latencies with threading logic and thread state management. Figure 13 shows the architecture of a typical GPU system.

Deep neural networks have been successfully exploiting the GPUs data-level SIMD parallelism. Software frameworks (e.g Caffe and Tensorflow) take advantage of the underlying system architecture often optimising parts of the code that are best suitable for running on a CPU (sequential) and GPU (parallel). The latter includes all the arithmetic matrix and vector operations performed in model training and validation. Novel parallel stochastic gradient descent algorithms have been recently proposed to train more efficiently DNNs in parallel systems such as clusters of GPUs [15]–[17].

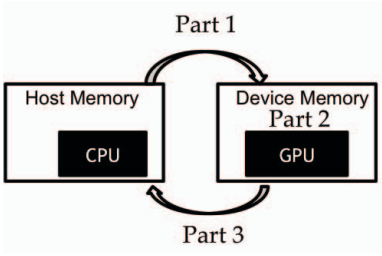

Fig. 14. CPU-GPU Interplay.

B. CPU-GPU Interplay Problem

Deep learning software frameworks specify in a given format the network architectures and linked hyperparameters. Also, networks learn differently in terms of training rates, methods, and other aspects. The analogy here is that different learners learn differently. High-level tools include Keras, Tensorboard, or APIs with conventional programming languages. The software attempts to build a processing graph of operations. A small portion of this runs in the CPU, but the majority should run in GPUs. Figure 14 shows the interplay between the CPU and GPU in commonly implemented DNNs. The code listed is a straw-man implementation of two vectors addition in CUDA[8]. This is a typical simple operation employed in the computing tasks associated with deep neural networks. The performance bottleneck is illustrated through the data transfers between CPU (host) and GPU (device). Parts 1-3 shown in Figure 14 demonstrate such a memory issue.

Listing 1. CPU-GPU Interplay

```
#include <cuda.h>
void vecAdd(float *h_A, float *h_B, ...
float *h_C, int n)
{
    int size = n* sizeof(float);
    float *d_A, *d_B, *d_C;
    // Part 1
    // Allocate GPU memory for A, B, and C
    // Copy A and B to GPU memory

    // Part 2
    // Kernel launch code
    // The GPU performs the addition

    // Part 3
    // Copy C from the GPU memory
    // Free GPU vectors
}
```

C. Domain Specific Architectures (DSAs)

A few application domains have recently benefited from DSAs. The task of mining in decentralised cryptocurrency systems has escalated the processing power from CPUs in the early stage of cryptocurrency blockchain lifetime to more customised architectures based on custom ASICs (latest generation from 2014 onwards). Such architectures deliver significant performance-power (0.07 W per Giga-hash/sec) leading to 8K+ times more energy efficiency over standard GPUs [45]. In many applications, the non-recurring engineering costs turn ASIC design prohibitive. DSAs in other domains include wearable edge-computing to accelerate multimedia and sensor data analysis. Stitch [3] is a many-core architecture where tiny, heterogeneous, configurable and fusible accelerators, called polymorphic patches are effectively enmeshed with the cores. Evaluation results across wearable applications show an average of 2.3X enhancement in run-time compared to many-core processor baseline at a modest area and power overhead.

[8]https://developer.nvidia.com/cuda-zone

D. DSA Design Issues

We present a few proposals for DSAs built for deep learning applications. These proposals are contrasted using the following guidelines [44]:

1) **Dedicated memories:** data movement can create significant bottlenecks in the system. Dedicated memories placed as close as possible to the processing units can improve the system performance.
2) **Larger arithmetic units:** data processing benefits from more arithmetic units or bigger memories.
3) **Easy parallelism:** exploits the parallelism that matches the application domain.
4) **Smaller data size:** reduce data size and type to the simplest form needed for the application.
5) **Domain-specific language:** capitalise on the programming language and framework specifically used for the application development.

The design issues listed above provide a baseline for deep learning systems aiming at a speedup of 100 times for the tasks of model training and real-time inference. The performance-power metric has been used to benchmarking new design for applications either running in data centres (cloud-based tasks) or edge devices (edge-computing tasks).

E. Deep Neural Networks (DNNs) on DSAs

DSAs have emerged as promising accelerator systems to the tasks commonly explored in a deep learning data pipeline. Every year new DSA design proposals are put forward into the academic and industrial communities. This section focuses on a small sample of this space by briefly discussing recent proposals: (i) Brainwave Neural Processing Unit (NPU) and (ii) Tensor Processing Unit (TPU) for cloud-based model training and inference. A summary of these proposals already aligned to the DSA design issues is presented in Table I.

F. Brainwave Neural Processing Unit (NPU)

Real-time interactive applications started using deep neural networks (DNNs) for voice and image recognition. Low-latency and the ability to process small data are requirements to achieve a smooth user experience and data efficiency (power and cost effective issues). High data throughput GPUs

TABLE I
DESIGN ISSUES FOR DOMAIN SPECIFIC ARCHITECTURES (DSAS)

	Brainwave NPU	**PROMISE NPU**	**TPU**
Design target	Data centre ASIC/FPGA	Data centre ASIC/edge device	Data centre ASIC
1. Dedicated memories	DNN model weights in distributed on-chip SRAM	Single-bank with 512×256 bit-cell SRAM (single vector 128 elements). Multiple banks support multiple vectors of 128 elements	24 MiB Unified Buffer, 4 MiB Accumulators
2. Larger arithmetic unit	Up to 96,000 multiply-accumulator units		64 KiB Multiply-accumulators
3. Easy parallelism	Pipeline and SIMD	Single-threaded, SIMD	Single-threaded, SIMD
4. Smaller data size	Narrow precision data types (BFP FPs)	8-bit integer size	8-bit, 16-bit integer size
5. Domain specific language/framework	TensorFlow	Caffe/TensorFlow on Julia	TensorFlow

can explore large batches of data. Neural processing units (NPUs) have emerged as an alternative for real-time small-data applications. The problem is the extent a system is sufficiently flexible to support various types of DNNs with low-latency exploiting the parallelism inherently present in the individual requests in cloud-based real-time applications.

The Brainwave NPU [12] is a large scale production system for real-time A.I applications. The main system architecture features are: (i) soft processor with the logic synthesis in FPGAs, (ii) single-thread with SIMD parallelism, (iii) focus on the vector-matrix multiplication operation, (iv) DNN model embedded with weights stored in distributed on-chip SRAM. The proposed implemented microarchitecture explores multi-functional units (MFU) and matrix-vector multipliers, as depicted in Figure 15.

Different FPGA systems were used to implement the Brainwave NPU. Parameters were adjusted to select the most suitable system among the ones tested, namely Stratix V D5, Arria 10 1150 and Stratix 10 280.

1) Performance on RNNs: The Brainwave NPU was implemented on the 40-lanes FPGA model Stratix 10 280 that reaches a peak of 48 TFLOPS. DeepBench benchmarking [35] is a set of benchmarks that comprise representative layers from popular DNN architectures including LSTMs (h = hidden dimension and t = time steps). The validation focused on inference at small batch sizes. The Brainwave NPU FPGA implementation (termed *BW_S10*) was compared to a modern NVIDIA Titan Xp GPU in terms of latency and computing throughput.

Table in Figure 16 shows the raw TFLOPS and the execution latency of each DeepBench benchmark. The BW NPU can run all DeepBench benchmarks at under 0.425ms, achieving 22.6 effective TFLOPS for an LSTM with 2048 hidden dimensions and 25-time steps. This represents orders of magnitude advantage over the Titan Xp. Such a performance is in part attributed to the BW NPUs high peak TFLOPS at narrow precision, but more significantly, this is due to better hardware utilisation [12].

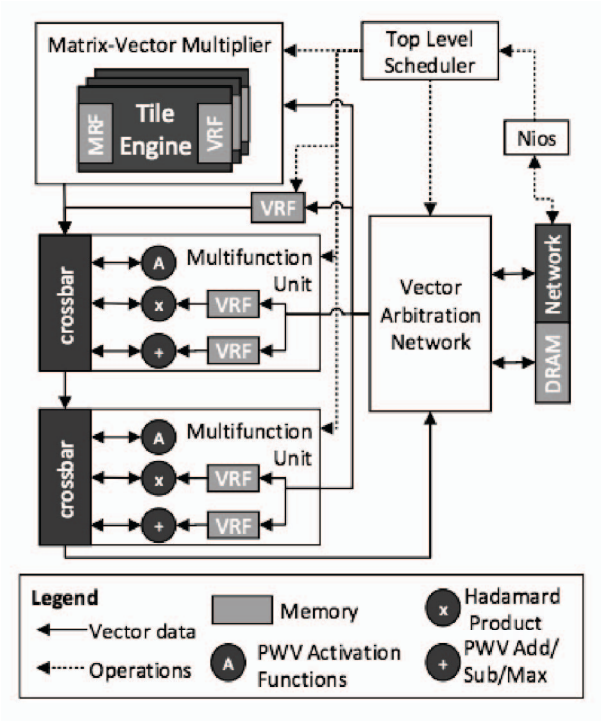

Fig. 15. Brainwave NPU Microarchitecture [12].

2) Performance on CNNs: Validation results obtained on a BW NPU variant customised to CNNs running a production-grade ResNet-50 image-based featuriser [36].

Table in Figure 17 compares the latency and throughput to run the ResNet-50-based featuriser standalone on a BW NPU hosted on an Arria 10 1150 against a high-end NVIDIA P40 GPU. The BW NPU achieves 555 inferences per second (IPS), a slightly higher performance compared to the high-end P40 using INT8 precision. On an unloaded system, the BW NPU serves a single instance of the model in 1.8 ms, while the P40 takes 2.17 ms. These results show that the BW NPU is a

LSTM h=2048 t=25	SDM	0.037	-	-
	BW	0.074	22.62	47.1
	Titan Xp	5.27	0.32	2.7
LSTM h=1536 t=50	SDM	0.043	-	-
	BW	0.145	13.01	27.1
	Titan Xp	6.20	0.30	2.5
LSTM h=1024 t=25	SDM	0.011	-	-
	BW	0.074	5.68	11.8
	Titan Xp	1.87	0.22	1.9
LSTM h=512 t=25	SDM	0.0038	-	-
	BW	0.077	1.37	2.8
	Titan Xp	1.26	0.08	0.7
LSTM h=256 t=150	SDM	0.0126	-	-
	BW	0.425	0.37	0.8
	Titan Xp	1.99	0.08	0.7

Fig. 16. LSTM Benchmarking Results [12].

	Nvidia P40	BW_CNN_A10
Technology node	16nm TSMC	20nm TSMC
Framework	TF 1.5 + TensorRT 4	TF + BW
Precision	INT8	BFP (1s.5e.5m)
IPS (batch 1)	461	559
Latency (batch 1)	2.17 ms	1.8 ms

Fig. 17. CNNs Benchmarking Results [12].

capable architecture for a single batch, low latency alternative to a high-end, newer generation GPUs on compute-intensive CNNs and orders of magnitude faster on RNNs [12].

G. Tensor Processing Unit (TPU)

TPU is Google's DSA based on a custom ASIC specifically designed for the tasks of deep neural networks training and online inference. The system was designed and validated in a variety of workloads including MLP, CNN and LSTM, which represent 95% of Google datacenter neural network application workloads [11]. The original goal was to improve cost-performance by 10X over general-purpose GPUs. The TPU system was designed, built and deployed in datacenters in record time (just within 15 months). The original TPU comprised a 256 x 256 8-bit MAC matrix multiply unit providing a peak throughput of 92 TeraOps/second (TOPS). Large software-managed on-chip memory (28 MiB) and a co-processor design allow the TPU to plug into existing servers just as GPU does. The host server sends TPU instructions for it to execute rather than fetching them itself. The instructions were custom defined for deep neural network data pipelines. The TPU instruction set architecture (ISA) includes instructions for host/memory data transfer, DNN weights management, matrix-matrix multiply and convolve operations and instruction for computing activation function. Table II presents the instructions specified in the TPU ISA.

A key feature of the TPU design is to keep the data as much as possible inside the system. Commonly used data transformers of the DNN data pipeline are implemented inside the TPU. Figure 18 shows the block diagram of the TPU system. The control and data paths can be summarised as

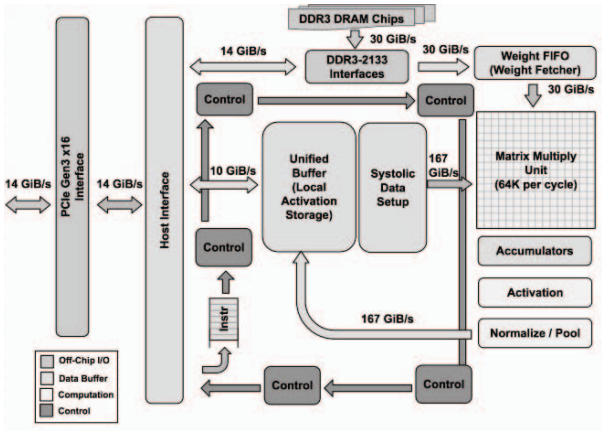

Fig. 18. TPU Design: Block Diagrams customised to DNNs [11].

follows: (i) the TPU instructions (Table II) are sent from the host CPU to the TPU instruction buffer via the PCIe Gen3 x16 bus (left side of Figure 18), (ii) the specific data transformers of the datapath start in the upper-right corner of Figure 18. The Matrix Multiply is the core unit of the system that contains 256x256 MACs (8-bit multiply-and-adds) capable of operating on signed or unsigned integers. (iii) The 16-bit products are collected in the 4 MiB (4096, 256-element) 32-bit Accumulators below the matrix unit. The matrix unit produces one 256-element partial sum per clock cycle. (iv) the Matrix Unit reads and writes 256 values per clock cycle and can perform either a matrix multiply or a convolution when using a mix of 8-bit weights and 16-bit activations (or vice-versa). (v) Finally, the weights for the matrix unit are fed into the on-chip Weight FIFO that reads from an off-chip 8 GiB DRAM called Weight Memory (for inference, weights are read-only; 8 GiB supports many simultaneously active models). The intermediate results are held in the 24 MiB on-chip Unified Buffer, which can serve as inputs to the Matrix Unit. A programmable DMA controller transfers data to or from CPU Host memory and the Unified Buffer.

Type	*MLP0*	*MLP1*	*LSTM0*	*LSTM1*	*CNN0*	*CNN1*	*GM*	*WM*
GPU	2.5	0.3	0.4	1.2	1.6	2.7	1.1	1.9
TPU	41.0	18.5	3.5	1.2	40.3	71.0	14.5	29.2
Ratio	16.7	60.0	8.0	1.0	25.4	26.3	13.2	15.3

Fig. 19. TPU Inference Performance for six DNN applications [11].

1) TPU System Validation: The workload was coded in the high-level TensorFlow framework. Six production neural network applications that represent 95% of Google's datacenters neural network inference demand (MLPs, CNNs, and LSTMs) have been used for the system validation (Figure 20). One DNN is RankBrain [37]; one LSTM is a subset of GNM Translate [38]; one CNN is Inception, and the other CNN is DeepMind AlphaGo [39]. The systems for comparison reasons used in the TPU validation include a mix of processing

TABLE II
TPU INSTRUCTION SET ARCHITECTURE (ISA) [11].

Instruction	Description
Read_Host_Memory	Reads data from the CPU memory into the TPU unified buffer
Read_Weights	Reads weights from the Weight Memory unit into the Weight FIFO as input to the Matrix Multiply Unit
MatrixMatrixMultiply/Convolve	Perform a matrix-matrix multiply, an element-wise matrix multiply, an element-wise vector multiply, a vector-matrix multiply, or a convolution from the Unified Buffer into the accumulators
Activate	Computes the non-linear function of the artificial neuron with options for ReLu, Sigmoid and others. It can also perform the pooling operations needed for convolutions using the dedicated hardware on the die, as it is connected to non-linear function logic.
Write_Host_Memory	Writes resulting data from the unified buffer into CPU host memory

power in configured servers: Intel Haswell (E5-2699 v3) CPU, NVIDIA K80 GPU and TPUs. Haswell has 18 cores, and the K80 has 13 SMX processors.

Name	LOC	Layers					Nonlinear function	Weights	TPU Ops / Weight Byte	TPU Batch Size	% of Deployed TPUs in July 2016
		FC	Conv	Vector	Pool	Total					
MLP0	100	5				5	ReLU	20M	200	200	61%
MLP1	1000	4				4	ReLU	5M	168	168	
LSTM0	1000	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1500	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1000		16			16	ReLU	8M	2888	8	5%
CNN1	1000	4	72		13	89	ReLU	100M	1750	32	

Fig. 20. Neural Network Applications [11].

Table in Figure 19 provides the relative inference performance per system, including the host server overhead for the two accelerators versus the CPU system. The next-to-last column shows the geometric mean of the relative performance for the six applications, which suggests the K80 is 1.1X the speed of a Haswell system, that the TPU is 14.5 times as fast, and thus the TPU is 13.2 times as fast as the GPU system.

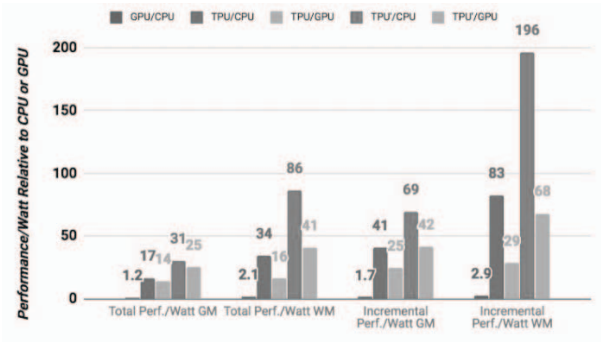

Fig. 21. TPU Inference Performance for six DNN applications [11].

2) Performance/Watt: Figure 21 shows the geometric (GM) and weighted mean (WM) performance/Watt for the K80 GPU and TPU relative to the Haswell CPU. Two different calculations of performance/Watt were done. The first (total) includes the power consumed by the host CPU server when calculating performance/Watt for the GPU and TPU. The second (incremental) subtracts the host CPU server power from the GPU and TPU beforehand.

For total-performance/Watt, the K80 GPU server is 1.2 - 2.1X Haswell CPU. For incremental-performance/Watt, when Haswell CPU server power is omitted, the K80 GPU server is 1.7- 2.9X. The TPU server has 17 to 34 times better total-performance/Watt than Haswell, which makes the TPU server 14 to 16 times the performance/Watt of the K80 GPU server. The relative incremental-performance/Watt which was the TPU's designers justification for a custom ASIC is 41 to 83 for the TPU, which lifts the TPU to 25 to 29 times the performance/Watt of the GPU.

Overall, the authors of [11] claim that a TPU is on average about 15X - 30X faster than its contemporary GPU or CPU, with TOPS/Watt about 30X - 80X higher. Moreover, using the GPUs GDDR5 memory in the TPU would triple achieved TOPS and raise TOPS/Watt to nearly 70X the GPU and 200X the CPU.

The results presented in the previous sections suggest that Domain-Specific Architectures (DSAs) are promising system design to fulfil the high demand for neural networks applications for both model training and online inference.

VIII. CONCLUSION

This paper is intended to offer a **companion material** to the tutorial **NVIDIA Deep Learning for Computer Vision, SIBGRAPI 2019**. It introduced a few selected topics in the field of *deep learning for computer vision* exploring the software and hardware interface for deep neural network applications. Systems used in GPUs and new DSAs are discussed with the perspective of advancing novel deep neural networks. The course is targeted to an audience who have not yet fully experienced deep learning in a practical computing environment. This should perhaps their first opportunity to train and validate deep neural networks for image-based classification and object detection. The set of tasks structured in customised Jupyter Notebooks[9] explore the deep learning workflows in NVIDIA GPUs.

ACKNOWLEDGEMENT

The authors wish to acknowledge the ongoing support received from the Deep Learning Institute - DLI as part of

[9]https://jupyter.org/

the NVIDIA University Ambassador Program (HQ San Jose - California). This initiative has been enabling FURG's teaching and research activities in deep learning, autonomous vehicles and IoT sensors.

REFERENCES

[1] Patterson, David and Andrew Waterman. 2017. "The RISC-V Reader: An Open Architecture Atlas.", Strawberry Canyon, First edition, November 2017. Free Portuguese Version by L.G Xavier, N. Formentin and M. Pias. URL: http://riscvbook.com/portuguese/

[2] Li Fei-Fei, R. Fergus and P. Perona, "One-shot learning of object categories," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 4, pp. 594-611, April 2006.

[3] Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. Y. "Zero-shot learning through cross-modal transfer". In 27th Annual Conference on Neural Information Processing Systems (NIPS 2013).

[4] Lowe, David G. et al. Object recognition from local scale-invariant features. In: ICCV. 1999. p. 1150-1157

[5] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In IEEE Conference on Computer Vision and Pattern Recognition, Vol. 1. IEEE, 886893.

[6] Wallach, Hanna M. "Topic modeling: beyond bag-of-words." Proceedings of the 23rd international conference on Machine learning. ACM, 2006.

[7] Doshi-Velez, Finale; Kim, Been. "Towards A Rigorous Science of Interpretable Machine Learning", https://arxiv.org/abs/1702.08608, November 2017.

[8] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. "ImageNet classification with deep convolutional neural networks". Communications of the ACM. 60 (6): 8490.

[9] Goodfellow, Ian; Benjio, Yoshua; Courville, Aaron. "Deep Learning", MIT Press, 2016.

[10] P. Srivastava, et al., "PROMISE: An End-to-End Design of a Programmable Mixed-Signal Accelerator for Machine-Learning Algorithms," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 2018 pp. 43-56.

[11] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, et al. "In-datacenter performance analysis of a Tensor Processing Unit," in Proc. 44th Annual International Symposium on Computer Architecture, pp.1-12. ACM, 2017.

[12] J. Fowers et al., "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in Proc. of ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, 2018, pp. 1-14.

[13] Amir Yazdanbakhsh et al. "GANAX: A Unified MIMD-SIMD Acceleration for Generative Adversarial Networks" arXiv:1806.01107

[14] Nandita Vijaykumar et al. "The Locality Descriptor: A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs", ISCA 2018.

[15] G. Lan, S. Lee, and Y. Zhou. "Communication-efficient algorithms for decentralized and stochastic optimization". arXiv preprint arXiv:1701.03961, 2017.

[16] A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. NIPS, 2011.

[17] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent". In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17).

[18] Quanshi Zhang, Ying Nian Wu, Song-Chun Zhu; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 8827-8836.

[19] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." In European conference on computer vision, pp. 818-833. Springer, 2014.

[20] E. Protas, J. D. Bratti, J. F. O. Gaya, P. Drews-Jr and S. S. C. Botelho, "Visualization Methods for Image Transformation Convolutional Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems.

[21] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, Kevin Murphy; The European Conference on Computer Vision (ECCV), 2018, pp. 19-34.

[22] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter. Neural Architecture Search: A Survey. Journal of Machine Learning Research 20 (2019) 1-21.

[23] Maryam, M. Najafabadi, Flavio Villanustre, Taghi M. Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. 2015. Deep learning applications and challenges in big data analytics. Journal of Big Data 2, 1 (2015), 121.

[24] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. 2016. YouTube-8M: A large-scale video classification benchmark. CoRR abs/1609.08675 (2016). Retrieved from http://arxiv.org/abs/1609.08675.

[25] ImageNet. 2017. Retrieved from http://image-Net.org. Accessed August 31, 2019.

[26] Krizhevsky, Alex, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Vol. 1. No. 4. Technical report, University of Toronto, 2009

[27] David H. Hubel and Torsten N. Wiesel. 1962. Receptive fields, binocular interaction and functional architecture in the cats visual cortex. Journal of Physiology 160, 1 (1962), 106154.

[28] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A Survey on Deep Learning: Algorithms, Techniques, and Applications. ACM Comput. Surv. 51, 5, Article 92 (September 2018), 36 pages. DOI: https://doi.org/10.1145/3234150.

[29] Kyunghyun Cho, Bart van Merrienboer, aglar Gulehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In The Conference on Empirical Methods in Natural Language Processing. 17241734.

[30] Xiangang Li and Xihong Wu. 2015. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 45204524.

[31] K. He et al., Deep Residual Learning for Image Recognition, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 770778.

[32] Goodfellow, Ian and Pouget-Abadie, Jean and Mirza, Mehdi and Xu, Bing and Warde-Farley, David and Ozair, Sherjil and Courville, Aaron and Bengio, Yoshua. "Generative adversarial nets."Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 26722680.

[33] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." The handbook of brain theory and neural networks 3361.10 (1995): 1995.

[34] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436.

[35] Narang, S., and G. Diamos. "Baidu DeepBench." (2017). Available online on https://github.com/baidu-research/DeepBench

[36] K. He et al., Deep Residual Learning for Image Recognition, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 770778.

[37] Clark, J. October 26, 2015, Google Turning Its Lucrative Web Search Over to AI Machines. Bloomberg Technology, www.bloomberg.com

[38] Wu, Y. et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, September 26, 2016, http://arxiv.org/abs/1609.08144.

[39] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587).

[40] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A Survey on Deep Learning: Algorithms, Techniques, and Applications. ACM Comput. Surv. 51, 5, Article 92 (September 2018), 36 pages. DOI: https://doi.org/10.1145/3234150

[41] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui and J. Collomosse, "Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask," 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T), Niteroi, 2017, pp. 17-41.

[42] Karen Simonyan and Andrew Zisserman. Networks for Large-scale Networks for Large-scale image recognition. In Proc. of ICLR 2015.

[43] KrizhevskyIlya, A. Sutskever Geoffrey E.. ImageNet Classification with Deep Convolutional Neural Networks December 2011Advances in neural information processing systems 25(2)

[44] John Hennessy and David Patterson, A new golden age for computer architecture: Domain-specific hardware/software co-design, en-

hanced security, open instruction sets, and agile chip development,"2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, 2018, pp. 27-29.
[45] M. Bedford Taylor, "The Evolution of Bitcoin Hardware," in Computer, vol. 50, no. 9, pp. 58-66, 2017.
[46] C. Tan, M. Karunaratne, T. Mitra and L. Peh, "Stitch: Fusible Heterogeneous Accelerators Enmeshed with Many-Core Architecture for Wearables," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, 2018, pp. 575-587.
[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
[48] David B. Kirk and Wen-mei W. Hwu. 2016. Programming Massively Parallel Processors, Third Edition: A Hands-On Approach (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.