Chapter 2
Deep Learning on Edge: Challenges and Trends

Mário P. Véstias
https://orcid.org/0000-0001-8556-4507
INESC-ID, ISEL, Instituto Politécnico de Lisboa, Portugal

ABSTRACT

Deep learning on edge has been attracting the attention of researchers and companies looking to provide solutions for the deployment of machine learning computing at the edge. A clear understanding of the design challenges and the application requirements are fundamental to understand the requirements of the next generation of edge devices to run machine learning inference. This chapter reviews several aspects of deep learning: applications, deep learning models, and computing platforms. The way deep learning is being applied to edge devices is described. A perspective of the models and computing devices being used for deep learning on edge are given, as well as what challenges face the hardware designers to guarantee the vast set of tight constraints like performance, power consumption, flexibility, etc. of edge computing platforms. Finally, a trends overview of deep learning models and architectures is discussed.

INTRODUCTION

Machine learning algorithms and, in particular, deep learning brought Artificial Intelligence to many application domains. In a deep learning workflow data is gathered and prepared for training the machine learning model. In the training step, deep learning models are trained with a large set of known instances so that they can classify new inputs not used during the training step. These trained models are then deployed for inference. Inference is when the trained model is used to classify new and unknown data instances.

Training is computationally heavy and requires high-performance computing platforms that still take hours or even days to train large deep learning models. Inference is orders of magnitude less demanding in terms of computation and can also be deployed in the same computing platform used for training. The common process is to use the high-performance computing platform for both training and inference. In many cases, data to be processed by the deep neural model is received from an edge device (any

DOI: 10.4018/978-1-7998-2112-0.ch002

hardware device that serves as an entry point of data and may store, process and/or send the data to a central server) and the inference result is sent back to the edge device. However, in a vast set of applications (security, surveillance, facial recognition, autonomous car driving, industrial, etc.) this round-trip method of doing inference is inefficient or unfeasible. Running the inference near the source of data is advantageous and in some cases necessary so that important information can be extracted in site and if necessary at real-time instead of sending data to the cloud and wait for the inference classification. Whenever the communication latency and data security violations are undesirable, like autonomous vehicles, local processing at the sensor is a requirement. In these cases, inference is done at the edge avoiding long data communications and high computing latencies. For these reasons, many deep learning tasks are migrating from the cloud of high-performance computing platforms to the low cost, low density embedded devices at the edge.

This brings new problems and open issues to the design of machine learning models at edge devices since running deep learning on edge is subject to different performance, memory and cost requirements then those considered by cloud computing design processes. Cloud inference is focused on delivering high performance inference with the highest model accuracy. Edge inference benefits from high accuracy models but achieving the highest accuracy is not the main metric. Cost, performance, energy, real-time, size are some of the most important design parameters considered when implementing computing platforms for edge inference on edge.

Deep learning on edge has been attracting the attention of researchers and companies looking to provide solutions for the deployment of machine learning computing at the edge. A clear understanding of the design challenges and the application requirements are fundamental to understand the requirements of the next generation of edge devices to run machine learning inference.

In this chapter several aspects of deep learning: applications, deep learning models and computing platforms, will be reviewed. Then the way deep learning is being applied to edge devices will be describes. A perspective of the models and computing devices that are being used for deep learning on edge will be given, what challenges are facing the hardware designers to guarantee the vast set of tight constraints like performance, power consumption, flexibility, etc. of edge computing platforms. Finally, a trends overview of deep learning models and architectures will be discussed.

BACKGROUND

Machine learning is a subfield of artificial intelligence whose objective is to give systems the capacity to learn and improve by its own without being explicitly programmed to do it. Machine learning algorithms extract features from data and build models from it so that new decisions and new outcomes are produced without being programmed a priori with these models and rules.

There are many types of machine learning algorithms with different approaches and application targets: Bayesian (Barber, 2012), clustering (Bouveyron et al., 2019), instance-based (Keogh, 2011), ensemble (Zhang, 2012), artificial neural network (Haykin, 2008), deep learning network (Patterson & Gibson, 2017), decision tree (Quinlan, 1992), association rule learning (Zhang & Zhang, 2002), regularization (Goodfellow et al., 2016), regression (Matloff, 2017), support-vector machine (Christmann & Steinwart, 2008) and others.

Different problems require different models and algorithms and so each of these algorithms apply to different types of data sets and applications. All these algorithms can be broadly classified accord-

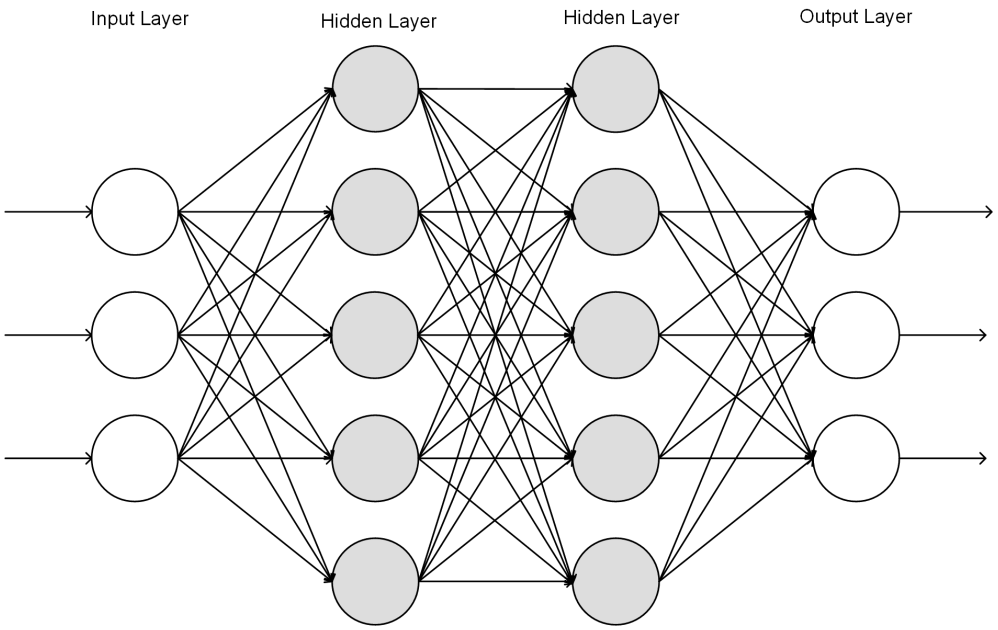
ing to the learning style: supervised, unsupervised and semi-supervised. Supervised machine learning algorithms (e.g., regression, artificial neural network and deep learning network) train the model of the algorithm with training data. Each instance of the training data has an associated label that identifies the expected result for each particular input. In the training process, the model is corrected and adjusted according to the expected outcomes. In the unsupervised class of algorithms (e.g., dimensionality reduction, k-means clustering, etc.), the training data do not have an expected outcome. The algorithms in this class try to extract features from the input data and cluster input data according to these features without any previous knowledge of the input data characteristics. The semi-supervised algorithms mixes both previous classes, that is, there is a desired outcome but the model must learn features to classify data.

Among the many machine algorithms this chapter is concerned with deep learning algorithms whose ground are artificial neural networks (ANN). ANNs are inspired by the structure of the human brain consisting of interconnected neurons. Theoretically, an ANN is a universal model capable to learn any function (Hornik et al., 1989). Deep learning is basically deep artificial neural networks with several and more complex layers designated deep neural networks. Since the introduction of deep learning that several models have been proposed, like convolutional neural network, recurrent neural network, deep belief network, deep Boltzmann machine, Kohonen self-organizing neural network, modular neural network and stacked auto-encoder.

Deep Learning

The grounds of deep learning models are artificial neural networks. Before proceeding with a description of deep neural networks the following section describes the fundamentals of ANNs

Figure 1. Artificial neural network



Artificial Neural Network

An artificial neural network (Haykin, 2008) consists of a basic structure known as perceptron or neuron organized in a series of layers. The first layer is the input layer, the last one is the output layer and all the other layers between the input and the output layer are known as hidden layers (see figure 1).

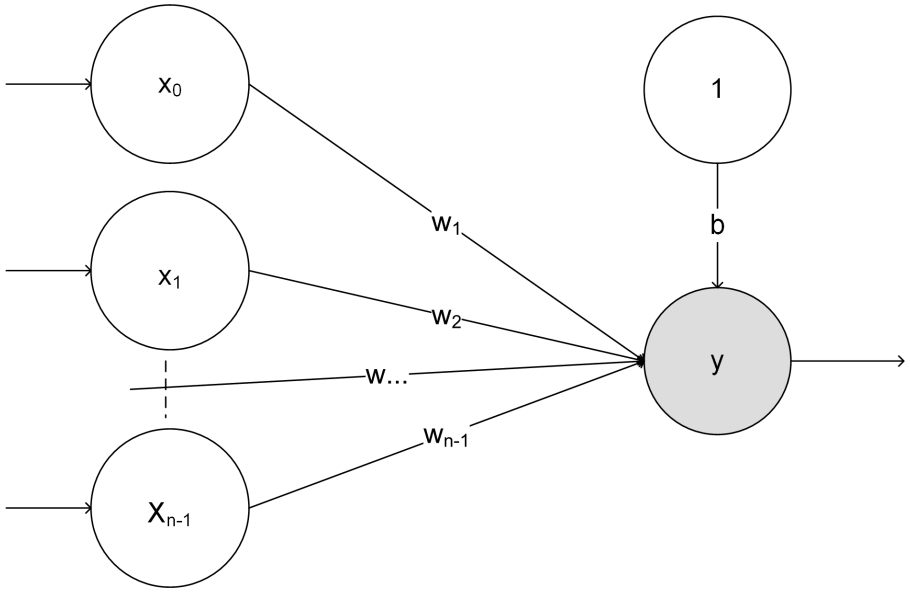
Neurons in the input layer receive input data and generate outcomes for the neurons of the first hidden layer, while the output layer receives the outcomes from the last hidden layer and produces the classification associated with the input data. These are feedforward networks which the underlying graph contains no feedback connections or cycles and are the focus of this chapter. Each neuron of an artificial neural network generate an output which is a function of all its inputs and sends it to all nodes of the next layer, except the output layer that does not have a next layer and so the outcomes of the neuron are the output results. The first artificial neural networks had only one hidden layer. Recently, this number has increased considerably and according to (Bengio, 2009) when a neural network has more than three layers is referred to as deep neural network.

A perceptron encodes *n* inputs, $\{x_1, x_2, x_3, \ldots, x_n\}$, from neurons of the previous layer using a vector of weights or parameters $\{w_1, w_2, w_3, \ldots, w_n\}$ associated with the connections between previous perceptrons and the target perceptron that determines the importance of the corresponding input to the perceptron being calculated. Each perceptron still has an additional bias value that is used to shift the output to better fit the data (see figure 2).

The output of a perceptron, *y*, its prediction value, is computed as a function of the weights and the bias:

$$y = f\left(b + \sum_{k=0}^{n-1} w_k x_k \right)$$

Figure 2. Perceptron or neuron



Function *f(.)* referred to as activation function determines the output of the neuron. In its simplest form the activation function is binary, that is, the output is inactive, '0', or active, '1'. While simple, requires that many neurons are used for a non-linear separation of classes. A linear function can be used instead that linearly rates the output between two values. These functions exhibit similar problems of a binary function and since the output is unbounded, it leads to an unstable convergence. Instead, normalized functions are used with better properties for classification and learning stability (Nwankpa et al, 2018).

One of the first activation functions was the sigmoid given by $\frac{1}{1+e^{-z}}$ that predicts the probability of the output with values between 0 and 1, and the hyperbolic tangent, $(e^x - e^{-x})/(e^x + e^{-x})$ that increases the output range to]-1, 1[. Many other activation functions were proposed during the last decades but one of the most recently used is the ReLU (Rectified Linear Unit) that is 0 if the output is less or equal than 0, and 1 otherwise. In (Glorot et al., 2011) it was shown that ReLU leads to better training of deep neural networks. Previous activation functions apply to a single set. In many models, the output layer provides outputs for multiple classes. To associate values to multiple classes the softmax function is used. This function takes as input a vector of *k* values and normalizes it into a probability distribution of *k* probabilities as follows:

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

The weights of a neural network model must be adjusted for each specific problem and for the best network accuracy. Determining all weights of the network for best accuracy is known as the training step. Training can be supervised, unsupervised or semi-supervised. In the supervised training the set of weights, *W*, is found with the help of an objective function that quantifies the error, *E*, between the measured outputs for a particular set of weights, y_m, and the expected outputs, t_m, through all *N* data inputs, x_n. The error is calculated as the sum of the squared error:

$$E(W) = \sum_{n=1}^{N}\sum_{m=1}^{M}\left(y_m(x_n, W) - t_m\right)^2$$

The training algorithm iteratively runs the forward propagation and the backpropagation steps. Starting with an initial set of weights, an input is applied at the input layer of the network and propagated until the output layer to find an output classification. The loss function, E(W), is then applied to determine the mean squared error between the obtained output and the required output. This finishes the forward propagation step.

Then the backpropagation step starts with the objective to adjust the weights to minimize the error function. This is achieved by propagating backwards the value of the loss function to all neurons that contribute to each output neuron. Neurons with a higher contribution to the loss function value of an output neuron receive a higher fraction of this value. When all neurons have received its loss fraction, weights are adjusted to reduce the loss. The adjustment of weights is done with the *gradient descent* (Ruder, 2016) technique as follows:

$$\Delta W[i] = -^{3}\left(\frac{\partial E_n}{\partial w[i]}\right)$$

In gradient descendent, weights are incrementally changed based on the derivative of the loss function and a learning rate, γ. It means that the loss function must be differentiable.

Unless the network is very simple, the model do not achieve 100% accuracy. Therefore, a criteria must be used to stop the training process. Normally the process stops when the accuracy improvement between two training iterations is below a certain threshold.

Gradient descent is an heuristic method and so it does not guarantee the global minimum. Finding a good local minimum depends on weight initialization. Some works have shown that better results can be achieved if the initial weights are randomly chosen within specific ranges (Glorot & Bengio, 2010).

Unsupervised training follows a different process since there are no expected outputs. The model is trained using extracted features from the input data. Semi-supervised training is a mix of both techniques where training is done with some labeled data and some unlabeled data.

Deep Neural Network

Deep neural networks (DNN) are an extension of the traditional artificial neural networks with more layers and different types of layers. Therefore in DNN each layer is trained with the output features extracted by the previous layer. So as the data progress through the network model more features are aggregated that represent more complex representations, like a hierarchy of features.

Unlike most traditional machine learning algorithms, deep learning is able to extract features from the input data without being explicitly programmed to do it. An important fact about deep learning is that the more data is used to train the network the better the accuracy, contrary to other machine learning algorithms. Since the size of the DNN can be freely increased it means that it can be applied to any complex classification problem with a high dimensionality of features.

The concept of a DNN with multiple layers is not new but its feasibility is recent. A neural network with many layers requires intensive computations to be trained. The required computational power is now possible with the recent high-performance computing platforms. To achieve high accuracy DNN also need a large set of data instances to train the network and it is a fact that today designers and developers have access to huge amounts of data to do it.

So, the accuracy of deep neural networks gets better with larger models and when trained with more data. The consequence is that both training and inference of DNN requires high memory size to store weights and feature maps and computational capacity to train it in reasonable times.

Since its introduction as a machine learning model, several types of deep neural networks were proposed differing mainly on how neurons are interconnected. In the following the most important types of DNNs will be briefly explained.

Types of Deep Neural Networks

Feed-Forward Neural Network is the traditional neural network model as explained in the previous section. All layers are dense and have the same structure. Dense layers means that all neurons of a layer connect to all neurons of the previous layer, except the first layer that receives inputs. Theoretically these

networks can model any relationship between inputs and outputs with enough hidden neurons but this may lead to impractical implementations and so different neural networks models are adopted.

Convolutional neural networks (CNN) were introduced in (LeCun et al., 1989) to image classification.

A feed-forward neural network can be naturally applied to classify image. The problem is that input pixels are modeled with input neurons and so, for reasonably sized images the number of neurons of the input layer is large which requires many parameters from the input to the first hidden layer. Considering images of size 128×128, the first layer would have 2^{14} neurons. Assuming the next layer with the same number of nodes, the first hidden layer would require 2^{28} weights. Since a deep neural networks is being used, this number easily increases to an order that turns the training process too hard.

Instead of using a neural network with dense layers, the interconnections between layers take into consideration the type of input data (LeCun, 1989). CNNs takes advantage of the spatial correlation between neighbor pixels to establish dependencies between neurons of different layers, that is, the output of a neuron is the result of the convolution between a small window of weights and the respective output of neurons of the input map. These layers are therefore designated convolutional layers. A CNN contains one or more convolutional layers. Each convolutional layer identify features of the image which are then correlated by the next convolutional layer to learn complex features.

The set of convolutional layers may be followed by one or more fully connected layers which are dense layers identical to those used in feed-forward layers. Since these layers interconnect all neurons of previous layers the complex features extracted so far are globally correlated.

Recurrent neural networks (RNN) were introduced in (Elman, 1990) and are basically dense networks with state. RNNs have a hidden state distributed through all neurons that allows them to store information of previous data. State information is updated in a non-linear way which permits the model to follow complex state sequences. This type of networks is very powerful but must be carefully designed to avoid the vanishing problem where weights converge to extreme values loosing previous information. RNNs have a vast set of applications including also those without an explicit association with a sequence of events. A picture, for example, can be processed as a sequence of pixels. A common application of RNNs is autocompletion where the information of a sequence is automatically determined.

The vanishing problem of RNNs happens because the state of the network is hard to keep for a long time. The **long-short term memory network** – LSTM - (Hochreiter, 1998; Hochreiter & Schmidhuber, 1997) reduces the vanishing problem of RNNs with the introduction of gates and an explicit memory to store states. The memory stores the state until a gate cell tells the memory to forget a state. LSTMs add a cell layer to remember information from a previous iteration of the model. With these improvements LSTM networks were able to execute complex tasks, like music composition.

RNN and LSTM networks may have an unpredictable behavior by following a non-deterministic path or oscillating. To overcome this instability problems the **Hopfield network** (HN) introduced in (Hopfield, 1982) can be used. The HN is the densest neural network since all neurons connect to all other neurons. The network is trained for a set of patterns and only these can be identified by the network, that is, for a particular input the network will converge to one of the stable patterns learned during training. The HN has been shown to be very limited in the number of patterns (15% of the number of neurons) it can learn because of the spurious minima. This limitation is associated with the fact that if two local minima correspondent to two training patterns are too close it may create a single local minima for both and therefore none of the two patterns will be memorized.

The **Boltzmann Machine** (BM) network is an unsupervised model that was introduced in (Hinton & Sejnowski, 1986). His model is similar to the Hopfield network but only considers input and hidden

neurons. After a network update during training the input neurons become output neurons. During the learning process, the BM maximizes the product of probabilities assigned to the elements of the training set. BM are used for dimensionality reduction, classification, feature learning, among others.

Deep Belief Network (DBN) introduced in (Bengio et al., 2006) are probabilistic generative models with multiple layers of the so called latent variables. The first two layers have undirected connections while the next layers have directed connections between layers. This type of networks is used recognize and generate images and videos.

The **autoencoder** (AE) network model was designed for unsupervised learning and is used to encode an input with a representation with less number of dimensions (Bourlard & Kamp, 1988). A decoder is then used to decode the data and obtain the original data. So they are basically used to reduce the size of the inputs into a compact representation.

Deep Neural Networks in Practice

From among the many different neural network models, CNNs have gained most of the attention due to its good image classification results better than other deep neural networks and because there has been an exponential increase of applications requiring image classification. In the following, some of the most known CNN for its results and novelty will be described.

LeNet (LeCun et al., 1995) proposed one of the first convolutional neural network for hand digit classification with high accuracy. The model accepts and classifies grayscale images of size 32×32 according to ten different classes representing the ten possible digits. The network has a total of 60K weights and an overall accuracy above 99%.

AlexNet was the first large CNN (Krizhevsky, 2012) with very good results for image classification. Compared to LeNet, AlexNet is deeper and has 1000× more weights than LeNet. The input images are also larger (227×227×3). AlexNet has top-5 error rates around 17.0% and a top-1 error rate of 37.5% when used to classify images from ImageNet.

In 2013 a multiplayer deconvolutional neural network – ZefNet - was proposed in (Zeiler & Fergus, 2013). The authors propose a methodology to help in the design of the network based on a process to observe the network activity of neurons (Erhan, 2009; Le, 2013). Following the methodology, they were able to improve AlexNet to a top-5 error rate of 11.2%.

One year later the VGG neural network (Simonyan & Zisserman, 2014) increased the size of CNNs published so far with 138 million parameters. VGG improved previous year top-5 error rate to 7.3%. In spite of several filter size improvements to reduce the number of parameters, VGG still have a huge number of parameters requiring long training times and high inference times.

In the same year, a new CNN - GoogleNet (Szegedy et al., 2014; Szegedy, 2016) - introduced a new layer that has groups of convolutions running in parallel within a module designated Inception. In the inception module, several convolutional layers run in parallel like a small neural network inside a larger model. With 6.8 million parameters GoogleNet achieved a top-5 error rate of 6.7% for ImageNet.

A very deep neural network - ResNet (He et al., 2015) - has increased the number of layers to 152 and achieved a top-5 error rate of 3.6% in the ImageNet contest. Similar to GoogleNet, ResNet includes a new block named *Residual Block* where the output map of a series of two convolutional layers are added to the input of the block. An optimization of ResNet was proposed in (Xie et al., 2017) with the ResNeXt CNN with a top-5 error of 3.03%. Another improvement of ResNet was proposed in (Huang,

2018) - DenseNet - with a similar accuracy but with about half of the parameters. This was possible with a modification of the residual block where a layer has dependencies with all previous layers.

Recently, SENet (Hu, Shen & Sun, 2018) introduced a new network block based on the residual block – squeeze-and-excitation that emphasizes important features and cancel less useful ones. SENet won the ILSVRC competition in 2017 with a top-5 error rate of 2.25%.

Deep Learning on Edge

Deep learning algorithms are very demanding in terms of memory resources to store weights and computing power for training and inference. For these reasons, deep learning networks run of high-performance computing platforms. Data collected from edge sensors are sent to high-performance computing centers to be processed and when required the result is sent back to the edge to be presented or to help taking decisions.

Today edge devices are almost everywhere in a large set of applications in industrial environments, automotive, surveillance and security cameras, drones, satellites, medical equipment and new applications appear every day. All these devices collect an enormous amount of data to be processed by the central high-performance computing platform. This reduces the complexity and the energy required for edge devices. However, for an increasing set of applications, the processing of collected data to take decisions has to be done near the sensor for several reasons: unreliable channel transmission, large communication round-trip delay, real-time processing, security and privacy of data.

Hence, data processing algorithms and decision taking is migrating from the cloud to the edge, in particular deep learning models. The problem is that to run an inference of a deep learning network requires high computing and memory resources, scarce resources on an edge computing device. Even with the increasing capacity of edge and mobile devices, it is still a challenge to run deep neural networks within the embedded constraints of the device.

Until recently, the main concern of DNN designers was to achieve the best accuracy. With the advent of deep learning on edge, in the design and development of DNN accuracy is traded-off by energy, cost, computing resources and several other metrics associated with edge computing devices.

Two lines of research are being followed for the optimization of deep neural networks on edge: model optimization and optimized computing platforms for edge deep learning.

Model Optimization

As became evident from the neural networks describe above, the trend is to consider more layers and more weights. Sometimes, a new neural network is proposed that optimizes a previous one with the reduction of parameters but without ever reducing accuracy. DNN for edge and mobile processing reduces the number of computations and weights with eventually a slight reduction in accuracy. MobileNet (Howard et al, 2017) is a CNN for mobile devices that reduces the number of parameters by manipulating the kernels. It applies a single kernel to each input map and then combines the convolution outputs with a pointwise convolution. This leads to a reduction in the number of parameters and consequently the number of computations. Other optimizations were considered, like reducing the number of input and output maps and the image resolution. Different networks with different trade-offs were implemented achieving accuracies from 50% to 70%, a number of parameters from 0.5 to 4.2 millions and a number of multiply-accumulate (MAC) operations ranging from 41 to 559 millions. MobileNetv2 (Sandler et

al., 2018) is an optimization of MobileNet that introduced some optimizations that reduce the number of parameters about 30% and the number of operations about 50% and still achieving higher accuracy.

ShuffleNet (Zhang et al., 2018) is another CNN proposed for mobile devices. Different convolutions are applied to separate parts of the input maps to reduce the number of operations. The output of convolutions are then shuffled so that the information from different groups can be mixed. The model has a complexity similar to MobileNetV2 but with better accuracy.

The optimizations proposed in the previous neural network models somehow changes the deep learning algorithm by considering different number and types of layers trying to reduce the number of parameters while keeping the accuracy. A different approach consists of reducing the complexity of the model at a lower level. In this approach, two types of techniques have been considered: data quantization and data reduction. Data quantization consists of techniques to reduce the arithmetic complexity and the number of bits used to represent parameters and activations. Data reduction is the set of techniques used to reduce the number of parameters or the volume of data transferred to the computing platform.

A commonly used data quantization technique is the conversion from single-precision floating-point representations to half-precision floating-point (Micikevicius et al., 2017) or 8-bit floating-point (Wang et al., 2018), fixed-point or integer representation. One advantage of this conversion is that the new representations are easier to implement and calculate than single-precision floating-point arithmetic. Also, with data represented with less bits the arithmetic operators are also less complex and so many operators can be implemented with the same silicon area.

Several works have shown that neural network models can use weights and activations represented with only 16 or even 8 bits and still keep accuracies close to the accuracy obtained with data represented with single-precision floating-point (Gysel et al., 2016), (Gupta et al., 2015), (Anwar et al., 2015), (Lin et al., 2016). Neural network models with weights and activations represented with a single bit have also been proposed - BNN (Binary Neural Networks) (Courbariaux et al., 2016; Umuroglu et al., 2016). BNN reduce considerably the bitwidth of data at the cost of some accuracy degradation. To reduce the impact of binarization over the network accuracy the number of weights must be considerably increased. Also, instead of using a binary representation, some works consider 2-bits to reduce the impact of the representation over the accuracy (Ubara et al., 2016).

A different approach to reduce the data volume of the network is to remove and compress data. In (Han et al., 2015) a DNN is compressed with punning and Huffman coding. Pruning is a process that removes some connections between neurons. For example, a reduction of about 90% of the weights belonging to the dense layers have a very small impact over accuracy. The disadvantage of pruning is that it introduces sparsity in the matrix of weights which complicates its implementation in hardware. When applied to dense layers, pruning is more efficient then when applied to convolutional layers, because the number of weights in dense layers is generally much higher than the set of all remaining weights.

Another technique to reduce the effects of large transfers of weights in fully connected layers is batching (Zhang et al., 2016). The batching technique stores several output feature maps of the last non-dense layers before being executed. It permits to reuse the same kernel for different input images.

Computing Platforms for Edge Deep Learning

Different technologies are available to deploy deep learning algorithms on edge devices. The right device depends on the design requirements, including delay, latency, area, energy, cost, flexibility, etc. Chips or devices for artificial intelligence at the edge try to optimize energy and performance efficiencies,

that is, get the lowest energy consumption and enough performance to run a DNN model within design constraints.

General-purpose processors (CPU) can run any deep learning model and their programmability permits them to run any new DNN model without any modifications to the computing platform. The problem with CPUs is that they have a low energy and performance efficiency. GPUs (Graphics Processing Unit) are one of the most used platforms for training DNN because they are a many-core architecture with massive computing parallelism offering high-performance computing and at the same time offer a high level of programmability. They are energy and performance efficient but have high energy consumption which is infeasible for most edge platforms due to their restrictions on available energy. The highest performance and best energy efficiency is achieved with ASICs (Applications Specific Integrated Circuit). ASICs have limited programmability because the algorithm implementation is hardwired in silicon. Some hardware programmability can be considered at the cost of extra silicon to implement extra computing modules that are chosen according to the target algorithm to be implemented. FPGAs (Field-Programmable Gate Array) are more flexible than ASICs since the hardware can be reconfigured for new and different functions but are harder to reprogram than CPUs or GPUs. SoCs (Sytem-on-Chip) FPGAs are an attractive option to run deep learning models since they contain a general-purpose processor tightly connected to reprogrammable logic. The reprogrammable logic is used to design and implement the most time-consuming operations of DNNs, while the CPU is used to control the system, to run the less frequent operations and allowing the implementation of new functions whose hardware implementation is inefficient.

Most commercial solutions are based on ASICs since they provide the best solutions in terms of performance and energy consumption. Some companies provide IP (Intellectual Property) cores as DNN accelerators to be integrated in a computing system, while others provide full SoC solutions implemented on chip.

High-performance IP processors are common approaches to run machine learning algorithms. DesignWare EV6x (Synopsis, 2017) is an IP processor for vision processing on embedded devices. It consists of a 32-bit processor, a vector DSP and a dedicated accelerator for CNNs. The accelerator supports many CNN models, including regular and irregular CNNs, like GoogleNet, and supports 8 and 12 bits data quantization. The whole core has a peak performance of 4.5 TMACs with 2 TMACs/W.

DNA (Cadence, 2017) is another IP SoC processor from Cadence designed for the acceleration of deep neural networks on edge devices. The core integrates a Tensilica DSP, and the DNN accelerator. The architectures optimize the execution of the algorithm using techniques like zero-skipping (multiplications with zero are not computed), pruning, data compression and decompression. The core can be configured with different number of MACs and each MAC can be configured with different data representations (8 or 16 bits integer or 16-bits floating-point). The configuration with the highest performance has a peak performance of 12 TMACs with 3.7 TMACs/W.

NeuPro (Linley Group., 2018) is also an IP core for machine learning to be deployed in embedded devices for advanced driver-assistance systems, surveillance systems, among others. The core is a SoC with an accelerator that can execute any layer of a CNN and a vector processing unit and to control the accelerator and to run other functions not supported by the accelerator. The IP core is configurable in terms of number of MACs. MACs are configurable for the execution of MACs with different data sizes (8 or 16). The smallest configuration of the IP has a performance of 2 TOPs and the larger one has a performance of 12.5 TOPs.

In (Gyrfalcon Technology - 2018) a many core architecture with 168 processing units, each with local memory and a MAC unit, was proposed for audio and video processing, including deep learning networks. The core element of the architecture is an engine to speed-up matrix processing. With an operation frequency of 300 MHz the chip delivers 16.8 TOPs with a consumption of 700 mW corresponding to a power efficiency of 24 TOPs/W.

Movidius Myriad X processor (Intel, 2017) is a SoC vector processor with an accelerator for DNN inference at the edge. The MAC units support 16-bit fixed- and floating-point operations and 8-bit fixed-point. The accelerator has a peak performance of 1 TOPs and the whole processor has a total peak performance of 4 TOPs.

ASIC offer the best solutions but with a reduced flexibility. Deep learning networks are still in its infancy and therefore are constantly being modified and improved. Therefore, deploying an ASIC solution for deep learning is always a risk. Turning the ASIC architecture more flexibility reduces its silicon efficiency which reduces performance and increases energy consumption. These aspects open the set of available platforms for edge computing to reconfigurable devices. Coarse and fine-grained solutions were already proposed to run inference in low cost devices.

Eyeriss (Chen et al., 2017) is a coarse-grained reconfigurable accelerator for CNNs. It contains 168 processing elements connected with a network-on-chip (NoC). The NoC is configurable to adapt the dataflow of the architecture to the dataflow of the model to run. The architectures uses compression and decompression to reduce the data volume between the chip and external memory. With an operating frequency, the accelerator was tested with the inference of AlexNet with data quantized to 16-bits fixed-point has an energy efficiency of 166 GOPs/W with a measured average power of 278 mW.

DNPU is another coarse-grained reconfigurable processor (Shin et al., 2017) for CNNs and RNNs. The chip has dedicated units to the execution of convolutional layers. Pooling and activation function are executed by a centralized module shared by all convolution modules. Dense layers are implemented with a dedicated unit for matrix multiplications and multipliers can be configured (4, 8 or 16 bits fixed-point). The architecture with 4-bit multipliers has a peak performance of 1.2 TOPs with an energy efficiency of 3.9 TOPs/W.

DRP is a dynamically coarse-grained reconfigurable core to accelerate embedded machine learning algorithms (Fujii et al., 2018). The core has an array of dynamically reconfigurable processing elements. Both 16-bit fixed- and floating-point and binary precisions are supported. Dynamic reconfigurability is used to support large networks by reconfiguring the architecture for different layers at execution time. The chip achieved a performance near 1 TOPs.

Fine-grained reconfigurable devices, FPGAs, permit to optimize the hardware architecture for each particular deep learning model (Sze et al., 2017). The first FPGA implementations had the sole objective of improving performance and therefore considered high density FPGA devices (Shawahna et al., 2019). Now, with the necessity to deploy DNNs on edge devices, small to medium density FPGAs are also considered (Guo et al., 2018; Venieris et al., 2018). Recently, a solution was proposed to execute large CNNs in low density FPGAs (ZYNQ XC7Z020) with a peak performance of 400 GOPs (Véstias et al., 2018). With 8-bit fixed-point quantization, the architecture explores several levels of parallelism and proposes a method to run convolutions independently of the size of the convolution window. With all these optimizations, the architecture has a peak performance around 400 GOPs and an energy efficiency near 50 GOPs/W.

FUTURE RESEARCH DIRECTIONS

Deep learning models have improved during the last years. Typically, good accuracies are only achieved with large models. However, the evolution of DNN models have shown that with appropriate techniques it is possible to reduce the complexity of the models with a negligible accuracy loss. New models are needed that emphasize performance and energy efficiency. Binary neural networks are promising solutions with a great impact over hardware complexity and memory storage but still requires a lot of improvements to avoid large accuracy degradations.

Training and inference are still two separated steps. Training is done in high-performance computing platforms and the results are used by the same platform or by an edge device. Considering that an edge device is constantly receiving new data, these could be used to dynamically train the network to keep improving accuracy. Incremental training is executed on high-performance platforms, but the process could be also implemented in the edge device for the same reasons enumerated before.

Designing neural network models for specific problems is still an empirical process that leads to oversized networks with redundant parameters. It is important to better understand how particular layers and neurons influence the final accuracy and how to redesign the model so that the best accuracy is achieved. This will help to tailor models for specific applications, which is particularly important for edge devices.

Concerning the computing platforms, its design is somehow influenced by the fact that DNNs are still under constant research and evolution. Several ASIC solutions already exist but the risk is high since the architectures are optimized for particular neural networks. Any improvements or changes to the original network model reduces the efficiency of the ASIC solution since new functions or modules have to be executed by general purpose processors.

Reconfigurable architectures help us to overcome some of the limitations of ASICs since the hardware architecture can be upgraded on-board with new modules and/or operations. It is the only platform whose hardware can follow the constant evolution of DNNs. FPGAs allow optimized implementations of binary neural networks contrary to architectural solutions based only on CPUs or GPUs. A major problem of FPGA devices is that it is difficult to design them compared to implementations based on software only. Specific frameworks to automatically map neural networks on FPGA already exist but the results suffer some degradation compared to a hand-made design. The proliferation of reconfigurable devices as solutions for deep learning on edge depends on the availability of tools to automatically map neural network models on FPGA.

Given the heterogeneity of layers in high accuracy neural network models, it is important to consider flexible architectures with dedicated accelerators for the common operations requiring massive parallelism integrated with a high-performance processor that can execute the remaining operations or functions whose execution cannot be done by the accelerator. SoCs with a processor and dedicated hardware are the most appropriate solutions for these cases. Many of the ASIC architectures for deep learning proposed so far consider a SoC architecture. An example of this trend is the recently announced FPGA for deep learning (Xilinx, 2018) with software programmable processors, fine-grained reconfigurable hardware and an intelligent device for tasks associated with deep learning. The new FPGA upgrades previous devices with a new engine for deep learning inference.

Inference is still the only operation executed in deep learning platforms in the edge. However, the possibility to train or retrain a network in the edge opens the possibility of constant learning whenever new data is collected. Training still requires full precision and its computational complexity is much

higher than that of inference. This mixed precision requires new architectures that can perform both training and inference with different data representations.

CONCLUSION

Deep learning algorithms have successfully improved the accuracy results of many machine learning algorithms. The set of applications that take advantage of these algorithmic improvements is increasing. Many of these applications are associated with edge devices and therefore running deep learning models on edge devices is now a major challenge.

This article describes the fundamentals of deep learning and known deep learning models proposed in the literature. Most of these models are only concerned with accuracy and only a few are optimized for mobile and edge computing. Neural network models for edge devices must be optimized even if this implies some accuracy degradation traded-off by lower energy consumption and improved execution times. Two main classes of optimizations have been applied so far: data quantization and data reduction. These reduce memory and computing requirements and in some cases without accuracy degradation.

ASICs and FPGAs are the most appropriate technologies for edge inference since they offer good energy and performance efficiencies. These metrics are better with ASICs, but FPGAs offer hardware flexibility to optimize the implementation of new neural network models. A brief description of recent commercial chips and published FPGA implementations were also given in this chapter.

REFERENCES

Anwar, S., Hwang, K., & Sung, W. (2015). Fixed point optimization of deep convolutional neural networks for object recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing* (pp. 1131–1135). 10.1109/ICASSP.2015.7178146

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127. doi:10.1561/2200000006

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.). In *Proceedings of the 19th International Conference on Neural Information Processing Systems* (153-160). Cambridge, MA.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, *59*(4-5), 291–294. doi:10.1007/BF00332918 PMID:3196773

Bouveyron, C., Celeux, G., Murphy, T., & Raftery, A. (2019). *Model-Based Clustering and Classification for Data Science: With Applications in R (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge, UK: Cambridge University Press. doi:10.1017/9781108644181

Cadence: Tensilica. (2017). DNA Processor IP For AI Inference.

Chen, Y., Krishna, T., Emer, J. S., & Sze, V. (2016). Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, *52*(1), 127–138. doi:10.1109/JSSC.2016.2616357

Christmann, A., & Steinwart, I. (2008). *Support Vector Machines*. Springer-Verlag.

Courbariaux, M., & Bengio, Y. (2016) BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. In CoRR, abs/1602.02830.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, *14*(2), 179–211. doi:10.120715516709cog1402_1

Erhan, D., Bengio, Y., Courville, A., & Vincent, P. (2009). Visualizing higher-layer features of a deep network. *Univ. Montr.*, *1341*, 1.

Fujii, T., Toi, T., Tanaka, T., Togawa, K., Kitaoka, T., Nishino, K., ... Motomura, M. (2018). New generation dynamically reconfigurable processor technology for accelerating embedded AI applications. In *Symposium on VLSI Circuits* (41-42). 10.1109/VLSIC.2018.8502438

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics* (249–256).

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (315-323).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Guo, K., Sui, L., Qiu, J., Yu, J., Wang, J., Yao, S., ... Yang, H. (2018). Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(1), 35–47. doi:10.1109/TCAD.2017.2705069

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015) Deep Learning with Limited Numerical Precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning:* Vol. 37. (1737–1746).

Gysel, P., Motamedi, M., & Ghiasi, S. (2016). Hardware-oriented Approximation of Convolutional Neural Networks. In *Proceedings of the 4th International Conference on Learning Representations*.

Han, S., Mao, H., & Dally, W. J. (2015). "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". *CoRR*, abs/1510.00149.

Haykin, S. (2008). *Neural Networks and Learning Machines* (3rd ed.). Pearson.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *Multimedia Tools and Applications*, *77*, 10437–10453.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart, J. L. McClelland, & CORPORATE PDP Research Group (Eds.), Parallel distributed processing: explorations in the microstructure of cognition. Vol. 1, MIT Press (282-317).

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, *6*(02), 107–116. doi:10.1142/S0218488598000094

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *8*(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735 PMID:9377276

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, *79*(8), 2554–2558. doi:10.1073/pnas.79.8.2554 PMID:6953413

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. doi:10.1016/0893-6080(89)90020-8

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861.

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-Excitation Networks, In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition* (7132-7141). IEEE.

Huang, G., Liu, Z., Maaten, L., & Weinberger, K. (2018). Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized Neural Networks. In D. D. Lee, M. Sugiyama, I. Guyon, & R. Garnett (Ed.), Advances in Neural Information Processing Systems: Vol. 4107–4115. *Curran Associates, Inc*.

Intel. (2017). Movidius Myriad X VPU.

Keogh, E. (2011). Instance-Based Learning. In C. Sammut, & G. I. Webb (Eds.), *Encyclopedia of Machine Learning*. Boston, MA: Springer.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Adv. Neural Inf. Process. Syst. 1–9.

Le, Q. V. (2013). Building high-level features using large scale unsupervised learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing* (8595–8598). 10.1109/ICASSP.2013.6639343

LeCun, Y. (1989). Generalization and network design strategies. In Connectionism in Perspective.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551. doi:10.1162/neco.1989.1.4.541

LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., ... & Vapnik, V. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. In Neural networks: the statistical mechanics perspective, 261-276. Mech. Perspect.

Lin, D. D., Talathi, S. S., & Annapureddy, V. S. (2016). Fixed Point Quantization of Deep Convolutional Networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. Vol. 48. (pp. 2849–2858).

Linley Group. (2018). Ceva NeuPro Accelerates Neural Nets.

Matloff, N. (2017). *Statistical Regression and Classification: from Linear Models to Regression* (1st ed.). Chapman and Hall. doi:10.1201/9781315119588

Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., García, D., … Wu, H. (2017). Mixed Precision Training. *CoRR*, abs/1710.03740.

Nwankpa, C., Ijomah, W., Gachagan, A. & Marshall, S. (2018). Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *Corr*. abs/1811.03378.

Patterson, J., & Gibson, A. (2017). Deep Learning: A Practitioner's Approach. O'Reilley Media, 1st ed.

Quinlan, R. (1992). *C4.5: Programs for Machine Learning* (1st ed.). Morgan Kaufmann.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. In CoRR.

Sandler, M. B., Howard, A. G., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (4510-4520). 10.1109/CVPR.2018.00474

Shawahna, A., Sait, S. M., & El-Maleh, A. H. (2018). FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access: Practical Innovations, Open Solutions*, *7*, 7823–7859. doi:10.1109/ACCESS.2018.2890150

Shin, D., Lee, J., Lee, J., & Yoo, H. (2017). 14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *IEEE International Solid-State Circuits Conference* (240-241). 10.1109/ISSCC.2017.7870350

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *arXiv preprint arXiv:1409.1556*.

Synopsys DesignWare. (2017). EV6x Vision Processors.

Sze, V., Chen, Y., Yang, T., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, *105*(12), 2295–2329. doi:10.1109/JPROC.2017.2761740

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, (2818-2826). 10.1109/CVPR.2016.308

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2014). Going Deeper with Convolutions. *arXiv:1409.4842*.

Gyrfalcon Technology. (2018). Lightspeeur 2803S Neural Accelerator.

Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P. H. W., Jahre, M., & Vissers, K. A. (2016). FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In CoRR, abs/1612.07119.

Venieris, S. I., & Bouganis, C. (2018). fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(2), 326–342. doi:10.1109/TNNLS.2018.2844093 PMID:29994725

Véstias, M. P., Duarte, R. P., Sousa, J. T., & Neto, H. C. (2018). Lite-CNN: A High-Performance Architecture to Execute CNNs in Low Density FPGAs. In *28th International Conference on Field Programmable Logic and Applications* (pp. 393-399). 10.1109/FPL.2018.00075

Wang, N., Choi, J., Brand, D., Chen, C., & Gopalakrishnan, K. (2018). Training Deep Neural Networks with 8-bit Floating Point Numbers. *CoRR* abs/1812.08011.

Xilinx, V. (2018). The first adaptive compute acceleration platform (acap).

Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. arXiv. vol. 30 (pp. 225–231).

Zhang, C., & Ma, Y. (2012). *Ensemble Machine Learning*. New York: Springer-Verlag. doi:10.1007/978-1-4419-9326-7

Zhang, C., Wu, D., Sun, J., Sun, G., Luo, G., & Cong, J. (2016). Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster. In *Proceedings of the International Symposium on Low Power Electronics and Design* (pp. 326–331). 10.1145/2934583.2934644

Zhang, C., & Zhang, S. (2002). Association Rule Mining. In *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018) ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6848–6856). 10.1109/CVPR.2018.00716

ADDITIONAL READINGS

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. New York: Springer Verlag.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, *11*, 625–660.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., & Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. *In International Conference on Artificial Intelligence and Statistics* (153–160).

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In *International Conference on Computer Vision*.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554. doi:10.1162/neco.2006.18.7.1527 PMID:16764513

Jarrett, K., Kavukcuogl, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *International Conference on Computer Vision* (2146–2153). 10.1109/ICCV.2009.5459469

Kalinowski, I., & Spitsyn, V. (2015). Compact Convolutional Neural Network Cascade for Face Detection. *CoRR*, abs/1508.01292.

Lawrence, S., & Giles, C. Lee, Tsoi, Ah C. & Back, A. (1997). Face Recognition: A Convolutional Neural Network Approach. In IEEE Transactions on Neural Networks. 8 (1): 98–113.

LeCun, Y., Kavukvuoglu, K., & Farabet, C. (2010). Convolutional networks and applications in vision. In *International Symposium on Circuits and Systems* (253–256).

Lei, T., Barziley, R., & Jaakkola, T. (2016). Rationalizing Neural Predictions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (107-117). 10.18653/v1/D16-1011

Matsugu, M., Mori, K., Mitari, Y., & Kaneda, Y. (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, *16*(5–6), 555–559. doi:10.1016/S0893-6080(03)00115-1 PMID:12850007

Pengcheng, Y., & Neubig, G. (2017). A Syntactic Neural Model for General-Purpose Code Generation, In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vol. 1 (440-450).

Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. In *IEEE Conference on Computer Vision and Pattern Recognition* (6517-6525).

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition". In *International Conference on Artificial Neural Networks* (92–101). 10.1007/978-3-642-15825-4_10

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated Residual Transformations for Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. 10.1109/CVPR.2017.634

Zeiler, M. D. & Fergus, R.. (2013). Visualizing and understanding convolutional networks. *Computing Research Repository*, abs/1311.2901.

KEY TERMS AND DEFINITIONS

Activation Function: The activation function defines the output of a neuron given a set of inputs from the previous layer or data input.

Artificial Neural Network (ANN): It is a computing model based on the structure of the human brain with many interconnected processing nodes that model input-output relationships. The model is organized in layers of nodes that interconnect to each other.

Autoencoder: An unsupervised learning network and is used to encode an input with a representation with fewer dimensions.

Boltzman Machine: An unsupervised network that maximizes the product of probabilities assigned to the elements of the training set.

Convolutional Layer: A network layer that applies a series of convolutions to a block of input feature maps.

Convolutional Neural Network (CNN): A class of deep neural networks applied to image processing where some of the layers apply convolutions to input data.

Deep Belief Network: A probabilistic generative model with multiple layers of the so called latent variables tha keep the state of the network.

Deep Learning (DL): A class of machine learning algorithms for automation of predictive analytics.

Deep Neural Network (DNN): An artificial neural network with multiple hidden layers.

Edge Device: any hardware device that serves as an entry point of data and may store, process and/or send the data to a central server.

Feature Map: A feature map is a 2D matrix of neurons. A convolutional layer receives a block of input feature maps and generates a block of output feature maps.

Fully Connected Layer: A network layer where all neurons of the layer are connected to all neurons of the previous layer.

Hopfield Network: A dense neural network where all neurons connect to all other neurons.

Long-short Term Memory Network: A variation of recurrent neural networks to reduce the vanishing problem.

Machine Learning: A subfield of artificial intelligence whose objective is to give systems the ability to learn and improve by its own without being explicitly programmed to do it.

Network Layer: A set of neurons that define the network of a CNN. Neurons in a network layer are connected to the previous and to the next layer.

Perceptron: The basic unit of a neural network that encodes inputs from neurons of the previous layer using a vector of weights or parameters associated with the connections between perceptrons.

Pooling Layer: A network layer that determines the average pooling or max pooling of a window of neurons. The pooling layer subsamples the input feature maps to achieve translation invariance and reduce over-fitting.

Pruning: An optimization technique for deep neural networks that removes some connections between neurons to reduce the complexity of the network.

Recurrent Neural Network (RNN): A class of deep neural networks consisting of dense networks with state.

Semi-Supervised: A training process of neural networks that mixes supervised and unsupervised training.

Softmax Function: A function that takes as input a vector of k values and normalizes it into a probability distribution of k probabilities.

Supervised Training: A training process of neural networks where the outcome for each input is known.

Unsupervised Training: A training process of neural networks where the training set does not have the associated outputs.