# Deep Learning for Edge Computing: Current Trends, Cross-Layer Optimizations, and Open Research Challenges

Alberto Marchisio[1,*], Muhammad Abdullah Hanif[1,*], Faiq Khalid[1], George Plastiras[2], Christos Kyrkou[2], Theocharis Theocharides[2], Muhammad Shafique[1]
[1]*Technische Universität Wien (TU Wien)*, Vienna, Austria
[2]*University of Cyprus (UCY)*, Nicosia, Cyprus
{alberto.marchisio, muhammad.hanif, faiq.khalid, muhammad.shafique}@tuwien.ac.at,
{gplast01, kyrkou.christos, ttheocharides}@ucy.ac.cy

***Abstract*—In the Machine Learning era, Deep Neural Networks (DNNs) have taken the spotlight, due to their unmatchable performance in several applications, such as image processing, computer vision, and natural language processing. However, as DNNs grow in their complexity, their associated energy consumption becomes a challenging problem. Such challenge heightens for edge computing, where the computing devices are resource-constrained while operating on limited energy budget. Therefore, specialized optimizations for deep learning have to be performed at both software and hardware levels. In this paper, we comprehensively survey the current trends of such optimizations and discuss key open research mid-term and long-term challenges.**

***Index Terms*—pre-processing, pruning, quantization, DNN, accelerator, hardware, software, performance, energy efficiency, low power, deep learning, neural networks, edge computing, IoT.**

## I. Introduction

Deep Neural Networks (DNNs) have become popular due to the (1) availability of large datasets, (2) accessibility of hardware resources for compute-intensive workloads (like GPGPUs) and (3) open-source Deep Learning libraries. Nowadays, they are widely used in several applications like image classification [59], detection [58] and segmentation [66]. Usually few years are required between the invention of a novel DNN algorithm and its successful hardware deployment, as for the case of [34] and [67] implemented in [28]. To achieve high efficiency products, optimizations at different levels of abstractions are required.

DNNs undoubtedly perform better the larger and deeper they are, but *this effect demands a continuously increased complexity on the hardware perspective to design specialized accelerators for Deep Learning*. Currently, there are several use-case scenarios where hardware acceleration is beneficial for DNNs: (1) offline DNN training in data centers, (2) inference in data centers, (3) online learning on mobile devices and (4) inference on mobile devices. While the discussed techniques are beneficial for multiple scenarios, in this paper, we focus mostly on the 4th scenario. Moreover, besides the hardware acceleration, it is extremely important to *start from an algorithm which is highly-optimized at the software level*, e.g., by reducing the number of DNN inferences through pre-processing, and reducing the computations for each inference through pruning and quantization. Although these kind of optimizations are transparent while considering inference in edge devices, they are beneficial to achieve several order of magnitudes of energy improvements. If matched with specialized hardware accelerators and optimized dataflows, these improvements will grow further.

After discussing the *main scientific questions and challenges* (**Section I-A**), we survey the *current trends of deep learning for edge computing* (**Section II**). We then present *our methodology* (**Section III**) of different cross-layer optimizations, supported by *case study analyses* (**Section IV**), before raising questions for future deep learning hardware practitioners about the *security* (**Section V**) and other *open research challenges* (**Section VI**). An overview of our work is depicted in Figure 1.

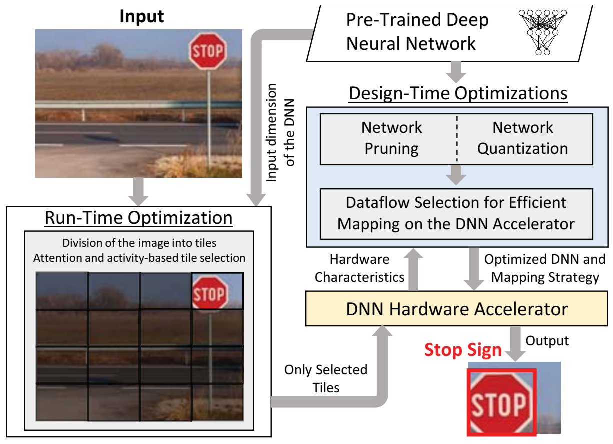
Input
Pre-Trained Deep Neural Network
Input dimension of the DNN
Design-Time Optimizations
Network Pruning
Network Quantization
Dataflow Selection for Efficient Mapping on the DNN Accelerator
Run-Time Optimization
Division of the image into tiles
Attention and activity-based tile selection
Hardware Characteristics
Optimized DNN and Mapping Strategy
DNN Hardware Accelerator
Only Selected Tiles
Stop Sign
Output

**Fig. 1:** Our Flow for Cross-Layer Optimizations for Deep Learning.

### A. Key Scientific Questions and Associated Challenges

**Low Power & Memory Budget:** Performing DNN inference on edge devices, which are typically resource- and power-constrained, is a challenging task. For example, the ResNet-50 [23] requires more that 95MB of memory to store the weights and more than 3.8 billion multiplications to process a single image. *Such amount of processing is infeasible to be deployed in edge devices to return real-time results*.

**Latency:** While mobile voice recognition applications like Apple Siri, Amazon Alexa and Google Assistant have the processing based on the cloud, for other critical applications (e.g., autonomous vehicles, drones, and wearable healthcare devices) the near-sensor processing is necessary to get a fast response from the DNN, as well as due to privacy and security reasons. Moreover, not only latency, but also security and privacy issues motivate near-sensor processing. Therefore, *specialized hardware accelerators are required to efficiently perform the DNN inference at the edge to meet the latency, security, and privacy requirements*.

**Accuracy vs. Speed and Efficiency:** High accuracy DNNs are extremely computational and memory intensive. Even though some of the recent trends hint towards designing DNNs with small memory footprint [26] [25], the most promising approach is to compress the DNN by parameter pruning, sharing and quantization. Several *dense* DNN accelerators have been proposed, but to facilitate compression optimizations like pruning, *sparse* DNN accelerators can achieve better results in terms of efficiency.

**Redundant Operations:** DNNs usually contain several redundant operations, like multiplications with zero, and correlated inputs in

*Alberto Marchisio and Muhammad Abdullah Hanif have equal contributions.

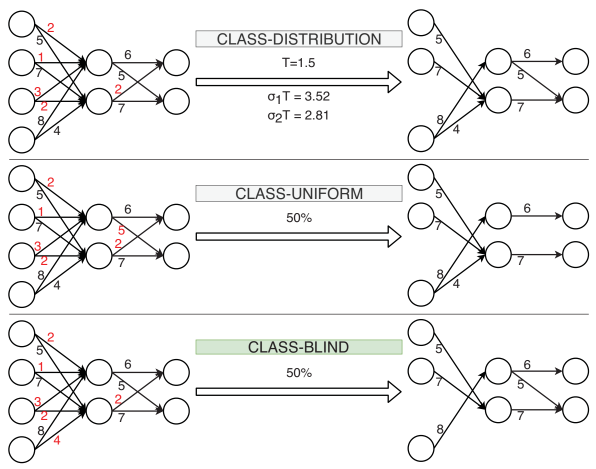

**Fig. 2:** Different magnitude-based pruning schemes.

streaming applications. Therefore, we do not necessarily need to process the complete set of inputs at every stage. *A challenging task is finding these redundancies and eliminating them efficiently*.

**Memory:** Significant cycles and energy may be required for memory data transfer to/from the computational array, which necessitates efficient memory architectures and data organization strategies for DNNs hardware. Moreover, following the in-memory computing trends, memristor devices allow to use resistive memories for analog computation, with additional cost of ADC/DAC overhead. *Is a CMOS-based design with traditional memory hierarchy the optimal solution for DNN processing, or should we adopt in-memory computing for deep learning at the edge?*

**Security:** Due to outsourcing of training and data dependencies, DNNs possess several security vulnerabilities that can be exploited to perform security attacks, e.g., adversarial examples, backdoors and data poisoning, for confidence reduction (ambiguity in classification), random or targeted misclassification and model stealing. These security vulnerabilities raise fundamental challenges like model privacy and secure execution of DNNs, regarding ensuring the robustness of DNN-based systems. Traditionally, the pre-processing, data encryption and watermarking are used, however, all these defenses can be neutralized by sophisticated model stealing or black-box attacks. *Therefore, there is a dire need to develop more sophisticated and efficient defenses to ensure model privacy and secure execution of the DNNs.*

## II. Current Trends

DNN compression is an attractive solution to reduce the complexity of a given network. The work of [14] proposed a 3-step method (pruning, quantization and encoding) to significantly reduce the memory footprint of a given DNN. **Network pruning** was first used in [10] to reduce the number of connections. Several different pruning methodologies have been explored in the literature Different magnitude-based pruning methods are shown in Figure 2. Structured pruning [75] employs constraints on some DNN parameters (e.g., kernel, filter, channel) to maintain a certain structure. Another approach is to prune the redundant and least significant weights, regardless of the structure of the DNN itself [15] [45], and share the weights to reduce the dimensionality [14]. Other compression methods, based on variational dropout [44], knowledge transfer [24] and low-rank approximations [70] are promising as well. On the other hand, techniques which are focusing on **reducing the precision**, like quantization [79] [71], binarization [54] and approximate computing [4] [18] have to leverage the trade-off between accuracy and efficiency.

**Hardware Accelerators:** The optimizations at the software level should be supported by specialized hardware accelerators in a co-design fashion [47] [19]. Recent advances in the datacenter computing deep learning [27] have inspired accelerators for edge devices. Specialized accelerators like [5] [28] exploit the concurrency and the parallelism available in the processing of the DNNs, especially for convolutional leyers, while [20] takes care also of the fully-connected layers. These architectures, however, accelerate *dense* DNNs, and cannot exploit the sparsity introduced by pruning. Therefore, *specialized accelerators for sparse DNNs are required* [13] [52]. Challenging aspects of these accelerators are flexibility, reconfigurability and data reuse [35] [39] [65]. Moreover, particular types of DNNs, like CapsuleNets [60] and GANs [11] present several differences in the computation patterns, as compared to traditional DNNs. These challenges are addressed by their specialized accelerators. For example, CapsAcc [46] adopts a data reuse policy to efficiently process the routing-by-agreement algorithm on a systolyc array-based accelerator for CapsuleNets, and GANAX [76] propose a unified MIMD-SIMD design for concurrent execution of GANs.

**Optimizations for Object Detection:** DNNs have been used successfully in a variety of tasks such as classification and detection. Numerous detectors have been proposed by the deep learning community, including Faster R-CNN [58], R-FCN [7], YOLO [56] and SSD [40]. These object detectors are separated into two main categories: *1) Region-based detectors*, a two-stage approach, with a region proposal stage followed by a classifier, and *2) Single-shot detectors*, consist of a single Convolutional Neural Network (CNN) trained to perform object detection. Region-based detectors use a region proposal method, such as Selective Search algorithm, to produce regions-of-interest (RoIs) for object detection. These RoIs are then warped into fixed size images and feed into a CNN network one-by-one. This process is time consuming due to the large number of RoIs that can be extracted ( 2000) and processed by a single CNN. Considering a typical $1000 \times 600$ image, there will be roughly 20000 potential RoIs per image, where different methods, such as non-maximum suppression (NMS), is applied on the proposed RoIs to reduce their count to 2000.

Single-shot detectors such as YOLO [56], have shown significant potential, especially for resource-constrained applications, compared to region proposal approaches by trading accuracy with real-time processing speed. To this end, *single-shot detectors avoid the multi-stage process by processing the whole image at once*. The detector receives an input image, resizes the image based on the CNN input size and then splits the input image into a grid, where for each grid it generates bounding boxes and class probabilities based on the number of objects. Thus, *the whole image is processed only once*, which makes this approach faster than the region proposal based approach.

## III. Cross-Layer Optimizations for Deep Learning Systems

Combining the current trends for targeting the above-discussed scientific challenges, we propose a methodology to apply optimizations across different software and hardware layers. The flow of our cross-layer methodology (shown in Figure 3), can be summarized in the following key steps:

- **Software-Level Optimizations:** The software-level optimizations mainly include *network pruning* (Step-1 in Fig. 3) and *quantization* (Step-2 in Fig. 3) of the parameters. Network pruning is usually performed iteratively, where, in each iteration, a small number of

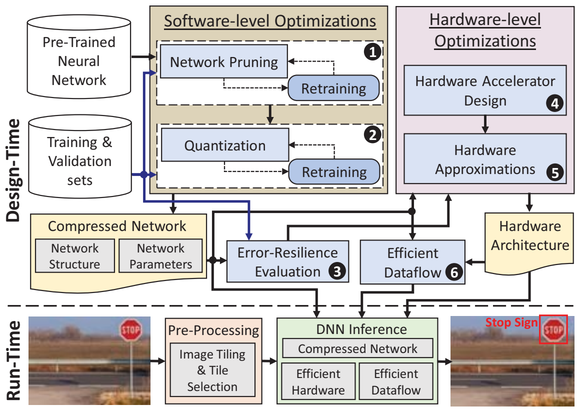

**Fig. 3:** Cross-Layer Optimizations for Deep Learning.

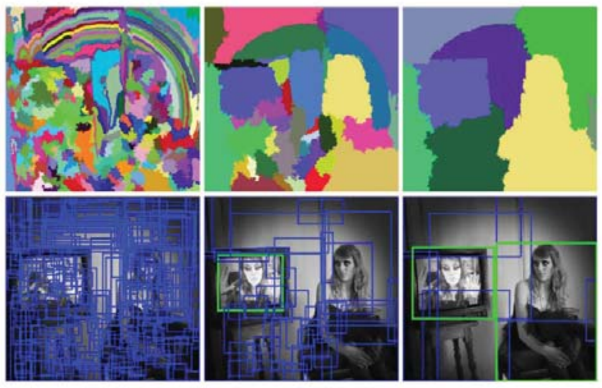

**Fig. 4:** Selective Search approach proposed in [72].

parameters are removed and the resultant network is fine-tuned by retraining for a limited number of *epochs*. The accuracy of the network is then analyzed and, on the basis of the accuracy loss, a decision is made whether to continue the pruning process or to proceed to the quantization phase. Once the network is pruned, an effective quantization level is selected for the weights and the activations of the network. The accuracy loss in the quantization phase is also compensated by fine-tuning the network parameters. Note, although here pruning and quantization are listed as two independent processes, they can be combined in a single loop where the level of both optimizations (i.e., pruning and quantization) is selected jointly. One such work, where both pruning and quantization are enclosed in a single loop, is CLIP-Q [71].

- **Hardware-level Optimizations:** An efficient hardware architecture (Step-4 in Fig. 3) can process the compressed network in a highly energy- and performance-efficient manner. This phase requires a thorough knowledge of the dataflow and the reuse of different parameters of the network. Once the hardware architecture has been finalized, different types of approximations can be applied (Step-5 in Fig. 3) to further improve its energy/power efficiency. The level and type of approximation are based on the error-resilience of the compressed network to different types of errors (Step-3 in Fig. 3). Once the hardware has been designed, a highly efficient dataflow is decided based on the network characteristics and the hardware architecture (Step-6 in Fig. 3).
- **Run-Time Optimizations:** Based on the application, certain optimizations can also be employed at run-time to reduce the number of samples to be processed. For example, in case of object detection application [58], a high-resolution image can be divided into multiple smaller images (known as tiling) and a selection criterion can be applied to select images with high activity regions. This process enables us to design DNNs which accept smaller inputs and thus are more computationally and latency-wise efficient.

By exploiting the above optimizations, we can significantly improve the efficiency of complex and inherently resource-intensive DNNs.

## IV. Optimizations Case Studies

### A. *Selective Data Processing During Inference*

A general trend of DNN design has been to add more layers and designing deeper models to get better accuracy, without considering the memory or power budget. Thus, it is challenging in many cases to use existing DNN models on resource-contrained enviroments without developing specialized DNN models for edge inference. For example, a basic DNN model, such as AlexNet [34], consists of five convolutional and three fully connected layers, and contains a total of 61 million parameters. Recent approaches propose the design of DNN models from scratch using different optimizations, such as successive smaller $3 \times 3$ convolutions to approximate the receptive field of $7 \times 7$ filters, along with a deeper network [67] leading to a decrease of the number of parameters. Our previous work [36] shows that by parameter space exploration in the design of a DNN based on a specific application, such as an Unmanned Aerial Vehicle (UAV), it is possible to design an efficient architecture that can perform vehicle detection up to $40\times$ faster with minimal impact on the accuracy.

There are different pre-processing techniques aiming to increase the accuracy of a detector while reducing the processing time. For region based approaches, a brute-froce approach used for object detection is the sliding windows approach [8] [22] [74]. Using windows of varied sizes and aspect ratios, to detect different object types at different viewing distances, a window is slided on the input image from right to left, and from up to down, to identify objects using classification. Each windows produced by the previous step, is warped into a fixed size image and fed into a CNN classifier to extract a large amount of features. Moreover, an SVM classifier is used to identify the class and the bounding box of the particular object.

A more sensible way to select regions of the image, is to use a region proposal method to extract RoIs for object detection. The work in [72] proposed a selective search algorithm based on hierarchical grouping, starting with each individual pixel as its own group. For each group, the texture is calculated. Afterwards, the groups that are the closest with respect to the texture are combined. First, smaller groups are grouped together, and the region merging continues until everything is combined together as shown in Fig. 4 (see details in [72]). Although these methods have shown remarkable accuracy results, they often lead to increased execution time and power consumption, especially when they are deployed on edge devices.

On the other hand, for single shot detectors a pre-processing technique that has been widely used is resizing the input image in fed into the DNN. This technique basically is a key step to reduce the processing time along with the processed data. Most DNNs are trained on datasets that involve low-resolution images with considerably large objects with large pixel coverage. The trained models perform really well on those types of input data, and the resizing has no impact on the accuracy of the DNN. When these models are tested on high-resolution images they yield significantly lower accuracy. In particular, when a high resolution image is resized there is a significant loss of information, where the pixels representing

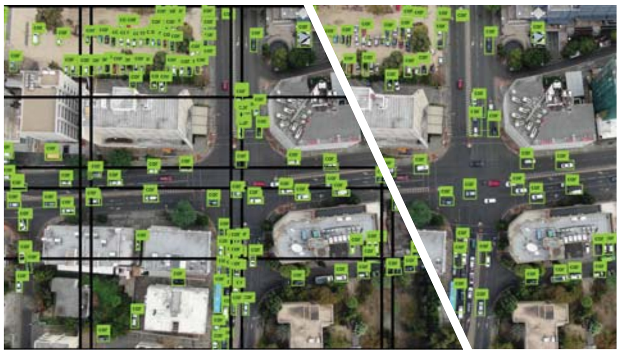
**Fig. 5:** Comparison between predictions using DroNet [36] with STP [53] (Left) and Resizing (Right).

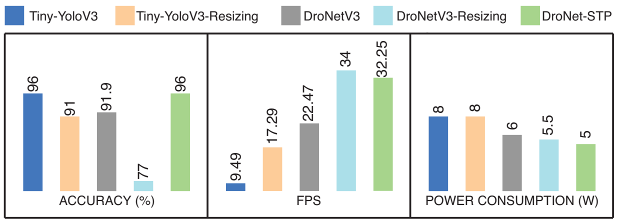

**Fig. 6:** Accuracy, frames-per-second (FPS), and power consumption using different DNNs with original, resized image and images with STP.

an object are reduced, leading to an accuracy reduction.

To this end, we proposed a Selective Tile Processing (STP) [53] approach where the input image is separated into smaller regions, called tiles, in order to avoid resizing the input image and to maintain the object resolution. Given an input image and the DNN input, the number of tiles are extracted and are uniformly distributed across the input image while maintaining a constant overlap between the tiles. Fig. 5 shows how the tiles are distributed across a high-resolution image using $512 \times 512$ tiles compared to resizing the image to $512 \times 512$ before fed into the DNN. It is clear that the localization of the object and the accuracy are better using the tiling approach. On the other hand, there is a notable increase of the number of images that need to be fed into the DNN for processing. Given the image on Fig. 5, there are 12 tiles extracted compared to the original image, leading to a $12\times$ increase of the processing time for each frame.

To solve the aforementioned problem, we propose two mechanisms that utilize statistical information gathered over time in order to select only a few tiles for processing, while keeping track of the activity in non-processed tiles. A *Memory Mechanism* that is able to use prior information from previous processed frames aiming to store and load the position of previously detected targets in an efficient way. Using the Intersection Over Union (IoU) metric, we are able to categorize each detected bounding box as new or already detected object. Moreover, with the use of a memory buffer, we are able to have an estimate of the position of the objects in a frame without having to again process the specific tile that contains the object.

The second mechanism is an *Attention Mechanism*, which selects the tiles that need to be processed by the DNN on the next frame. Using both memory and attention mechanism, we combine gathered information from previous frames, such as the number of objects detected in each tile, to select one or more tiles for processing by selecting the tiles with the highest weighted score presented in [53].

Fig. 6 shows the impact that resizing and tiling has on the accuracy of different DNNs trained and evaluated on the same pedestrian dataset tested on Jetson TX2. Note, by avoiding the resizing step, and feeding the original image to the DNN, there is an increase in the accuracy of each detector. On the contrary, the performance in terms of Frames-per-Second (FPS) is decreased along with the Power Consumption which makes this method suitable for exploration for edge devices. Our Selective Tile Processing technique shows that with minimal impact on the performance of an existing DNN ($-3FPS$), there is an increase of $20\%$ of the accuracy and $1.5\times$ decrease of the power consumption when implemented on an edge device.

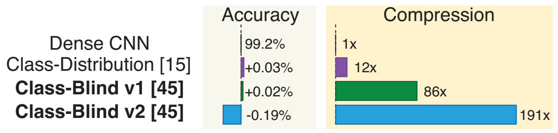

**Fig. 7:** Comparison of Pruning Methods for the LeNet-5.

## B. Software-Level Optimizations at Design Time

For model compression, among the pruning techniques, the *magnitude-based pruning* method is the most common one, since it is effective in terms of memory savings, and at the same time relatively simple to implement. Iteratively pruning and retraining the sparse network guarantees to maintain a good level of accuracy, while significantly reduces the memory footprint. Although this approach requires more retraining iterations, it is beneficial to reduce the computations of the sparse model at the inference stage, i.e., amenable to the edge devices. In our previous work in [45], we presented a class-blind pruning method for achieving significant DNN sparsity. Figure 7 presents the tradeoff between accuracy and compression achieved by pruning. The results have been measured, as a case study using the LeNet-5 on the MNIST dataset. The compression gains achieved at this stage, however, are not completely reflected into gains during inference. Indeed, some additional memory for storing the locations of nonzero weights/parameters in the sparse matrix must be allocated. Even though a structured pruning scheme usually requires less overhead than unstructured pruning, sparse coding schemes like Compressed Sparse Row/Column (CSR, CSC) [14] have been demonstrated to be effective.

To further reduce the complexity of the network, without accuracy loss, a *fixed-point quantization* can be applied. Our previous work in [19] shows that 12-bit fixed-point is the optimal design choice for the computations during inference, because the quantized DNN has the same accuracy as the original one. Further reduction to an 8-bit representation (or even to a lower bit-width) can be obtained, but it may require re-training to compensate for the accuracy loss depending upon the application requirements and the type of DNN used. More aggressive type of quantizations include binary [54] and ternary [80] quantizations where the weights are represented using 1 and 2 bits, respectively. However, these methods require complex training algorithms, which can ensure reasonable accuracy even when the weights have been quantized to 1 or 2-bits.

As an example, Figure 8 shows how to chose Pareto-optimal points of the design space for Class-Blind pruning and Fixed-Point quantization, by analyzing the Pareto-frontiers between accuracy and efficiency. Following this design flow, we obtain a compressed DNN. Alternatively, another approach is described in the work of [71], where pruning and quantization are combined in a parallel fashion before retraining the network.

## C. Hardware-Level Optimizations

DNNs require massive computations, which are typically difficult to deploy at the edge. Most of the computations involved are matrix

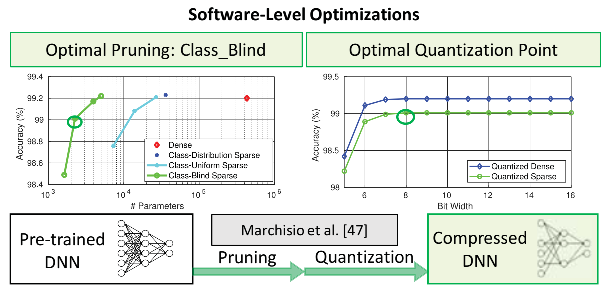

**Fig. 8:** Combining Pruning and Quantization as software-level optimizations.

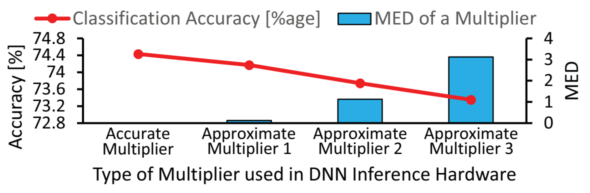

**Fig. 9:** Effects of approximations in the multipliers of a DNN inference hardware on the classification accuracy of the LeNeT network when used for the CiFAR-10 dataset [16]. MED represents Mean Error Distance of a multiplier computed using uniform input distribution.

multiplications, which require multiply-and-accumulate (MAC) units. A MAC multiplies the weight and the activation, and updates the partial sum. Depending upon the architecture of the accelerator and the dataflow mapping strategy (e.g., Weight Stationary, Output Stationary, Row Stationary [1] [5]), different data reuse scenarios can be exploited, like weight, input activation, and output activation reuse [20], for convolutional and fully-connected layers.

A DNN hardware accelerator can further benefit from the network sparsity, introduced by pruning. For this reason, specialized designs for handling relative indexing and skipping multiplications in which one of the terms is equal to zero. The load imbalance problem can be mitigated by the utilization of queues [13]. The dataflow (e.g., PlanarTiled-InputStationary-CartesianProduct-sparse [52]) has to manage the coordinates of all the nonzero weights, input and output activations. Moreover, the support of different bit-widths can be handled by having flexible-size processing elements [65].

**DNN Inference at the Edge: Accurate or Approximate?** DNNs are considered to be inherently error-resilient [38] and, therefore, can leverage approximate computing for achieving significant efficiency gains at the cost of minor accuracy loss, that may be compensated through re-training. The efficiency gain-per-unit accuracy drop depends on the error-resilience of the DNNs, which also depends on the type of application and other characteristics of the DNNs [38]. Applications like image classification, which generate only one output (i.e., class of the image) per input sample are considered to be more error-resilient as compared to the applications like object detection, which produce more sophisticated output. Various techniques based on fault/noise injection have been proposed to evaluate the error-resilience of the DNNs [55] [18]. These techniques help in quantifying the amount of approximation that can be applied in a DNN.

Approximations can be employed both at the hardware and the software level. However, here we mainly talk about hardware-level approximations, because the pruning and the quantization techniques, perfect examples of software-level optimizations/approximations, have already been discussed in Section IV-B. Hardware-level approximations include architecture- and circuit-level simplifications. These types can further be classified into data, and functional approximations [63], where *data approximations* refer to approximations in data storage [61] (i.e., memories) and *functional approximations* refer to approximations in the functionality of the processing units [17] [62] [57].

In memories, aggressive voltage scaling is one of the most prominent approaches which can lead to significant efficiency gains. Towards this, Kim et al. [33] proposed MATIC, a memory-adaptive training approach that enables aggressive voltage scaling of accelerator weight memories. Most of the works towards hardware-level approximation in DNN-based systems have been carried out in approximating computational modules of the DNN accelerators, like adders and multipliers. Few of the prominent works include [73] [78] [49] [50]. To highlight what impact of approximations in the multipliers used for DNN inference, Fig. 9 shows how the accuracy of the LeNeT network for the CiFAR-10 dataset decreases when the approximation level of the multipliers is increased. To cater the accuracy loss due to approximations, the work in [73] proposed to incorporate approximations in the forward pass of the training process to tune the network for the introduced approximations.

All the above-mentioned approaches result in some accuracy loss and, therefore, can hardly be used in any safety-critical application because of their stringent accuracy constraints. To address this, we proposed CANN [16], an approach where curable approximations are applied in the system such that approximation errors introduced by one module are completely cured by the subsequent module/s while ensuring efficiency gains of approximate computing.

**In-Memory Computing:** The main operations in state-of-the-art DNNs are vector-matrix and matrix-matrix multiplications, which are highly data intensive. Therefore, the memory access latency and access energy can potentially become the critical bottlenecks. In-memory computing and near-memory computing have emerged as promising paradigms for addressing such bottlenecks. Several architectures have been proposed which make use of ReRAM crossbars for realizing in-memory computing, i.e., performing computations where the data is stored. PRIME [6] reported around 895x efficiency gains in the overall energy consumption of the accelerator compared to the then state-of-the-art. PIPELAYER [68] proposed a hardware architecture for improving the overall throughput of ReRAM crossbar based accelerators. However, there are some practical issues associated with ReRAM crossbars when used for computations which limit the offline trained networks to perform as expected on such these accelerators. To address these issues, recently, a device variability-aware training methodology has also been proposed in [42], which trains a network while adding stochastic noise in the parameters of the network. The noise is modeled based on the variation-characteristics of the hardware and, therefore, helps in maintaining high accuracy even when there are significant variations in the network parameters because of the device variations.

## V. Machine Learning Security

Recently, security for machine learning, especially in DNNs, has become one of the prime challenges to ensure robustness of DNN-based systems. This is because these systems are highly vulnerable to data poisoning, model stealing and adversarial example [64] [21]. In this section, we present a brief overview of the recent advancements in security attacks and corresponding defenses for DNNs (Fig. 10).

[1]Weight Stationary: maximize convolutional reuse and filter reuse, Output Stationary: maximize partial sum accumulation and input feature map reuse. Row Stationary: maximize all these parameters.

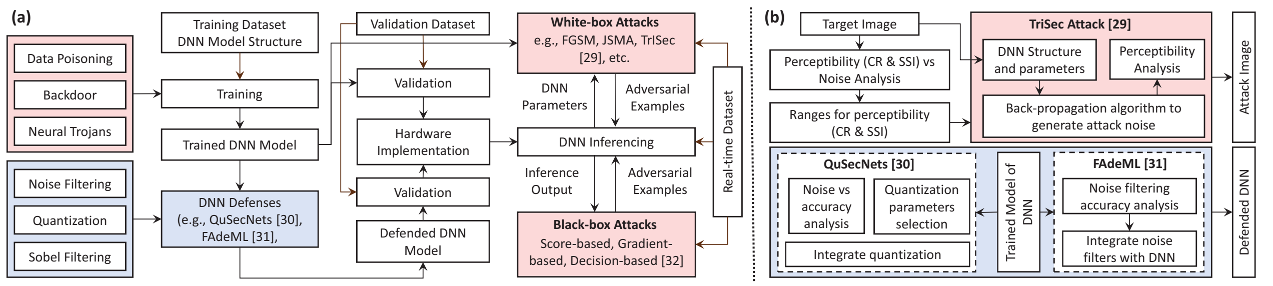
(a)
Data Poisoning
Backdoor
Neural Trojans
Noise Filtering
Quantization
Sobel Filtering
Training Dataset DNN Model Structure
Training
Trained DNN Model
DNN Defenses (e.g., QuSecNets [30], FAdeML [31],
Validation Dataset
Validation
Hardware Implementation
Validation
Defended DNN Model
White-box Attacks e.g., FGSM, JSMA, TrISec [29], etc.
DNN Parameters
Adversarial Examples
DNN Inferencing
Inference Output
Adversarial Examples
Black-box Attacks Score-based, Gradient-based, Decision-based [32]
Real-time Dataset
(b)
Target Image
Perceptibility (CR & SSI) vs Noise Analysis
Ranges for perceptibility (CR & SSI)
TriSec Attack [29]
DNN Structure and parameters
Perceptibility Analysis
Back-propagation algorithm to generate attack noise
Attack Image
QuSecNets [30]
Noise vs accuracy analysis
Quantization parameters selection
Integrate quantization
Trained Model of DNN
FAdeML [31]
Noise filtering accuracy analysis
Integrate noise filters with DNN
Defended DNN

**Fig. 10:** (a) An overview of the different security attacks and corresponding defense strategies for machine learning, especially DNNs. (b) An example of our training dataset-unaware imperceptible attack (e.g., TrISec [29]) and pre-processing based defense strategies (e.g., QuSecNets [30], FAdeML [31]).

### A. *Security Attacks:*

Security vulnerabilities in DNNs can be exploited at different development phases of DNN-based systems, i.e., training, hardware implementation and inferencing. During the DNN **training**, attacker either can exploit the training dataset by introducing small patterned noise, adding specially crafted backdoors [12], or modifying the DNN structure and parameters (i.e., neural Trojans [41]), to train the DNNs for particular noise patterns or backdoor triggers. In these kind of attacks, the attacker either requires complete access to the training dataset or DNN structure. In most of the cases, the specially crafted attack noise is perceptible and may or may not have the hidden backdoor triggers. Therefore, these attacks can be detected during the DNN inference using subjective and objective analysis [29].

On the other hand, attacks during the DNN **inferencing** do not depend on the whole training datasets but may require some samples from training dataset. In these attack, attacker can exploit both black-box[2] and white-box[3] settings to generate adversarial examples for misclassification or confidence reduction [51]. The attack noise generated from such attacks may or may not be imperceptible depending upon their attack algorithm. Based on the attack algorithm and optimization to ensure imperceptibility, several adversarial attacks have been proposed [77], e.g., the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS), the Fast Gradient Sign Method (FGSM), and the Jacobian-based Saliency Map Attack (JSMA), etc. However, these attacks do not consider the subjective and objective analysis for ensuring the imperceptibility.

To address these limitation, recently, training data-unaware imperceptible, TrISec, attack was proposed [29]. This attack leverages the back-propagation algorithm to perform the targeted misclassifcation and incorporates the subjective and objective analysis, i.e., correlation coefficient and structural similarity, into its attack algorithm to ensure the imperceptibility under the white-box settings. Note, all the attacks during the inferencing under the white-box setting can be used in black-box setting by combining them with model stealing attacks [30]. Similarly, several attacks under black-box setting have been proposed, e.g., decision-based [32] and score-based attacks, that either use the search algorithm or perform statistical analysis to estimate the behavior of the DNNs.

### B. *Defenses:*

Several security defense strategies have been proposed based on the different threat models and the development phases of DNN-based systems. Backdoor and training data poisoning attacks can be neutralized by using pruning [12] or retraining the DNN on a subset of the dataset. These defense strategies can be countered by using pruning or weight sensitivity analysis before adding the neural Trojans or before training it for backdoors [12]. Several defense strategies have been proposed to counter the adversarial attacks [77], e.g., DNN masking, gradient masking, adversarial learning, generative adversarial network based defense, data augmentation, and pre-process the input data (e.g., noise filtering [31], quantization [30]) to detect remove or make the adversarial noise perceptible. For example, recently, it has been studied that even low pass noise filtering at the input of the DNN can neutralize the adversarial examples if the attacker is unaware of it [31]. Note, all of these defense strategies can be countered by decision-based attacks which can estimate the DNN behavior even in masked DNNs under the black-box settings [1]–[3], [32].

[2]Black-Box Attack: the attacker does not has any access to the trained DNN structure or the output probabilities

[3]White-Box Attack: the attacker has access to the DNN structure which an attacker can exploit but cannot modify.

## VI. Open Research Challenges

In the following, we list some key research challenges which can potentially have a huge impact for improving the efficiency of deep learning for edge computing.

- **Hardware Software Co-Design:** A common trend is to optimize the DNN for achieving high accuracy, without caring much about the underlying hardware complexity and energy consumption of a computing device. On the other hand, hardware designers have to implement a-posteriori architectures to exploit the software-level optimizations. However, hardware-aware software-level optimizations, e.g., for DNN architecture exploration [69] or compression [43] are promising and need further efforts to succeed.
- **In-Memory Computing:** It seems to be a promising paradigm for developing accelerators that can offer orders of magnitude of energy-efficiency gains compared to the conventional CPU and GPU based systems. However, the high variation characteristics associated with ReRAM and other non-volatile memories limit the accelerators which are based on them to offer precise functionality. Towards this, the multi-level cell (MLC) ReRAM technology has to be mature enough to offer reasonable precision while offering high data density. Also, a significant amount of work is required to develop methods which can be used to train networks such that they can offer high accuracy even when operated on NVM-based in-memory computing devices.
- **Hardware-Aware Hyperparameter Tuning and DNN Architectural Exploration:** Several software-level optimization techniques have been proposed which highlight that sparse DNNs, i.e., having lesser number of parameters, can also offer nearly the same level of output accuracy as dense DNNs. Systematic methodologies are required which, while being aware of the underlying hardware architecture and the system, can tune the network such that it offers near-optimal energy and performance efficiency while maintaining the baseline accuracy.

- **Event-based Spiking Neural Networks:** They have the potential to be much more energy-efficient, as compared to digital-based DNNs, because the power is only consumed when a spike is firing. Such event-driven processing are promising. Therefore, companies like IBM and Intel are investing into their respective neuromorphic architecture chips and its accelerators [48] [9].

## References

[1] A. Athalye et al. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[2] A. Athalye et al. On the robustness of the CVPR 2018 white-box adversarial example defenses. *arXiv preprint arXiv:1804.03286*, 2018.

[3] W. Brendel et al. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[4] C. Chen et al. Exploiting approximate computing for deep learning acceleration. In *DATE*, 2018.

[5] Y. H. Chen et al. Eyeriss: A spatial architecture for energy efficient dataflow for convolutional neural networks. In *ISCA*, 2016.

[6] P. Chi et al. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News*, 2016.

[7] J. Dai et al. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *NIPS*, 2016.

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[9] M. Davies et al. Loihi: A neuromorphic manycore processor with on-chip learning. In *IEEE Micro*, 2018.

[10] X. Dong et al. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *arXiv preprint arXiv:1705.07565*, 2017.

[11] I. J. Goodfellow et al. Generative adversarial nets. In *NIPS*, 2014.

[12] T. Gu et al. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. In *IEEE Access*, 2019.

[13] S. Han et al. EIE: Efcient Inference Engine on Compressed Deep Neural Network. In *ISCA*, 2016.

[14] S. Han et al. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[15] S. Han et al. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

[16] M. A. Hanif et al. CANN: Curable Approximations for High-Performance Deep Neural Network Accelerators. In *DAC*, 2019.

[17] M. A. Hanif et al. QuAd: Design and analysis of quality-area optimal low-latency approximate adders. In *DAC*, 2017.

[18] M. A. Hanif et al. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In *DATE*, 2018.

[19] M. A. Hanif et al. X-DNNs: Systematic Cross-Layer Approximations for Energy-Efficient Deep Neural Networks. In *Journal of Low Power Electronics*, 2018.

[20] M. A. Hanif et al. MPNA: A Massively-Parallel Neural Array Accelerator with Dataflow Optimization for Convolutional Neural Networks. *arXiv preprint arXiv:1810.12910*, 2018.

[21] M. A. Hanif et al. Robust machine learning systems: Reliability and security for deep neural networks. In *IOLTS* 2019.

[22] H. Harzallah et al. Combining efficient object localization and image classification. In *ICCV*, 2009.

[23] K. He et al. Deep residual learning for image recognition. In *CoRR, vol. abs/1512.03385*, 2015.

[24] G. E. Hinton et al. Distilling the knowledge in a neural network. In *NIPS*, 2015.

[25] A. G. Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[26] F. N. Iandola et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$0.5MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

[27] Z. Jia et al. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. *arXiv preprint arXiv:1804.06826*, 2018.

[28] N. P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.

[29] F. Khalid et al. TrISec: training data-unaware imperceptible security attacks on deep neural networks. In *IOLTS*, 2019.

[30] F. Khalid et al. QuSecNets: Quantization-based Defense Mechanism for Securing Deep Neural Network against Adversarial Attacks. In *IOLTS*, 2019.

[31] F. Khalid et al. FAdeML: understanding the impact of pre-processing noise filtering on adversarial machine learning. In *DATE* 2019.

[32] F. Khalid et al. RED-Attack: Resource efficient decision based attack for machine learning. *arXiv preprint arXiv:1901.10258*, 2019.

[33] S. Kim et al. MATIC: Learning around errors for efficient low-voltage neural network accelerators. In *DATE*, 2018.

[34] A. Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[35] H. Kwon et al. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *ASPLOS*, 2018.

[36] C. Kyrkou et al. Dronet: Efficient convolutional neural network detector for real-time UAV applications. In *DATE*, 2018.

[37] C. H. Lampert et al. Efficient subwindow search: A branch and bound framework for object localization. In *TPAMI*, 2009.

[38] G. Li et al. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *HPCA*, 2017.

[39] J. Li et al. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. In *DATE*, 2018.

[40] W. Liu et al. SSD: Single Shot MultiBox Detector. In *ECCV*, 2016.

[41] Y. Liu et al. Neural trojans. In *IEEE ICCD*, 2017.

[42] Y. Long et al. Design of Reliable DNN Accelerator with Un-reliable ReRAM. In *DATE*, 2019.

[43] Q. Lou et al. AutoQB: AutoML for Network Quantization and Binarization on Mobile Devices. *arXiv preprint arXiv:1902.05690v1*, 2019.

[44] C. Louizos et al. Bayesian compression for deep learning. In *NIPS*, 2017.

[45] A. Marchisio et al. PruNet: Class-Blind Pruning Method For Deep Neural Networks. In *IJCNN*, 2018.

[46] A. Marchisio et al. CapsAcc: An Efficient Hardware Accelerator for CapsuleNets with Data Reuse. In *DATE*, 2019.

[47] A. Marchisio et al. HW/SW co-design and co-optimizations for deep learning. In *INTESA@ESWEEK*, 2018.

[48] P. A. Merolla et al. A million spikingneuron integrated circuit with a scalable communication network and interface. in *Science*, 2014.

[49] V. Mrazek et al. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *ICCAD*, 2016.

[50] V. Mrazek et al. AutoAx: An Automatic Design Space Exploration and Circuit Building Methodology utilizing Libraries of Approximate Components. In *DAC*, 2019.

[51] N. Papernot et al. Practical black-box attacks against machine learning. In *ACM CCS*, 2017.

[52] A. Parashar et al. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *ISCA*, 2017.

[53] G. Plastiras et al. Efficient ConvNet-based object detection for unmanned aerial vehicles by selective tile processing. In *ICDSC*, 2018.

[54] M. Rastegari et al. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[55] B. Reagen et al. Ares: A framework for quantifying the resilience of deep neural networks. In *DAC*, 2018.

[56] Y. Redmon et al. You Only LookOnce: Unified, Real-Time Object Detection. In *CVPR*, 2016.

[57] S. Rehman et al. Architectural-space exploration of approximate multipliers. In *ICCAD*, 2016.

[58] S. Ren et al. Faster R-CNN: towards real-time object detection with region proposal networks. In *CoRR, vol. abs/1506.01497*, 2015.

[59] O. Russakovsky et al. Imagenet large scale visual recognition challenge. In *International Journal of Computer Vision*, 2015.

[60] S. Sabour et al. Dynamic routing between capsules. In *NIPS*, 2017.

[61] F. Sampaio et al. Approximation-aware multi-level cells STT-RAM cache architecture. In *CASES*, 2015.

[62] M. Shafique et al. A low latency generic accuracy configurable adder. In *DAC*, 2015.

[63] M. Shafique et al. Cross-layer approximate computing: From logic to architectures. In *DAC*, 2016.

[64] M. Shafique et al. An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era. In *DATE*, 2018.

[65] H. Sharma et al. Bit Fusion: Bit-level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *ISCA*, 2018

[66] E. Shelhamer et al. Fully convolutional networks for semantic segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[67] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *CoRR, vol. abs/1409.1556*, 2014.

[68] L. Song et al. Pipelayer: A pipelined reram-based accelerator for deep learning. In *HPCA*, 2017.

[69] D. Stamoulis et al. HyperPower: Power-and memory-constrained hyper-parameter optimization for neural networks. In *DATE*, 2018.

[70] C. Tai et al. Convolutional neural networks with low-rank regularization. *vol. abs/1511.06067*, 2015.

[71] F. Tung and G. Mori. Clip-q: Deep network compression learning by inparallel pruning-quantization. In *CVPR*, 2018.

[72] J. R. Uijlings et al. Selective Search for Object Recognition. In *IJCV*, 2014.

[73] S. Venkataramani et al. AxNN: energy-efficient neuromorphic systems using approximate computing. In *ISLPED*, 2014.

[74] P. Viola and M. J. Jones. Robust real-time face detection. In *IJCV*, 2004.

[75] W. Wen et al. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[76] A. Yazdanbakhsh et al. GANAX: A Unified SIMD-MIMD Acceleration for Generative Adversarial Network. In *ISCA*, 2018.

[77] X. Yuan et al. Adversarial examples: Attacks and defenses for deep learning. In *IEEE Transactions on neural networks and learning systems*, 2019.

[78] Q. Zhang et al. ApproxANN: An approximate computing framework for artificial neural network. In *DATE*, 2015.

[79] A. Zhou et al. Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights. In *ICLR*, 2017.

[80] C. Zhu et al. Trained ternary quantization. In *ICLR*, 2017.