Chapter 7

Summary and concluding remarks

CONVOLUTIONAL Neural Networks (CNNs) are biologically inspired computational models that are extremely effective at processing multidimensional data and solving tasks such as images classification, objects detection and others [4]. Nowadays, to ensure high responsiveness, low energy cost and data privacy, many CNN-based applications execute CNNs on edge (mobile and embedded) platforms. However, while execution of CNNs at the Edge is desirable and beneficial, it is also challenging due to numerous requirements, posed on the CNNs by the application and the target edge platform. Among these requirements, the most common are high CNN accuracy, high CNN throughput, low CNN latency, low CNN memory cost, and low CNN energy cost. The aforementioned requirements make the design of a CNN executed at the Edge a complex task. Typically, this task is performed by means of the state-of-the-art (SOTA) design flow, shown in Figure 1.3. The heart of the SOTA design flow are platform-aware NAS and the CNN optimization methodologies. These methodologies explore CNNs with different architectures and parameters (weights and biases) and try to find a CNN which adheres to all the requirements posed on it. This CNN is then implemented by means of existing DL frameworks such as Keras [19], Tensorflow [1], TensorRT [72] and others [74], and deployed on an edge platform.

The SOTA design flow, however, has limitations that negatively affect the design of CNN-based applications executed at the Edge. First of all, it restricts the execution of a CNN to a so called sequential (layer-by-layer) manner. The sequential manner of CNN execution is widespread due to its simplicity but it cannot always guarantee efficient CNN execution (i.e., efficient utilization of resources, available on the target edge platform by a CNN). As a dataflow kind of model, characterized with large amount of parallelism available within and among its layers, a CNN can be executed in alternative (non-sequential) manners, that ensure efficient utilization of the target edge platform resources, and thus significantly improve the platform-aware characteristics of the CNN [14,101]. However, this is not explored in the SOTA design flow. Secondly, the SOTA design flow assumes that a CNN-based application only uses one CNN to perform its task. As a result, the SOTA design flow lacks means for inter-CNN optimizations and run-time adaptivity, important to some CNN-based applications. In this thesis, we try to relax the two limitations, mentioned above, and reduce their negative impact on the design of CNN-based applications executed at the Edge. To this end, we have extended the SOTA design flow as shown in Figure 1.5 and explained in Section 1.5. To implement the extended design flow shown in Figure 1.5, we have proposed four novel methodologies, presented in Chapters 4 to 6 in this thesis. Below, we summarize the proposed methodologies and give some concluding remarks.

To relax the first limitation, mentioned above, we have proposed the methodologies, presented in Chapter 3 and Chapter 4. These methodologies explore and exploit alternative (non-sequential) manners of CNN execution, thereby improving platform-aware characteristics of a CNN.

The methodology presented in Chapter 3 ensures high CNN inference throughput, required by many CNN-based applications executed at the Edge [14]. To ensure high CNN inference throughput, our methodology efficiently distributes (maps) the computations within a CNN to the computational resources of a target edge platform. The mapping, found by our methodology, features combined exploitation of two types of parallelism, namely task-level parallelism and data-level parallelism, available within a CNN. This feature distinguishes our methodology from other existing methodologies because these methodologies utilize only task-level or only data-level parallelism, when mapping a CNN onto an edge platform. Based on the experimental results, we conclude that for CNNs executed on the Jetson TX2 edge platform [71], our methodology offers a 1.36% to 42% higher inference throughput, compared to the mapping methodology employed by the best-known and state-of-the-art TensorRT DL framework for the Jetson TX2 edge platform.

The methodology presented in Chapter 4 ensures low-memory CNN inference at the Edge, required for CNN-based applications executed on edge platforms with extremely small memory resources, such as embedded Internet-of-Things (IoT) devices [17]. To ensure low CNN memory footprint, our methodology splits the data exchanged between layers of a CNN into parts, processed in a specific order, and efficiently reuses the platform memory among the data parts. However, as the data processing by parts may cause CNN execution time overheads (e.g., CNN layers may require time to switch among the data parts), our methodology may decrease the CNN throughput. The evaluation results show that, compared to the memory reduction, achieved by the most relevant CNN buffers reuse methodology, employed by the TensorRT DL framework for efficient CNN execution at the Edge, our memory reduction methodology allows to reduce the CNN memory footprint by 2.8% to 38% at the cost of 2% to 23% decrease of the CNN throughput.

The methodologies proposed in Chapter 5 and Chapter 6 of this thesis, are aimed at relaxation of the second limitation, mentioned above.

The scenario-based run-time switching (SBRS) methodology in Chapter 5 introduces the use of multiple CNNs and run-time adaptive switching between these CNNs to a CNN-based application. Every CNN in the application corresponds to a scenario and is designed to adhere to a specific set of requirements, posed on the CNN. During the application execution, the application environment can trigger the application to switch between the scenarios, thereby adapting the characteristics of a CNN-based application to changes in the application environment (such as a change in device battery level or a change of the throughput of the input data stream). Thus, our methodology ensures efficient execution of an application which needs are affected by the application environment at run-time. To the best of our knowledge, our proposed methodology is the first methodology, able to design an adaptive CNN-based application, which considers platform-aware requirements and constraints that are specifically affected by environment changes at run-time. The most relevant to our methodology, the MSDNet methodology, does not consider real-world platform-aware requirements and constraints and does not offer means for automated switching among the application scenarios, based on the changes in the application environment. Based on the experimental results, we conclude that: 1) by introducing run-time adaptivity into CNN-based applications, affected by changes in the application environment at run-time, our methodology significantly improves the applications' characteristics; 2) our methodology outperforms the most relevant MSDNet methodology.

The methodology proposed in Chapter 6 extends our low-memory CNN inference methodology, proposed in Chapter 4, with support for pipeline parallelism and inter-CNN memory reuse. Thus, our methodology offers efficient memory reduction to a wide spectrum of applications, designed using both the SOTA design flow and our extended design flow. Based on the evaluation results, we conclude that our methodology: 1) enables for up to 5.9 times memory reduction compared to deployment of CNN-based applications with no memory reduction, and 7% to 30% memory reduction compared to other memory reduction methodologies that reduce the CNN memory cost without CNN accuracy decrease; 2) can be efficiently combined with well-known pruning and quantization methodologies (that are orthogonal to our methodology) in order to offer high rates of CNN memory compression without significant decrease of the application accuracy and throughput.