5 | Dynamic optimisation of energy consumption

In the previous chapter, we proposed a solution to distribute a DNN between the cloud and multiple edge devices while optimising accuracy and execution time. In this chapter, the objective is to optimise energy consumption and accuracy.

Indeed, high energy consumption in DNNs is a major concern, particularly for real-time or continuous processing applications, such as surveillance systems, autonomous vehicles, and smart cities. Therefore, researchers have been investigating different approaches to optimise the energy consumption while maintaining their performance and accuracy. In this chapter, we propose an energy optimisation approach adapted to distributed DNN inference between the cloud and multiple edge nodes, each including one early-exit.

The main objective of this chapter is to propose an effective model to optimize energy consumption, based on a rapid consumption estimation approach with good precision. The idea is to develop an approach that is fairly easy to use and at the same time generic enough to be applicable to all the components of common hardware architectures. These estimates are then used in a dynamic method for optimizing energy consumption. This optimization is based on dynamic linear programming for maximum performance and better responsiveness to changes in the quantities of energy available over time.

This chapter is organised as follows. First, we present our energy consumption estimate approach (Section 5.1), before exposing our proposed methodology for energy consumption optimization (Section 5.2). Then, we define a case study (Section 5.3) which will be used in our results section (Section 5.4).

5.1 Energy consumption estimate

5.1.1 Introduction

With the increasing use of embedded systems powered by batteries with limited capacities, estimating and optimising energy consumption have become paramount challenges for designers and users of such systems. This is particularly important in distributed video surveillance systems, which comprise heterogeneous cameras with varying quantities of energy. Some cameras may be powered by rechargeable batteries, while others may be connected to permanent power sources but can be unplugged and switched to battery power, etc.

Accurately estimating energy consumption is a tedious task that requires developing prototypes corresponding to all the hardware architectures considered in the approach and using electronic probes. Unfortunately, this is not suitable for our approach, as it does not target a single specific hardware architecture or a particular application.

The goal of the approach proposed in this chapter is to achieve estimates with acceptable accuracy, while supporting various architectures and applications. The methodology must also be fast enough to operate on application models without the need to develop physical prototypes. Finally, the last characteristic of our methodology is its dynamic adaptability during run time to potential changes in energy sources and the amount of energy available at each edge.

5.1.2 General principle

The energy consumption of an operating embedded system comprising processors and memories consists of two parts:

- Static consumption: This corresponds to the hardware architecture components per unit time, while functioning but without performing any activity (idle mode). This part of energy consumption is completely independent of the application being executed and depends only on the physical characteristics of the components and the clock frequency.
- Dynamic consumption: This refers to the energy resulting from the activity of hardware components, i.e., the energy cost directly linked to the execution of the application.

Thus the total energy consumption of the system corresponds to the sum of the static and dynamic consumptions.

5.1.3 Energy consumption estimate during run time

5.1.3.1 Identification of the energy-consuming activities

One way to estimate energy consumption which is suitable for our constraints, is to annotate the DNN code with relevant activity counters in terms of energy consumption. This approach involves identifying the energy-consuming activities of each component of the hardware architecture, such as processors and memories. While listing such activities precisely and exhaustively can be a challenging task for many hardware components, such as hardware accelerators, it is relatively simple for processors and memories. A simple processor is either idle or active, and its activity primarily involves the execution of instructions. Depending on the processor type, the power consumption associated with basic instructions (Mult/Div, Add/Sub, Logical) is either provided by the hardware manufacturer or measured using prototyping hardware. In the memory case, it is even simpler than for processors. Indeed, the 3 main activities of a memory are simple and clearly identifiable. They are: reading, writing and idle.

Our adopted approach involves retaining the following energy-consuming activities :

- Processors : We model a processor as a (Finite-state machine (FSM)) with four states that correspond to an abstraction of the potentially power-consuming CPU activities.
  - Idle: This is the state in which the processor does not execute any instructions, but it is still powered.
  - Multiplication/division: Although these are basic arithmetic operations, they are often more expensive in terms of energy consumption than simple additions or subtractions.
  - Addition/subtraction: These are the simplest and least energy-consuming arithmetic operations.
  - Logic: These are boolean operations such as comparisons of values with NOT, AND, and OR.
- Memories : The memory model is relatively simple due to its reduced number of states and transitions. In fact, since it is a high-level abstraction, advanced modes of memory operation, such as burst mode, are excluded. This is due to our memory activity counting approach, which consists of counting a reading for each variable on the right side of an assignment and a writing for those on the left side.
  - Idle : when memory is in a waiting state and not performing a read or write.
  - Read: this is a simple read access.
  - Write: this is a simple write access.

- Communication networks : The communication model contains two states:
  - Idle: this is the state in which a communication link (wired or wireless) is not transferring any data.
  - Transmission: this is the state corresponding to the transmission of a unit of data (one byte).

Once these activities have been identified, the next step involves counting the number of occurrences of each of them in the application code and knowing their energy cost.

5.1.3.2 Estimation of the energy consumption of each activity

There are several ways to estimate the power consumption relative to the various activities of a given hardware component. The main methods are as follows:

- Prototyping and emulation: A time-consuming process which involves creating an architecture with the same characteristics as the targeted architecture, using specialised hardware components. Physical probes are then used to measure the energy consumption.
- Using manufacturer data: Manufacturers often provide power consumption data for components in the accompanying data sheet. This particularly common for memories.
- Use of simulation software dedicated to energy consumption: This involves using specialised tools, sometimes only for certain hardware components, to estimate energy consumption by entering the component's properties.

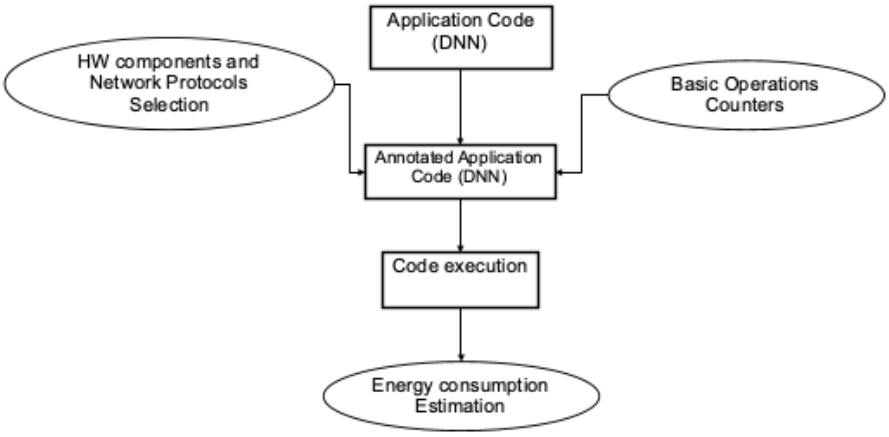
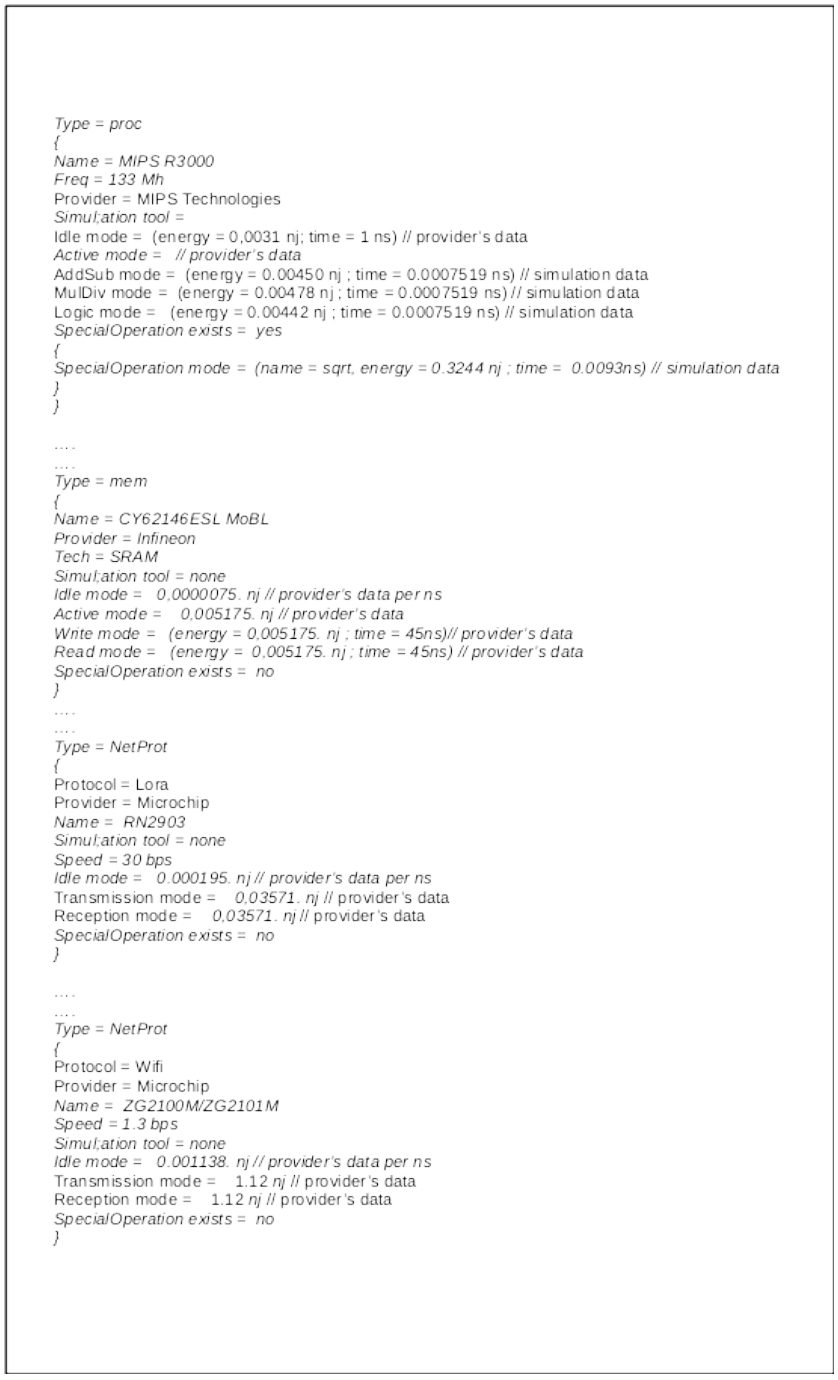


Figure 5.1: Energy consumption estimation flow.

5.1.3.3 Energy consumption estimation flow

The steps of our approach for estimating energy consumption are illustrated in Figure 5.1.

- Inputs
  - Core power library: These are average power consumption values per instruction type for each processor potentially used in our hardware targets, as well as power costs for memory reads and writes (per memory type), and for the transmission of one data byte for each communication protocol. An example illustrating the organisation of this information in a library is given in Figure 5.2.
  - Basic activities' counters: these are counters dedicated to each energy-consuming activity identified for any hardware component in the target architecture.



```
Type = proc
{
Name = MIPS R3000
Freq = 133 Mh
Provider = MIPS Technologies
Simul;ation tool =
Idle mode =  (energy = 0,0031 nj; time = 1 ns) // provider's data
Active mode =   // provider's data
AddSub mode =  (energy = 0.00450 nj ; time = 0.0007519 ns) // simulation data
MulDiv mode =  (energy = 0.00478 nj ; time = 0.0007519 ns) // simulation data
Logic mode =   (energy = 0.00442 nj ; time = 0.0007519 ns) // simulation data
SpecialOperation exists =  yes
{
SpecialOperation mode =  (name = sqrt, energy = 0.3244 nj ; time =  0.0093ns) // simulation data
}
}

....
....
Type = mem
{
Name = CY62146ESL MoBL
Provider = Infineon
Tech = SRAM
Simul;ation tool = none
Idle mode =   0,0000075. nj // provider's data per ns
Active mode =    0,005175. nj // provider's data
Write mode =   (energy = 0,005175. nj ; time = 45ns)// provider's data
Read mode =   (energy =  0,005175. nj ; time = 45ns) // provider's data
SpecialOperation exists =  no
}
....
....
Type = NetProt
{
Protocol = Lora
Provider = Microchip
Name =  RN2903
Simul;ation tool = none
Speed = 30 bps
Idle mode =   0.000195. nj // provider's data per ns
Transmission mode =    0,03571. nj // provider's data
Reception mode =    0,03571. nj // provider's data
SpecialOperation exists =  no
}

....
....
Type = NetProt
{
Protocol = Wifi
Provider = Microchip
Name =  ZG2100M/ZG2101M
Speed = 1.3 bps
Simul;ation tool = none
Idle mode =   0.001138. nj // provider's data per ns
Transmission mode =    1.12 nj // provider's data
Reception mode =    1.12 nj // provider's data
SpecialOperation exists =  no
}
```

Figure 5.2: Architecture components energy consumption.

  - Application code: this is the application code, i.e. the DNN itself.

- Code annotation: This consists of annotating the application code to count all occurrences of processor, memory, and data communication operations.
- Execution : This is the actual execution step of the annotated application code.
- Output : Our flow outputs the values of dynamic energy consumption (linked to the components' activities) and static energy consumption (linked to the presence and power-up of the components in idle mode).

5.2 Energy consumption optimisation

5.2.1 Integer linear programming model

In the same spirit as the approach of Chapter 4, our approach here consists in modelling the problem in the form of a linear program to find a compromise between the minimisation of the total quantity of energy consumed and the accuracy of the results provided by the DNN.

5.2.1.1 Controlled parameters

With N the number of early-exits, the controlled parameters of the linear programs are the following :

- $NBIm_i, i = 1, \ldots, N+1$.
- $S_i, i = 1, \ldots, NB$.

5.2.1.2 Non-controlled parameters

The non-controlled parameters of the linear program are the following :

- $ErrAll_i$: Error percentage of output i
- $ExcT_i$: Execution time for an image taking output i
- $EnCons_i$: Energy consumption for an image taking output i
- $AvEn_i$: Available energy for each hardware device i
- $NbImT$: Total number of images
- BI : big integer

5.2.1.3 LP model

$$\begin{aligned} \text{minimise } W_{time} * \sum_{i=1}^{N+1}(NBIm_i * EnCons_i) \\ +W_{err} * \sum_{i=1}^{N+1}(NBIm_i * ErrAll_i) \end{aligned} \tag{5.1}$$

$$\begin{aligned} 0 \leq W_{err} \leq 1 \text{ (precision weight)} \\ 0 \leq W_{time} \leq 1 \text{ (energy consumption weight)} \end{aligned} \tag{5.2}$$

subject to

$$\sum_{i=1}^{N+1} NBIm_i = NbImT \tag{5.3}$$

$$NBIm_i \leq BI * Si. \forall i \tag{5.4}$$

$$BI * NBIm_i \geq Si. \forall i \tag{5.5}$$

$$NBIm_i * EnCons_i + \sum_{j=1}^{j \leq i} NBIm_j * EnCons_i \leq AvEn_i. \forall i \tag{5.6}$$

5.2.2 Design decisions coming out of the model

Design decisions coming out of the model are results of optimization trade off between accuracy and energy consumption. This consumption takes into account processing (on cloud and edge devices) and data communication. The main advantage of this approach is its genericity, because it can be applied to cloud/edge systems regardless of the calculation elements and communication protocols used on each device. Indeed, results obtained with substantially identical systems with quite different communication protocols in terms of energy consumption will very likely be significantly different. The same reasoning remains valid by varying the consumption due to calculation on the edge or on the cloud.

5.2.3 Limitations of the linear program

The linear program model has multiple advantages mainly related to its simplicity and belonging to a class of approaches that produce optimal solutions. However, it suffers from certain limitations that make its effectiveness relative in the contexts and constraints linked to application and hardware targets. These limitations include the lack of responsiveness to changes in energy sources during run time: The context of power consumption optimisation during runtime involves some power changes as the runtime progresses. Indeed, camera batteries can recharge and discharge and camera connections (or disconnections) to power sources can occur over time. A completely static approach is clearly incapable of taking this type of change into account, which making it ineffective.

5.2.4 Dynamic linear programming

General principle Consider a dynamic system, with discrete time and finite horizon T, whose state at time t $(0 \leq t \leq T)$ is noted x_t; Thus T = number of early exits.

Let $\Gamma(x_t)$ be the set of possible actions when the system is in state x_t. If the action at $\in \Gamma(x_t)$ is chosen, the system goes to the state $\Gamma(x_{t+1})$ = $\Phi(x_t, a_t)$. $\Gamma(x_t)$ contains a single action which consists of deciding the number of images taking the early exit t.

Let us also note $F(x_t, a_t)$ as the cost of the action at when the system is in the state x_t. This corresponds to the execution time for images taking the early exit t (weighted) + error percentage at exit t (weighted).

Starting from the initial state x_0, the global cost after T time steps is then:

$$\sum_{t=0}^{T} F(x_t, a_t) \tag{5.7}$$

Minimising the overall profit therefore amounts to solving Bellman's equation:

$$\begin{aligned} &\text{minimise} \sum_{t=0}^{T} F(x_t, a_t), \\ \text{so that } &\forall 0 \leq t \leq T, a_t \in \Gamma(x_t) \\ &\text{and } \forall 0 \leq t \leq T, x_{t+1} \in \Phi(x_t, a_t) \end{aligned} \tag{5.8}$$

Application to the context of energy consumption optimisation This type of dynamic model applies quite systematically to the context of energy consumption optimisation that we are targeting. Indeed, by considering the resolution of an instance of the LP described above and by removing an EE from the model at each

Algorithm 1 Dynamic Linear Program for Execution Time Optimisation

Require: $NBEE \geq 1$
Ensure: Each image takes only one early exit.
The states of the system correspond to the different early exits.
The choices made at each state consists of determining the number of images taking the current early exit.
The cost of the choice corresponds to the value of the LP objective function
repeat
Solve the current LP in a classic way
From the previous step, we obtain the number of images taking the current early exit; then we update the LP and we delete the corresponding variable $NBIm_i$. Subsequently, we obtain a LP corresponding to the next state of the dynamic program
until no remaining early exits

iteration, the convergence of the approach is guaranteed in a number of iterations equal to the number of EEs.

The different steps of the resolution process are described in algorithm 1.

5.2.4.1 Optimisation flow

The main idea of our optimisation approach is based on the principle of dynamic programming. However, the simple version of the dynamic programming model does not take into account the possible charging or discharging of the cameras.

Our methodology consists of an iterative process based on the principle of dynamic optimisation, where successive linear programs are resolved, but within a framework that is more responsive to changes in the execution environment, such as the amount of available energy.

Indeed, this approach not only takes into account the number of images leaving the DNN on the various early-exits at any time, but also is reactive to the following events :

- Switching of a camera to charging mode : this is an event sent by a software or hardware probe (generally available in all marketed cameras), notifying the event corresponding to the switch of the camera from discharging mode to that of energy charging. This happens when connecting a camera, during operation, to an electrical outlet , or starting charging via solar panels.
- Switching of a camera to discharging mode : this is the reverse event to the one described above. It is the fact that an operating camera connected to a power source goes into discharging mode, i.e., the disconnection of the camera from the power source.

We begin with an initialisation iteration during compile time, followed by successive iterations during run time as long as there is a flow of images. The optimisation flow is exposed in algorithm 2

Note

- In predicate A, we have chosen to regenerate a new LP to ensure that the available quantities of energy, on each camera, are dynamically updated with the real available quantities. This is an additional guarantee that, at every 100 000 inferences, the available energy is not exceeded.
- In predicate B, an EE is removed from the LP at each iteration. For the proper functioning of the optimisation approach we assume that no camera can run out of battery during execution. In fact, it is assumed that cameras operating on battery without being connected to any permanent power source are provided with new batteries before running out of power.

Algorithm 2 Energy consumption optimisation flow

Require: Define the following predicates:
- A: Number of images processed = 100 000
- B: Number of remaining EE = 0
- C: Percentage of images given by the LP for an EE is reached
- D: A camera switches to charge mode
- E: A camera switches to discharge mode

Run the LP to know the initial workload sharing between the EE
while $\neg A$ and $\neg B$ and $\neg C$ and $\neg D$ and $\neg E$ **do**
    Run the DNN
    if A **then**
        Regenerate a new LP with an update of available energies
        Number of processed images $\leftarrow 0$
        $A \leftarrow$ false
    end if
    if B **then**
        Number of EE $\leftarrow$ Initial number of EE
        Regenerate a new LP with an update of available energies
        Number of processed images $\leftarrow 0$
        B $\leftarrow$ false
    end if
    if C **then**
        Regenerate a new LP with an update of available energies and one less EE
        C $\leftarrow$ false
    end if
    if D **then**
        Regenerate a new LP with infinite available energy for the concerned EE (and an update of other available energies)
        D $\leftarrow$ false
    end if
    if E **then**
        Regenerate a new LP with an update of available energies
        E $\leftarrow$ false
    end if
end while

5.2.5 Dynamic threshold adaptation

In order to introduce even more adaptability, to the characteristics of real images, our proposed approach adopts almost the same principle of dynamic adaptation of the threshold presented in Chapter 4. However, in the context of energy consumption, changes in the quantities of available energy occur over time, making it necessary to update the threshold accordingly. Therefore, in our approach, the threshold is updated inside each iteration when generating a new LP. If the percentage of image taking a certain EE is lower than that given by the LP, the threshold of the corresponding camera is reduced by 10%. Conversely, if rate of images taking the EE is higher than that given by the LP, the threshold is increased by 10%, as in the approach of Chapter 4. The update of the parameters in the successive LPs ensure that the available quantities of energy are not exceeded.

5.2.6 Optimisation algorithm execution

As explained in the previous paragraphs, the proposed approach for the optimisation of energy consumption consists of an iterative and dynamic algorithm. Thus, several linear programs are likely to be generated during run time. Generating and running these models can be costly in terms of computational resources and energy. It is for this reason that we constrain the execution of all the successive optimisation models to the cloud level in this methodology. This is an inexpensive solution because the cloud is theoretically supposed to have infinite resources. This choice still requires some communications from the cameras to the cloud and vice versa.

These communications convey the quantity of energy available as well as the state of charging or discharging in the direction of cameras towards the cloud, and the values of the controllable variables given by the LP resolutions from the cloud to

the cameras. This is a very small number of integer type values, which do not overload the network excessively.

5.3 Case study

5.3.1 Activities counting

5.3.1.1 Processor

To estimate the energy consumption linked to the processor as accurately as possible, we counted the number of arithmetic and logical operations executed in each function of the DNN model. The model includes only basic arithmetic operations (add, sub, mul and div) in addition to square root calculations (SQRT). Since the latter cannot be directly calculated in a single instruction with certain Reduced instruction set computer (RISC) processors such as the MIPS R3000, we used an algorithm for calculating square roots using only the four basic arithmetic operations. This is the Newton-Raphson algorithm, which is described below.

```
1
2  Function SQRT_NewtonRaphson(N, epsilon, x0) :
3      x_next = x0
4      x_previous = 0
5
6      Tant que abs(x_next - x_previous) >= epsilon :
7          x_previous = x_next
8          x_next = 0.5 * (x_previous + N / x_previous)
9
10     Return x_next
11
```

In this pseudo-code, N is the number whose square root is to be calculated, epsilon is the tolerance for convergence (e.g., 1e-6 for sufficient precision), and x0 is the initial estimate for the square root. We can initialise x0 to an approximate value of the square root of N (e.g. 1.0) or use a more advanced method to improve convergence speed. The algorithm continues to iterate until the difference between successive estimates of the square root becomes less than or equal to epsilon, indicating that the algorithm has converged. The result is then returned.

Thus, the approach that we have adopted to estimate the energy consumption corresponding to the calculation of a square root is based on simulation. Indeed, the implementation of the algorithm in MIPS R3000 assembler and its execution on a simulation tool, a large number of times, gives a fairly precise estimate of the energy consumption. Concretely, for 10 000 executions of the assembler program the average energy consumption is 0.3244 nJ;

Table 5.1 shows the operation count for each processor power-consuming operation. For each EE_i, this is the number of operations between EE_{i-1} and EE_i.

Table 5.1: Processor's activities counting.

	Add	Sub	Mul	Div	Sqrt
Early exit 1	915281920	46080	915192320	94720	46080
Early exit 2	1192181760	149760	1191887360	304640	149760
Early exit 3	1324380160	170240	1324047360	348160	170240
Final exit	2560983040	421120	2560158720	860160	421120

5.3.1.2 Memory

As shown in Table 5.2, reading and writing memory in each EE and the previous EE correspond to the operation of reading and writing variables. Therefore, a read

corresponds to a variable appearing on the right-hand side of an assignment and a write corresponds to a variable appearing on the left-hand side of an assignment instruction.

Table 5.2: Memory's activities counting.

	Read	Write
Early exit 1	1889354890	115332200
Early exit 2	2465752340	147661520
Early exit 3	2737330590	163517240
Final exit	6411063720	168177440

The results of the two tables 5.2 and 5.1 have been approximated to the upper ten in particular to compensate for any loss during floating point calculations.

5.3.1.3 Communication

Counting communication activities involves calculating the calculation of the amount of data (in bytes) that is transferred from each EE to the next in the execution platform. These data quantities were estimated in Chapter 4 and are presented in Table 4.3.

5.3.2 Architecture elements activities energy consumption estimation

In this section, energy consumption relating to the activities of the basic hardware architectures main components is estimated, measured or calculated from manufacturers' data. These consumptions are given in different tables with great precision (several digits after the decimal point). This is not done for the sake of perfect accuracy of the values but is simply due to the fact that the consumptions are extremely low and the number of operations in our experiments is very large.

Consequently, any approximation in consumption values can cause significant differences in total consumption.

5.3.2.1 Processor

We employed a low-power-consuming processor for our experiments, which is suitable for use in an edge type component without requiring an electrical outlet. The processor we used is the MIPS R3000, a 32-bit RISC microprocessor, developed by MIPS Computer Systems. It has a simple scalar architecture that executes one instruction per cycle [DominicSweetman, 2023]. It is used in various domains, such as gaming with the Sony PlayStation One and Nintendo 64. Additionally, the Tesla model S cars integrate a high-performance, MIPS-based Mobileye EyeQ3 computer vision chip that enables the Model S to detect road signs, lane markings, obstacles and other vehicles [Wilson, 2023]. The choice here is mainly the use of a processor with a fairly simple structure and low energy consumption. Thus, the use of MIPS, in particular, among processors of this type was dictated by the availability of its HDL code that could be synthesized and used during various experiments.

The energy consumption corresponding to each type of instruction that we have identified was measured the simulation and consumption estimation tool Xilinx Vivado [XilinxInc, 2023]. This tool enables the setting of different parameters and characteristics of a processor (such as number of Arithmetic-logic unit (ALU)s, clock frequency, etc.) on the processor's model [BabakAminazad, 2023], followed by simulation of the execution of any assembler instruction. The result is a fairly accurate estimate of power consumption for each instruction type on average, as described in table 5.3.

Table 5.3: Energy consumption, for a MIPS R3000, corresponding to each instruction type.

Instruction	Energy consumption (nJ)
ADD/SUB	0.0045
MUL/DIV	0.00478
LOGIC	0.00442
IDLE	0.0031

These estimates are calculated with Xilinx's Vivado software on a faithful and synthesizable model of the processor. Their precision is based exclusively on the effectiveness of Vivado (which has already proven itself in both the industrial and academic worlds) as well as that of the synthesizable HDL model (having already been tested, simulated and validated quite extensively).

The estimation was made for a small processor that consumes reduced energy amounts. However, since the execution platform targeted includes a server-type calculator on the cloud side, it is also necessary to estimate the consumption per activity on the server. Such computers are generally several thousand times more energy-consuming than a processor of the MIPS R3000 type [IBMcorp., 2023]. Nevertheless, to simplify the calculations while maintaining a consistent order of magnitude, we assume that a server consumes 100 times more energy than a MIPS R3000. Such a ratio (100 times) is taken as an illustration, but it is still quite realistic. Indeed, servers are generally either very energy intensive, or (for a small part) having an efficient dynamic consumption but generate a significant energy overhead due to the static consumption of the very large quantity of logic, potentially unused, that they embed.

5.3.2.2 Memory

The estimation of energy consumption is relatively simple in the case of memory thanks to the simplicity of its energy-consuming activities automaton. Memory chip manufacturers, such as NEC, Micron, ST or Infineon systematically provide datasheets including consumption in idle, write and read modes, or at least idle and access modes (when energy consumption of the read and write modes are not significantly different).

Table 5.4 summarises the energy costs of the SRAM CY62146ESL MoBL. Its a very low energy consuming memory produced by Infineon corp. These values are calculated from the read and write times as well as the power dissipated during the read, write and idle modes provided in the datasheet [InfineonInc, 2023].

Table 5.4: Energy consumption, for an Infineon CY62146ESL MoBL SRAM, corresponding to each instruction type.

Activity	Energy consumption (nJ)
Read	0.0005175
Write	0.0005175
IDLE	0.0000075

The values are expressed in nJ because they correspond to the energy consumption of read/write operation, each lasting 45 ns. All these values are calculated from the manufacturer's datasheets which provide the power dissipation, the read and write times as well as the operating frequency. Therefore, the energy consumption can be calculated simply using the following formula : $E(j) = P(W) \times T(s)$.

In the same way as for processors, we will consider that cloud memories are 100 times more expensive in energy than those of cameras because of their nature and size. This is a lower bound because the ratio is often greater than 100.

5.3.2.3 Communication

As shown in Table 5.5, we selected two network controllers for our study: LoRa and Wifi. The main energy-consuming activities for these controllers are: transmission, reception and idle modes. The results presented in the table are for Microchip Inc.'s network controllers, specifically LoRa Microchip RN2903 [MicrochipTechnologyInc, 2023a] and Wifi Microchip ZG2100M [MicrochipTechnologyInc, 2023b].

Table 5.5: Energy consumption for Microchip RN2903 and ZG2100M network controllers.

	Lora	Wifi
Transmission (nJ)	0.03571	1.12
Reception (nJ)	0.03571	1.12
IDLE (nJ)	0.000195	0.001138

5.3.3 Energy consumption optimisation flow

5.3.3.1 Initialisation iteration

PL parameters The primary parameters of the initial linear program are established during compile time, similar to the optimisation during execution discussed in Chapter 4.

- Dynamic energy consumption for each early-exit : By employing the approach of counting energy-consuming activities for one DNN inference, we were able to derive the results presented in Table 5.6.
- Static and total energy consumption : The energy consumption of hardware components during idle mode can be summed up to obtain the static energy consumption, while the total energy consumption is the sum of the static

Table 5.6: Dynamic energy consumption for each EE per image (in μ J)

	Processor				Memory			Network			Total
	Add/Sub	Mul/Div	Sqrt	Total	Read	Write	Total	Trans.	Recep	Total	
EE1	4118.9	4375	14.9	8509	977.7	59.6	1037.4	1772	0	1772	11318.4
EE2	5365.5	5698.6	48.5	11112.8	1276	76.4	1352.4	518.9	1772	2291	14756.3
EE3	5960.4	6333	55.2	12346.3	1416.56	84.62	1501.1	6.33	518.9	525.23	14372.5
FE	11526200	12241600	1366100	90456800	331770	8700	340470	0	6.33	6.33	90797276.3

and dynamic energies. Nevertheless, the activities performed by hardware components are mutually exclusive and non-interchangeable, implying that a processor cannot replace a memory and vice versa. As a result, minimising dynamic energy consumption inevitably leads to an increase in the amount of time components spend in idle mode. Given that idle mode consumes considerably less energy than active mode, our approach prioritises minimising the overall energy consumption. This explains why we decided not to include static consumption in the LPs, which also contributes to their simplicity.

- Available energy and energy sources for the cloud and each edge: This refers to quantity of energy accessible to each device during the iteration. For devices in charging mode, this value is infinite, while for devices in discharging mode, the available energy is equal to the amount that is currently available. This information is provided by hardware sensors that are integrated into the devices or by specific software processes.

  As shown in Table 5.7, for our experiments, we chose classical smart cameras with 5w power capacity. At the beginning, we fixed available power to 5 Wh, 0.64 Wh and 1 Wh for cameras integrating EE1, EE2 and EE3 respectively. The available power was set to infinite for the cloud server. Camera 1 operates exclusively on battery power and cannot be connected

to any other energy source. Conversely, cameras 2 and 3 can be connected to a plug and a solar panel, respectively. Initially, camera 2 and 3 were not connected to their respective power sources, while the cloud server was connected. Camera 1 is processing only on batteries, and we suppose that the battery is replaced by a fully charged one after each iteration.

Table 5.7: Available energy and energy sources for the cloud and each edge.

	Initial available power (Wh)	Potential energy source	Initial state
Camera 1	5	Battery	none
Camera 2	0.64	Plug	NC
Camera 3	1	Solar	NC
Cloud server	inf	plug	C

PL execution Executing the linear program during compile time provided the percentages of images taking each EE, as illustrated in Table 5.8.

Table 5.8: Images taking each early exit according to the initial LP resolution.

	Exiting images percentage
Early exit 1	61%
Early exit 2	28%
Early exit 3	7%
Cloud server	4%

These results are reasonably expected, considering the number of activities that each camera is capable of performing. Cameras 1 and 2 have fewer activities, which explains their higher percentage of images taking EE. However, the number of images exiting by the EE1 is less significant than the results obtained in Chapter 4. This can be attributed to the limited amount of energy available for camera 1 and the absence of an energy source.

5.3.3.2 Other iterations

Energy events scenario To demonstrate the effectiveness of the proposed approach and its adaptability to changes in energy sources, we elaborate on an energy scenario that encompasses various energy sources for the cameras, along with some changes that occur during run time. Therefore, camera 2 is disconnected from the electrical outlet during the fourth first iterations, then it is reconnected again after 200 images since the beginning of the 5th iteration. As for camera 3, it goes into solar recharging mode at the start of the second iteration, then into discharging mode after six iterations.

5.4 Results and discussion

The experiments have been done using the same DNN architecture as in Chapter 4, on the ImageNet dataset [StanfordVisionLab., 2023] with 1,281,167 training images, 50,000 validation images and 100,000 test images with 71.6% baseline accuracy on VGG16.

5.4.1 Sensitivity to the energy scenario

Several particular cases arise during run time when applying the energy scenario described in the previous section. Camera 2 consumes almost all of its available energy during the four first iterations, and ends up with only 0.00005 Watts for the start of the fifth iteration. As a result, the LP generated at this stage states that only 0.2% images must exit through EE2, and 0% through EE3 and cloud. This is imputable to the energy bottleneck constituted by camera 2, which, in addition to preventing images output by EE2, also prevents their passage through EE3 and the cloud, because this would also consume energy on camera 2.

However, after 200 inferences in the same iteration, the switching of camera 2 to recharging mode causes the generation of a new LP, which considers that the available energy in camera 2 is infinite. This relieves the "energy bad" results of the previous LP. The new LP gives results are more in line with those of the initialization iteration (about 30% on EE2 and 60% on EE1).

The event of camera 3 switching to solar charging mode is also taken into account in the LP generated during the 6th iteration, which considers that camera 3 has infinite energy. However, the quantity of energy of camera 3 being already sufficient before recharging, the LP remains largely in line with that of iteration zero.

5.4.2 Effectiveness of the dynamic threshold adaptation

Table 5.9 presents the results in terms of the percentage of images taking each EE and their corresponding power consumption. The energy consumed values in this table are cumulative across different EEs. Therefore, the energy consumed by an outgoing inference through EE3 is equal to the energy consumed to traverse all the layers of the network up to EE3.

We notice that as the threshold increases, images go deeper into the DNN, resulting in increasing energy consumption. However, the dynamic threshold adaptation approach proves to be efficient, not only in terms of energy consumption but also in terms of accuracy of the results.

In column Opt1, the weights given to precision and energy consumption are 0.9 and 0.1 respectively, whereas they are 0.5 for each in Opt2, and finally 0.1 and 0.9 in Opt3. Therefore, it is quite clear that the more weight given to the energy criterion in the model, the more the total quantity of consumed energy is optimised, while the precision of the model decreases simultaneously. In fact, the

model's precision in Opt1, Opt2 and Opt3 configurations is 71.2%, 67.7% and 60.3% respectively.

These results are fairly predictable because the optimisation approach mandates a decrease in the threshold of any EE when the number of images taking it is insufficient. However, this number of images remains bounded and limited by the amount of available energy.

Table 5.9: Images exiting at each EE and corresponding energy consumption with fixed thresholds - for 1M inferences

	0.3		0.5		0.7		0.9	
	% im.	En.(J)	% im.	En.(J)	% im.	En.(J)	% im.	En.(J)
EE1	88	11318.4	80	11318.4	69	11318.4	27	11318.4
EE2	6	1770,7	10	2951,3	17	4574.4	19	10772.1
EE3	5	862.35	7	1437.25	10	2012.15	9	7761,15
Cloud	1	907.9 10^3	3	2723.9 10^3	4	3631.9 10^3	45	40856.8 10^3
Total	100	921.8 10^3	100	2739.6 10^3	100	3649.8 10^3	100	40886.6 10^3

Table 5.10: Images exiting at each EE and corresponding energy consumption with dynamic threshold configuration - for 1M inferences

	Opt1		Opt2		Opt3	
	% im.	En.(J)	% im.	En.(J)	% im.	En.(J)
EE1	20	11318.4	67	11318.4	72	11318.4
EE2	11	11805	20	4869.6	21	4131.8
EE3	9	9917	11	1868.4	6.7	1006.1
Cloud	60	54478.4 10^3	2	1815.9 10^3	0.3	272.4 10^3
Total	100	54511.4 10^3	100	1833.9 10^3	100	288.8 10^3

Table 5.11: Performance sensitivity to the presence/absence of permanent energy sources

	Opt2			Opt4			Opt5		
	% im.	En.(J)	Acc.	% im.	En.(J)	Acc.	% im.	En.(J)	Acc.
EE1	67	11318.4		93	11318.4		79	11318.4	
EE2	20	4869.6		0.1	1032.9		19	3098.8	
EE3	11	1868.4	67.7%	6.1	991.7	58.4%	1.5	287.4	52.6%
Cloud	2	1815.9 10^3		0.8	726.4 10^3		0.5	454 10^3	
Total	100	1833.9 10^3		100	739.7 10^3		100	468.7 10^3	

5.4.3 Sensitivity to the presence/absence of permanent energy sources

As shown in Table 5.11, several experiments were conducted to test the sensitivity of the proposed approach to the presence or absence of permanent energy sources for the cameras.

In the first column, the test was carried out with the same scenario as in Table 5.7. The obtained solution optimises the overall energy consumption and the DNN's accuracy with equal weights.

In the second column, all the cameras are still connected with the same energy scenario except for camera 2. The amount of energy available for camera 2 is systematically fixed at 0.0001 Wh during the generation of successive linear programs, creating a bottleneck in the amount of available energy for camera 2. Thus, the optimisation approach will limit the number of inferences passing through to camera 3 as much as possible. Consequently, the energy availability constraints are respected at the cost of a significant drop in the accuracy of the DNN.

In the third column of the table, the energy bottleneck is shifted to camera 3, which operates continuously with only 0.008 Wh as long as camera 2 is in normal energy

scenario. This prevents a large number of images from passing through EE3 and going to the cloud, as they mostly exit through EE1 and EE2. Consequently, the precision of the DNN is expectedly reduced, while the energy consumption is further minimised.

5.4.4 Comparison with cloud-only processing

Tables 5.12, 5.13 and 5.14 show a comparison, in terms of energy consumption, of the approaches: camera + cloud, cloud only without EE and cloud only with EE for 1M inferences. For each of the last 2 approaches, several communication protocols have been tested. Two configurations were also tested for the hardware architecture of the cloud: identical to the cameras and classic server. The EE approaches use a 0.7 threshold.

We can see that our solution (column 1) is much less energy consuming than all the cloud-only type configurations without EE. In fact, even if the total amount of data transmitted via the network decreases by around 45% in the cloud-only approach, the overall consumption remains very high because of the costs of calculations on the cloud with inferences that cross all the layers of the DNN.

For cloud-only with EE case, the energy consumption remains very high when we consider a classical server hardware architecture because of the cost of the calculations. However, the particular case of cloud only with EE embedding a very low consumption RAM and a MIPS R3000 type processor gives a lower overall consumption than that found using the proposed methodology. Nevertheless, this is a totally improbable or even impossible case in real life.

Table 5.12: Energy consumption (in MJ) of Cameras + Cloud approach

Cam. + Cloud	
Wifi - LoRa	Wifi - 4G
1.83	1.84

Table 5.13: Energy consumption (in MJ) of Cloud-only without EE approach

Cloud-only without EE			
MIPS & LoRa	MIPS & 4G	Server & LoRa	Server & 4G
63.2	67	63.2	63.2

Table 5.14: Energy consumption (in MJ) of Cloud-only with EE approach

Cloud-only with EE			
MIPS & LoRa	MIPS & 4G	Server & LoRa	Server & 4G
3.27	4.90	5.61	5.78

5.4.5 Limitations

The main constraint, which is always assumed to be verified in the hardware architectures running the DNN, is that at least camera 1 (EE1) has available energy at all times, even if it may be in very small quantities. Indeed, if this camera is completely out of energy, this would mean that no more images can be processed by the DNN, because they will no longer be able to cross the first DNN layers.

Such a constraint is not necessary for the other cameras of the system, because an absence of energy in any camera after the first EE would create a cut in the DNN but will not prevent the processing of the images by the first layers of the DNN, at least until the first EE.

5.5 Summary

In this chapter, we introduce an energy consumption estimation and optimisation approach for DNNs with multi early exits. This approach is executed on a platform composed of several cameras and a cloud server, which enables real-time analysis and decision-making. Our approach leverages machine learning techniques to estimate the energy consumption of DNNs with different configurations and to identify the optimal configuration that minimises energy consumption while maintaining the desired accuracy level. We demonstrate the effectiveness of our approach through experiments on a real-world dataset and show that it can achieve significant energy savings compared to traditional DNNs.

Overall, our approach presents a promising solution for energy-efficient DNNs with multi early exits, which can be applied to various computer vision applications. The approach consists of a dynamic LP-based algorithm that is able to estimate energy consumption at compile time and then optimises the overall energy consumption, while taking into account energy sources changes during run time.

By reducing the energy consumption of DNNs, we can not only save costs but also promote sustainability and environmental friendliness in the development and deployment of AI systems.

6 | Conclusion and future work

6.1 Conclusion

Artificial intelligence (AI) and machine learning (ML) algorithms have revolutionised the field of computer vision by enabling automatic interpretation and understanding of images and videos. DNNs, in particular, have demonstrated remarkable performance in various visual recognition tasks, like object detection, segmentation, and tracking. However, due to the increasing complexity and size of DNNs, their computational requirements have significantly increased, leading to high energy consumption and execution time.

Early exits enable DNNs to make predictions at multiple stages of their computation, reducing their computational requirements and energy consumption by allowing early termination when a satisfactory accuracy level is achieved, rather than waiting for the final layer to output a prediction. However, optimising and designing DNNs with early exits is a complex task that requires careful consideration of various factors, such as the number and location of early exits, accuracy threshold for early termination, and tradeoffs between accuracy, execution time and energy consumption. This is even more challenging when the DNN is distributed accross hybrid cloud/edge platform.

The main contributions presented in this thesis are the following:

- An approach to split neural network inference between the edge and the cloud while optimising both bandwidth and memory constraints. For this, we proposed a novel method which jointly trains knowledge distillation at the edge and intermediate data compression with an auto-encoder. This approach allows to find the optimal partition under specific constraints. It

makes it possible to execute part of the network on very constrained devices such as MCUs, and under poor network conditions (LoRa) while limiting the accuracy drop. Moreover, we showed that the optimal splitting layers are not necessarily the upper ones, as suggested by previous work, but must be chosen carefully according to the constraints. This is a promising technique for applications with limited bandwidth and memory size at the edge devices. It enables to have guidelines to choose the splitting layer depending on these constraints. This optimal splitting may vary over time according to the evolution of the constraints.

- A methodology for distributing DNN inference across multiple clients at the edge and the cloud. By including early-exits and bottlenecks in the neural network, the solution we propose makes it possible to distribute computation according to time and accuracy requirements. In scenarios where the communication with the cloud is very limited (e.g. LoRa), it allows a 80% improvement in average latency compared to a traditional cloud solution.
- An energy consumption estimation and optimisation approach for DNNs with multi early exits in a platform composed of several cameras and a cloud server. The approach utilises machine learning combined with dynamic linear programming techniques to estimate the energy consumption of different DNN configurations and identify the optimal configuration that minimises energy consumption while maintaining the desired accuracy level. This approach has important implications for promoting sustainability and environmental friendliness in the development and deployment of AI systems while reducing energy consumption costs.

All contributions have been systematically validated by detailed experiments using datasets of significant sizes. The results were exposed and analysed in order to show the relevance and effectiveness of each contribution.

6.2 Directions for future work

The design and optimisation of distributed DNNs in a multi-camera environment with a cloud server implies a great scientific and technical complexity. It concerns a wide spectrum of disciplines such as: hardware design, multi-criteria optimisation, networking, modeling, development, etc. Thus, there several potential improvements to the contributions made in this thesis work, as presented below.

6.2.1 Mobile cameras

6.2.1.1 Research problem

In real life video surveillance systems, there can be mobile cameras, such as cameras fixed on motorised wheels or drones. This introduces more complexity into any performance optimisation approach, because it adds dynamicity to certain parameters of the hardware execution platform. Indeed, the mobility of certain cameras implies possible variations in protocols and network speeds, and also leads to additional energy consumption due to mechanical movements. Moreover, such cameras may not be easily connected to a power outlet, which presents an additionnal constraint.

6.2.1.2 Ideas for potential solutions

A potential solution is to design an hybrid dynamic energy consumption estimation approach. In the case of a system including mobile cameras, there would be

three potential dynamic energy consumption types in addition to those we have seen in Chapter 5 :

- Consumption due to the movements made in order to capture specific scenes (motion detection for example).
- Consumption due to the normal movement of the camera independently of the scene's context (regular and systematic rotations of a drone for example).
- Consumption due to the camera's movements within the collaboration framework (moving physically closer to the next camera in order to use a faster communication protocol for example).

These different movements can be estimated using an approach that has, in a library, the energy costs of the various atomic movements. With these potential movements and the interactions between cameras, a dynamic methodology of the heuristic or meta heuristic type, sensitive to events of interest, could be well adapted as a resolution approach. In fact, the estimation of the mechanical movements cannot be obtained as simply and quickly as in Chapter 5, because the trajectories and the corresponding movements must be estimated at each event. This would therefore make approaches based on exact LP-type methods potentially too slow.

6.2.2 Fault-tolerant multi-camera system and task migration

6.2.2.1 Research problem

Fault tolerance in a multi-camera surveillance system (including several video sensors) refers to the ability of the system to continue functioning even in there is failure or malfunction in one or more of its components. It constitutes an essential

feature in many real life application domains, and it is important for ensuring continuous and reliable surveillance coverage in critical areas, such as airports, government facilities, and high-security locations.

Thus, ensuring fault tolerance in multi camera systems executing a multi early exits DNN is crucial, as failure in a single camera or a battery depletion can lead to significant downtime and affect system performance. Such cases can be particularly problematic when the DNN layers distribution is done statically, i.e. the assignment of the early-exits on the various cameras remains fixed after the deployment of the DNN. Indeed, such approaches can pose particular problems in the following cases:

- Critical amount of available energy on a given camera.
- Hardware failure of a camera.
- Communication network problems.

6.2.2.2 Ideas for potential solutions

- Tasks migration: One solution to this challenge could be tasks migration, where the workload of a failed or low-battery camera is reassigned to other cameras in the system. Task migration can help ensure fault tolerance by maintaining system performance even when a camera fails. Additionally, this approach can increase the overall lifespan of the system by balancing the workload between cameras, which can prolong the battery life of each camera. We can investigate the impact of different task migration strategies on fault tolerance and overall system performance, including execution time and energy consumption. Specifically, we could evaluate the effectiveness of two migration strategies: (1) distributing the workload of the failed or low-battery camera to other available cameras with early exits (executing a

part of the DNN, and (2) distributing the workload to cameras without early exits (free cameras that are not executing any DNN layers).

- Run time executable tasks selection : This involves introducing some redundancy in terms of compiled executable code, i.e. having several executable codes at compile time on the cameras and the cloud, instead of having only one. A controller, at the cloud level, would dynamically assign the executable codes to the different devices as needed during run time. This would consist, for example, of having different alternative executable codes available for camera 1. These codes would be: 1) the first DNN layers up to EE1; 2) DNN layers from EE1 to EE2, 3) DNN layers from start to EE2; 4) all the DNN layers, etc.

6.2.3 FPGA-based execution platforms

6.2.3.1 Research problem

Field Programmable Gate Arrays (FPGAs) are digital integrated circuits that can be programmed and reprogrammed to perform a variety of digital functions. They are commonly used in applications that require high performance, low latency, and low power consumption, such as digital signal processing, networking, and image processing. In fact, they are more and more used to implement AIs. An AI accelerator is a kind of specialized hardware designed to accelerate artificial intelligence and machine learning applications, including artificial neural networks and machine vision. Reconfigurable devices such as FPGA make it easier to evolve hardware, frameworks, and software alongside each other. Such devices start being more and more used to accelerate inference processing [MicrosoftResearch, 2023].

Recent FPGAs, from Xilinx in particular, make it possible to do what is called partial dynamic reconfiguration (PDR). This consists in reprogramming a part of the FPGA in a hardware way to implement a new function replacing the previous one, without stopping the execution of the rest of the application.

In multi-camera surveillance system with mobile cameras, taking advantage of PDR feature for a global optimisation approach implies several challenges.

6.2.3.2 Ideas for potential solutions

The use of modern FPGAs capable of making PDRs, with a hardware implementation of the DNN layers, could meet a large number of scientific and technical challenges as described below :

- Possibility of integrating the PDR at the communication control level: during run time, if necessary, the switch from a communication protocol to a another one, more adapted to the application context or to the external environment of the system, could be realised in an efficient and fast way using the PDR. The change of protocol can also be consecutive to a failure of one of the network controllers or to a change of optimisation criterion, passing from minimising the execution time to minimising energy consumption for example.
- Fast hardware tasks and context switching: the PDR not only makes it possible to configure the hardware logic to perform functions, but also the RAM and FIFO zones for data storage.

Therefore, FPGAs seem to be an excellent target for multi-camera video surveillance systems using distributed DNN. However, their use poses certain problems of great complexity, like the design of the reconfiguration controller, which is a tedious and error prone task, also largely conditioning the final performance. Such

a controller could be realised in software or hardware on the FPGA. It can also be designed in a centralised, distributed or hybrid way on the different cameras.

References

[Aghajan and Cavallaro, 2009] Aghajan, H. and Cavallaro, A. (2009). Front matter. In Aghajan, H. and Cavallaro, A., editors, *Multi-Camera Networks*, page iii. Academic Press, Oxford.

[Ahmed et al., 2020] Ahmed, M., Mumtaz, R., Zaidi, S. M. H., Hafeez, M., Zaidi, S. A. R., and Ahmad, M. (2020). Distributed fog computing for internet of things (iot) based ambient data processing and analysis. *Electronics (Basel)*, 9(11):1–20.

[Alessandrelli et al., 2012] Alessandrelli, D., Azzarà, A., Petracca, M., Nastasi, C., and Pagano, P. (2012). Scantraffic: Smart camera network for traffic information collection. volume 7158, pages 196–211.

[Alex et al., 2012] Alex, K., Ilya, S., and Geoffrey, H. (2012). Classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Arlo, 2023] Arlo (2023). Arlo pro 4 outdoor security camera.

[Ashish et al., 2015] Ashish, V., Carlo, C., Philip, G., Thomas, J., Konstantinos, K., Jitendra, P., and George, V. (2015). Wanalytics : Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1087–1092. ACM.

[BabakAminazad, 2023] BabakAminazad (2023). Mips r3000 verilog code.

[Bengio et al., 2016] Bengio, E., Bacon, P.-L., Pineau, J., and Precup, D. (2016). Conditional computation in neural networks for faster models.

[Bengio et al., 2013] Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation.

[Berestizshevsky and Even, 2019] Berestizshevsky, K. and Even, G. (2019). *Dynamically Sacrificing Accuracy for Reduced Computation: Cascaded Inference Based on Softmax Confidence*, pages 306–320. Springer.

[Bhattacharya and Lane, 2016] Bhattacharya, S. and Lane, N. D. (2016). Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, page 176–189, New York, NY, USA. Association for Computing Machinery.

[Bolei et al., 2014] Bolei, Z., Agata, L., Jianxiong, X., Antonio, T., and Aude, O. (2014). Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495.

[Bonato and Bouganis, 2021] Bonato, V. and Bouganis, C.-S. (2021). Class-specific early exit design methodology for convolutional neural networks. *Applied Soft Computing*, 107:107316.

[Chen et al., 2018] Chen, Z., Li, Y., Bengio, S., and Si, S. (2018). Gaternet: Dynamic filter selection in convolutional neural network via a dedicated global gating network. *CoRR*, abs/1811.11205.

[Cheng et al., 2018] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2018). Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136.

[Chien et al., 2015] Chien, S.-Y., Chan, W.-K., Tseng, Y.-H., Lee, C.-H., Somayazulu, V. S., and Chen, Y.-K. (2015). Distributed computing in iot: System-

on-a-chip for smart cameras as an example. In *The 20th Asia and South Pacific Design Automation Conference*, pages 130–135.

[Chien-Chun et al., 2018] Chien-Chun, H., Ganesh, A., Leana, G., Minlan, Y., and Mingyang, Z. (2018). Wide-area analytics with multiple resources. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–16. ACM.

[Cho and Bengio, 2014] Cho, K. and Bengio, Y. (2014). Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning.

[Choi et al., 2020] Choi, H., Cohen, R. A., and Bajic, I. V. (2020). Back-and-forth prediction for deep tensor compression.

[Chowdhery et al., 2019] Chowdhery, A., Warden, P., Shlens, J., Howard, A., and Rhodes, R. (2019). Visual wake words dataset.

[Courbariaux et al., 2015] Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 3123–3131. Curran Associates, Inc.

[Curtin and Matthews, 2019] Curtin, B. and Matthews, S. (2019). Deep learning for inexpensive image classification of wildlife on the raspberry pi. pages 0082–0087.

[DominicSweetman, 2023] DominicSweetman (2023). Idt r30xx family software reference manual.

[Dong et al., 2020] Dong, Y., Zhao, P., Yu, H., Zhao, C., and Yang, S. (2020). Cdc: Classification driven compression for bandwidth efficient edge-cloud collaborative deep learning. *arXiv preprint arXiv:2005.02177*.

[Elbayad et al., 2019] Elbayad, M., Gu, J., Grave, E., and Auli, M. (2019). Depth-adaptive transformer. *CoRR*, abs/1910.10073.

[Emmons et al., 2019] Emmons, J., Fouladi, S., Ananthanarayanan, G., Venkataraman, S., Savarese, S., and Winstein, K. (2019). Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, HotEdgeVideo'19, page 27–32, New York, NY, USA. Association for Computing Machinery.

[Eshratifar et al., 2019] Eshratifar, A. E., Esmaili, A., and Pedram, M. (2019). Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6.

[Frosst and Hinton, 2017] Frosst, N. and Hinton, G. E. (2017). Distilling a neural network into a soft decision tree. *CoRR*, abs/1711.09784.

[Gatys et al., 2015] Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A neural algorithm of artistic style.

[Gerry, 2023] Gerry (2023). 100 sports image classification.

[Google, 2016] Google (2016). The Google Brain is a real thing but very few people have seen it. http://www.businessinsider.com/what-isgoogle-brain-2016-9.

[Gou et al., 2021] Gou, J., Yu, B., Maybank, S. J., and Tao, D. (2021). Knowledge distillation: A survey.

[Grulich and Nawab, 2018] Grulich, P. M. and Nawab, F. (2018). Collaborative edge and cloud neural networks for real-time video processing. *Proc. VLDB Endow.*, 11(12):2046–2049.

[GSMArena, 2023a] GSMArena (2023a). Apple iphone 13.

[GSMArena, 2023b] GSMArena (2023b). Samsung galaxy s20.

[Guo et al., 2017] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. *CoRR*, abs/1706.04599.

[Guo et al., 2016] Guo, Y., Yao, A., and Chen, Y. (2016). Dynamic network surgery for efficient dnns. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29, pages 1379–1387. Curran Associates, Inc.

[Gupta et al., 2015] Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, ICML'15, page 1737–1746. JMLR.org.

[Han et al., 2017] Han, S., Pool, J., Narang, S., Mao, H., Gong, E., Tang, S., Elsen, E., Vajda, P., Paluri, M., Tran, J., Catanzaro, B., and Dally, W. J. (2017). Dsd: Dense-sparse-dense training for deep neural networks.

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 1135–1143. Curran Associates, Inc.

[Han et al., 2021] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. (2021). Dynamic neural networks: A survey.

[Hauswald et al., 2015] Hauswald, J., Kang, Y., Laurenzano, M. A., Chen, Q., Li, C., Mudge, T., Dreslinski, R. G., Mars, J., and Tang, L. (2015). Djinn and tonic: Dnn as a service and its implications for future warehouse scale comput-

ers. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 27–40. IEEE.

[Hinton et al., 2012] Hinton, G., Deng, l., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29:82–97.

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[Hu and Krishnamachari, 2020] Hu, D. and Krishnamachari, B. (2020). Fast and accurate streaming cnn inference via communication compression on the edge. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 157–163.

[Huang et al., 2017a] Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. (2017a). Multi-scale dense convolutional networks for efficient prediction. *t ICLR 2018*, page 14.

[Huang et al., 2017b] Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. Q. (2017b). Multi-scale dense convolutional networks for efficient prediction. *CoRR*, abs/1703.09844.

[Hubara et al., 2016] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Infor-*

mation Processing Systems, volume 29, pages 4107–4115. Curran Associates, Inc.

[IBMCorp., 2023] IBMCorp. (2023). Ibm cplex optimizer.

[IBMcorp., 2023] IBMcorp. (2023). Ibm power7 energy consumption.

[InfineonInc, 2023] InfineonInc (2023). Cy62146esl mobl data sheet.

[Jankowski et al., 2020] Jankowski, M., Gündüz, D., and Mikolajczyk, K. (2020). Joint device-edge inference over wireless links with pruning. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5.

[Jie et al., 2021] Jie, Z., Sun, P., Li, X., Feng, J., and Liu, W. (2021). Anytime recognition with routing convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(6):1875–1886.

[Jong Hwan et al., 2018] Jong Hwan, K., Taesik, N., Mohammad Faisal, A., and Saibal, M. (2018). Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In *Proceedings of the 15th IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 1–6. IEEE.

[Jouppi et al., 2017] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12.

[Kang et al., 2017] Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. (2017). Neurosurgeon: Collaborative intelligence be-

tween the cloud and mobile edge. In *ACM SIGPLAN Notices*, volume 45, page 615–629, New York, NY, USA. Association for Computing Machinery.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Klemens et al., 2021] Klemens, J. A., Tripepi, M., and McFoy, S. A. (2021). A motion-detection based camera trap for small nocturnal mammals with low latency and high signal-to-noise ratio. *Methods in Ecology and Evolution*, 12(7):1323–1328.

[Laskaridis et al., 2020a] Laskaridis, S., Venieris, S. I., Almeida, M., Leontiadis, I., and Lane, N. D. (2020a). SPINN: synergistic progressive inference of neural networks over device and cloud. *CoRR*, abs/2008.06402.

[Laskaridis et al., 2020b] Laskaridis, S., Venieris, S. I., Almeida, M., Leontiadis, I., and Lane, N. D. (2020b). Spinn: Synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, New York, NY, USA. Association for Computing Machinery.

[Li et al., 2018] Li, E., Zhou, Z., and Chen, X. (2018). Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *The SIGCOMM Workshop on Mobile Edge Communications*, pages 31–36.

[Li et al., 2019] Li, H., Zhang, H., Qi, X., Yang, R., and Huang, G. (2019). Improved techniques for training adaptive deep networks. *CoRR*, abs/1908.06294.

[Lin et al., 2020] Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., and Han, S. (2020). Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*.

[Lin et al., 2017] Lin, J., Rao, Y., Lu, J., and Zhou, J. (2017). Runtime neural pruning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Liu and Deng, 2017] Liu, L. and Deng, J. (2017). Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *CoRR*, abs/1701.00299.

[Liu et al., 2016] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

[Lo et al., 2017] Lo, C., Su, Y.-Y., Lee, C.-Y., and Chang, S.-C. (2017). A dynamic deep neural network design for efficient workload allocation in edge computing. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 273–280.

[Lovejoy, 2015] Lovejoy, B. (2015). Apple moves to third-generation Siri backend, built on opensource Mesos platform. http://9to5mac.com/2015/04/27/siribackend-mesos/.

[Luo et al., 2017] Luo, J.-H., Wu, J., and Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression.

[Mathur and Khattar, 2019] Mathur, A. and Khattar, S. (2019). Real-time wildlife detection on embedded systems.

[Matsubara and Levorato, 2020] Matsubara, Y. and Levorato, M. (2020). Neural compression and filtering for edge-assisted real-time object detection in challenged networks.

[Matsubara et al., 2022] Matsubara, Y., Levorato, M., and Restuccia, F. (2022). Split computing and early exiting for deep learning applications: Survey and research challenges.

[Microchip, 2023] Microchip (2023). Atsam4e16e.

[MicrochipTechnologyInc, 2023a] MicrochipTechnologyInc (2023a). Low-power long range lora technology transceiver module.

[MicrochipTechnologyInc, 2023b] MicrochipTechnologyInc (2023b). Zg2100m/zg2101m wi-fi module data sheet.

[MicrosoftResearch, 2023] MicrosoftResearch (2023). Project brainwave.

[Molchanov et al., 2017] Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*.

[Monteiro et al., 2021] Monteiro, A., Santos, S., and Gonçalves, P. (2021). Precision agriculture for crop and livestock farming—brief review. *Animals*, 11(8).

[Neshatpour et al., 2019] Neshatpour, K., Behnia, F., Homayoun, H., and Sasan, A. (2019). Exploiting energy-accuracy trade-off through contextual awareness in multi-stage convolutional neural networks. In *20th International Symposium on Quality Electronic Design (ISQED)*, pages 265–270.

[NumPy, 2023] NumPy (2023). numpy.savez compressed.

[NvidiaCorporation, 2023] NvidiaCorporation (2023). Meet jetson, the platform for ai at the edge.

[NXP, 2023a] NXP (2023a). Nxp i.mx rt series: Crossover processor.

[NXP, 2023b] NXP (2023b). Nxp mcus.

[Osia et al., 2020] Osia, S. A., Shahin Shamsabadi, A., Sajadmanesh, S., Taheri, A., Katevas, K., Rabiee, H. R., Lane, N. D., and Haddadi, H. (2020). A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet of Things Journal*, 7(5):4505–4518.

[Park et al., 2015] Park, E., Kim, D., Kim, S., Kim, Y.-D., Kim, G., Yoon, S., and Yoo, S. (2015). Big/little deep neural network for ultra low power inference. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 124–132.

[Patel, 2019] Patel, D. (2019). *Distributed Computing for Internet of Things (IoT)*, pages 84–109. IGI Global.

[Peisong et al., 2018] Peisong, W., Qinghao, H., Yifan, Z., Chunjie, Z., Yang, L., and Jian, C. (2018). Two-step quantization for low-bit neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4376–4384. IEEE.

[Phuong and Lampert, 2019] Phuong, M. and Lampert, C. (2019). Distillation-based training for multi-exit architectures. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1355–1364.

[Prado et al., 2020] Prado, M. D., Su, J., Saeed, R., Keller, L., Vallez, N., Anderson, A., Gregg, D., Benini, L., Llewellynn, T., Ouerhani, N., Dahyot, R., and Pazos, N. (2020). Bonseyes ai pipeline—bringing ai to you: End-to-end integration of data, algorithms, and deployment tools. *ACM Trans. Internet Things*, 1(4).

[Price et al., 2018] Price, E., Lawless, G., Ludwig, R., Martinovic, I., Bülthoff, H. H., Black, M. J., and Ahmad, A. (2018). Deep neural network-based coop-

erative visual tracking through multiple micro aerial vehicles. *IEEE Robotics and Automation Letters*, 3:3193–3200.

[Qassim et al., 2017] Qassim, H., Feinzimer, D., and Verma, A. (2017). Residual squeeze vgg16.

[Ramos-Arredondo et al., 2020] Ramos-Arredondo, R. I., Carvajal-Gámez, B. E., Gendron, D., Gallegos-Funes, F. J., Mújica-Vargas, D., and Rosas-Fernández, J. B. (2020). Photoid-whale: Blue whale dorsal fin classification for mobile devices. *PLOS ONE*, 15(10):1–19.

[RaspberryPiLtd, 2023] RaspberryPiLtd (2023). Raspberry pi 4 model b specifications.

[Ray, 2022] Ray, P. P. (2022). A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623.

[Remagnino et al., 2004] Remagnino, P., Shihab, A., and Jones, G. (2004). Distributed intelligence for multi-camera visual surveillance. *Pattern Recognition*, 37(4):675–689. Agent Based Computer Vision.

[Rigamonti et al., 2013] Rigamonti, R., Sironi, A., Lepetit, V., and Fua, P. (2013). Learning separable filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Rohilla et al., 2020] Rohilla, R., Banga, P., Garg, P., and Mittal, P. (2020). Gpu based re-trainable pruned cnn design for camera trapping at the edge. pages 558–563.

[Romero et al., 2015] Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). Fitnets: Hints for thin deep nets. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Represen-*

tations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

[Rota Bulo and Kontschieder, 2014] Rota Bulo, S. and Kontschieder, P. (2014). Neural decision forests for semantic image labelling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Ryan et al., 2014] Ryan, D., Denman, S., Fookes, C., and Sridharan, S. (2014). Scene invariant multi camera crowd counting. *Pattern Recognition Letters*, 44:98–112. Pattern Recognition and Crowd Analysis.

[Sandler et al., 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

[Saputra et al., 2019] Saputra, M. R. U., Gusmao, P. P. B. d., Almalioglu, Y., Markham, A., and Trigoni, N. (2019). Distilling knowledge from a deep pose regressor network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[Segalla et al., 2020] Segalla, A., Fiacco, G., Tramarin, L., Nardello, M., and Brunelli, D. (2020). Neural networks for pest detection in precision agriculture. In *2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, pages 7–12.

[Shadrin et al., 2020] Shadrin, D., Menshchikov, A., Somov, A., Bornemann, G., Hauslage, J., and Fedorov, M. (2020). Enabling precision agriculture through embedded sensing with artificial intelligence. *IEEE Transactions on Instrumentation and Measurement*, 69(7):4103–4113.

[Shafiee et al., 2019] Shafiee, M. S., Shafiee, M. J., and Wong, A. (2019). Dynamic representations toward efficient inference on deep neural networks by decision gates.

[Shao and Zhang, 2020] Shao, J. and Zhang, J. (2020). Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems.

[Shen et al., 2020] Shen, J., Fu, Y., Wang, Y., Xu, P., Wang, Z., and Lin, Y. (2020). Fractional skipping: Towards finer-grained dynamic CNN inference. *CoRR*, abs/2001.00705.

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

[StanfordVisionLab., 2023] StanfordVisionLab. (2023). Download imagenet data.

[STMicroelectronics, 2023a] STMicroelectronics (2023a). Stm32 high performance mcus.

[STMicroelectronics, 2023b] STMicroelectronics (2023b). Stm32 nucleo boards.

[STMicroelectronics, 2023c] STMicroelectronics (2023c). Stm32f7 series of mcus.

[Sylvain Gugger, 2023] Sylvain Gugger, J. H. (2023). Fastai - external data.

[Teerapittayanon et al., 2017a] Teerapittayanon, S., McDanel, B., and Kung, H. (2017a). Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339.

[Teerapittayanon et al., 2017b] Teerapittayanon, S., McDanel, B., and Kung, H. T. (2017b). Branchynet: Fast inference via early exiting from deep neural networks. *CoRR*, abs/1709.01686.

[Thomassen, 2017] Thomassen, S. (2017). Embedded analytics of animal images. Master's thesis, The Artic University of Norway.

[Torres-Zapata et al., 2019] Torres-Zapata, E., Guerra, V., Rabadan, J., Perez-Jimenez, R., and Luna-Rivera, J. M. (2019). Vehicular communications in tunnels using vlc. In *2019 15th International Conference on Telecommunications (ConTEL)*, pages 1–6.

[Vincent et al., 2011] Vincent, V., Andrew, S., and Mark, M. (2011). Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, volume 1.

[Wang et al., 2019] Wang, M., Mo, J., Lin, J., Wang, Z., and Du, L. (2019). Dynexit: A dynamic early-exit strategy for deep residual networks. In *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 178–183.

[Wang et al., 2018] Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., and Gonzalez, J. E. (2018). Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[Wilson, 2023] Wilson, R. (2023). Playstation to tesla, mips r2000 still going strong at 30.

[Wolczyk et al., 2021] Wolczyk, M., Wójcik, B., Balazy, K., Podolak, I. T., Tabor, J., Smieja, M., and Trzcinski, T. (2021). Zero time waste: Recycling predictions in early exit neural networks. *CoRR*, abs/2106.05409.

[Wong et al., 2020a] Wong, A., Famouri, M., Pavlova, M., and Surana, S. (2020a). Tinyspeech: Attention condensers for deep speech recognition neural networks on edge devices.

[Wong et al., 2020b] Wong, A., Famouri, M., and Shafiee, M. J. (2020b). Attendnets: Tiny deep image recognition neural networks for the edge via visual attention condensers. *CoRR*, abs/2009.14385.

[Wołczyk et al., 2021] Wołczyk, M., Wójcik, B., Bałazy, K., Podolak, I., Tabor, J., Śmieja, M., and Trzciński, T. (2021). Zero time waste: Recycling predictions in early exit neural networks.

[Wu et al., 2018] Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L., Grauman, K., and Feris, R. (2018). Blockdrop: Dynamic inference paths in residual networks. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[XilinxInc, 2023] XilinxInc (2023). Vivado overview.

[Xin et al., 2020] Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. (2020). Deebert: Dynamic early exiting for accelerating BERT inference. *CoRR*, abs/2004.12993.

[Xu et al., 2020a] Xu, S., Zhang, Z., and Kadoch, M. (2020a). A collaborative cloud-edge computing framework in distributed neural network. *J Wireless Com Network 2020*, (211).

[Xu et al., 2020b] Xu, S., Zhang, Z., Kadoch, M., and Cheriet, M. (2020b). A collaborative cloud-edge computing framework in distributed neural network. *EURASIP Journal on Wireless Communications and Networking*, 2020:211.

[Yang et al., 2022] Yang, S., Ding, F., Li, P., and Hu, S. (2022). Distributed multi-camera multi-target association for real-time tracking. *Scientific Reports*, 12:11052.

[Yaniv et al., 2014] Yaniv, T., Ming, Y., MarcAurelio, R., and Lior, W. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708.

[Yao et al., 2020a] Yao, S., Hao, Y., Zhao, Y., Shao, H., Liu, D., Liu, S., Wang, T., Li, J., and Abdelzaher, T. (2020a). Scheduling real-time deep learning services as imprecise computations. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10. IEEE.

[Yao et al., 2020b] Yao, S., Li, J., Liu, D., Wang, T., Liu, S., Shao, H., and Abdelzaher, T. (2020b). Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, page 476–488. Association for Computing Machinery.

[Yao et al., 2017] Yao, S., Zhao, Y., Zhang, A., Su, L., and Abdelzaher, T. (2017). Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–14.

[Yim et al., 2017] Yim, J., Joo, D., Bae, J.-H., and Kim, J. (2017). A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7130–7138.

[Yu et al., 2018] Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. S. (2018). Slimmable neural networks. *CoRR*, abs/1812.08928.

[Zeng et al., 2019] Zeng, L., Li, E., Zhou, Z., and Chen, X. (2019). Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things. *IEEE Network*, 33(5):96–103.

[Zhang et al., 2020] Zhang, M., Zhang, F., Lane, N. D., Shu, Y., Zeng, X., Fang, B., Yan, S., and Xu, H. (2020). Deep learning in the era of edge computing: Challenges and opportunities.

[Zhao et al., 2018] Zhao, Z., Barijough, K. M., and Gerstlauer, A. (2018). Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359.

[Zhou et al., 2021] Zhou, A., Muller, R., and Rabaey, J. M. (2021). Memory-efficient, limb position-aware hand gesture recognition using hyperdimensional computing. *CoRR*, abs/2103.05267.

[Zualkernan et al., 2022] Zualkernan, I., Dhou, S., Judas, J., Sajun, A. R., Gomez, B. R., and Hussain, L. A. (2022). An iot system using deep learning to classify camera trap images on the edge. *Computers*, 11(1).

[Zumimmal, 2023] Zumimmal (2023). Zumimall 2k outdoor rechargeable battery wifi security camera with solar panel-q1prok.