MPSoC FPGA Implementation of Algorithms of Machine Learning for Clinical Applications Using High-Level Design Methodology

Mario Daniel Guanche Hernández, Pedro Pérez Carballo, Sonia Raquel León Martín
Instituto Universitario de Microelectrónica Aplicada (IUMA), Universidad de Las Palmas de Gran Canaria (ULPGC)
Las Palmas de Gran Canaria, Spain
mguanche@iuma.ulpgc.es

***Abstract*— This work presents the design of a hardware/software system designed using high-level design methodologies, consisting of a software application running on a host and a set of hardware acceleration nodes implemented in an FPGA device. The ultimate goal is to have an integrated system on an MPSoC FPGA device for skin cancer detection using both hyperspectral imaging and a k-means algorithm. The hardware acceleration system is designed using three FPGA kernels. The first two filter and normalize the hyperspectral image. The last kernel then runs k-means to segment the image into three regions: healthy skin and a lesion. FPGA acceleration significantly improves the performance and power consumption of the application compared to software execution or other alternatives such as implementation on GPUs.**

Keywords – hyperspectral imaging; k-means; MPSoC; FPGA; HLS; skin cancer.

I. INTRODUCTION

Hyperspectral imaging (HSI) is a technique that combines spatial and spectral information. By spanning hundreds of spectral bands, HSI can capture a wealth of information, both within and beyond the range of vision of the human eye. Each pixel in an HSI image contains a near-continuous spectrum called a spectral signature[1]. These characteristics make hyperspectral imaging useful for assessing the dispersion of material components. As a result, this technique is gaining relevance in many fields, such as medicine and healthcare in general [2], [3].

In this work, a set of design techniques have been developed to enable the development of an application that uses hyperspectral imaging for skin cancer detection. To improve its performance, the application takes advantage of hardware acceleration using FPGA MPSoC devices. As this is a data-intensive problem, it is necessary, on the one hand, to study the computational requirements and, on the other hand, the data transfer to obtain results in reasonable times with reasonable power consumption requirements.

The application requires a first pre-processing stage to minimize the unwanted influence of various factors generated during the hyperspectral image capture, apart from the material nature of what is captured in the image, including artifacts in the cameras, and lighting effects, among others. In addition, it includes a second stage that allows automatic discrimination of the regions of skin tissue evaluated with different states of affectation by using the spectral information of the pre-processed pixels. For this purpose, an unsupervised automatic learning algorithm based on k-means is used[2], [4].

This application allows skin cancer cases to be diagnosed with a better combination of speed and accuracy when compared to other techniques, such as direct visual examination [5]. The speed improvements are due to the use of hyperspectral imaging that provides a broader and more detailed electromagnetic spectrum in the evaluation of the skin than direct visual examination can provide. This quality increases the accuracy in delineating areas of skin involvement, with the advantage that hyperspectral imaging is noninvasive and nonionizing [3], [5].

The main drawback of applications of techniques using hyperspectral imaging and machine learning is the enormous amount of data and processing required. This situation makes the execution in serial or sparsely parallelized computation models of the application, as is the case of microprocessors, very time-consuming and energy-intensive. However, the algorithms associated with the processing of these applications offer good possibilities of parallelism, the exploitation of which would greatly reduce the execution time, in addition to improving the ratio between time and energy consumption. For this purpose, it has been decided to implement a set of kernels to perform the hardware acceleration of the application. For this purpose, a GPU-based implementation has been used as a reference [5].

II. MATERIALS & METHODS

A. HS skin cancer dataset

The input data set for this application consists of three hyperspectral images (P15_C1000, "P15_C2000", "P60_C1003). These images have a spatial dimensionality of 50 x 50. The spectral dimensionality consists of 125 bands. However, the first 4 bands and the last 5 bands are discarded because they do not include significant information. Within the image, each data is a 16-bit unsigned integer. The image content is serialized following the band sequential criterion (BSQ). Figure 1 presents these images as grayscale images.

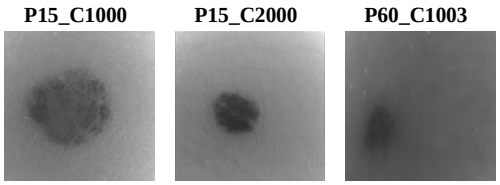

Figure 1. Hyperspectral images [6]

B. *Target system*

The prototyping platform used for this application is the Xilinx ZCU102 board, which includes a Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC device. The MPSoC is organized into two distinct systems, PS (Processing System), and PL (Programmable Logic), all included inside the same chip. The main components of the PS are the microprocessors, where the APU (Application Processing Unit), the RPU (Real-Time Processing Unit), and the GPU (Graphics Processing Unit) are located. APU consists of four Cortex™-A53 MPCore processors, an L2 cache, and related circuits. PL is constituted by the FPGA fabrics, including hardware resources that are highly replicated, like as LUTs (Look-Up Table), BRAMs (Block RAM), and DSPs (Digital Signal Processors) [7], [8], [9].

III. ARCHITECTURE DESIGN

The basic execution scheme of this application consists of a host, executed on a microprocessor system, which sequentially processes the computation flow with high data dependency and is supported by a set of kernels implemented in hardware, optimized to perform specific functions. The host manages data transfers and synchronization with hardware accelerators.

The execution of the application has two stages. The first stage corresponds to pre-processing. This stage has three main steps: (1) remove extreme spectral bands; (2) filter the spectral signatures of the pixels; (3) normalize the spectral signatures of the pixel.

Because hyperspectral cameras have poor sensitivity in extreme spectral bands, it is necessary to filter these bands. For this application, the first four and the last five spectral bands are discarded. It is possible to set other configurations during application construction.

The next step is smooth filtering for each pixel in the image, which reduces the influence of spectral noise on the image data. The filtering algorithm processes each spectral band by averaging the pixel value in that band with a symmetric number of upper and lower bands lying next to that band. The total amount of data on average for each spectral band is typically five. However, another number can be set during compilation. Particular cases are those spectral bands that are close to the upper or lower limit of the spectrum considered and therefore do not have enough spectral bands on one of their sides. For such cases, less data is considered, but symmetry is maintained.

For the filtering step, the host uses a dedicated kernel, implemented on the FPGA. This kernel uses the hardware implementation of the smooth filtering algorithm and saves the maximum, minimum, and scale (maximum-minimum) values found in the filtered pixels in the evaluation. This process is parallelized for a specific number of pixels, defined during kernel configuration. Figure 2 shows a simplified diagram of the operation of the kernel for filtering.

Due to the operation of this first FPGA kernel, the image data must be sent to this kernel in a specific order. During this process, values of the rejected spectral bands are avoided, thus fulfilling the first step in the preprocessing.

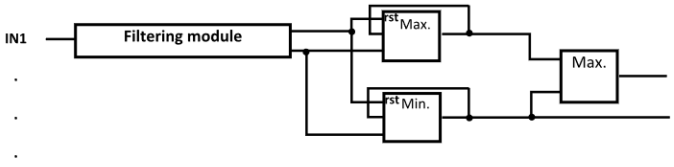

Figure 2. The basic architecture of the filtering kernel.

Each data in the same batch belongs to the same spectral band. In addition, data in the same batch belongs to adjacent pixels if the spatial dimensionality of the image has been serialized. When there are not enough new pixels to fill a batch, padding values are included. Figure 3 shows how the image content is taken and transferred to the kernel for the case of a BSQ image and kernel processing parallelism of 6. This procedure is analogous to any other possible case.

Figure 3. Ordering and transfer of the image content to the filtering kernel

The normalization process consists of scaling the spectral signature of each pixel to values between 0 and 1. This homogenizes the amplitude in the spectral signature of the different pixels in the image. A new FPGA kernel is used to perform the normalization process. This kernel takes the data, maximum, and scale values of the filtered pixels calculated by the previous kernel. The normalization kernel must be parallelized for the same pixel sizes as the filtering kernel. In addition, the filtered pixels are normalized in the same order in which they were generated.

The second step is to use k-means to distinguish regions in the skin tissue from the image according to their different states of affection. The number of regions is equal to the number of clusters in the k-means algorithm. According to [5], the best results are obtained for segmentation in 3 clusters. In this application, the same number of clusters is considered, but the application can be rebuilt for another number. The k-means algorithm is executed on FPGA as a kernel. This kernel has 4 execution modes: (1) centroid initialization; (2) distance estimation; (3) centroid update; (4) centroid transfer.

The first mode corresponds to the centroid initialization, where for each centroid a different pixel is selected as its initial spectral signature. These pixels are sent one by one to be

loaded into the FPGA registers. These pixels are sent in batches of equal size so that one is introduced to the FPGA per kernel execution. This mode is started once per image for evaluation.

In the second mode, the spectral signature of a target pixel is compared with the centroids to determine the new centroid to which the pixel will be reassigned. For this purpose, the pixel to be evaluated is sent in batches of the same size as in the first mode. Each batch in this mode is used only once per kernel execution. On each run, the squared difference between the pixel and a centroid is calculated for the spectral bands sent in the batch. The resulting values are put into an adder to get their sum "S" according to (1), where "px(i)" and "ctr(i)" are the spectral bands of the pixel and the centroid, respectively.

$$S = \sum((px(i) - ctr(i))^2) \quad (1)$$

The adder will follow a tree structure to simplify pipelining and reduce runtime. The calculated "S" means the distance between the pixel and the centroid for the spectral bands taken in this execution. This value is accumulated in a register. This register stores the "S" value for the spectral bands compared between the pixel and the centroid over several consecutive kernel executions. This process is repeated within the same executions for all centroids. In this way, after the execution for the last batch of pixels is finished, this register will store the distances between the pixel and each centroid. These distances are passed to a comparator. This comparator, following a tree structure, determines the centroid for which a minimum distance has been obtained and returns it to the host as the new centroid for the pixel. When this process is complete, this execution mode is reset and the kernel switches to centroid update mode.

The third mode corresponds to the centroid update. In this mode, the centroids affected by the last pixel redistribution are updated. The spectral signature of the centroids consists of the average spectral signature of the pixels belonging to the centroids. This average is obtained by dividing the sum of the pixels in each spectral band by the number of pixels within the centroid. Therefore, updating the centroids means that in each spectral band, the pixel is subtracted from the sum of the old centroid and added to one of the new centroids before the averages are recalculated. These processes are parallelized to the same number of spectral bands as the previous two execution modes. Thus, each execution of this mode processes only one batch of pixel data of the same size as in the other modes. These batches are not stored in the FPGA from the previous runs for distance estimation. Therefore, they must be resent to the kernel from the host.

If the pixel evaluated for the centroids update is not the last pixel of the image, the distance evaluation execution mode is set for the next pixel. On the other hand, if such a pixel is the last pixel of the image, the k-means termination criteria are evaluated. These termination criteria return true if less than a specified proportion of pixels from the image have been redistributed in the last k-means iteration, or if a specified number of k-means iterations has been exceeded. If it returns true, the k-means kernel is set to centroid transfer mode. Otherwise, the k-means algorithm is repeated for the entire image, and the distance estimation is restarted for the first pixel.

In centroids transfer mode, the final centroids are transferred back to the host. This transfer is divided into batches of the same size as the previous execution modes. One batch is sent per kernel run. After the centroids transfer is complete, the kernel returns to the centroids initialization mode for possible new images.

IV. Application Development

From the application development point of view, the Vitis IDE, a development platform provided by Xilinx, is used. The host is programmed using C/C++ and standard C/C++ libraries. On the other hand, the kernels are designed according to the HLS (High-Level Synthesis) methodology. This methodology allows the hardware architecture of the kernels to be described at the algorithmic level, deriving the optimized RTL architecture through the introduction of compilation directives and synthesis constraints. These directives are marked as "#pragma HLS" in the kernel source code [10], [11], [12]. Interaction and synchronization between the host and the kernels are programmed using OpenCL via C++ wrappers from the host.

V. Implementation

The Zynq Ultrascale+ MPSoC ZCU102 prototyping board was used for this research.

The entire application is designed to be implemented on the MPSoC. Thus, the host is designed to run on the PS while the accelerator kernels are implemented on the PL.

The platform uses AXI4 (Advanced eXtensible Interface) interfaces for communication between the host and the kernels. In this sense, four AXI4 interfaces are used for data transfers between the host and the filtering kernel, three for the normalization kernel, and two for the k-means kernel. However, all these ports are connected to the same PS-PL port called HP0 via a dedicated interconnect IP. An important aspect of the AXI4 interfaces is that they support a maximum data transfer width of 512 bits. Therefore, if a batch of data to be transferred on a line exceeds this length, the application is configured to transfer the data in as many transfers as necessary in burst mode [13]. Figure 4 presents the basic schematic of the application.

This application can be adapted to other SoCs and MPSoCs if they contain at least a microprocessor, an FPGA, and sufficient AXI4 interfaces. However, the parallel processing of the core must be adapted to the availability of FPGA resources. The application is loaded into the MPSoC via the SD card slot on the prototyping board. It must contain the executable file of the application to be run on the host and the binary file to configure the FPGA. In addition to these files, some additional files are needed to boot the board and to implement a Linux operating system and file system to support the application. The interaction with the system is done through a PC terminal connected to the MPSoC via Ethernet. For this interaction, an IP address for the prototyping board must be set and known in advance.

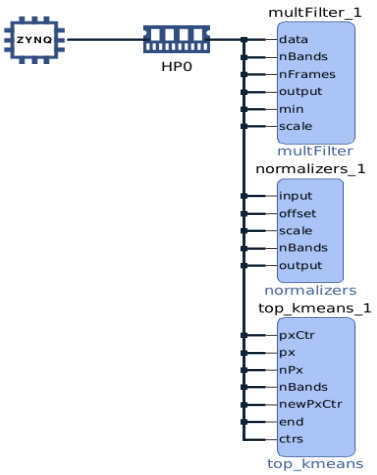

Figure 4. Basic diagram for the platform

VI. Design validation

Following the HLS methodology, the design was validated using both software and hardware emulations. Both emulations are run by the QEMU (Quick Emulator) emulator on a computer running Linux OS. This emulator pretends to be the target APU for the host in both emulations. However, the emulations differ for the FPGA kernels [14].

Software emulation interprets FPGA kernels to run on the APU as host code. This emulation option is compiled and executed first because it is faster. However, software emulation can only verify that the kernels are algorithmically correct. As a result, aspects such as FPGA utilization and timing are not considered for each kernel. Also, this type of emulation does not take into account the parallelism in kernel processing that occurs in the FPGA implementation.

In hardware emulation, kernels are interpreted to be implemented and executed on the target FPGA. This change makes this emulation choice more time-consuming than software emulation. However, it allows further validation of the FPGA kernel by checking the feasibility and performance of its implementation. In addition, running this emulation makes it possible to validate that any problem in the FPGA implementation does not change the correctness of the output returned by the kernels. These features make the hardware emulation compile and run after the software emulation, as a second level to validate the kernels. [14].

VII. Final application

This application is intended to be invoked from the shell command window. The main input data is the hyperspectral image to be evaluated, specified as a binary file. Optionally, the user can specify the dimensions of the hyperspectral image (length, width, and spectrum) and the serialization criterion followed by the image data in the binary file. These features can be specified directly in the application call, or indirectly via a text file. If not specified, default values are used.

The application can adapt to hyperspectral images of different dimensionalities. Thus, the application can be configured to work with images no larger than 2^31-1 bytes. In addition, the dimensions of the evaluated images can change from one run to another without rebuilding the application, if they do not exceed the limits configured at compile time about the maximum number of pixels and spectral bands allowed. The serialization of the input image data can be Bands Interleaved by Pixel (BIP), Bands Interleaved by Line (BIL), or Band Sequential Sequential [1].

This application can adapt to any of these without rebuilding. Another configurable aspect is the data type of each datum from the hyperspectral input image content. On this topic, the application can only work with unsigned integers or fixed-point data. However, the application can be configured during compilation to work with data of 1, 2, 4, or 8 bytes.

VIII. Performance results

This section discusses the performance of the application. This performance is measured in terms of FPGA resource utilization, time, and power. Finally, the clustering results of the kernel are evaluated.

A. *FPGA utilization*

According to the results of the design, the FPGA kernels can operate at a clock speed of up to 150 MHz. The utilization of the FPGA by the kernels is shown in TABLE I. For this evaluation, the kernels are considered to have a parallelism of 16 data for the preprocessing kernels and 8 data for the k-means kernel. The Platform row refers to the FPGA resources required by the entire system. The HP0 row refers to the resources involved used to switch the HP0 port in the PS to each data port in each FPGA kernel. The other row refers to resources used for additional purposes, such as switching between host and kernels and resetting the kernels. This usage is distributed throughout the FPGA as shown in Figure 5, where each kernel and HP0 is identified by the colors shown there.

TABLE I. Utilization of FPGA resources.

IP		**FPGA resources**			
		FF	***LUT***	***DSP***	***BRAM***
filtering		22342	16887	0	16.5
normalization		32304	26294	0	12.5
k-means		29391	30271	25	53
HP0		21986	22907	0	136.5
Others		1947	1148	0	0
Platform		**107970**	**105109**	**25**	**219**

As shown in TABLE I, the amounts of FPGA resources utilized for switching are quite significant. However, they are mainly targeted to switch data frame transfers. Besides, the utilization of every resource in “top_kmeans” is greater or very similar to that obtained in the other kernels. This occurs even though the kernel’s parallelization is half of the other kernels.

In the case of flip-flops, it must be emphasized that “top_kmeans” requires more complex coordination among its iterations than others. This is because it must coordinate the 4 execution modes and the access to the information about

centroids stored within. Moreover, the "top_kmeans" utilizes more LUTs as it involves more and more complex operations.

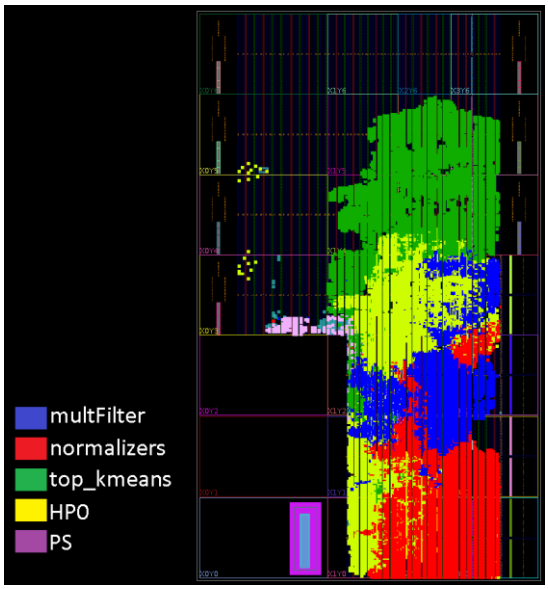

Figure 5. FPGA utilization

Besides, the special complexity of some of these operations leads to the use of DSPs in "top_kmeans" for multiplication, while its usage is inexistent in the other kernels. Also, the utilization of BRAMs in "top_kmeans" is much higher than in the other kernels due to the need to store information about the centroids. It must be remembered that this information implies the entire spectral signature of every centroid and the summation of pixels for every spectral band at each centroid.

B. *Kernel's time performance*

Time performance results are shown in TABLE II for every kernel, being quantified in kernel clock cycles. The term "Initiation Interval" means the minimum elapsed time between two 2 consecutive executions of the kernel thanks to pipelining.

TABLE II. TIME PERFORMANCE BY EVERY KERNEL.

Kernel function	**Time feature (FPGA clock cycles)**	
	Initiation Interval	***Latency***
filtering	2	239
normalization	1	250
k-means	40	151

The most significant difference between the kernels is found in the initiation interval. In that way, while this value is minimal for the filtering and normalization kernels, it is much greater for the k-means kernel. This last situation occurs because of the mutual data dependence between the execution modes for the distance assessment and the centroids update. More specifically, while the distance assessment needs the last centroids update finishes to know the spectral signatures of the centroids, the centroids update must wait for the last distance assessment to know the new centroid for reallocating the pixel.

C. *Application time performance*

Employing the HS dataset, the average runtime on one iteration has been measured for every kernel. These runtimes include the kernels' execution, but also the host's execution required. Also, they consider the synchronization and parallelism between the host and kernels. Besides, the runtime has been measured for the entire application, except for the display of the results. These results were obtained for software emulation (row "sw_emu") and actual execution on the MPSoC (row "hw").

Moreover, the "GPU 1" and "GPU 2" rows are results for the GPU-accelerated similar application from [5], for which 2 implementation options are assessed. This GPU application also includes Spectral Angle Mapper (SAM) and Support Vector Machines (SVM) as part of its running. "GPU 1" refers to the NVIDIA RTX 2080 GPU implementation in [5], while "GPU 2" means the NVIDIA Tesla K40 GPU.

TABLE III. RUNTIME FOR EVERY KERNEL AND THE WHOLE APPLICATION

Execution	**Part of the application**			
	multFilter	***normalizers***	***top_kmeans***	***Total***
sw_emu	217 ms	221 ms	181 ms	48886 s
hw	95.56 µs	97.3 µs	90.4 µs	22.77 s
GPU 1				0.8 s
GPU 2				0.33 s

If results from "hw" are compared to those from "sw_emu", FPGA acceleration is concluded to relevantly enhance the application's throughput. In addition, both "multFilter" and "normalizers" leverage parallelism in terms of throughput with the same effectiveness. This last happening means that making kernels "multFilter" and "normalizers" can have different parallelism capabilities is useless.

D. *Energy performance*

The estimated power needed by the application's running goes around 6.16 W. This consumption is mostly dynamic power (88%), whereas static power is the minor part (12%). The static power is mainly consumed by the FPGA (86%), due to the high number of components and configuration that its utilization requires. The dynamic power is majorly consumed by the host's running (48%) but stays close to FPGA's running (43%). The remaining 9% is employed in clock signals.

By this power data, it is seen that FPGA kernels involve a part of the power consumption in the application which is more meaningless than the part of the computational load they process. In this way, FPGA acceleration is concluded to help in saving energy, both by reducing runtime and power.

E. *Qualitative results*

According to the results, the segmentations made by the application fit well to the spread of the skin cancer lesion. This happening can be seen in Figure 6, where images are also

represented in grayscale to ease the comparison. Besides, results obtained by a CPU function written in Matlab that follows a similar algorithm are also shown.

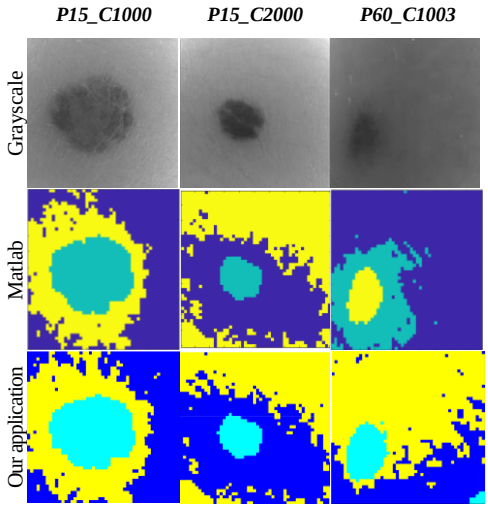

Figure 6. Image segmentation results

In the image "P60_C1003", the clustering from the application differs from Matlab. According to the grayscale image, this difference is because the affection is more blurred than in the other images. Besides, pixels from the lower-right corner of "P60_C1003" are misclassified. It happens for the influence of "dead pixels", consisting of a flaw in the hyperspectral camera.

IX. Conclusions and Future Work

The final application provides good aid to dermatologists in detecting skin cancer. This quality is seen in the segmentation images, which fit quite well with the spread of the lesion seen in the grayscale images. Moreover, using FPGA acceleration helps to shorten runtime and energy consumption if compared to not hardware accelerated choices. Besides, this application has runtime and compile-time adaptation to some aspects of the input hyperspectral image. This guarantees the application's utility under many work circumstances.

Regarding the FPGA kernels, compile-time configuration options have been enabled to ease the application's adaptability. The most characteristic and common option to configure for these kernels is their parallelism, by which they can be adapted to the availability of FPGA resources. Also, these kernels have been designed to have as much pipelining capability as possible due to data dependencies among their algorithm.

Some future works for this project may be: (1) exploring dynamic FPGA reconfiguration to make every FPGA kernel have more available FPGA resources; (2) implementing labeling of every cluster by SAM (Spectral Angle Mapper); (3) changing k-means stage to have a better pipelining.

References

[1] J. Qin, "Hyperspectral imaging instruments," in *Hyperspectral Imaging for Food Quality Analysis and Control*, D. Sun, Ed. Academic Press, 2010, pp. 129–172 [Online]. Available: https://doi.org/10.1016/B978-0-12-374753-2.10005-X

[2] M. Borengasser, W. S. Hungate, and R. Watkins, *Hyperspectral remote sensing : principles and applications*, 1st ed. CRC Press, 2007 [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&db=cat07429a&AN=ulpgc.576063&site=eds-live

[3] H. Fabelo *et al.*, "Spatio-spectral classification of hyperspectral images for brain cancer detection during surgical operations," *PLOS ONE*, vol. 13, no. 3, pp. 1–27, Mar. 2018, doi: 10.1371/journal.pone.0193721. [Online]. Available: https://doi.org/10.1371/journal.pone.0193721

[4] M. Kubat, *An Introduction to Machine Learning*. Springer International Publishing, 2015 [Online]. Available: https://www.springer.com/gp/book/9783319348865. [Accessed: Apr. 26, 2020]

[5] E. Torti *et al.*, "Parallel Classification Pipelines for Skin Cancer Detection Exploiting Hyperspectral Imaging on Hybrid Systems," *Electronics*, vol. 9, no. 9, 2020, doi: 10.3390/electronics9091503. [Online]. Available: https://doi.org/10.3390/electronics9091503

[6] R. León Martín *et al.*, "Non-Invasive Skin Cancer Diagnosis Using Hyperspectral Imaging for In-Situ Clinical Support," *Journal of Clinical Medicine*, vol. 9, no. 6, 2020, doi: 10.3390/jcm9061662. [Online]. Available: https://www.mdpi.com/2077-0383/9/6/1662

[7] Xilinx Inc, "Zynq UltraScale+ MPSoC Data Sheet: Overview (DS891)," 2019 [Online]. Available: www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf. [Accessed: Apr. 14, 2021]

[8] L. Crockett, D. Northcote, C. Ramsay, F. Robinson, and B. Stewart, *Exploring Zynq ® MPSoC With PYNQ and Machine Learning Applications*. Glasgow, Scotland, UK: Strathclyde Academic Media, 2019 [Online]. Available: https://www.zynq-mpsoc-book.com/. [Accessed: Apr. 16, 2021]

[9] Xilinx Inc, "ZCU102 Evaluation Board User Guide," 2019 [Online]. Available: www.xilinx.com/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf. [Accessed: Apr. 22, 2021]

[10] Xilinx Inc, "HLS Pragma," 2021. [Online]. Available: https://docs.xilinx.com/r/2021.2-English/ug1399-vitis-hls/pragma-HLS-interface. [Accessed: Nov. 14, 2022]

[11] Xilinx Inc, "Hardware Function Optimization Methodology," in *Vivado HLS Optimization Methodology Guide*, 2018, p. 12 [Online]. Available: https://www.xilinx.com/htmldocs/xilinx2017_4/sdaccel_doc/asf1516906387725.html

[12] Xilinx Inc, "High-Level Synthesis," in *Vivado Design Suite User Guide*, 2021 [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2020_2/ug902-vivado-high-level-synthesis.pdf#nameddest=xApplyingOptimizationDirectives

[13] Xilinx Inc, "Controlling AXI4 Burst Behavior," *Vitis High-Level Synthesis User Guide (UG1399)*, 2021. [Online]. Available: https://docs.xilinx.com/r/2021.2-English/ug1399-vitis-hls/Controlling-AXI4-Burst-Behavior. [Accessed: Jul. 28, 2022]

[14] Xilinx Inc, "Building and Running the Application," 2022 [Online]. Available: https://docs.xilinx.com/r/2021.2-English/ug1393-vitis-application-acceleration/Building-and-Running-the-Application. [Accessed: Nov. 14, 2022]