# CNN-LSTM implementation methodology on SoC FPGA for Human Action Recognition based on Video

Daniel Suárez
*Microelectronics Engineering Group*
*Universidad de Cantabria*
Santander, Spain
ORCID: 0000-0002-5722-4051

Víctor Fernández
*Microelectronics Engineering Group*
*Universidad de Cantabria*
Santander, Spain
ORCID: 0000-0003-0614-151X

Héctor Posadas
*Microelectronics Engineering Group*
*Universidad de Cantabria*
Santander, Spain
ORCID: 0000-0002-1427-7524

***Abstract*—The growing use of AI-driven video applications like surveillance or healthcare monitoring underscores the need for embedded solutions capable of accurately categorizing human actions in real-time videos.**

**A methodology is proposed for implementing a customized CNN-LSTM architecture on AMD-Xilinx SoC FPGA devices for human action categorization from video data. In this approach, CNN operations are accelerated by the Vitis-AI DPU within the FPGA, offering flexibility to support a range of CNN architectures without requiring individual hardware description language development. This adaptability is crucial given the varying performance of CNN models across datasets. LSTM operations are executed on the SoC processors, overcoming limitations in the support provided by DPU IP cores for such networks, while maintaining flexibility to assess different configurations. Additionally, a pipeline strategy is proposed to enable parallel execution of both CNN and LSTM components, optimizing resource utilization and minimizing idle times.**

**To demonstrate the validity of the proposed implementation methodology, experiments were conducted on the ZCU102 development board, equipped with a Zynq Ultrascale+ MP-SoC, and involved the use of the VGG16 CNN model along with the exploration of different LSTM configurations. The results demonstrate remarkable computational performance, achieving frame rates of up to 44.34 FPS for videos recorded at a resolution of 320x240 pixels, surpassing real-time requirements. Aditionally, the proposed implementation maintains high accuracy levels, exemplified by the single bidirectional LSTM layer achieving a competitive accuracy of 73.33% based on the UCF101 dataset.**

***Index Terms*—SoC FPGA, Zynq Ultrascale+ MPSoC, ZCU102, Vitis-AI DPU, CNN-LSTM, UCF101, HAR, Deep Learning, AMD-Xilinx**

## I. Introduction

The rise of deep learning has revolutionized the capacity to analyze intricate data patterns, especially in critical domains like video surveillance [1], [2] and healthcare monitoring [3]–[5], emphasizing the importance of computer vision. In this context, a pivotal challenge emerges: accurately categorizing human actions in video sequences. This challenge stem from factors such as variability in capture conditions, visual complexity, appearance variability, context and semantics, ambiguity, data volume, and generalization requirements. Simultaneously, embedded systems, often resource-constrained, have become indispensable in applications requiring real-time video data processing on-site [6]–[8]. This intersection has generated a growing demand for efficient solutions enabling embedded systems to proficiently recognize and classify actions in video streams [9], [10]. However, deploying deep learning solutions for human action recognition in videos, while meeting precision and real-time requirements on resource-limited platforms, presents intricate challenges.

Recognizing actions in dynamic video data, composed of sequences of frames containing rich visual information, involves extracting spatial features from each frame and identifying temporal patterns across multiple frames. A popular solution for addressing this complexity involves adopting a combined solution that leverages the strengths of *Convolutional Neural Networks (CNNs)* for spatial feature extraction and *Recurrent Neural Networks (RNNs)*, particularly *Long Short-Term Memory (LSTM) networks*, for capturing temporal dependencies [11], [12].

Simultaneously, focusing on the challenges of achieving real-time performance on resource-constrained devices, *System-on-Chip (SoC)* devices featuring *Field-Programmable Gate Arrays (FPGAs)* emerge as a promising solution. *SoC FPGAs*, blending hardware and software, enable precise hardware customization and adaptability. Their proficiency in handling parallelizable operations, combined with processor managing sequential and control tasks, provides an ideal platform for adaptive video classification solutions [13], [14].

However, custom hardware development, in *FPGA* implementation, introduces challenges in terms of cost, time, and expertise. Addressing these complexities, highly specialized *AMD-Xilinx Deep Learning Processing Units (DPUs)* have been developed for deep learning computation acceleration [15]. Nevertheless, limitations persist, as not all deep learning operations or architectures, including *LSTM* networks, are supported by *DPUs* [16].

This work has been supported by Project PID2020-116417RB-C43, funded by Spanish MCIN/AEI/10.13039/501100011033 and by Project No 101007273 ECSEL DAIS, funded by EU H2020 and by Spanish pci2021-121988.

In this context, the proposed contribution aligns with a *CNN-LSTM* hybrid design, essential for precise action classification in videos. The approach distributes computationally intensive tasks, like *CNNs*, to *DPUs* implemented on the *PL* part of the *SoC FPGA*, capitalizing on the inherent parallelizability of convolution operations. In contrast, *LSTM* networks, intrinsically less parallelizable due to their sequential nature, lack support in *DPU* accelerators. Their recurrent structure and temporal dependencies, along with the limited resources of the *FPGA*, make them more suitable for software execution on *SoC* processors.

This computing distribution, along with an optimized pipeline to minimize idle times, enables the deployment of configurable *CNN-LSTM* designs on low-power embedded platforms. This configuration allows meeting real-time requirements with resolutions high enough to address complex human action recognition tasks, delivering precision results comparable to the state of the art.

The article is structured as follows: *Section II* reviews previous work related to video action classification, addressing both techniques and architectures used in action recognition as well as implementations in embedded systems. *Section III* presents in detail the proposed implementation methodology, including the *CNN-LSTM* design, training procedure, *SoC FPGA* resource distribution and the execution flow strategy. *Section IV* discusses the results obtained through experiments and performance evaluations, while *Section V* presents the conclusions and discusses potential future research directions.

## II. Previous Work

### A. Techniques and Architectures in Video Action Recognition

In the domain of computer vision, the evolution of video action recognition has progressed from classical techniques to deep learning paradigms, bringing significant advancements in each of the three stages that constitute the complete process: feature extraction, temporal representation, and classification.

Initially, in the first stage, classical feature extraction techniques played a crucial role. Methods such as the *Histogram of Oriented Gradients (HOG)* [17] and *Scale-Invariant Feature Transform (SIFT)* [18] were employed to capture static object attributes and contours, while dynamic features related to object motion were addressed through techniques like the *Histogram of Optical Flow (HOF)* [19] and *Motion Boundary Histograms (MBH)* [20].

In the following stage, efforts were focused on integrating these diverse features into a standardized video-level representation. Methodologies such as *Bag of Words (BoW)* [21] were commonly employed to organize visual descriptors into dictionaries, aiming to provide a coherent and structured representation of video content.

The final stage of the classical process involved video content classification, often achieved through machine learning techniques like *Support Vector Machines (SVM)* [22]. These algorithms represent the culmination of the traditional video action recognition pipeline.

However, the field of video action recognition experienced a significant transformation with the introduction of deep learning algorithms. The *CNNs* [23]–[26] revolutionized feature extraction by autonomously learning hierarchical representations from static images. Leveraging the effectiveness of *CNNs*, two key strategies expand their applicability to sequences of images commonly encountered in videos, addressing both spatial and temporal relationships.

One strategy involves *2D CNNs* [27], [28], where neural network topologies are tailored to handle reduced frame sequences, enabling the effective extraction of spatial features and motion patterns. The other strategy utilizes *3D CNNs* [29], capturing spatial and temporal information directly from video sequences. However, they often require substantial labeled training data and can be computationally intensive, limiting real-time applicability.

Other approaches emphasized freeing *CNNs* from extracting temporal patterns, proposing two distinct strategies. The first one uses pooling techniques with *2D CNNs* [30], [31] to extract temporal patterns without the computational overhead of *3D CNNs*. The second one employs *LSTM* networks [32], [33], adept at capturing temporal dependencies within video data, accommodating varying temporal distances and requiring fewer training samples compared to *3D CNNs*.

Authors in [11], [12] employ an *LSTM*-based approach, utilizing the *CNN-LSTM* design as the foundation for their experiments. They assess its efficacy in action recognition and other proposed tasks using data encompassing diverse types of information. Additionally, both studies demonstrate significant precision when applying this network to raw frames, capitalizing on the spatial information conveyed by the images. This underscores that leveraging *LSTM* networks as detectors of motion patterns yields superior results compared to utilizing *CNN* networks for this purpose.

### B. Embedded System Implementations for Video Action Recognition

The deployment of *CNN-LSTM* neural networks on resource-constrained embedded systems for real-time video-based applications has encountered challenges related to the vast amount of data that must be processed within a short period of time. Within the literature addressing this issue, three distinct approaches emerge.

Firstly, there are those focusing on software implementations on microprocessor embedded systems [34]. For instance, a solution for video surveillance on a *Raspberry Pi* achieved a commendable performance of *10-13 frames per second (FPS)*. However, this impressive performance comes at the cost of reduced input frame resolution. Additionally, this implementation faces limitations in the selection of suitable *CNNs*. While *MobileNetV2* is proposed with *3.4 million* parameters and around *0.6 Giga Floating Point Operations Per Second (GFLOPs)*, it falls short in comparison to networks with significantly higher requirements.

In [35], the performance of a *CNN-LSTM* combined design is compared across various platforms, including both typical workstation devices and embedded systems like the *Pynq Z1 FPGA SoC*. Although using only its *ARM A9* cores for testing, the proposed architecture with *InceptionV3*, having *23 million* parameters and requiring around *6 GFLOPs*, achieves a performance of *1-2 FPS*, which is insufficient for real-time applications.

The second approach focuses on completely hardware-based implementations. [36] introduces a reconfigurable accelerator design based on an *FPGA*, capable of operating in both *CNN* and *RNN* modes, but it is specifically designed and tested for one-dimensional time series. Conversely, [37] suggests an *SoC* design on *65 nm CMOS* technology to serve as a general-purpose processor within deep learning.

In a slightly different research, [38] explores various architectures for music genre classification, including *CNNs*, *LSTMs*, and the *CNN-LSTM*. These networks undergo experimentation and comparison across different platforms, encompassing a workstation equipped with an *Intel Core i7 8th Gen CPU* and the *Zynq Ultrascale+ MPSoC* found in the *ZCU104* board. However, only performance results for the *CNN* network are presented on the embedded platform. This limitation arises from challenges encountered in developing an efficient and optimized *CNN-LSTM* network for an *MPSoC FPGA*, despite the conclusion that the *CNN-LSTM* design exhibits the highest capability for the given task.

In the third approach, [39] proposes an *SoC FPGA* for a complete implementation of a *CNN-RNN* model targeting *Remaining Useful Life Prediction (RUL)* for machine health monitoring. The hardware accelerator is dedicated to the *CNN*, featuring three convolutional layers with *3x3* convolution filters, while the *RNN* computations are handled by *ARM A9* microprocessors. This implementation is evaluated using small data matrices derived from information collected by *21* sensors over a specific time span. It achieves a performance of up to *5* predictions per second from sequences consisting of *5* time steps.

The implementation methodology proposed in this research addresses distinct applications with varying requirements, meeting real-time high-definition imagery while maintaining the precision of deep learning algorithms essential for various computer vision tasks. Furthermore, the proposed design offers flexibility, enabling adaptation to different neural network architectures, with computational power tailored to the complexity of the tasks, facilitated by the utilization of *DPU* units.

## III. Proposed Implementation Methodology

This section provides a comprehensive overview of the methodology of implementation, detailing the hybrid *CNN-LSTM* design, the training procedure, the utilization of *SoC FPGA* resources, and the pipeline execution strategy. The visual representation is encapsulated in *Fig. 1*, illustrating the distribution of computational load across the *SoC FPGA*.

### A. Hybrid CNN-LSTM Design

The proposed solution features a hybrid design, seamlessly integrating *CNNs* and *LSTM* networks. This combination is designed to extract spatial features through *CNNs* and capture temporal dependencies across frames using *LSTMs*.

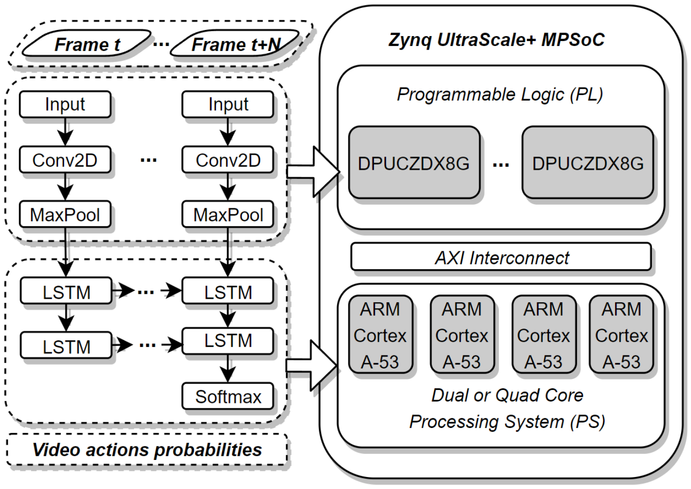

Fig. 1. *CNN-LSTM implemented on Zynq Ultrascale+ MPSoC.*

The solution offers a flexible implementation approach for experimenting with different neural network architectures for both *CNN* and *LSTM* components, facilitating a selection based on specific application needs.

Within the *CNN* domain, the standardized workflow allows for deployment across various network families, such as *VGG*, *ResNet*, or *Inception*. As for the *LSTM* component, the employed methodology simplifies experimentation with various configurations, enabling adjustments to parameters such as layer count, units per layer, and learning direction.

To predict the category of a given action in a video, the *CNN-LSTM* network analyzes a sequence of frames representative of the action. Notably, the *CNN* processes a single frame at a time, while the *LSTM* requires a fixed-length sequence as input. Therefore, videos, which typically consist of varying-length sequence of frames, need to be standardized into fixed-length sequences. This standardization process is achieved through the implementation of a sampling strategy designed to retain maximal contextual information of the action, with adjustments made to the sampling rate based on the individual duration of each video. Consequently, this method yields fixed-length sequences wherein frames are uniformly spaced apart by a consistent temporal interval, ensuring compatibility across varying video lengths.

Balancing the need for real-time processing and effective prediction, a mixed strategy is employed. Although previously utilized, its application varies based on the specific problem, leading to nuanced adaptations.The approach involves dividing the fixed-length sequence corresponding to a single action into multiple shorter subsequences. Each subsequence undergoes prediction independently, and a final prediction for the entire action is derived through a fusion strategy. The implementation methodology offers flexibility concerning the specific values selected for the fixed length of sequences representing an action post-sampling, as well as the number and length of subsequences utilized for prediction.

### *B. Training procedure*

The training procedure, adaptable across various architectures and datasets, consists of two main stages: first, fine-tuning the *CNN* and, then extracting features for the *LSTM*.

To initiate the training process, a pre-trained *CNN* model sourced from the *ImageNet* dataset serves as the foundation. Leveraging the features learned from a wide array of images, this model embodies an understanding of visual patterns and structures. However, to tailor its capabilities to the specific domain of human action recognition in videos, a standard fine-tuning approach is applied. This process involves adjusting the parameters of the *CNN* model through iterative training on the target dataset, enabling it to adapt and specialize in discerning spatial features relevant to human actions.

Once the *CNN* model is fine-tuned, it serves as a feature extractor for the subsequent *LSTM* network. By processing fixed-length frame sequences sampled from each video in the dataset, the *CNN* extracts high-level spatial features that encapsulate the visual characteristics indicative of different human actions. These spatial features are then compiled into a new dataset, prepared to serve as the training input for the *LSTM* network.

Before proceeding with the training of both components, standard customizations are introduced to enhance their effectiveness within the context of the target video dataset. Specifically,it is common practice to append a *fully connected* layer tailored to the categories present in the video dataset is appended to each model. This additional layer acts as a bridge between the extracted features and the output predictions. Furthermore, a *softmax* layer is typically incorporated to interpret the output as probabilities, a standard practice in classification tasks that ensures coherent predictions.

Having established the architectural foundations, standard training techniques are applied to both the *CNN* and *LSTM* networks. These techniques encompass the utilization of the *cross-entropy* loss function to quantify the disparity between predicted and actual labels, the *Adagrad* optimizer to dynamically adjust the learning rate for efficient convergence, and the integration of a *dropout* layer to mitigate overfitting by randomly deactivating neurons during training.

### *C. SoC FPGA Resource Distribution*

Efficiently addressing computational demands involves distributing tasks across *SoC FPGA* resources. *Fig. 1* visually illustrates the distribution of the computational burden between the *FPGA* and *SoC* processors. The neural network architecture is depicted on the left side, while the *SoC FPGA* resources are shown on the right side. Arrows delineate the allocation of each neural network component within the *SoC FPGA*: *CNN* operations are accelerated by the *AMD-Xilinx Vitis-AI DPU* in the *FPGA*, while *LSTM* operations are executed on the processors of the *SoC*.

The *AMD-Xilinx Vitis-AI DPU*, a commercial *IP* core, accelerates *CNN* convolutional operations, facilitating parallel spatial feature extraction by utilizing multiple *DPU IP* cores, contingent upon the resource availability of the *FPGA*. *Vitis AI* offers a comprehensive development environment, streamlining workflow from neural network creation to compilation into *DPU IP* core-compatible binary instructions. The process includes graph quantization, intermediate representation, and compilation to the *DPU* instruction set, resulting in binary instructions alongside neural network weights. The initial software implementation employs *32-bit floating-point* data type to ensure maximal precision during weight updates in training. Subsequently, during the quantization process, the data type transitions to *8-bit integer*, significantly reducing the size of the weights of the network without compromising prediction accuracy.

Additionally, *AMD-Xilinx* provides *PetaLinux*, a reduced *Linux OS* integrating drivers and libraries for application development. *PetaLinux* serves as the enabler for *LSTM* network software implementation through the *TensorFlow* framework on the *SoC* processing system. *TensorFlow*, employed for all proposed *LSTM* implementations, utilizes *intra-operation* parallelization to optimize critical operations within *LSTM* cells across all processing cores of the *SoC*. The capabilities of this software approach are further enhanced by the *VART (Vitis AI Runtime)* library atop *XRT (Xilinx Runtime)*, extending to managing *CNN* operation execution, orchestrating signals for *DPU IP* core initiation and termination, and ensuring seamless collaboration between hardware accelerators, off-chip *DDR* memory and *ARM* processors.

### *D. Execution flow*

The execution flow of the implemented pipeline strategy is designed to leverage both hardware and software capabilities, ensuring efficient execution for both *CNN* and *LSTM* components. This approach consists of two interconnected parallel software processes described here:

*a) CNN Process:* A thread is initiated for each *DPU IP* core present in the *SoC FPGA*, responsible for managing the execution of *CNN* operations. Each *DPU* runs a set of binary instructions, representing neural network operations, loaded into the external off-chip memory along with network weights and the corresponding input frame. Upon thread initiation, a *'start'* signal activates the corresponding *DPU*. The thread then awaits the *'end'* signal from the *DPU*, indicating the completion of the *CNN* block execution. After receiving the initiation signal, the *DPU* utilizes *Direct Memory Access (DMA)* to load instructions, weights, and input data from the external *DDR* off-chip memory, storing results back into the same external *DDR* memory. A *First In, First Out (FIFO)* queue associated with each thread has been implemented, storing spatial features extracted from the frame processed by the respective *DPU*. This parallel approach optimizes the efficiency of the *CNN* process by allowing simultaneous extraction of spatial features from multiple frames.

*b) LSTM Process:* On the other side, the *LSTM* process waits until a sufficient number of spatial features are enqueued in *FIFO* structures associated with each *CNN* thread. Subsequently, the input subsequence for the *LSTM* is generated, and inferences are conducted for each one. The probabilities obtained from each subsequence are accumulated, leading to a final prediction that determines the composite action derived from all the subsequences.

The temporal organization and resource distribution of the designed software processes are graphically depicted in *Fig. 2*, providing a overview of the implemented pipeline. This temporal diagram presents multiple timelines, delineated by solid and dashed lines, each symbolizing a specific task inherent to consecutive subsequence inferences. Solid lines delineate primary tasks, while dashed lines depict subtasks derived from them. Specifically, the dashed lines in the diagram refer to threads that decompose the *CNN* process to handle interactions with each *DPU*. The resources assigned to each task are represented by shaded rectangles, categorized with labels such as *"HW"* (hardware) or *"SW"* (software), indicating whether the task takes place within the *FPGA* or the *SoC* processors, respectively.

The duration termed as *"spatial features time"* represents the time needed for the *CNN* process to complete, which includes extracting spatial features from frames forming a subsequence. Subsequently, a period is necessary for inter-process communication, achieved through read and write operations of *FIFO* queues, bridging the gap between the *CNN* process's completion and the initiation of *LSTM* process execution. This combined duration, along with the spatial features time, is defined as *"subsequence generation time"*. Lastly, the time taken to conduct an inference in the *LSTM* network utilizing a subsequence of spatial features, when added to the previously mentioned durations, is referred to as *"subsequence inference time"*.

The pipeline strategy is reflected in the simultaneous execution of the *LSTM* process for the inference of the first spatial features subsequence and the commencement of spatial features generation for the second frame subsequence by the *CNN* process, thus ensuring parallel execution of the *CNN* process and its respective threads alongside the *LSTM* process, maintaining the operability of both *HW* and *SW* parts of the platform.

The latency of each process varies depending on the specific architecture chosen for each component. This variability may lead to scenarios where the latency of the *LSTM* process exceeds that of the *CNN* process, or vice versa. However, this pipeline approach ensure that the hardware *(CNN)* and software *(LSTM)* components work simultaneously, minimizing wait times, and optimizing the utilization of *SoC FPGA* resources. Smooth and efficient communication between these processes, facilitated by *FIFO* queues and synchronization mechanisms, enables real-time and effective video classification on embedded systems.

## IV. Results

Several experiments were conducted on the *ZCU102* development board, equipped with the *Zynq Ultrascale+ MPSoC*, which consists of a *XCZU9EG FPGA* and four *ARM A-53 cores* interconnected via the *AXI* bus. The configuration utilized the *DPUCZDX8G IP* core, synthesized to maximize computational capacity at *4096 MACs (Multiply-Accumulate Operations)* per clock cycle. To fully exploit *FPGA* resources, three instances of this *DPU* were deployed, enabling simultaneous processing of up to three frames.

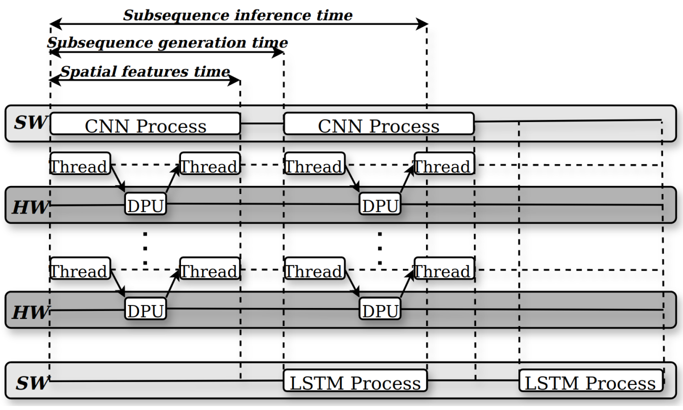

Fig. 2. *Pipeline execution flow diagram.*

The *CNN* is structured based on the *VGG16* architecture, which boasts *30.96 GFLOP*. Chosen for its reputation as a benchmark in computer vision deep learning, *VGG16* consists of thirteen convolutional blocks, each containing a *convolutional layer* with *3x3* filter size, *max-pooling layer*, and *ReLU* activation, followed by three *fully connected layers*. In the *LSTM* component, various configurations were explored, adjusting the number of layers (one or two), units per layer (*128*, *256*, or *512*), and learning direction (unidirectional or bidirectional).

For performance evaluation, we utilized the widely recognized *UCF101* dataset [40], which comprises *13,320* videos categorized into *101* actions. This dataset offers extensive diversity in actions and features significant variations in camera motion, object appearance and pose, object scale, viewpoint, cluttered background, and illumination conditions. Additionally, the actions can be grouped into five types based on the nature of the depicted interactions: *Human-Object Interaction*, *Body-Motion Only*, *Human-Human Interaction*, *Playing Musical Instruments*, and *Sports*.

Initially, the videos were at *320x240* pixels resolution and *25 FPS*. Then, they were sampled into fixed-length sequences of fifteen frames, further divided into five subsequences of three frames each to optimize the utilization of the three DPUs deployed in the FPGA. This allowed for the parallel processing of each frame within a subsequence by an independent DPU. The inference process involved independently processing subsequences of three frames through the *CNN-LSTM* design. Probabilities corresponding to the *101* categories in the *UCF101* dataset were obtained from each *3-frame* subsequence and averaged to generate the final prediction for the entire *15-frame* sequence, providing inferred likelihoods for each action category in the dataset.

The precision evaluation assessed results for each *LSTM* configuration are presented in *Table I*, focusing on the first *train/test* split of the dataset, which comprised *3.7K* videos for testing. The nomenclature used to refer to the different *LSTM* configurations is as follows: based on the number of layers, *"single"* denotes a single layer, and *"stacked"* denotes two layers; based on the learning direction, *"LSTM"* is used for a unidirectional network and *"Bi-LSTM"* for a bidirectional network; finally, the number of units per cell is indicated by *128, 256, or 512*.

The highest precision of *73.65%* is achieved with the *Stacked_Bi_LSTM_512* architecture, which closely aligns with values reported in previous works for similar *CNN-LSTM* architectures. For instance, [11] achieved a *71.12%* accuracy in the first split of the *UCF101* dataset using a *CNN* architecture very similar to *AlexNet* and a unidirectional *LSTM* configuration with *256 units* in its cells. It employs an averaging strategy to obtain complete video categorization but performs inference on subsequences of *16* frames without subsampling each video. Additionally, [12] achieved a *77.36%* accuracy for the first test split of the *UCF101* dataset using the *Inception-ResNetV2* architecture. Similarly, it employs various configurations of recurrent networks, including the bidirectional *LSTM* with *512* units per cell, achieving the previously mentioned accuracy. However, it adopts a different evaluation strategy by classifying sequences of *180* frames, which encompass the complete context of the action in each video, thanks to prior adaptative subsampling.

TABLE I
*VGG16 + Several LSTM Configurations*

| CNN | Direction | Layers | Units | Acc |
|---|---|---|---|---|
| VGG16 | Unidirectional | single | 128 | 71.13 |
| | | | 256 | 72.83 |
| | | | 512 | 73.43 |
| | | stacked | 128 | 69.39 |
| | | | 256 | 71.03 |
| | | | 512 | 72.19 |
| | Bidirectional | single | 128 | 73.21 |
| | | | 256 | 73.33 |
| | | | 512 | 73.54 |
| | | stacked | 128 | 72.09 |
| | | | 256 | 71.9 |
| | | | 512 | 73.65 |

To illustrate the performance enhancement achieved with the pipeline implementation, computational performance results were compared in *Fig. 3* against a sequential implementation, where the *LSTM* software component waited for the *CNN* hardware component to finish, and vice versa. The latency for processing a single frame and generating a spatial feature was approximately *60 milliseconds*, consistent across all three instances of simultaneously implemented *DPUs*. However, the latency for processing a sequence of three spatial features varied depending on the *LSTM* configuration being tested, ranging from *20.3 milliseconds* to *89.73 milliseconds* for the *Single_LSTM_128* and *Stacked_Bi_LSTM_512* architectures, respectively. The maximum processing speed was achieved with the *Single_Bi_LSTM_256* architecture, reaching a performance of *44.34 frames per second (FPS)*. The largest absolute improvement provided by the pipeline implementation over the sequential approach is *17.13 FPS* for the *Stacked_Bi_LSTM_256* architecture, while the most significant improvement in percentage terms occurs with the *Single_LSTM_512* architecture, showing a *69.40%* enhancement in *FPS* performance compared to the sequential implementation. The *FPS* results indicated a maximum limit of around *44 FPS*, primarily dictated by the minimum latency of the VGG16 network. The improvement provided by the pipeline implementation diminished significantly only for those *LSTM* configurations whose latency exceeds the *60 milliseconds* of the *VGG16* network, moving away from the maximum limit of *44 FPS*.

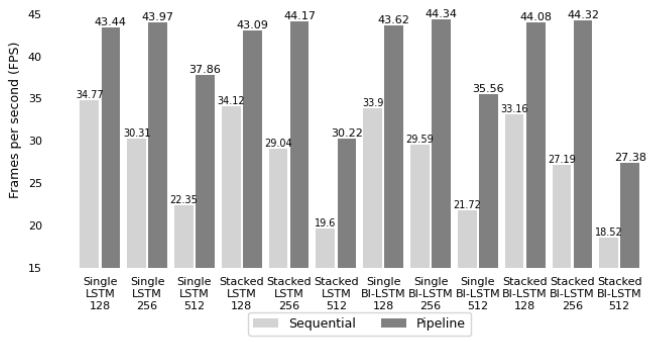

Fig. 3. *Sequential vs Pipelined FPS performance for several LSTM architectures.*

A visual representation in *Fig. 4* illustrates the trade-off between computational performance and accuracy in successful classification. All *LSTM* architectures are depicted by two crosses on the graph, each representing one of the two implementations: sequential and pipeline, and indicating their respective accuracy and *FPS* performance. Additionally, an arrow connects both points for the same architecture, symbolizing the improvement brought by the pipeline implementation. Notably, architectures with higher accuracy may not have the highest *FPS*, and vice versa. However, some architectures excel in both aspects, such as the *Single_Bi_LSTM_256*, achieving *73.33%* accuracy, merely *0.32* decimals below the best accuracy, and delivering a peak performance of *44.34 FPS*, surpassing the highest accuracy architecture by *16.96 FPS*.

The presented methodology allows for alternative algorithm selection and utilization of DPUs based on the requirements and available hardware resources on other platforms.

## V. Conclusions and future work

This study introduces an efficient methodology of implementation of a *CNN-LSTM* network on an *SoC FPGA* tailored for video action recognition. Leveraging the *AMD-Xilinx Vitis AI DPU IP* core as a hardware accelerator for convolutional operations, the *FPGA* serves as a spatial feature extractor

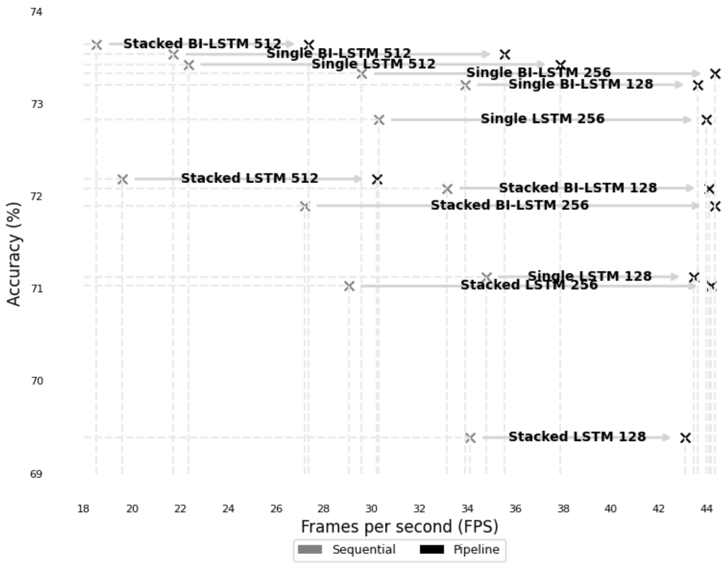

Fig. 4. *Accuracy vs FPS for several LSTM architectures.*

from raw frames. The significant advantage of this hardware accelerator lies in its flexibility, allowing users to select *CNN* architectures without the need for individual hardware description language development, such as *VHDL* or *Verilog*. This adaptability is essential as the efficacy of a particular *CNN* architecture on one dataset does not guarantee its performance across all datasets. Concurrently, the utilization of the processors of *SoC* for *LSTM* network deployment addresses the *DPU IP* cores limited support for alternative network types, while preserving flexibility in assessing various configurations for the dataset at hand.

The achieved results demonstrate competitive performance, aligning with typical benchmarks for this architecture as evidenced by state-of-the-art literature. Discrepancies in precision may stem from the chosen *CNN* architecture for feature extraction, input sequence length during *LSTM* inference, prediction strategy, or dataset preprocessing methods like data augmentation and video subsampling. However, this study does not aim to surpass current state-of-the-art in human action recognition in videos but rather to present a competitive implementation capable of real-time operation on low-power embedded systems with constrained resources. Notably, the computational performance, indicated by *FPS*, reaches up to *44 FPS*, comfortably exceeding real-time requirements typically ranging between *15* and *30 FPS*. These findings motivate further exploration to enhance precision, albeit potentially at the cost of *FPS* performance.

In the proposed *CNN-LSTM* design, the *LSTM* network identifies motion patterns based on spatial features extracted by the *CNN* from raw frames. However, enhancing predictions by providing more detailed information about specific elements in the frame is crucial.

Various approaches, including optical flow techniques and networks like *YOLO* or *PoseDetection*, have been suggested to extract motion information and identify relevant objects or key points, improving the *LSTM* network's ability to recognize motion patterns.

Nevertheless, the challenge lies in the increased computational resources required for detailed object movement extraction while meeting real-time constraints. Future research may focus on optimizing *CNN* and *LSTM* architectures to reduce computational load and latency, enabling the integration of supplementary information without sacrificing performance.

Exploring sequences of different lengths and sampling strategies could provide insights into the impact of context and action duration on prediction accuracy. By varying sequence lengths and sampling methods, researchers can assess how temporal context influences action recognition performance, informing the design of more effective video classification systems.

## Acknowledgment

The work has been partially supported by DAIS project, funded from Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement No 101007273. The KDT JU receives support from the European Union's Horizon 2020 research and innovation program and Sweden, Spain, Portugal, Belgium, Germany, Slovenia, Czech Republic, Netherlands, Denmark, Norway, Turkey.

## References

[1] A. S. M and N. Thillaiarasu, "A Survey on Different Computer Vision Based Human Activity Recognition for Surveillance Applications," 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2022, pp. 1372-1376, doi: 10.1109/ICCMC53470.2022.9753931.

[2] R. Khurana and A. K. Singh Kushwaha, "A Deep Survey on Human Activity Recognition in Video Surveillance," 2018 International Conference on Research in Intelligent and Computing in Engineering (RICE), San Salvador, El Salvador, 2018, pp. 1-5, doi:10.1109/RICE.2018.8509080.

[3] M. H. Siddiqi et al., "A Unified Approach for Patient Activity Recognition in Healthcare Using Depth Camera," in IEEE Access, vol. 9, pp. 92300-92317, 2021, doi: 10.1109/ACCESS.2021.3092403.

[4] Z. A. Khan and W. Sohn, "Abnormal human activity recognition system based on R-transform and kernel discriminant technique for elderly home care," in IEEE Transactions on Consumer Electronics, vol. 57, no. 4, pp. 1843-1850, November 2011, doi: 10.1109/TCE.2011.6131162.

[5] M. Hossain, S. B. Siraj, M. M. Haque, G. Song, B. Ryu and M. T. Bin Iqbal, "Physiotherapy Exercise Classification by Exploring Knowledge Distillation with Attention Modules," 2023 International Conference on Next-Generation Computing, IoT and Machine Learning (NCIM), Gazipur, Bangladesh, 2023, pp. 1-11, doi: 10.1109/NCIM59001.2023.10212747.

[6] S. Fan, M. Li and C. Han, "Intelligent Video Monitoring for Real-Time Detection and Recognition of Elderly Falls on the Embedded Platform," 2023 IEEE International Conference on Image Processing and Computer Applications (ICIPCA), Changchun, China, 2023, pp. 630-635, doi: 10.1109/ICIPCA59209.2023.10257766.

[7] H. Liu, J. Gao, B. Shi and W. Zhao, "Design of Campus Parking Guidance Experimental Platform Based on Raspberry Pi," 2023 IEEE 6th International Conference on Electronic Information and Communication Technology (ICEICT), Qingdao, China, 2023, pp. 386-390, doi: 10.1109/ICEICT57916.2023.10245461.

[8] R. Rimal, M. Herceg, M. Vranješ and R. Grbić, "Driver Monitoring System using an Embedded Computer Platform," 2023 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2023, pp. 1-7, doi: 10.23919/SoftCOM58365.2023.10271657.

[9] S. Pasricha, ”Lightning Talk: Efficient Embedded Machine Learning Deployment on Edge and IoT Devices,” 2023 60th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2023, pp. 1-2, doi: 10.1109/DAC56929.2023.10247845.

[10] K. P. Seng and L. -M. Ang, ”Embedded Intelligence: State-of-the-Art and Research Challenges,” in IEEE Access, vol. 10, pp. 59236-59258, 2022, doi: 10.1109/ACCESS.2022.3175574.

[11] J. Donahue et al., ”Long-term recurrent convolutional networks for visual recognition and description,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 2625-2634, doi: 10.1109/CVPR.2015.7298878.

[12] D. Tkachenko, ”Human Action Recognition Using Fusion of Modern Deep Convolutional and Recurrent Neural Networks,” 2018 IEEE First International Conference on System Analysis & Intelligent Computing (SAIC), Kyiv, UKraine, 2018, pp. 1-6, doi: 10.1109/SAIC.2018.8516860.

[13] A. Safaei, Q. M. J. Wu, Y. Yang and T. Akılan, ”System-on-a-chip (SoC)-based hardware acceleration for extreme learning machine,” 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Batumi, Georgia, 2017, pp. 470-473, doi: 10.1109/ICECS.2017.8292050.

[14] G. Sciangula, F. Restuccia, A. Biondi and G. Buttazzo, ”Hardware Acceleration of Deep Neural Networks for Autonomous Driving on FPGA-based SoC,” 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 2022, pp. 406-414, doi: 10.1109/DSD57027.2022.00061.

[15] ”DPU for convolutional neural network v3.0 DPU IP product guide”, [online] Available: https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_0/pg338-dpu.pdf.

[16] A. Kalantar, Z. Zimmerman and P. Brisk, ”FA-LAMP: FPGA-Accelerated Learned Approximate Matrix Profile for Time Series Similarity Prediction,” 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Orlando, FL, USA, 2021, pp. 40-49, doi: 10.1109/FCCM51124.2021.00013.

[17] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In CVPR, volume 1, 2005.

[18] D. G. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 2004.

[19] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In Proc. ECCV, pages 428-441, 2006.

[20] I. Laptev, M. Marszaek, C. Schmid and B. Rozenfeld, ”Learning realistic human actions from movies”, Proc. CVPR, pp. 1-8, 2008.

[21] X. Sun, M. Chen, and A. Hauptmann. Action recognition via local descriptors and holistic features. In CVPR, 2009.

[22] J. Liu, J. Yang, Y. Zhang and X. He, ”Action Recognition by Multiple Features and Hyper-Sphere Multi-class SVM,” 2010 20th International Conference on Pattern Recognition, Istanbul, Turkey, 2010, pp. 3744-3747, doi: 10.1109/ICPR.2010.912.

[23] Szegedy, C.; Liu, W.; Jia, Y. Q.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1–9, 2015.

[24] W. Shang, K. Sohn, D. Almeida, and H. Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In ICML, 2016.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 2818–2826, http:// ieeexplore.ieee.org/document/7780677/

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015

[27] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, ”Large-Scale Video Classification with Convolutional Neural Networks,” 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 1725-1732, doi: 10.1109/CVPR.2014.223.

[28] K. Simonyan and A. Zisserman, ”Two-stream convolutional networks for action recognition in videos”, Proc. NIPS, pp. 568-576, 2014.

[29] S. Ji, W. Xu, M. Yang and K. Yu, ”3D convolutional neural networks for human action recognition”, IEEE Trans. PAMI, vol. 35, no. 1, pp. 221-231, Jan. 2013

[30] Joe Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga and G. Toderici, ”Beyond short snippets: Deep networks for video classification,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 4694-4702, doi: 10.1109/CVPR.2015.7299101.

[31] S. Cho and H. Foroosh. Spatio-temporal fusion networks for action recognition. In Asian Conference on Computer Vision, pages 347–364. Springer, 2018.

[32] S. Hochreiter and J. Schmidhuber, ”Long Short-Term Memory,” in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[33] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, ”LSTM: A Search Space Odyssey,” in IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 10, pp. 2222-2232, Oct. 2017, doi: 10.1109/TNNLS.2016.2582924.

[34] W. Iqrar, M. Z. Abidien, W. Hameed and A. Shahzad, ”CNN-LSTM Based Smart Real-time Video Surveillance System,” 2022 14th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 2022, pp. 1-5, doi: 10.1109/MACS56771.2022.10023018.

[35] A. B. Mahjoub and M. Atri, ”Implementation of convolutional-LSTM network based on CPU, GPU and pynq-zl board,” 2019 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS), Gammarth, Tunisia, 2019, pp. 1-6, doi: 10.1109/DTSS.2019.8915287.

[36] Y. Yang et al., ”Implementation of Reconfigurable CNN-LSTM Accelerator Based on FPGA,” 2021 IEEE 21st International Conference on Communication Technology (ICCT), Tianjin, China, 2021, pp. 1026-1030, doi: 10.1109/ICCT52962.2021.9657920.

[37] D. Shin, J. Lee, J. Lee and H. -J. Yoo, ”14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks,” 2017 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2017, pp. 240-241, doi: 10.1109/ISSCC.2017.7870350.

[38] Faizan M, Intzes I, Cretu I, Meng H. Implementation of Deep Learning Models on an SoC-FPGA Device for Real-Time Music Genre Classification. Technologies. 2023; 11(4):91.

[39] X. Feng, J. Yue, Q. Guo, H. Yang and Y. Liu, ”Accelerating CNN-RNN Based Machine Health Monitoring on FPGA,” 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hsinchu, Taiwan, 2019, pp. 184-188, doi: 10.1109/AICAS.2019.8771482.

[40] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. CoRR, 2012.