Video Action Recognition in SoC FPGAs driven by Neural Architecture Search

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

Pedro Hernández-Fernández
Institute Applied Microelectronics
Univ. de Las Palmas de Gran Canaria
Las Palmas de Gran Canaria, Spain
ORCID: 0000-0003-3848-2116

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

Gustavo M. Callico
Institute Applied Microelectronics
Univ. de Las Palmas de Gran Canaria
Las Palmas de Gran Canaria, Spain
ORCID: 0000-0002-3784-5504

***Abstract*—This work presents a hardware-aware Neural Architecture Search (NAS) framework for video-based human action recognition, targeting real-time deployment on FPGA-based System-on-Chip (SoC) platforms. The proposed method explores a constrained search space of Convolutional Neural Network (CNN)–Recurrent Neural Network (RNN) architectures aligned with a hardware-software pipeline where CNNs are mapped to FPGA Deep Learning Processing Units (DPUs) and RNNs to embedded ARM cores. A reinforcement learning (RL)-based controller, guided by a position-based discounted reward strategy, progressively learns to generate architectures that emphasize high-impact design decisions. Experiments on the UCF101 dataset demonstrate that the proposed architectures achieve 81.07% accuracy, among the highest reported for CNN-RNN models relying exclusively on spatial information. The results validate the effectiveness of the proposed framework in driving hardware-compatible and performance-optimized architecture exploration.**

***Index Terms*—Neural Architecture Search, SoC FPGA, CNN-RNN architectures, Video Action Recognition, Reinforcement Learning, Embedded AI, Hardware-aware NAS**

I. Introduction

The integration of artificial intelligence into embedded systems is increasingly relevant for applications requiring real-time processing, secure operation, or limited communication bandwidth [1]. Deploying deep learning algorithms on resource-constrained platforms introduces challenges in meeting timing constraints [2], particularly in computationally intensive domains such as computer vision.

Action recognition is a core task in this domain [3], [4], demanding models that capture spatial and temporal dependencies, such as convolutional neural network–recurrent neural network (*CNN-RNN*) architectures [5], [6]. To satisfy latency and throughput requirements, hardware acceleration is typically implemented using application-specific integrated circuits (*ASICs*), embedded graphics processing units (*GPUs*), or system-on-chip field-programmable gate arrays (*SoC FPGAs*), each with trade-offs in power, cost, and performance [7].

Among these, integrating *CNN-RNN* models with *SoC FPGAs* has shown promise [8]. Their flexibility supports customized software–hardware pipelines for inference within power and timing constraints. However, the vast architectural design space complicates deployment: each configuration impacts accuracy and hardware performance, and manual exploration is inefficient and not scalable.

This work introduces a hardware-aware neural architecture search (*NAS*) framework for video action recognition on *SoC FPGAs*. Unlike prior *NAS* approaches that overlook hardware limitations, our method aligns the search space with an implementation methodology [9], where convolutional neural networks (*CNNs*) map to FPGA-based deep learning processing units (*DPUs*) and recurrent neural networks (*RNNs*) run on *ARM* cores, enabling concurrent execution without custom hardware design.

Exploration proceeds through a three-stage process controlled by a generative long short-term memory (*LSTM*) model, which iteratively samples, evaluates, and updates candidate architectures based on task-specific metrics.

In summary, this work proposes a hardware-aware *NAS* framework that combines a *CNN-RNN* search space aligned with a real-time *SoC FPGA* execution pipeline and a reinforcement learning controller with causal reward design, enabling the systematic discovery of architectures that are both high-performing and compatible with embedded deployment constraints.

The remainder of the paper is organized as follows: **Section II** reviews related work on *NAS* for video action recognition. **Sections III–IV** present the proposed approach, including the search space, the three-stage procedure for sampling, training/evaluation, and policy update, as well as experimental

This work has been supported by Projects PID2023-148285OB-C42 and PID2023-148285OB-C43, funded by the Spanish MICIU/AEI/10.13039/501100011033 and by FEDER, UE, as part of the OASIS project (Open AI-Driven stack for enhanced HPEC platforms in embedded systems).

results. Finally, **Section V** concludes and outlines future research directions.

II. Previous Work

NAS in computer vision was initially applied to image-based tasks such as classification [10], object detection [11], and semantic segmentation [12]. Subsequent research has advanced in two main directions: improving search efficiency—using *RL* [13], [14], neuroevolution [15], [16], or gradient-based optimization [17]—and introducing hardware-aware constraints, such as latency or *FLOPs*, into the objective function [18]-[22]. However, these approaches largely remain focused on static image tasks and do not address the temporal modeling requirements of video understanding.

Extending *NAS* to video-based tasks presents additional challenges due to temporal dynamics and higher computational costs. Early video *NAS* work has centered on search space design, typically employing 3D convolutions [23], [24] or multi-stream 2D *CNN* architectures [25], with limited attention to deployment constraints.

A more recent approach [26] introduces a customizable neuron-level search space and a latency-constrained evolutionary algorithm for edge deployment. However, such methods rely on fully customized architectures trained from scratch, often underperforming relative to manually designed models with pretrained components, and requiring custom hardware adaptations for real-time execution.

In contrast to prior methods that rely on 3D convolutions or multi-stream 2D *CNN* architectures to capture temporal information, this work explores a modular *CNN-RNN* search space that explicitly separates spatial and temporal modeling, enabling the use of pretrained *CNN* backbones and simplifying hardware mapping. Moreover, while recent video *NAS* efforts [26] adopt evolutionary strategies with latency filtering, the proposed framework employs a *RL*-based controller with an *a priori* causal hierarchy and a position-based discounted reward, directly guiding architecture exploration toward designs optimized for both task performance and hardware deployability.

III. Search Space

The search space encompasses nine key parameters defining *CNN-RNN* architectures, balancing diversity and computational feasibility for *SoC FPGA* deployment (Figure 1). The convolutional component includes three backbone options: VGG16 (sequential 3×3 convolutions), ResNet50 (residual connections), and InceptionV3 (multi-scale parallel convolutions). The recurrent component is characterized by six additional parameters: number of layers (1–3), cell type (long short-term memory, *LSTM*, or gated recurrent unit, *GRU*), and three independent parameters specifying the number of units in each of the potential layers. Each units parameter can take values in $\{8, 16, 32, 64, 128, 256, 300, 512, 700, 900, 1024\}$, with second and third layer additionally allowing the value 0 to enable coherence between the number of active layers and the parameterization. The recurrent configuration is further

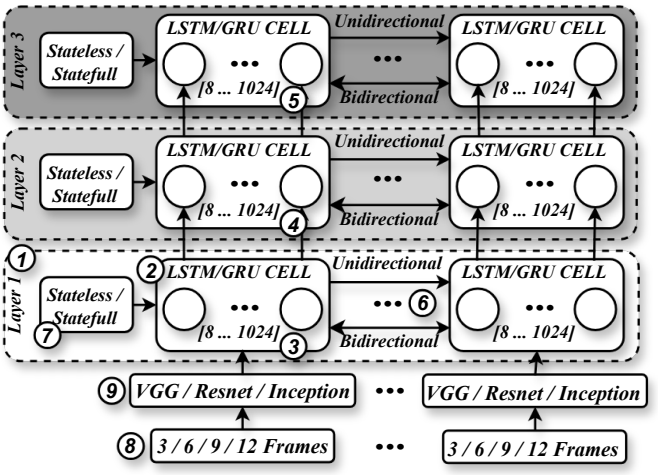

Fig. 1. *CNN-RNN architectures search Space.*

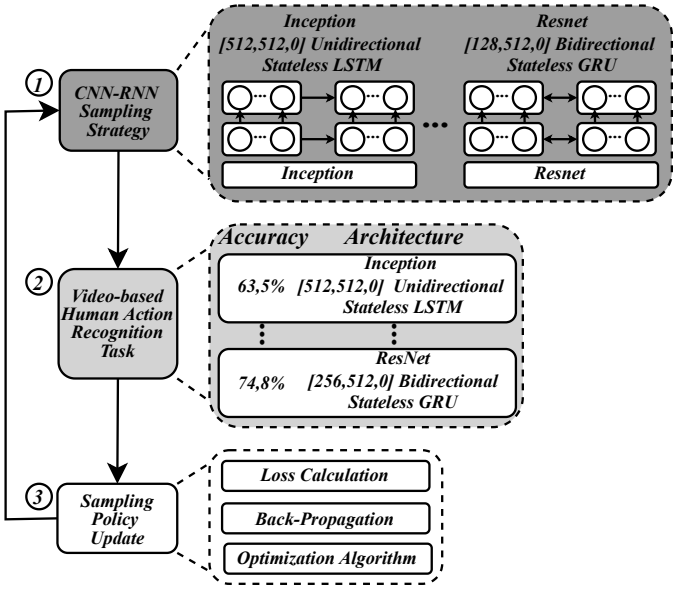

Fig. 2. *Exploration Strategy guided by LSTM-based Controller.*

controlled by directionality (uni- or bidirectional), state mode (stateless or stateful), and input sequence length (3, 6, 9, or 12). The search space comprises 32,064 valid configurations combining these convolutional backbones with parametrized *RNN* modules, where the first three parameters define the *RNN* topology and the remaining parameters govern its temporal processing.

The enumeration used in Figure 1 reflects the order in which these parameters are encoded into token sequences for sampling and optimization. This ordering is intentionally designed to align with an *a priori* causal hierarchy among architectural decisions, which will be further discussed in Section IV.

IV. Exploration Strategy

The proposed exploration strategy follows a three-stage process illustrated in Figure 2: (1) sampling candidate *CNN-RNN* architectures, (2) evaluating their performance on the target task, and (3) refining the sampling policy through iterative training.

A generative language model, implemented as an *LSTM*-based controller, orchestrates the process. It samples architectures from a structured search space via autoregressive token generation. Each token is sampled from a conditional probability distribution, ensuring syntactic validity through predefined constraints. Sampled architectures are evaluated on a video action recognition task, with their accuracy serving as the reward signal for policy refinement.

This policy is updated through a hybrid learning scheme combining self-supervised and *RL*. Rather than relying on fixed labels, the controller learns from the sampled architectures and their observed performance, enabling it to progressively associate architectural choices with task-specific rewards and improve future generations.

A. CNN-RNN Sampling Strategy

The controller operates over a vocabulary $\mathcal{V}$ encoding all valid values for the P parameters defining a *CNN-RNN* architecture. To organize the generation process, $\mathcal{V}$ is partitioned into P disjoint sub-vocabularies:

$$\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_P, \tag{1}$$

where each $\mathcal{V}_p$ specifies the set of allowable tokens for the p-th parameter.

At each cell p, the *LSTM* processes the previous sequence token and updates its hidden state h_p, which encodes the history of decisions $a_{<p}$.

The *LSTM* outputs a probability distribution over the entire vocabulary:

$$p(a \mid h_p) = \text{Softmax}(W h_p + b), \tag{2}$$

where W and b are the learnable parameters of the vocabulary projection layer in the final *LSTM* cell, a represents the predicted token sequence, and h_p is its corresponding hidden state.

To ensure parameter-specific sampling, a masking operation is applied to restrict $p(a \mid h_p)$ to the current sub-vocabulary $\mathcal{V}_p$:

$$\tilde{p}(a_p \mid h_p) = \text{Mask}_{\mathcal{V}_p}(p(a \mid h_p)), \tag{3}$$

where invalid tokens outside $\mathcal{V}_p$ are zeroed out and the distribution is re-normalized.

The token for parameter p is then sampled from this masked distribution:

$$a_p \sim \tilde{p}(a_p \mid h_p). \tag{4}$$

Thus, the full sequence $a = (a_1, \ldots, a_P)$ is generated according to the masked, conditioned distribution:

$$\pi_\theta(a) = \prod_{p=1}^{P} \tilde{p}(a_p \mid a_{<p}). \tag{5}$$

Additionally, a further conditional masking step is applied to ensure global architectural consistency, enforcing constraints such as requiring the number of specified layers to match the number of non-zero unit specification parameters.

B. Video-based Human Action Recognition Task

Each sampled *CNN-RNN* architecture is evaluated on a video action recognition task, where a reward signal guides the exploration process. The process comprises video sampling and preprocessing, architecture training, and prediction.

To handle variable video lengths, each input video is converted into a fixed-length frame sequence using a dynamic sampling strategy that ensures uniform temporal spacing and preserves contextual information.

The *CNN-RNN* training follows a two-stage scheme. The *CNN* is initialized with ImageNet-pretrained weights and fine-tuned on the target dataset to extract spatial features, which are then used as input to the *RNN*. The *RNN* is trained on feature sequences, with sequence generation depending on the *RNN* state mode: for stateless architectures, overlapping sequences are used during training and non-overlapping during validation; for stateful architectures, non-overlapping sequences preserve temporal continuity, enabling the hidden state to accumulate context across the full video.

Both components include task-specific classification heads and are trained using cross-entropy loss, the Adagrad optimizer, and dropout regularization.

The final video-level prediction also depends on the *RNN* state mode. For stateless architectures, independent subsequence predictions are aggregated using either average fusion or majority voting, with the best-performing strategy selected based on validation accuracy. For stateful architectures, the model processes the video sequentially, with only the final subsequence contributing to the optimization loss, while intermediate subsequences update the *RNN* hidden state to capture long-range temporal dependencies.

C. Sampling Policy Update: Self-Supervised and Reinforcement Learning

The controller is trained to optimize a unified objective that integrates syntactic learning and task-specific performance through a reward-weighted policy gradient loss:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(a_t^n \mid s_t^n) \cdot y_t \cdot \tilde{R}_t^n, \tag{6}$$

where $\tilde{R}_t^n$ is the normalized reward for token t in sequence n, N is the number of sampled architectures, and T the number of tokens per architecture.

This objective combines two complementary signals: a self-supervised learning (SSL) component that captures the syntactic structure of valid *CNN-RNN* architectures, and a *RL* component that incorporates task-specific feedback.

In SSL, the controller is trained autoregressively over token sequences representing architectures, with a right-shifted target:

Input: $x = [a_1, a_2, \ldots, a_{T-1}]$, *Target:* $y = [a_2, a_3, \ldots, a_T]$.

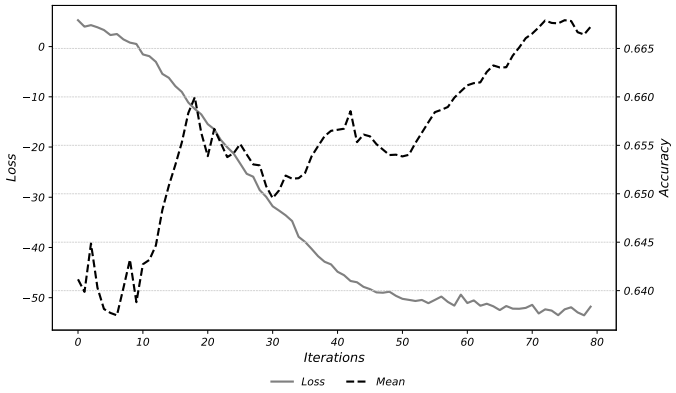

Fig. 3. *Controller loss and CNN-RNN accuracy metrics.*

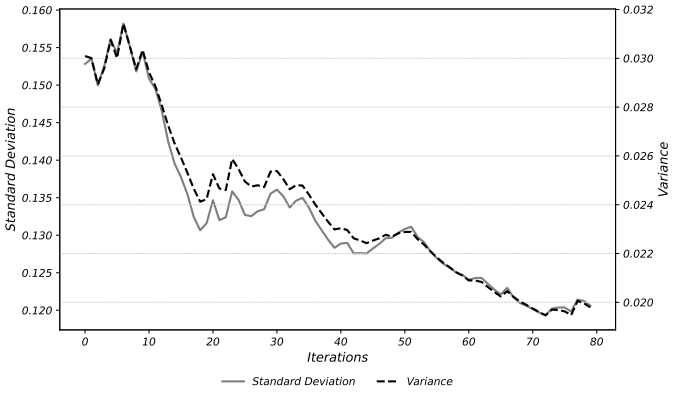

Fig. 4. *CNN-RNN candidate architectures accuracy dispersion metrics.*

and a categorical cross-entropy loss:

$$\mathcal{L}_{\text{SSL}} = -\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) \cdot y_t. \tag{7}$$

The RL component formulates architecture generation as a sequential decision process. At each time step t, the controller samples an action a_t from the policy $\pi_\theta(a_t|s_t)$, where s_t represents the sequence of previously selected tokens. Each generated architecture is evaluated through a validation-based scalar reward R, adjusted by a moving average baseline to reduce variance.

A key contribution of the proposed approach lies in the design of the RL signal. Rather than distributing R uniformly across tokens, a position-based discounted reward is applied, assigning greater weight to tokens appearing earlier in the sampling sequence. Importantly, the token ordering itself is deliberately designed to reflect an *a priori* causal hierarchy among architectural decisions, based on their expected influence on overall model performance. Early tokens in the sequence—such as those defining layer count, cell type, and units per layer—are positioned to represent decisions that fundamentally constrain model capacity and representational power. In contrast, tokens corresponding to parameters governing temporal behavior or convolutional settings appear later in the sequence, as their impact is considered more localized. By aligning the discounted reward with this token ordering, the proposed strategy seeks to guide the controller toward more efficient exploration, prioritizing high-impact architectural decisions and reducing the likelihood of suboptimal configurations driven by late-stage parameter choices.

The controller training follows an iterative deep learning procedure adapted for architecture search, where the outer loop runs for E_o epochs. In each iteration, a batch of N architecture sequences is sampled autoregressively from the controller, then each architecture is instantiated and trained on the downstream task for E_v epochs to obtain the best validation accuracy, which serves as the scalar reward R^n. Subsequently, the controller is updated over E_c internal epochs using the batch of architectures and their rewards, where each internal epoch involves computing the reward-weighted policy gradient loss, backpropagating gradients with respect to the controller parameters θ, and updating via a gradient-based optimizer.

V. Results

The proposed *NAS* framework was evaluated on the *UCF101* dataset [28], which contains 13,320 videos categorized into 101 distinct action classes. The actions are grouped into five categories based on interaction type: Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, and Sports. The original videos have a resolution of 320×240 pixels and a frame rate of 25 *Frames per second* (*FPS*). The training process for the *CNN* and *RNN* components employed distinct frame sampling strategies based on their respective computational and learning considerations. For the *CNN* block, 15 frames per video were sampled, leveraging pretrained models and transfer learning to reduce training cost. For the *RNN* block, which was trained from scratch without pretrained weights, 72 frames per video were sampled to provide richer temporal information, facilitating the learning of sequence-level representations.

In the conducted experiment, 100 iterations were performed. In each iteration, the controller generated a batch of four *CNN-RNN* architecture sequences, followed by 5 policy update epochs. For each architecture, the reward was computed after 10 training epochs and evaluation on the *UCF101* dataset, using an initial baseline value of 0.5.

Two key metrics were monitored to evaluate the controller's performance: the loss of the *LSTM*-based controller and the video classification accuracy of the generated architectures. Their evolution over 100 iterations is shown in Figure 3.

The controller's loss decreases progressively, indicating effective optimization of the defined objective function. However, this alone does not guarantee that the loss is properly aligned with the target task objective; it is therefore necessary to directly analyze the accuracy of the generated architectures.

Figure 3 reports the cumulative moving average of mean accuracy per iteration, highlighting long-term trends. A slight

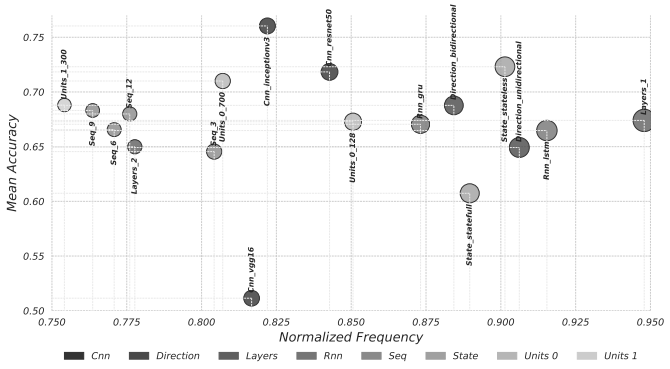

Fig. 5. *Token sampling frequency vs. token mean accuracy*

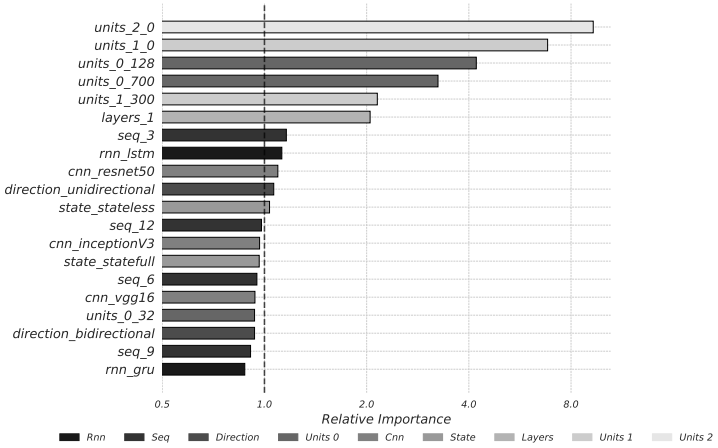

Fig. 6. *Top 20 tokens with higher relative importance to the LSTM-based controller*

upward trend is observed, indicating that the controller increasingly generates architectures with higher classification accuracy. The modest slope of this increase may reflect the relatively high baseline accuracy of most architectures, limiting reward gradients.

To further analyze learning dynamics, the variance and standard deviation of accuracy across each batch were computed (Figure 4). Both metrics progressively decrease, indicating that the controller converges toward generating more consistent architectures with reduced performance variability.

Having established that the controller progressively generates architectures with improved task performance, it is important to assess whether it maintains a diverse exploration of the architecture space or prematurely converges to a limited set of configurations.

Figure 5 presents a comparison between the selection frequency of individual tokens and the average accuracy of the architectures in which they appear. Each point corresponds to a unique token, with color indicating its architectural field. The x-axis shows the normalized global frequency of the token, and the y-axis shows the mean accuracy of architectures containing that token.

This visualization provides insight into the selection patterns of different architectural decisions and their performance contributions. The observed distribution indicates that the controller maintains a broad exploration strategy, with tokens from various components represented across a wide range of frequencies and accuracies, avoiding premature convergence to suboptimal configurations.

Beyond overall controller behavior, it is crucial to evaluate whether the proposed discounted reward strategy has induced the intended causal hierarchy in token selection. A token-level analysis was conducted to examine whether the controller prioritizes architectural decisions with greater structural impact, as encoded in the reward design.

The *relative importance* of each token is defined as the ratio between its empirical selection probability and its prior probability under uniform sampling within its field. Values above one indicate preferential selection; values below one indicate avoidance.

Figure 6 shows the 20 tokens with the highest relative importance. Tokens corresponding to early architectural decisions—such as number of layers and units per layer—are strongly prioritized, consistent with the intended hierarchy. Several later parameters (sequence length, directionality, state mode) also show substantial importance, suggesting their relevance to task performance.

These results indicate that the discounted reward strategy effectively shapes the controller's sampling behavior to emphasize high-impact architectural decisions, while preserving flexibility to adapt later parameters based on empirical performance signals.

The best-performing architecture identified by the controller combines an InceptionV3 backbone with a single-layer bidirectional *LSTM* (128 units per cell), processing input sequences of six spatial feature vectors without maintaining hidden state continuity across predictions. This configuration achieved an accuracy of **81.07%** on the *UCF101* dataset (split 1).

This result surpasses several related *CNN-RNN* approaches using spatial features on *UCF101*. A recent baseline study reported a top accuracy of *73.65%* with a two-layer bidirectional *LSTM* (512 units) configuration [9]. Similarly, [5] achieved *71.12%* using an AlexNet-like *CNN* with a unidirectional *LSTM* (256 units), classifying 16-frame subsequences via simple averaging. [6] reported *77.36%* using InceptionResNetV2 with bidirectional *LSTMs* (512 units), processing 180-frame sequences through adaptive subsampling to preserve full video context.

VI. Conclusions and Future Work

This work introduces a *NAS* framework that serves as a foundational step toward automated and hardware-aware architecture design for video-based human action recognition. The proposed approach integrates two key innovations: a constrained design space tailored for *SoC FPGA* deployment, and a discounted reward strategy aligned with the causal

hierarchy of architectural decisions. Together, these elements enable the systematic generation of high-performing *CNN-RNN* architectures, achieving an accuracy of **81.07%** on the *UCF101* dataset and surpassing prior benchmarks for models relying exclusively on spatial information.

Beyond this result, the study should be regarded as an initial exploration that establishes the groundwork for more extensive experimental research. The defined architecture space and reward formulation have demonstrated promising potential, but further experimentation is required to fully characterize the influence of individual design choices and their interactions.

Currently, hardware-awareness is incorporated through design constraints that ensure implementability on target platforms—specifically, *DPU*s for *CNN*s and ARM-based *SoC*s for *LSTM*s. However, this is a passive form of compatibility; future work will aim to extend the framework to multi-objective optimization, jointly considering accuracy and inference latency on the target *SoC FPGA* platform.

ACKNOWLEDGMENT

This work has been supported by the Spanish Ministry of Science, Innovation and Universities (MICIU), the Spanish State Research Agency (AEI), and the European Union through the European Regional Development Fund (FEDER). This work reflects only the authors' view, and the funding agencies are not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Zhang, Z., & Li, J. (2023). A Review of Artificial Intelligence in Embedded Systems. Micromachines, 14(5), 897. https://doi.org/10.3390/mi14050897

[2] Roth, W., Schindler, G., Klein, B., Peharz, R., Tschiatschek, S., Fröning, H., ... & Ghahramani, Z. (2024). Resource-efficient neural networks for embedded systems. Journal of Machine Learning Research, 25(50), 1-51.

[3] 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), pp. 1372-1376, 2022.

[4] M. H. Siddiqi et al., "A Unified Approach for Patient Activity Recognition in Healthcare Using Depth Camera", IEEE Access, vol. 9, pp. 92300-92317, 2021.

[5] J. Donahue et al., "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 4, pp. 677-691, 1 April 2017, doi: 10.1109/TPAMI.2016.2599174.

[6] 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.

[7] M. Cheshfar, M. H. Maghami, P. Amiri, H. G. Garakani and L. Lavagno, "Comparative Survey of Embedded System Implementations of Convolutional Neural Networks in Autonomous Cars Applications," in IEEE Access, vol. 12, pp. 182410-182437, 2024, doi: 10.1109/ACCESS.2024.3510677.

[8] 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), pp. 184-188, 2019.

[9] D. Suárez, V. Fernández and H. Posadas, "CNN-LSTM Implementation Methodology on SoC FPGA for Human Action Recognition Based on Video," 2024 27th Euromicro Conference on Digital System Design (DSD), Paris, France, 2024, pp. 202-209, doi: 10.1109/DSD64264.2024.00035.

[10] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 8697-8710).

[11] Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X., & Sun, J. (2019). Detnas: Backbone search for object detection. Advances in neural information processing systems, 32.

[12] Zhang, M., Jing, W., Lin, J., Fang, N., Wei, W., Woźniak, M., & Damaševičius, R. (2020). NAS-HRIS: Automatic Design and Architecture Search of Neural Network for Semantic Segmentation in Remote Sensing Images. Sensors, 20(18), 5292. https://doi.org/10.3390/s20185292

[13] Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

[14] Baker, B., Gupta, O., Naik, N., & Raskar, R. (2016). Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167.

[15] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V. & Kurakin, A.. (2017). Large-Scale Evolution of Image Classifiers. ¡i¿Proceedings of the 34th International Conference on Machine Learning¡/i¿, in ¡i¿Proceedings of Machine Learning Research¡/i¿ 70:2902-2911 Available from https://proceedings.mlr.press/v70/real17a.html.

[16] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized Evolution for Image Classifier Architecture Search. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 4780-4789. https://doi.org/10.1609/aaai.v33i01.33014780

[17] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," arXiv preprint arXiv:1806.09055, 2018.

[18] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 2820-2828).

[19] Yang, T. J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., ... & Adam, H. (2018). Netadapt: Platform-aware neural network adaptation for mobile applications. In Proceedings of the European conference on computer vision (ECCV) (pp. 285-300).

[20] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., ... & Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10734-10742).

[21] Cai, H., Zhu, L., & Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332.

[22] Xiong, Y., Mehta, R., & Singh, V. (2019). Resource constrained neural network architecture search: Will a submodularity assumption help?. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1901-1910).

[23] Peng, W., Hong, X., & Zhao, G. (2019, September). Video action recognition via neural architecture searching. In 2019 IEEE international conference on image processing (ICIP) (pp. 11-15). IEEE.

[24] Ryoo, M. S., Piergiovanni, A. J., Tan, M., & Angelova, A. (2019). Assemblenet: Searching for multi-stream neural connectivity in video architectures. arXiv preprint arXiv:1905.13209.

[25] Zhou, Y., Li, B., Wang, Z., & Li, H. (2021). Video Action Recognition with Neural Architecture Search. Proceedings of Machine Learning Research, 157(2021).

[26] Piergiovanni, A. J., Angelova, A., & Ryoo, M. S. (2022). Tiny video networks. Applied AI Letters, 3(1), e38.

[27] Zhu, Y., Li, X., Liu, C., Zolfaghari, M., Xiong, Y., Wu, C., Zhang, Z., Tighe, J., Manmatha, R., Li, M.: A comprehensive study of deep video action recognition. arXiv preprint arXiv:2012.06567 (2020)

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