niCNN: A Novel Neuromorphic Approach to Energy-Efficient and Lightweight Human Activity Recognition on Edge Devices

Preeti Agarwal
School of Technology Management and Engineering,
Narsee Monjee Institute of Management Studies, Maharashtra, India

***Abstract*—Recent years have seen a surge in the use of deep learning for human activity recognition (HAR) in various applications. However, running complex deep learning models on edge devices with limited resources, such as processing power, memory, and energy, is challenging. The objective of this study is to design a novel, lightweight and energy-efficient neuromorphic inspired CNN (niCNN) architecture for real-time HAR on edge devices. The niCNN architecture consists of four stages: design of a shallow CNN, conversion into an equivalent spiking network using Clamping and Quantization (CnQ) algorithm to minimize information loss, threshold balancing to calculate spiking neuron firing rate using Threshold Firing (TF) algorithm, and edge deployment. The experimental evaluation shows that the niCNN architecture achieves 97.25% and 98.92% accuracy on two publicly accessible HAR datasets, WISDM and mHealth. Furthermore, the niCNN technique retains a low inference latency of 2.25 ms and 2.36 ms, as well as a low memory utilization of 22.11 KB and 31.84 KB, respectively. Furthermore, energy usage is reduced to 5.2w and 5.8w. In comparison to various state-of-the-art and baseline CNN models, the niCNN architecture outperforms them in terms of classification metrics, memory usage, energy consumption, and inference delay. The CnQ algorithm reduces memory usage and inference latency, while the TF algorithm improves classification accuracy. The findings show that neuromorphic computing has a lot of potential for resource-constrained edge devices.**

***Keywords*—Neuromorphic computing; human activity recognition (HAR); edge computing; convolution neural network (CNN); spiking neural network (SNN); sensors**

I. Introduction

Human activity recognition (HAR) has garnered significant attention in recent years due to its potential applications in healthcare, sports, and security [1]. Many deep learning techniques, including Convolutional Neural Networks (CNNs), have been applied to HAR with promising results [2]. However, most of these techniques are designed to be run on high-performance computing systems, which can limit their practical applications.

Conversely, performing HAR on edge devices, such as smartphones or wearables has emerged as viable solution due to their ubiquity, low cost, and personalized features [3]. However, running complex deep learning models on edge devices poses many challenges, including limited processing power, memory, and energy resources [4] [5]. Therefore, there is a need to develop lightweight and energy-efficient deep learning models for HAR on resource-constrainted edge devices.

One promising approach can be the use of neuromorphic networks, which mimic the brain's neural structure in a new and alternative computer architecture. They significantly reduce energy consumption by replacing weight multiplications with additions [6]. Very few studies have explored the potential of neuromorphic computing for HAR.

The major objective of this study is to propose a novel neuromorphic CNN (niCNN) architecture for HAR that is specifically tailored to edge device. Traditional methods of transferring weights from a CNN to a neuromorphic network often result in accuracy loss and higher inference latency, making them unsuitable for real-time applications. To address this drawback, the niCNN architecture utilizes clamping and quantization (CnQ) algorithm along with improved threshold firing (TF) algorithm to improve accuracy while reducing inference latency, memory requirements and energy consumption. The key contributions of the study are:

- Designing a novel niCNN architecture for real-time edge processing.
- Introducing a CnQ algorithm that reduces inference latency, energy consumption, and memory requirements of the model while maintaining accuracy.
- Incorporating a novel TF algorithm to improve the accuracy of the model.
- Experimental evaluation is performed on two publicly available HAR datasets, WISDM [7] and mHealth [8].
- The performance of niCNN is evaluated in terms of classification metrics, memory consumption, energy consumption and inference latency. The results show that the proposed architecture takes the lead compared to many existing model.

The study is organized as follows: Section II presents a brief overview of human activity recognition (HAR) and related work. Section III describes the proposed niCNN model in detail, including the CnQ and TF algorithms. Section IV presents the experiment and compares the model with the state-of-the-art approaches. Section V provides the performance results of niCNN. Finally, Section VI concludes the study with suggestions for future research directions.

II. Background and Related Work

A. Background

The process of recognizing human activities through sensors involves four key stages, namely data acquisition, preprocessing, learning, and evaluation [9]. The data acquisition stage involves collecting data from sensors, which is then transmitted to the next stage for further processing. Since the data obtained from sensors may contain artifacts and noise, it needs to be preprocessed. This involves filtering, segmentation, feature extraction, and selection. Filtering helps remove unwanted values and fill in any missing ones. Segmentation involves breaking the data into smaller segments or windows, which are then used for feature extraction and selection. Finally, different classification algorithms are used for activity recognition, and their performance is assessed using defined metrics.

B. Related Work

For HAR, various researchers have proposed several neuromorphic approaches to reduce the energy consumption in on-edge deployment. These approaches include using spiking neural networks (SNNs) [10], bio-inspired artificial retinas [11], data fusion from multiple sensing systems [12] [13], and data preprocessing frameworks [14]. Additionally, researchers have proposed frameworks that integrate physics and neurobiology to model and recognize human actions and object activities, as well as a hardware module that uses a Hopfield Neural Network to detect falls through sensor data integration and classification [15] . In [16], the authors presents a hardware module that uses a Hopfield Neural Network to recognize falls through sensor data integration and classification. In study [17], the use of SNNs in HAR tasks has been proposed as a way to achieve spatio-temporal feature extraction and reduce energy consumption by up to 94%.

Table I provides the listing of different neuromorphic approaches to HAR, each with its unique advantages and limitations. However, these approaches have limitations in terms of accuracy or processing for real-time applications. In comparison to existing approaches, the proposed niCNN, inspired by neuromorphic computing and incorporating the CNQ and TF algorithms, achieves higher accuracy with more optimization parameters, as shown in the table.

TABLE I. Comparison of Approaches

Ref.	Focus of Work	Advantages	Limitations	Optimization Parameters considered			
				Energy	Memory	Latency	Accuracy
[10]	Neuromorphic approach for mitigating energy consumption for edge AIoT HAR applications using SNN.	High performance at low energy cost, effective dealing with temporal signals.	Limited comparison, only DNN, LMU architecture used, requiring application oriented hyperparameter optimization.	✓	×	✓	✓
[11]	Applied SNN and event memory surfaces for HAR using event based camera.	Eliminates data redundancy, allows for low latency and data sparisity.	Technique is complicated and difficult to easily adapt to domain.	✓	✓	×	✓
[12]	Performed multisensory data fusion for precise HAR using neuromorphic computing	Reliable, robust, and achieved high accuracy with multi-sensor data.	Requires additional hardware signal processing for flexible integration of data.	×	×	×	✓
[14]	Proposed data processing framework using neuromorphic computing to reduce the need for extensive sensor training samples.	Robustness with lesser number of training samples requirement.	Not full HAR deployment discussed.	×	✓	×	✓
[13]	HAR using Neuromorphic computing with one-shot learning on multi-sensor fused data	Increased robustness and accuracy of activity recogniton	Limited dataset	×	✓	×	✓
[15]	Integration of physics and neurobiology to model and recognize human actions	Global representation of motion, integration of neurobiological models, multiple hypothesis testing framework.	Very complex implementation	×	×	×	✓
[16]	Designing hardware module for fall detection using Hopfield Neural Network.	High accuracy, power-efficient, faster data classification.	Limited to fall detection, hardware specific design	✓	✓	✓	✓
[17]	Explored AI based SNN for HAR allowing spatio-temporal feature extraction.	Reduced energy consumption due to binary spikes with higher accuracy.	Relies on traditional artifiacl neural networks	✓	×	×	✓
Ours	Designed a four-stage neuromorphic model for HAR	Higher accuracy with low power consumption, memory and lesser inference time.	Training time higher with multiple training stages. Hyperparameter optimization required	✓	✓	✓	✓

III. Proposed Methodology

In this section, the niCNN architecture is discussed. The proposed architecture addresses the following neuromorphic architecture and CNN integration issues.

- In neuromorphic models, such as Spiking Neural Network (SNN), input values are binary [0,1] at each step, whereas in CNN they are floating point values,

inappropriate conversion reduces accuracy and inference speed.

- Secondly, the activation behavior of IF neurons in SNNs differs from ReLU neurons in CNNs.
- Thirdly, the choice of threshold balancing algorithm greatly affects the performance of the model by affecting the firing rate, energy consumption and information transfer of neurons. High firing rate can lead to overfitting, whereas low firing rate can lead to reduced information transfer.

The niCNN follows a series of basic steps. First, the data is acquired through sensors and preprocessed to create a shallow CNN architecture. Then, an optimal shallow CNN architecture is converted into an equivalent spiking neural network architecture using the CnQ transformation model that minimizes information loss. Next, the firing threshold of the spiking neurons is determined using TF, a threshold balancing technique. The final step is to deploy the inference model, which takes spike-encoded data as input, enabling efficient implementation of deep learning on resource-constrained devices. The schematic workflow of the model is shown in Fig. 1. The Algorithm 3 for niCNN implementation is as follows:

Algorithm 3: Procedure: niCNN

Phase I: Data Acquisition

Step1: Collect sensor data

Step 2: Aggregate sensor data

Phase II: Preprocessing

Step 3: Remove missing values and outliers

Step 4: Windowing and Segmentation

Phase III: Model Building

Step 5: Train a shallow CNN

Step 6: Apply CnQ algorithm for weight transformation

Step 7: Set firing threshold values using TF algorithm

Step 8: Deploy SNN inference model

Phase IV: Performance Evaluation

Step 9: Evaluate different performance metrics

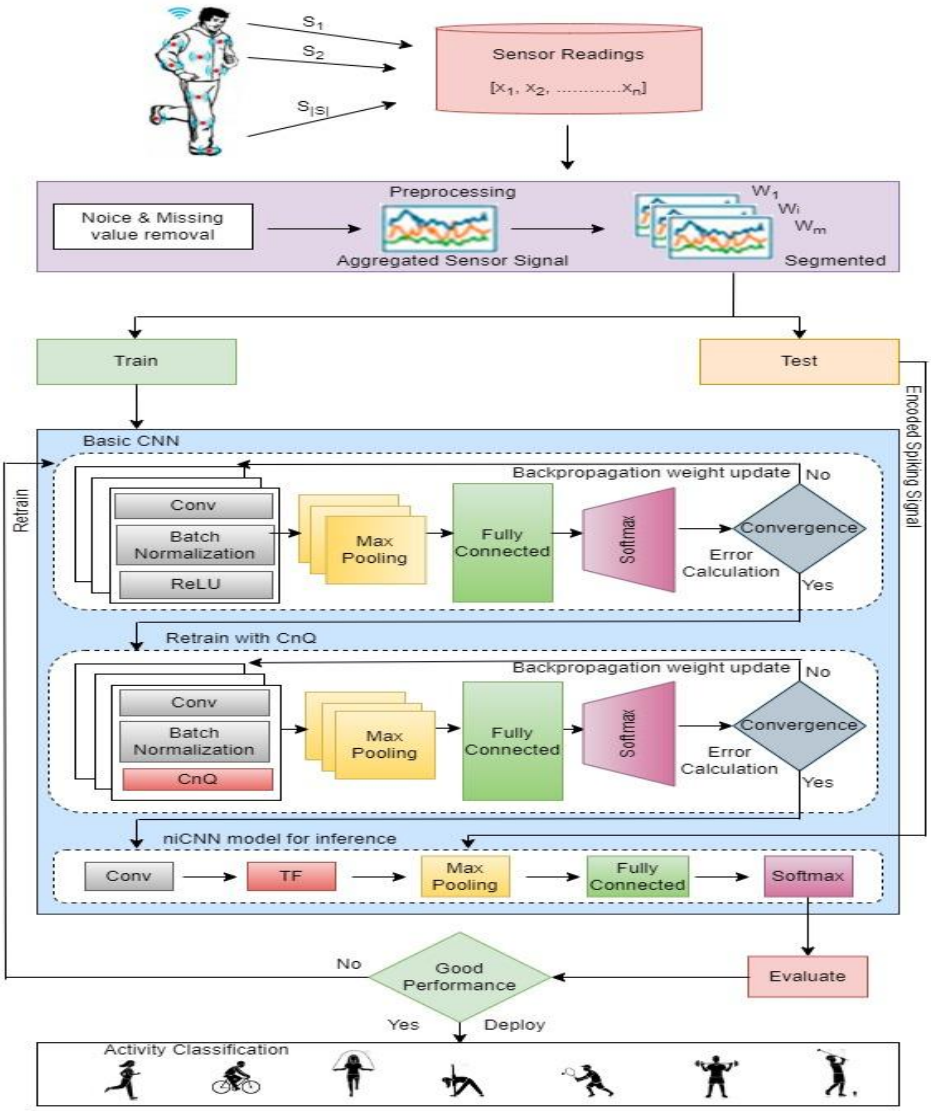

Fig. 1. niCNN architecture.

The description of each phase of the algorithm is as follows:

A. Phase 1: Data Acquisition

A collection of sensors $S = \{s_1, s_2, \dots, s_{|S|}\}$are used to acquire sensor data, each generating a time series reading $X = \{x_1, x_2, \dots x_n\}$. $X(t)$ represents the sensor readings at time step t and $A = \{a_1, a_2, \dots, a_{|A|}\}$ be the number of the activities to be predicted.

The values from the multiple sensors can be aggregated as time series window, as shown in Eq. (1):

$$X = \begin{bmatrix} x_1(1) & \cdots & x_1(t) \\ \vdots & \ddots & \vdots \\ x_n(1) & \cdots & x_n(t) \end{bmatrix} = [X(1), ., X(i), ., X(t)] \quad (1)$$

where, $X(i) = [x_1(i), \dots, x_n(i)]$ is the sensor input at time i for n number of sensor readings.

B. Phase 2: Preprocessing

The samples in the time segment are passed through filters for missing values, noise and outlier removal. After preprocessing, a set of segments W is produced that corresponds to activity A, as shown in Eq. (2).

$$W = \{ws_1, ws_2, \dots, ws_m\} \quad (2)$$

For each segment $ws_i = (t_1, t_2)$ represents a portion of samples from t_1 to t_2.

C. Phase 3: Model Building

The model building consists of the following stages:

1) Train a shallow CNN: The aim of this stage is to train a shallow CNN inference model ρ, A = ρ(X), such that the difference between the actual and predicted values is minimized.

- Define Input Layer: The input layer consists of time series signals. The input at time step t denoted as $X(t)$, and the input to the first convolution layer denoted as $l_o(t)$ is calculated using Eq. (3):

$$l_o(t) = X(t) = X^0 \quad (3)$$

- Convolution Layer: It consists of a series of L layers, $L = \{l_1, l_2, \dots l_{|L|}\}$, each with n number of neurons. The output of ith neuron at layer l is calculated using Eq. (4):

$$x_i^l = h\left(\sum_{j=0}^{N} (w_{ij}^l . x_j^{l-1} + b_i^l\right) \quad (4)$$

where, w_{ij}^l is the weight between the neuron j at layer $l-1$ and neuron i at layer l, b_i^l is the bias of neuron i at layer l, $h(\cdot)$ indicates the ReLU activation function, which is calculated using $\max\{0, \sum_{j=0}^{n}(w_{ij}^l . x_j^{l-1} + b_i^l)\}$.

- Batch Normalization: To converge the CNN network easily the batch normalization is performed using Eq. (5):

$$BN(x'^l_i) = \frac{\gamma(x_i^l - \mu_\beta)}{\sigma_\beta} + \beta \quad (5)$$

where, μ_β is mean and σ_β is the standard deviation of the current batch. The resultant values are scaled by γ and shifted by β. The γ and β are learned throughout training.

- Pooling Layer: The convolution layer is followed by pooling layer to reduce the size and parameters of the network. It is performed using Eq. (6):

$$Maxpool_i^l(t) = max_{r \in R}(x'^l_i(t * T + R)) \quad (6)$$

- Softmax classification layer: After pooling layer, the input passes through fully connected layer, and finally softmax layer for performing classification. The computed activity scores for each achievement are expressed as $P(^{a_i}/_{x_i, \theta})$, computed using inference method ρ, expressed as

$$P\left(\frac{a_i}{x_i}, \theta\right) = \rho(x_i, \theta), for\ a \in A$$

where, θ is trained parameters of model ρ.The maximum score will then be used to calculate the predicted activity a_i for the segment ws_i.

$$a_i^* = argmax\ P\left(\frac{a_i}{x_i}, \theta\right)$$

- The training is performed through backpropagation and the gradients are computed as:

$$\frac{\partial E}{\partial W^l} = \left(\frac{\partial E}{\partial X^l}\right)^T . X^{l-1} \quad (7)$$

$$\frac{\partial E}{\partial X^{l-1}} = (W^l)^T . \frac{\partial E}{\partial X^l} \quad (8)$$

where, Eq. (7) represents error with respect to weights at layer l and Eq. (8) represents error w.r.t the output of the previous layer. Once the gradients are computed, then weights are updated according to Eq. (10):

$$W^l = W^l - \alpha(\frac{\partial E}{\partial W^l}) \quad (9)$$

where, α is the learning rate of the optimization algorithm. This process is repeated for each training example in the dataset until the error is minimized to a satisfactory level.

2) Retrain using CnQ algorithm: To minimize transfer loss from CNN to spiking neural network, the designed CNN in step 5 is retrained using the CnQ algorithm, a variation of CQ [18]. For the CnQ, the input is first normalized using Eq. (10):

$$X\_norm = \frac{X}{X_{max} - X_{mean}} \quad (10)$$

Then quantized for spike train of length T, using Eq. (11):

$$Q_T(X\_norm(t)) = \left\lfloor \frac{X\_norm(t).T}{T} \right\rfloor \quad (11)$$

Since, batch normalization cannot be applied to the input spike trains, therefore batch normalization is applied to weights w^l and bias b^l using Eq. (12) and Eq. (13):

$$BN(w_i^l) = \frac{\gamma . w_i^l}{\sigma_\beta} + \beta \tag{12}$$

$$BN(b_i^l) = \frac{\gamma (b_i^l - \mu_i^l)}{\sigma_\beta} + \beta \tag{13}$$

Now the input data is transformed in spike trains with discrete values set $T \in \{0,1\}$. Instead of ReLU activation function, the CQ transform is applied. Clamping is first applied to the activation function to restrict them to [0,1] and is defined as C in Eq. (14):

$$C(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \le x \le 1 \\ 1, & x > 1 \end{cases} \tag{14}$$

Then Quantization is applied as in Eq. (15):

$$Q_T(x) = \left\lfloor \frac{x.T}{T} \right\rfloor \tag{15}$$

In CnQ, the forward propagation is modified as Eq. (4). The learning is performed through conventional backpropagation:

$$\frac{\partial E}{\partial W^l} = Q'.C'.\left(\frac{\partial E}{\partial X^l}\right)^T . X^{l-1} \tag{16}$$

$$\frac{\partial E}{\partial X^{l-1}} = Q'.C'.(W^l)^T . \frac{\partial E}{\partial X^l} \tag{17}$$

where, Eq. (16) shows error w.r.t weights at layer l and Eq. (17) represents error w.r.t the output of the previous layer. Once the gradients are computed, then weights are updated according to Eq. (9). Algorithm 1 explains CnQ training details.

Algorithm 1: CnQ Training

Input: Sensor data (X), Activity Label (A)

Output: Trained Layers $\langle l_1, \dots, l_{|L|} \rangle$

Parameters Initialized: Number of layers (L), Weights (W), Bias(B), convergence==false

```
Normalize X    //using eq. 10
Quantize X    //using eq. 11
while(!convergence==true)
    For i=1 to L
        For j=1 to N    //forward propagation
            Calculate x_j^i    //using eq. 4
            Apply BN    //using eq. 12 and 13
            Apply Clamping on x_j^i    //using eq. 14
            Apply Quantization on x_j^i //using eq. 15.
        End for
    End for
    For i = L to 1    //Backpropagation
        Calculate Gradients    //using eq. 16 and 17
        Adjust weights    //using eq. 9
    End for
    Update convergence
End while
Return Trained layers ⟨l_1, ..., l_|L|⟩
```

3) The SN Deployment: While CNN's transmit activation data among layers (x^l) as real numbers. SNN neurons receive series of binary input as spike trains. Formally, s_i^l is a spike train of neuron i at layer l where $s_i^l(t) \in \{0,1\}$ is a spike at time t. Unlike CNNs, the weight sum is merely a summation since the weights are binary.

The IF model is used to map CNN to SNN activation. At each time step t, a neuron i computes the weighted sum of the received input spikes $s_j^{l-1}(t)$ and the corresponding weights and integrates them as the membrane potential V_i. Whenever V_i exceeds a predefined threshold V_{th_i}, the neuron fires a spike '1' and decreases V_i by V_{th_i}. Otherwise, it outputs '0'. Formally,

$$v_i^l(t) = v_i^l(t-1) + (\sum_{j=0}^{n} (w_{ij}^l . s_j^{l-1}(t) + b_i^l)$$

$$where, s_i^l(t) = \begin{cases} 1, & v_i^l(t) \ge v_{th_i}^l \\ 0, & otherwise \end{cases}$$

$$v_i^l(t) = \begin{cases} v_i^l(t) - v_{th_i}^l, & v_i^l(t) > 0 \\ v_i^l(t), & otherwise \end{cases}$$

The firing threshold is set using the TF algorithm, inspired from threshold firing approach [19]. Given the desired inference latency t for SNN, TF records the maximum accumulated activation value for each neuron across timesteps at each layer l, as in Eq. (18):

$$thres_i^l = \begin{cases} \max(thres_i^l, \max\left(\sum_{j=1}^{N} w_{ij}^l x_j^l\right), & l = 1 \\ \max(thres_i^l, \max\left(\sum_{j=1}^{N} w_{ij}^l s_j^l(t)\right), & 1 < l < L \end{cases} \tag{18}$$

After the assignment of the firing threshold for a layer, it freezes the thresholds of the layer and repeats the threshold determination for the next layer. Algorithm 2 explains the working of TF.

Algorithm 2: TF setting

Input: Timesteps for inference (T)

Output: Firing threshold for each layer $\left[\{v_{th_i}^l\}_{i=1}^N \right]_{l=1}^L$

```
For l = 1 to L
    For i = 1 to N
        Set thres_i^l = 0
    End for
    If (l === 1)
        For i = 1 to N
            Calculate thres_i^l    //using eq. 18
            Set v_th_i^l = thres_i^l
```

```
        End for
    End if
    Else if (l == L)
        Set v_{th_i}^l = ∞
    End if
    else
        For t = 1 to T
            For i = 1 to N
                Calculate thres_i^l   //using eq. 18
                Set v_{th_i}^l = thres_i^l
            End for
        End for
    End if
End for
Return v_{th_i}^l list
```

D. Phase 4:Evaluation

- Evaluation metrics: After training network, the performance can be tested on the various metrics on test data and can fine-tuned as necessary as discussed in [20] accuracy ($\frac{tp+tn}{tp+tn+fp+fn}$), precision ($\frac{tp}{tp+fp}$), recall ($\frac{tp}{tp+fn}$), f1-score ($\frac{2*(precision*recall)}{precision+recall}$), specificity ($\frac{tn}{tn+fp}$), balanced accuracy ($\frac{1}{2}(\frac{tp}{tp+fn}+\frac{tn}{tn+fp}$), where tp is true positive, fp is false positive, tn is true negative. fn is false negative.
- Evaluate number of parameters: To estimate the number of parameters in a spiking CNN model, using Eq. (19):

$$np = ((ic * fs * nf + bias) * os\ ) \quad (19)$$

where, ic represents the number of channels in the input spike trains, fs represents the size of the filters, nf represents the number of filters in the convolutional layer, bias represents the bias term for each filter, and os represents the size of the output spike trains.

- Calculate memory usage: The memory utilized by a spiking CNN model can be estimated by multiplying the number of parameters by the number of bytes used to store each parameter. For example, if we assume that each parameter is stored as a 32-bit float, then the memory utilized can be calculated, as in Eq. (20), whereas niCNN uses 8 bit clamped and quantized signal; therefore each parameter is 8 -bit.

$$memory\ utilized\ (in\ bytes) = np * 4 \quad (20)$$

- Power Consumption: As per Intel specification [21], Intel core i7 processor consumes 65 watts, whereas Intel Loihi 2 consumes around 1 watt. To calculate the power consumed in execution, it is done using Eq. (21):

$$power\ consumption = execution\ time * watts\ of\ processor \quad (21)$$

IV. Experiment

This section elucidates the experimentation conducted on two widely used public datasets, namely WISDM and mHealth, to evaluate the efficacy of niCNN. All experiments were performed on a single system that was equipped with an i7-5500U processor, 16 GB of RAM, and a 1 TB hard disk drive. The implementation of niCNN was executed utilizing the dynamic and powerful programming language Python, alongside auxiliary packages such as Nengo, NengoDL, and NNI [22]. The proposed network architecture was constructed utilizing Keras, a Python-based high-level neural networks API that can run on top of TensorFlow.

A. Dataset Description

WISDM and mHealth datasets contain time series data for physical activity that were collected using wearable sensor devices. The WISDM includes data for six physical activities that were performed by 29 subjects at a sampling rate of 20 Hz, while the mHealth dataset includes data for 12 physical activities performed by 10 subjects at a sampling rate of 50 Hz. Table II provides an overview of each dataset's specifications, and Table III describes captured activity data.

TABLE II. Dataset Specifications

Dataset	Subjects	Sensors	Sampling Rate	Activities	Samples
WISDM	36	A	20 Hz	6	1,098,209
mHealth	10	A,G, M, ECG	50 Hz	12	1,215,745

[a.] A= Accelerometer, G= Gyroscope, M= Magnetometer, ECG= Electrocardiography

TABLE III. Activity Data

Dataset	Activities (Labels, Samples, % Distribution)
WISDM	Walk(1,424400,38.6), Jog(2,342177,31.2), Up(3.122869,11.20), Downstairs(4,100427,9.1), sit(5,59939,5.5), standing(6,48397,4.4)
mHealth	Standing(1,30720,9), Sitting(2,30720, 9), Lying down(3, 30720,9), Walking(4, 30720,9), Climbing stairs(5, 30720,9), Forward bending(6, 28315,8.3), Frontal Arm elevation(7,29441, 8.6), Knees bending(8, 29337, 8.5), cycling(9, 30720,9), Jogging(10, 30720,9)Runing (11, 30720,9), jumping(12,10342,3)

B. Preprocessing

A total of 1,098,209 and 3,44,116 samples were collected from WISDM and MHealth. To minimize computational effort for on-edge deployment, only segmentation was performed as the preprocessing step. The signal was partitioned into non-overlapping temporal windows of length 2s. After dividing the samples based on raw data without feature extraction, a split of 70:30 was used to create training and testing sets.

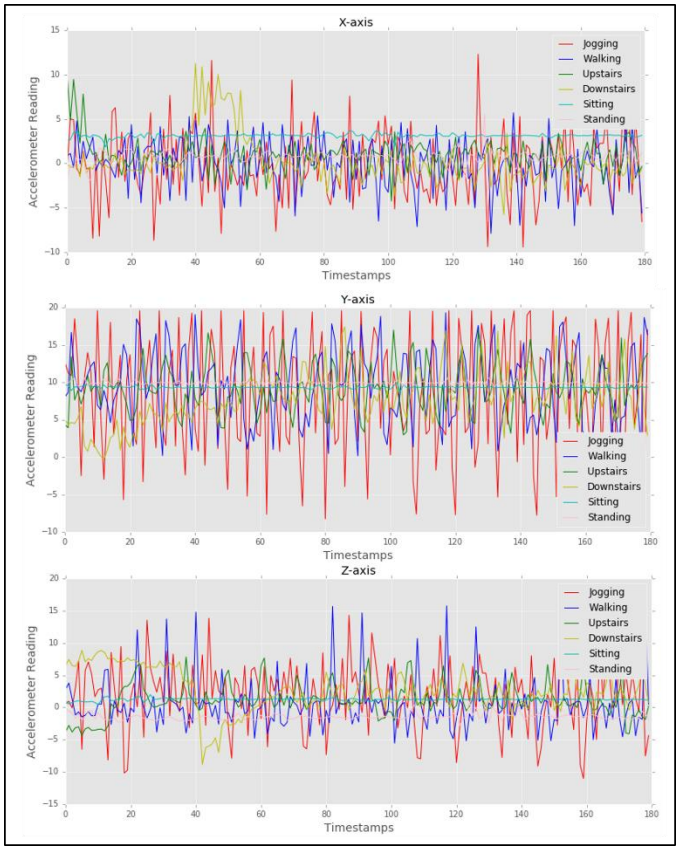

Fig. 2. WISDM activity visualization.

WISDM does not contain outliers, missing values, or null activities; thus, all samples were used for experimentation. Fig. 2 displays a visualization of the WISDM dataset at a timestep of 180. After eliminating null activities from mHealth, the final sample size consists of 343195 samples. At a timestep of 80, a visualization of the mHealth dataset is presented in Fig. 3.

C. Network Architecture

A shallow CNN architecture is developed, consisting of two convolution layers, a max pooling layer, a flattening layer, and two dense layers. The model underwent fully supervised training, with the gradient backpropagated from the Softmax layer.

In the context of the study, cross-entropy loss function is utilized to quantify the dissimilarity between the predicted and actual distributions.

Adam, a stochastic optimization algorithm founded on the first-order gradient, is utilized as the optimizer. The training process is performed over 25 epochs, with a batch size of 64 and a small learning rate of 0.001 to improve the fitting ability. To improve the model's robustness, the training set is randomly shuffled. The hyperparameters are chosen after analyzing the effect of various factors, including the number of filters, batch size, filter size, and dropout rate, on accuracy. Fig. 4 illustrates the impact of these factors on accuracy, and the final hyperparameters are listed in Table IV.

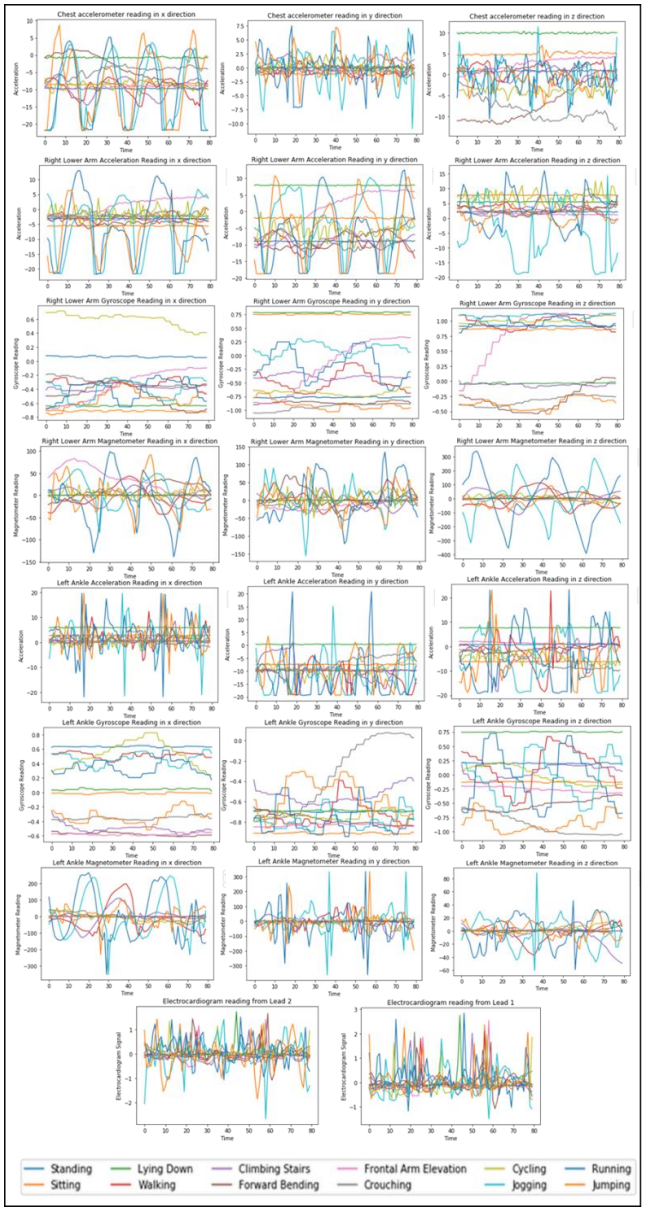

Fig. 3. mHealth activity visualization.

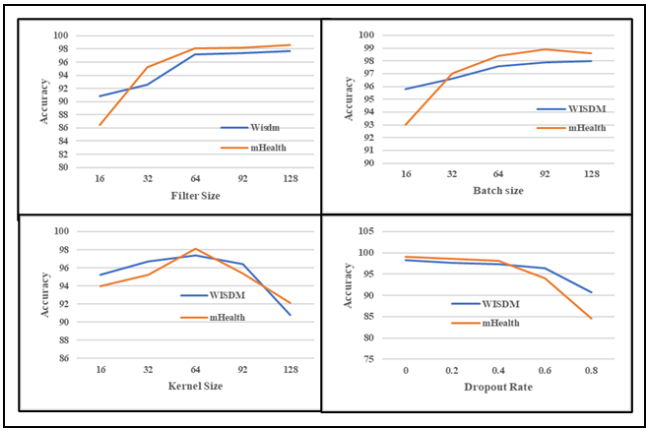

Fig. 4. Impact of hyperparameters on accuracy.

TABLE IV. SELECTED HYPERPARAMETERS

Stage	Hyperparameters	Selected Value
Preprocessing	Window_size	180(WISDM) 80(mHealth)
	Step_size	100
Architecture	Conv1_kernel_size	3
	Conv1_filter_size	64
	Maxpooling_size	2
	Conv2_kernel_size	3
	Conv2_filter_size	64
	Dense layer	128
	Dropout	0.5
Training	Optimizer	Adam
	Batch_Size	64
	Learning rate	.001
	Number of epochs	25

The niCNN is developed using the Nengo neural simulator on the Loihi neuromorphic chip. To construct the spiking CNN directly from its non-spiking counterpart, the NengoDL converter was used. Hyperparameter tuning is conducted using the Neural Network Intelligence (NNI) toolkit and Annealing algorithm, while ensuring a proper search space. Each optimization experiment comprised 1,000 trials, with the tuner being randomly re-initialized four times in equal intervals to avoid local minima. Following every trial of 25 training epochs, the weights yielding the best training accuracy are employed to evaluate the test accuracy. To train all investigated networks, including optimization of the learning rate throughout the experiment trials, Adam optimizer with a constant learning rate is utilized. Fig. 5 and Fig. 6 represent training progress of WISDM and mHealth over epochs.

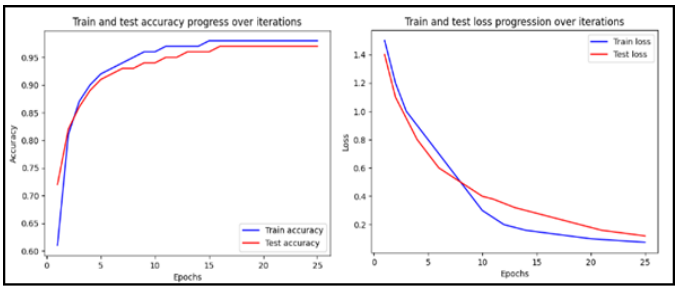

Fig. 5. WISDM dataset progress over epochs.

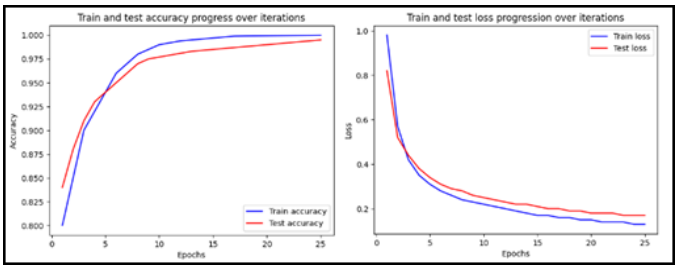

Fig. 6. mHealth dataset progress over epochs.

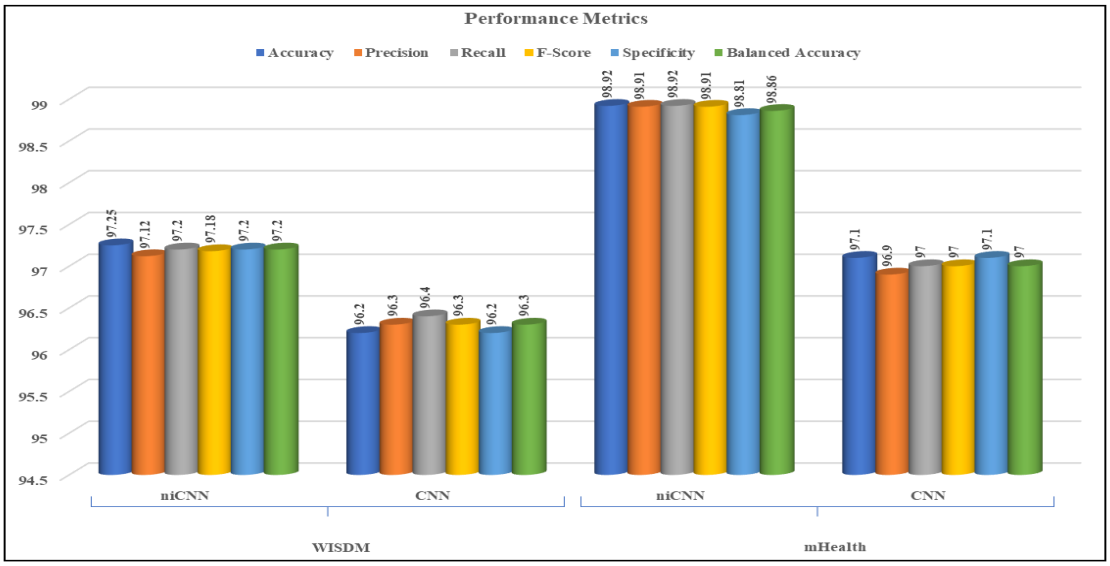

Fig. 7. Performance comparison.

V. RESULTS

The evaluation of performance was carried out using the metrics described in Section III, and the results are presented in Table V. niCNN achieved 98.08% average accuracy, 97.25% on WISDM dataset and 98.92% on mHealth. Fig. 7 provides a comparison of niCNN on both datasets against its baseline CNN on the same architecture. Additionally, the obtained confusion matrix for the WISDM and mHealth datasets are shown in Fig. 8 and Fig. 9, respectively. As per results, presented in Fig. 7 and Table V, niCNN obtained on average 1.1% increase in accuracy, 97.4% reduced power consumption, 60.04% reduced memory consumption, and 30.2% reduced inference time.

niCNN achieved a notably higher accuracy compared to other state-of-the-art works([10], [23]–[33]), as demonstrated in Fig. 10.

TABLE V. PERFORMANCE METRICS

	WISDM		mHealth	
	niCNN	***CNN***	***niCNN***	***CNN***
#P	22,642	14,326	32,612	20,286
#ET(s)	5.2	3.2	5.8	4.1
#M(kBytes)	22.11	55.96	31.84	79.24
#PC(watts)	5.2	208	5.8	266.5
#IT(ms)	2.25	3.27	2.36	4.02

b. #P=Number of parameters, #ET= Execution Time, #M=Memory used, #PC=Power Consumption, #IT=Inference Time

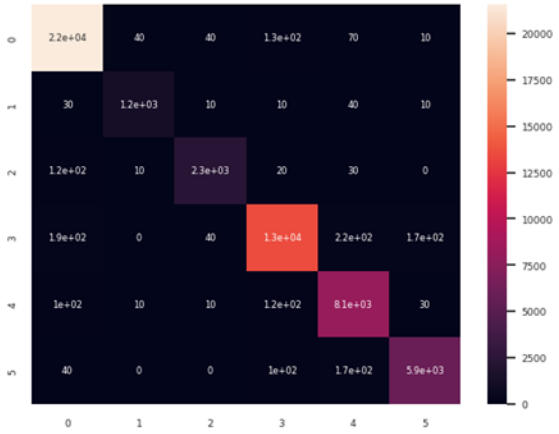

Fig. 8. WISDM confusion matrix.

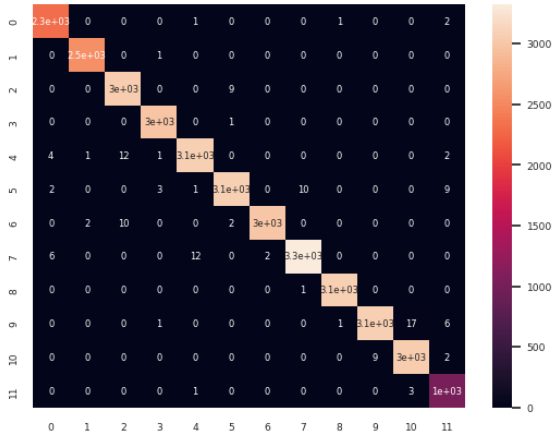

Fig. 9. mHealth confusion matrix.

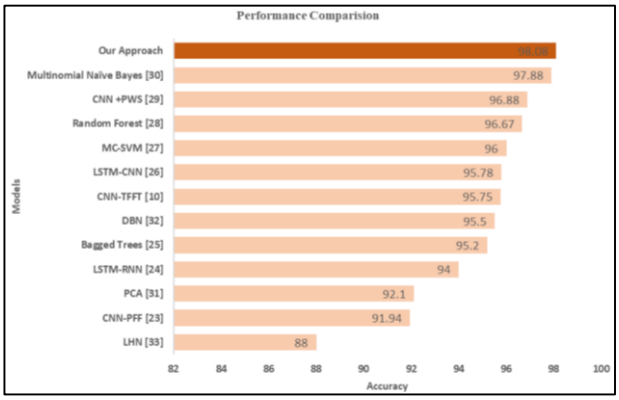

Fig. 10. Comparison of performance accuracy.

VI. CONCLUSIONS

The study demonstrates the capability of neuromorphic computing to perform real-time HAR on edge devices with limited resources. It presents a novel four stage niCNN architecture, overcoming the accuracy loss and higher inference latency associated with transferring weights from a CNN to a neuromorphic network by applying CnQ algorithm. The new improved threshold balancing approach, TF, further increases the inference accuracy of the system. The experimental results demonstrate that the niCNN architecture achieves high accuracy, low inference latency, memory usage, and energy consumption compared to the state-of-the-art and baseline CNN models.

In future, niCNN performance can be investigated on different HAR datasets. The CnQ and TF algorithms can also be further optimized to achieve better results. Further research can expand the model's capability to recognize more complex activities. The effectiveness of various optimization methods within the Nengo framework, including different windowing techniques can also be explored. Additionally, the niCNN architecture can be extended to other real-time processing applications on edge devices.

REFERENCES

[1] O. D. Lara and Miguel A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1192–1209, 2012, doi: 10.1109/SURV.2012.110112.00192.

[2] E. Ramanujam, Thinagaran Perumal, and S. Padmavati, "Human activity recognition with smartphone and wearable sensors using deep learning techniques: A review," *IEEE Sens J*, vol. 21, no. 12, pp. 13029–13040, 2021.

[3] P. Agarwal and M. Alam, "Edge optimized and personalized lifelogging framework using ensembled metaheuristic algorithms," *Computers and Electrical Engineering*, vol. 100, p. 107884, 2022, doi: 10.1016/j.compeleceng.2022.107884.

[4] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu, "Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities," *ACM Comput Surv*, vol. 54, no. 4, pp. 1–40, Jul. 2021, doi: 10.1145/3447744.

[5] P. Agarwal and M. Alam, "A Lightweight Deep Learning Model for Human Activity Recognition on Edge Devices," *Procedia Comput Sci*, vol. 167, pp. 2364–2373, Jan. 2020, doi: 10.1016/J.PROCS.2020.03.289.

[6] D. Wu, X. Yi, and X. Huang, "A Little Energy Goes a Long Way: Build an Energy-Efficient, Accurate Spiking Neural Network From Convolutional Neural Network," *Front Neurosci*, vol. 16, p. 759900, 2022, doi: 10.3389/FNINS.2022.759900.

[7] J. Kwapisz, G. Weiss, and S. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, Mar. 2011, doi: 10.1145/1964897.1964918.

[8] O. Banos, R. Garcia, and A. Saez, "UCI Machine Learning Repository: MHEALTH Dataset Data Set." https://archive.ics.uci.edu/ml/datasets/MHEALTH+Dataset.

[9] A. Bulling, U. Blanke, and B. Schiele, "A tutorial on human activity recognition using body-worn inertial sensors," *ACM Comput Surv*, vol. 46, no. 3, 2014, doi: 10.1145/2499621.

[10] V. Fra, E. Forno, R. Pignari, T. C. Stewart, E. Macii, and G. Urgese, "Human activity recognition: suitability of a neuromorphic approach for on-edge AIoT applications," *Neuromorphic Computing and Engineering* , vol. 2, no. 1, p. 014006, Feb. 2022, doi: 10.1088/2634-4386/AC4C38.

[11] B. R. Pradhan, Y. Bethi, S. Narayanan, A. Chakraborty, and C. S. Thakur, "N-HAR: A neuromorphic event-based human activity recognition system using memory surfaces," in *IEEE International Symposium on Circuits and Systems*, IEEE, 2019. doi: 10.1109/ISCAS.2019.8702581.

[12] Z. Yu *et al.*, "An Intelligent Implementation of Multi-Sensing Data Fusion With Neuromorphic Computing for Human Activity Recognition," *IEEE Internet Things J*, vol. 10, no. 2, pp. 1124–1133, Jan. 2023, doi: 10.1109/JIOT.2022.3204581.

[13] Z. Yu, A. Zahid, W. Taylor, H. Heidari, M. A. Imran, and Q. H. Abbasi, "Multi-Sensing Data Fusion for Human Activity Recognition based on Neuromorphic Computing," in *2021 IEEE USNC-URSI Radio Science Meeting (Joint with AP-S Symposium*, IEEE, 2021, pp. 64–65. doi: 10.23919/USNC-URSI51813.2021.9703558.

[14] Z. Yu *et al.*, "IMU Sensing–Based Hopfield Neuromorphic Computing for Human Activity Recognition," *Frontiers in Communications and Networks*, vol. 2, p. 68, Jan. 2022, doi: 10.3389/FRCMN.2021.820248.

[15] R. J. Sethi, A. K. Roy-Chowdhury, and S. Ali, "Activity recognition by integrating the physics of motion with a neuromorphic model of perception," in *2009 Workshop on Motion and Video Computing, WMVC '09*, 2009. doi: 10.1109/WMVC.2009.5399241.

[16] Z. Yu *et al.*, "Hardware-Based Hopfield Neuromorphic Computing for Fall Detection," *Sensors 2020, Vol. 20, Page 7226*, vol. 20, no. 24, p. 7226, Dec. 2020, doi: 10.3390/S20247226.

[17] Y. Li, R. Yin, H. Park, Y. Kim, and P. Panda, "Wearable-based Human Activity Recognition with Spatio-Temporal Spiking Neural Networks," *https://arxiv.org/abs/2212.02233*, Nov. 2022, Accessed: Apr. 24, 2023. [Online]. Available: https://arxiv.org/abs/2212.02233v1

[18] Z. Yan, J. Zhou, and W. F. Wong, "Near Lossless Transfer Learning for Spiking Neural Networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Association for the Advancement of Artificial Intelligence, May 2021, pp. 10577–10584. doi: 10.1609/AAAI.V35I12.17265.

[19] H. C. V. Ngu and K. M. Lee, "Effective Conversion of a Convolutional Neural Network into a Spiking Neural Network for Image Recognition Tasks," *Applied Sciences 2022, Vol. 12, Page 5749*, vol. 12, no. 11, p. 5749, Jun. 2022, doi: 10.3390/APP12115749.

[20] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf Process Manag*, vol. 45, no. 4, pp. 427–437, Jul. 2009, doi: 10.1016/J.IPM.2009.03.002.

[21] I. Corporation, "Taking Neuromorphic Computing with Loihi 2 to the Next Level Technology Brief."

[22] "Nengo." https://www.nengo.ai/ (last accessed Apr. 24, 2023).

[23] S. Ha and C. Seungjin, "Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors," in *International Joint Conference on Neural Networks (IJCNN),IEEE.*, 2016, pp. 381–388. doi: https://doi.org/10.1109/IJCNN.2016.7727224.

[24] S. W. Pienaar and R. Malekian, "Human Activity Recognition Using LSTM-RNN Deep Neural Network Architecture," in *IEEE 2nd wireless africa conference*, 2019, pp. 1–5. doi: https://doi.org/10.1109/AFRICA.2019.8843403.

[25] S. Khatun and B. I. Morshed, "Fully-Automated Human Activity Recognition with Transition Awareness from Wearable Sensor Data for mHealth," in *IEEE International Conference on Electro Information Technology*, 2018, pp. 934–938. doi: 10.1109/EIT.2018.8500135.

[26] K. Xia, J. Huang, and H. Wang, "LSTM-CNN Architecture for Human Activity Recognition," *IEEE Access*, vol. 8, pp. 56855–56866, 2020, doi: 10.1109/ACCESS.2020.2982225.

[27] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A Public Domain Dataset for Human Activity Recognition Using Smartphones," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Belguim, Apr. 2013.

[28] G. Chetty, M. White, and F. Akther, "Smart phone based data mining for human activity recognition," in *Procedia Computer Science*, 2015, pp. 1181–1187. doi: 10.1016/j.procs.2015.01.031.

[29] M. Zeng *et al.*, "Convolutional Neural Networks for human activity recognition using mobile sensors," in *6th International Conference on Mobile Computing, Applications and Services, IEEE.*, 2014, pp. 197–205. doi: https://doi.org/10.4108/icst.mobicase.2014.257786.

[30] L. Syed, S. Jabeen, M. S., and A. Alsaeedi, "Smart healthcare framework for ambient assisted living using IoMT and big data analytics techniques," *Future Generation Computer Systems*, vol. 101, pp. 136–151, 2019, doi: 10.1016/j.future.2019.06.004.

[31] A. S. Abdull Sukor, A. Zakaria, and N. Abdul Rahim, "Activity recognition using accelerometer sensor and machine learning classifiers," in *2018 IEEE 14th International Colloquium on Signal Processing and its Application, CSPA 2018*, IEEE, May 2018, pp. 233–238. doi: 10.1109/CSPA.2018.8368718.

[32] H. Li and M. Trocan, "Deep learning of smartphone sensor data for personal health assistance," *Microelectronics J*, vol. 88, pp. 164–172, Jun. 2019, doi: 10.1016/J.MEJO.2018.01.015.

[33] A. Jordao, R. Kloss, and W. R. Schwartz, "Latent HyperNet: Exploring the Layers of Convolutional Neural Networks," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2018-July, Oct. 2018, doi: 10.1109/IJCNN.2018.8489506.