AN EVOLUTIONARY APPROACH TO THE DESIGN OF CONVOLUTIONAL NEURAL NETWORKS FOR HUMAN ACTIVITY RECOGNITION

Stefan Tsokov
Department Computer Systems, Technical University of Sofia
stsokov@tu-sofia.bg

Milena Lazarova
Department Computer Systems, Technical University of Sofia
milaz@tu-sofia.bg

Adelina Aleksieva-Petrova
Department Computer Systems, Technical University of Sofia
aaleksieva@tu-sofia.bg

Abstract - Automated human activity recognition has a number of applications such as in elderly healthcare monitoring, fitness tracking and in various smart home systems that can adapt to the inhabitants' behavior. Deep learning using Convolutional Neural Networks (CNNs) is increasingly being used for recognition of human activities. However, the CNNs performance is highly dependent on the network architecture and usually the hyper-parameters are manually selected. Various approaches have been used to automate the design of CNN architectures. The paper proposes an evolutionary based approach for optimizing the architecture of one dimensional CNNs for human activity recognition. The suggested approach is tested on three accelerometric data sets. The experimental results show that the CNNs designed using the evolutionary based approach have comparable accuracy on the WISDM Actitracker data set and significantly better performance on the Smartphone-Based Recognition of Human Activities and Postural Transitions data set compared to other deep CNNs.

***Keywords*:** Convolutional neural networks; Genetic algorithms; Neural network architecture design; Deep learning; Human activity recognition.

1. Introduction

Human activity recognition (HAR) has a number of applications such as in elderly healthcare monitoring, fitness tracking and in various smart home systems that can adapt to the inhabitants' behavior. Recently, deep learning methods have been gaining popularity in HAR due to their ability to recognize more complex patterns in the data compared to shallow neural networks. One very popular deep network is the Convolutional Neural Network (CNN) [20]. The main advantage of CNNs over classical methods is their ability to automatically extract relevant features, usually directly from the raw signal. This feature extraction is done by the convolution of the input signal with filters, the result of which is usually passed through a pooling layer that reduces the size of the data and introduces translational invariance. Often CNNs consist of several successive convolutional and pooling layers, which leads to a hierarchical extraction of features – with each layer the data is represented in a more abstract way. Typically after these layers the convolutional networks contain a fully connected multilayer perceptron, which performs the classification according to the extracted features.

The performance of the CNN depends a lot on the architecture that's being used. Choosing the right architecture is not an easy task because not only does it depend on the specific problem but it also requires the selection of a number of hyperparameters such as the number of convolutional, pooling and fully connected layers as well as how these layers should be connected. In addition to that the parameters for each layer must be defined, for the convolutional layers this usually includes the number of kernels, the size of the kernel and the stride, for the pooling layers – the type of the pooling operation, the size of the kernel and the stride, and for the fully connected layers – the number of neurons. Because the number of hyperparameters is large especially as the network's depth increases and evaluating the performance of a particular CNN architecture usually takes considerable time, most researchers resort to manual optimization, relying on previous experience and using a trial and error approach. Thus in order to obtain relevant architectures for solving a particular problem the experimenter needs to have significant specialized knowledge. In that case, finding effective automated methods

for optimizing the CNN's architecture would be extremely useful for many researchers when solving specific problems.

This paper proposes an evolutionary based approach for optimizing the architecture of a 1D convolutional neural network for recognizing human activities. The paper is organized as follows: Section 2 presents related work on the automatic design of CNN architectures. Section 3 describes the proposed evolutionary approach for designing CNNs for HAR. Section 4 presents the experimental results on three different data sets and the conclusion is presented in Section 5.

2. Related work

In this section some approaches for automatic optimization of CNN architectures are discussed. The majority of the work done so far is in the field of image classification.

A classic approach for optimizing hyperparameters is grid search that covers all possible architectures that can be obtained with predefined possible values for each parameter. However, grid search is not very efficient especially as the number of hyperparameters and the range of their values increase. Instead of searching the entire space defined by the pre-selected discrete values of the hyperparameters, a random search can be used.

Random search shows better results than grid search especially when the performance of the model depends mainly on a relatively small subset of the hyperparameters [7]. The main disadvantage of random search is that the information obtained in the course of the optimization does not affect the search, i.e. the process cannot be directed to more promising areas of the solution space. To overcome this limitation a number of adaptive methods have been proposed for optimizing the CNN hyperparameters such as reinforcement learning and various types of metaheuristic algorithms.

2.1. *Reinforcement learning for CNN architecture design*

Reinforcement learning is one of the possible automatic approaches for optimizing CNN hyperparameters. A policy gradient reinforcement learning is used for segmenting medical images in [19] to optimize the hyperparameters of a base CNN consisting of an encoder and a decoder and the reported results show good performance of the approach compared to other methods for medical image segmentation.

In [5] the authors used Q-learning with an ϵ-greedy exploration strategy to automatically generate CNN architectures by incrementally adding layers. The method is tested on image classification problems with the found architectures performing better than networks using standard layer types as well as showing competitive results against networks using complex layer types. The method also shows better results than other automated network design approaches.

Reinforcement learning is used in [10] to transform an existing network by increasing the depth of the network by adding layers or expanding a layer by adding elements or kernels in such a way as to preserve its function, i.e. allowing the use of already learned weights and thus speeding up the training process. The method is tested experimentally on image recognition data sets and the resulting models performed better than many other network architectures as well as show competitive results against architectures using advanced techniques such as skip connections and branching.

In [37] a recurrent neural network is trained using reinforcement learning to generate convolutional architectures by predicting what the next layer and its parameter would be such as the number of kernels, kernel size and stride. The method generated convolutional architectures that perform better on image recognition than most manually designed architectures.

2.2. *Metaheuristic approaches for CNN architecture design*

Metaheuristic approaches are also utilized for optimization of CNN hyperparameters. In [12] the use of hill climbing is proposed to optimize CNN architectures starting with a small network that is modified and used to gradually generate larger and more complex networks by adding convolutional layers, increasing the number filters or adding skip connections. Two strategies are used for classification to obtain an ensemble of models. The results of experiments on image recognition problems show that the approach is competitive to other CNN design approaches and in addition requires fewer resources than most of them. Two metaheuristic algorithms – tree growth and firefly, are used in [4] to optimize the hyperparameters of a CNN for recognizing handwritten numbers. The approach shows better results in terms of accuracy and computational resources compared to other techniques.

2.2.1. Particle swarm optimization

A commonly used metaheuristic approach for optimizing hyperparameters is Particle Swarm Optimization (PSO). Several different metaheuristic algorithms are used in [11] – PSO, Bat Algorithm, Cuckoo Search and Firefly Algorithm, to optimize the dropout parameter of a CNN network for image recognition. The results show the performance of networks using dropout is better compared to those that do not use it and often the metaheuristic algorithms found just as good or even better values than the default one (0.5).

In [31] PSO is used to optimize the architecture of a CNN for image classification by developing a new strategy for encoding the solution inspired by IP addresses. The solution is a sequence of layers each encoded with a 2-byte IP address, each byte considered as a separate particle dimension.

A new version of PSO is proposed in [32] to further optimize an existing CNN. To improve the exploration ability of the algorithm new positions are generated for the worst particles by a confidence function defined by a compound normal distribution. For more efficient search taking into account that different hyperparameters may have differently sized ranges of values vector coefficients are used instead of scalar ones. To reduce the fitness evaluation time the authors suggested a model to predict the quality of the solution which allows to prematurely stop the training. Experiments on an image recognition problem show that the approach performs better than other methods for optimizing hyperparameters.

In [13] a modified version of PSO with adaptive coefficients is used to optimize the architecture of a CNN. The network consist of a fixed number of modules and the data size is gradually reduced with each module while at the same time the number of kernels gradually increases. The parameter optimized is the number of layers per module and for faster optimization a weight sharing technique is used for each module and its possible dimensions which allows gradual training of individual modules during the optimization process. The method is evaluated on image recognition data sets and is compared with manually designed architectures, reinforcement learning techniques and evolutionary approaches. The comparisons show that the method achieves better balance between performance and computational efficiency compared to most models.

This modular approach to PSO-based architecture optimization is extended in [14] by using an ensemble of models to obtain the classification. Two techniques are suggested for ensemble formation: in the first the models are chosen based on the best solutions found from each particle and in the second the base models are constructed based on the best results shown for each block individually. The most common classification in the ensemble, i.e. the one given by most models, is selected as the final classification. The method is evaluated on an image classification data set and the use of an ensemble gives better results compared to taking only the best model. The method is also compared to other evolutionary techniques for network architecture design showing better accuracy results than most of them while requiring similar or less computational time.

2.2.2. Evolutionary approaches

Evolutionary approaches are widely used metaheuristics for automatic optimization of CNN architectures. In [34] two metaheuristic approaches – standard PSO and two variants of Differential Evolution (DE) are proposed to optimize the architecture of a CNN–LSTM network used to forecast demand for a food shop. The solution is defined as a list of integers with each value representing a separate layer and the numeric value encoding both the layer type and its parameters. The suggested metaheuristic approaches show better results than a popular prediction approach with DE performing better than PSO.

The author of [8] used two metaheuristic algorithms – memetic and simulated annealing to optimize the activation function, the type of pooling function, the kernel size and the stride of a CNN. The suggested metaheuristic algorithms are tested on image recognition data sets showing better results compared to random search and an ordinary genetic algorithm.

Covariance Matrix Adaptation Evolution Strategy is used in [17] to optimize the hyperparameters of a base CNN architecture. The approach is tested on an image classification problem and shows better results compared with Bayesian optimization algorithms especially when the solutions are evaluated in parallel.

In [35] genetic algorithms implemented as a master-slave process are proposed for CNN design, where selection, crossover and mutation are executed on a single node and the evaluation of the population is made on multiple slave nodes. The authors used a fixed convolution architecture optimizing the number of kernels and the kernel size for each convolution layer.

Genetic algorithms are used in [2] to evolve the structure of convolutional networks to classify brain tumors from magnetic resonance data. To improve the performance an ensemble of models is formed by bagging from the best model obtained from the evolution. The results show that the suggested approach performs better than existing methods for classifying brain tumors.

An evolutionary algorithm is suggested in [9] to optimize the number of kernels and the kernel size for each convolutional layer as well as the size of each fully connected layer of a CNN ensemble. A fitness function that takes into account the classification error of the entire population is used to evaluate the solutions and the experimental results on an image classification data set show that the proposed approach performs better than other modern methods.

In [28] the authors propose to evolve a CNN architecture using ResNet and DenseNet units as building blocks. The solution is encoded as a sequence of modules which could be of three types – ResNet module, DenseNet module and a Pooling module. The ResNet and DenseNet modules could consist of a different number of ResNet/DenseNet blocks and the Pooling module consists of one pooling layer. A single-point crossover is applied and the mutation is made at the module level, i.e. as operations to remove, add or change a module. The approach is tested on image classification data sets and shows better results than other automatic and semi-automatic design methods as well as other modern manually designed CNNs.

In [33] genetic algorithms are used to optimize CNN architectures with networks consisting of a fixed number of modules, each module being followed by a pooling layer. Each module comprises a predetermined number of nodes representing convolutional operations and each node receives input data from all previous nodes connected to it. The connections between the nodes are evolved during the optimization process and the experimental results show that structures obtained using smaller data sets could be transferred to larger image recognition data sets.

An evolutionary approach is proposed in [22] to optimize the internal structure of two types of modules – normal and reduction, used in a fixed skeletal architecture for image recognition. All modules of a certain type have the same internal structure consisting of five pairwise combinations of operations and the results of the two operations are combined and transferred to subsequent operations. To obtain a greater variety of solutions and to improve the exploration of the optimization process the authors use a modified tournament selection in which the newly obtained solution replaces the oldest one in the population. The optimization is done on smaller models for faster search after which the best obtained models are enlarged and trained for a longer period of time. Compared to reinforcement learning the evolution produce more accurate models in the early stages of the process. The best evolved architecture is competitive with the best models having similar size (number of parameters) and surpasses them in accuracy when enlarged.

Genetic algorithms are used in [29] to evolve the architecture and the initial weights of a CNN for image recognition. Each solution consists of two parts with convolutional and pooling layers in the first part and fully connected layers in the second part. The lengths of the two parts are determined by generating numbers, then filling the first part, starting with a single convolutional layer and gradually adding convolutional or pooling layers with equal probability. After filling the first part, the fully connected layers are generated. Indirect encoding is used to initialize the network's weights – mean and standard deviation, instead of encoding all weights in the genome. Because the genomes could have different lengths, the authors propose a modified crossover operator that is applied separately to different types of layers. The layers are arranged in the order in which they appear in the network and crossover is applied by exchanging layers with the same positions between two parents. The algorithm shows better accuracy in image recognition problems than many other methods while having fewer parameters than the best ones.

A graph representation is used in [23] for the evolution of the architecture of a convolutional network for image classification in which nodes represent data and edges – convolutional operations. The initial architectures are simple and consist of a single layer without convolutions. To obtain new solutions mutation is applied that can insert, remove or alter entire convolutional layers. The optimization is done with very few restrictions – there is no fixed depth of the network, skip connections are allowed and the values of the parameters are not heavily restricted. The result of the evolutionary process is fully trained models which is accomplished by allowing the children to inherit the weights if the layers are the same and the mutation allows it. The best optimization models are included in an ensemble. Experiments show that the resulting networks are competitive with manually designed networks.

In [18] Neuroevolution of Augmenting Topologies (NEAT) is adapted for the evolution of deep network architectures by interpreting a node in the graph not as a neuron but as a layer and the edges only determine how the layers would be connected. This approach is extended for the coevolution of modules and blueprints for their wiring. Each module is a small deep network and in the blueprints each node point to a specific type of module. To construct the network each node in the blueprint is filled with a randomly selected module of the corresponding type to which the node pointed. The approach is tested on an image recognition problem by evolving the architecture of a convolutional network and on a language modeling task by evolving the architecture of an LSTM network with results comparable to other architectures. The authors also used the suggested approach to develop an application for image captioning on a website of an online shop.

An evolutionary algorithm is proposed in [16] to optimize the structure of a module to be used in a predefined skeletal structure. A hierarchical representation is applied in which smaller graph motifs form larger ones, i.e. lower level motifs are utilized as building blocks for next level motifs. Different types of convolutional and pooling operations are applied as primitive building blocks. For faster optimization during evolution a smaller model with fewer modules is used and then the resulting module is inserted in a larger model. When testing on image classification tasks this approach performs better than many manually designed models. The evolved module is also successfully applied to solve a classification problem with another data set.

A combination of genetic algorithms with grammatical evolution is suggested in [3] to obtain CNN architectures. The solution is represented as a list of layers, each layer encoded using a grammatical approach. Two types of crossover operators are applied – crossing layers of a certain type (convolutional/pooling, fully connected) between two parents or exchanging the whole set of layers of a certain type between two parents. The results of an experimental evaluation on image recognition data demonstrate that the approach performs better than other automated approaches used to generate CNNs.

In [6] genetic algorithms and grammatical evolution are used to evolve a large number of CNN hyperparameters. A sample of the training data is used to reduce the time for the quality evaluation – for each epoch 50% of the training data are randomly selected, and the networks are trained for small number of epochs. After the end of the optimization process the best networks are trained with the entire training set for a longer time. Handwritten number recognition experiments show competitive results of the used approach.

Cartesian genetic programming is used in [27] to automatically find best CNN architectures for image classification. The encoding is based on an acyclic graph placed in a two-dimensional grid, the rows could be considered as different branches and the columns as successive layers. The genome is of fixed length and each gene describes the type of the node and the connections to it. Each node represents a functional unit such as convolution, pooling, concatenation, and summation. The mutation can change the type of the node and its connections. The experimental results show that the obtained architectures are competitive with modern models.

3. Evolutionary based design of convolutional neural networks for human activity recognition

An evolutionary based approach is suggested for automatic design of one dimensional CNNs for human activity recognition. The goal of HAR is defined as matching an input data from inertial, physiological and motion sensors to corresponding human activity. The input data for HAR are usually based on inertial data and the output encodes the recognized human activity, for example walking, jogging, ascending stairs, descending stairs, sitting, standing, etc. A typical 1D CNN architecture used for HAR is shown in fig. 1.

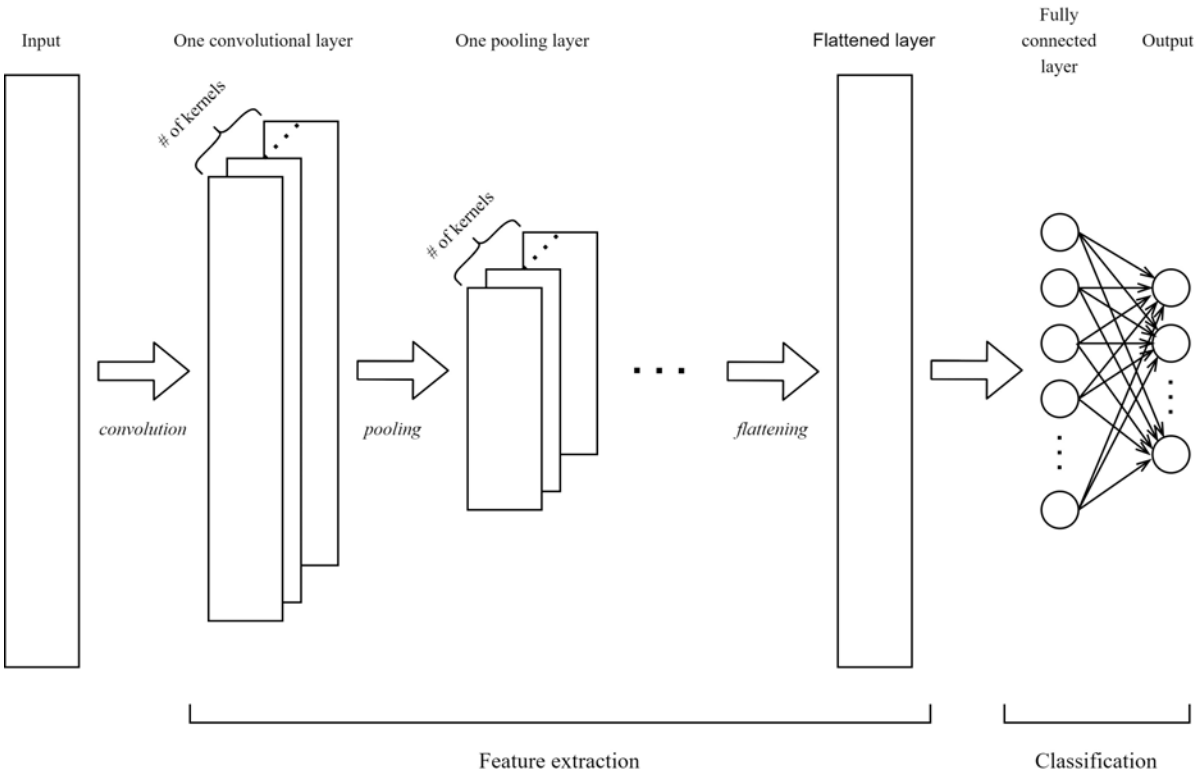

Fig. 1. A typical 1D CNN.

In order to use a genetic algorithm for the optimization of the architecture of a 1D CNN for HAR the following general sequence of steps of the evolutionary paradigm is applied:

(1) An initial population of random solutions is generated;
(2) The solutions in the current population are evaluated using a fitness function;
(3) Selection is applied for the solutions in the current population based on their fitness;
(4) The selected solutions are modified by applying crossover and mutation operations;
(5) A new population of solutions is generated and used at the next iteration (step 2-5) until predefined termination criteria are satisfied.

3.1. *Solution encoding*

The solutions in the population represent 1D CNN architectures and can have varying lengths. In order to simplify the genetic operators used for the modification of the solutions as well as to avoid invalid configurations the suggested encoding of the solutions consists of two parts – convolutional part and fully connected part. Each part of the solution comprises certain number of layers and their corresponding parameters. Thus the solution is represented as (see figure 2) $[C, D]$, $C = [c_1, c_2, c_3, ..., c_n]$, $D = [d_1, d_2, d_3, ..., d_m]$, $c_i = [hc_1, hc_2, hc_3, ..., hc_k]$, $d_i = [hd_1, hd_2, hd_3, ..., hd_l]$, where C is the convolutional part, D is the fully connected part, n is the number of convolutional layers, m is the number of fully connected layers, hc_j, $j = 1 \div k$, are the parameters of a convolutional layer c_i, and hd_j, $j = 1 \div l$, are the parameter of a fully connected layer d_i. Similar solution encoding is used in [4, 9] but additional restrictions are placed on the possible configurations by rearranging the layers by the kernel size and by the number of kernels. In this work these restrictions are not applied since the position of a layer in the solution corresponds to the position of that layer in the assembled network.

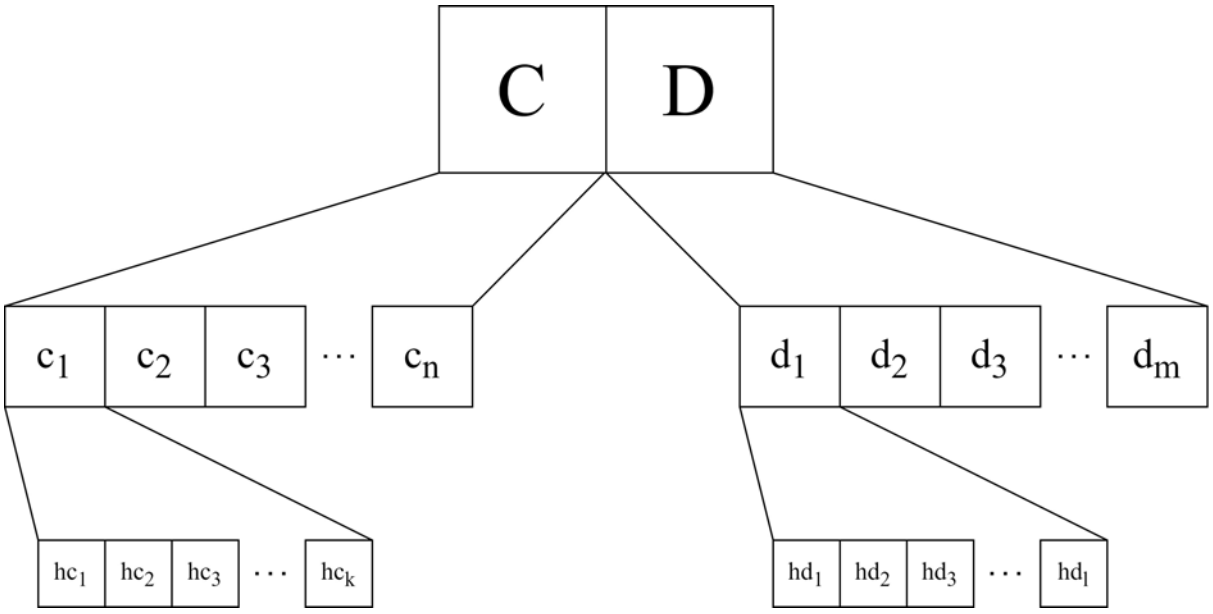

Fig. 2. Solutions‘ encoding.

3.2. *Initial population*

The initial population for the evolutionary procedure consists of random architectures of minimal size. To ensure that the randomly generated solutions are functional the smallest possible networks consist of one convolutional and one fully connected layer. Thus the evolutionary algorithm starts with a population of solutions representing simple networks, each with one convolutional and one fully-connected layer, and the architectures gradually become more complex with the progression of the evolutionary process.

3.3. *Fitness function*

For an architecture to be evaluated it first must be trained. Since the CNN training can take considerable time, to speed up the optimization process a similar approach to the one that is successfully utilized in [3, 5, 6, 13, 18, 27, 29, 31] is applied: the CNN architectures are trained for a small number of epochs, in this case the value is set to 5. During the evolutionary process the networks are trained on a training set and are then evaluated on a validation set. The quality of each solution is evaluated using a fitness function that is the accuracy of the classification obtained on the validation set.

3.4. *Crossover*

In order to avoid invalid CNN architectures the crossover operator is applied to one of the two parts of a selected solution.

The crossover is equally likely to be applied either to the convolutional or to the fully-connected part, and then the layers of the selected part of the CNN architecture are crossed between the two parent solutions. The probability of crossing is not constant, but dynamic and is evaluated as follows:

$$p = 1 - \frac{|l_1 - l_2|}{max(l_1, l_2) - 1}, \quad (1)$$

where l_1 and l_2 are the lengths in number of layers of the selected part (convolutional or fully connected) of the two solutions.

Thus the more similar the two parent solutions are in terms of the number of layers in the part to be crossed, the greater the probability of crossing them. In order to always have some possibility of crossover, a minimum probability is set to 0.05.

The single-point crossover is adapted for solutions with variable lengths. To do this, first the crossover point in the longer solution is randomly selected, then its position is scaled relative to the size of the smaller solution to obtain the crossover point for the shorter solution so that the two points have the same relative positions in both solutions. In the case when the selected part of one of the solutions consists of only one layer the crossover operator exchanges the whole sequence of layers (convolutional or fully-connected part) between the two parents. When no crossover operation is performed copies of both parents are created, mutation is applied on each of them with some probability and both are included in the population for the next generation.

3.5. *Mutation*

After crossover is applied either of the two newly obtained solutions can be mutated with a certain probability. The mutation probability is not fixed but varied starting from 0.8 in the first generation and decreasing by 0.05 with each subsequent one to reach a minimum probability of 0.2. Like the crossover operator, the mutation operator is applied to one of the two parts of the solution with equal probability. There are three possible equally likely mutation changes in the CNN architecture:

- Modifying a layer – the value of one of the hyperparameters of a randomly selected layer is changed with another valid value;
- Adding a layer – a new randomly generated layer is added at a randomly chosen position in the solution;
- Removing a layer – one of the existing layers is selected and removed.

Solution mutation by adding a layer is only allowed if the maximum length of the relevant part of the solution has not been reached. Solution mutation by removing a layer is not allowed when the relevant part of the solution has the minimum allowed length.

3.6. *Selecting the best architecture and full training*

After the optimization is completed by iteratively applying the evolutionary steps until a termination condition is reached, in this case a maximum number of generations, the best architecture of the last generation is selected and is then trained from scratch for a larger number of epochs on a combination of a training and a validation sets. In this case the best architecture is trained for 100 epochs and due to the random nature of the weight initialization, the training is performed 10 times. The CNN architecture performance evaluation is based on a test set that is not used during the architecture optimization.

4. Experimental results

In order to evaluate to performance of the proposed evolutionary based approach for CNN architecture design in an experimental environment with limited computational resources, a small population and a small number of generations are used. The population size for the experimental evaluation of the suggested approach is set to 20 and the number of generations is 50. A tournament selection is utilized for the selection of parents for the crossover operator with a tournament size of 2. Elitism is not applied in the evolutionary procedure. The maximum number of convolutional layers is set to 10 and the maximum number of fully connected layers is set to 5 thus limiting the depth and the complexity of the CNN. In addition to the number of convolutional and the number of fully connected layers the parameters of each layer are also optimized. For the convolutional layers these include the number of kernels, the size of the kernel and whether there is a pooling layer after the corresponding convolutional layer. For each fully connected layer the number of neurons is optimized as well as whether there would be dropout regularization and what the p would be. Table 1 shows the possible values for each hyperparameter used in the evolutionary based optimization.

Table 1. Possible values for each CNN hyperparameter.

Convolutional part	
Number of kernels	[16, 32, 64, 128, 256]
Kernel size	[2, 3, 4, 5, 6, 7, 8, 9]
Include a pooling layer	[True, False]
Fully connected part	
Number of neurons	[100, 250, 500, 750, 1000]
Include dropout	[True, False]
Dropout p	[0.1, 0.2, 0.3, 0.4, 0.5]

All convolutional and fully connected layers use ReLU as an activation function. The stride for all convolutional layers is fixed to 1 and valid padding is applied. The parameters of the pooling layers are not optimized and all pooling layers use max-pooling with size 2, stride 2 and valid padding. The last layer of the CNN architecture is always a softmax layer.

The neural networks are implemented using the Keras neural network library. The parameters of the networks are optimized by minimizing the cross entropy function by the Adam optimisation algorithm with learning rate set to 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 1.0 \times 10^{-7}$. Training and classification are performed using NVIDIA GeForce RTX 2060.

Several metrics are used to evaluate the best models obtained from the evolutionary process: precision, recall, specificity, F_1 score and accuracy, calculated as follows:

$$precision = \frac{TP}{TP + FP} \tag{2}$$

$$recall = \frac{TP}{TP + FN} \tag{3}$$

$$specificity = \frac{TN}{TN + FP} \tag{4}$$

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{5}$$

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{6}$$

where *TP* are true positives, *FP* are false positives, *TN* are true negatives and *FN* are false negatives.

4.1. *Data sets and processing*

The proposed evolutionary approach for CNN architecture design for HAR is experimentally evaluated on accelerometric data from three publicly available human activity recognition data sets. With the exception of the size of the sliding window, all other settings are identical in all three experiments as described above. Each of the data sets is divided into training, validation and test set.

4.1.1. WISDM Actitracker

The WISDM Actitracker [15] data set contains data collected from 36 people under controlled conditions. The data are recorded by a triaxial accelerometer with a sampling frequency of 20 Hz and are labeled for 6 activities – walking, jogging, ascending stairs, descending stairs, sitting and standing. Data from people with a user id of 1 to 29 were used for training, those with id 30 and 31 for validation, and all others, from 32 to 36, for testing. The sensor data is segmented by a sliding window with a size of 10s and an overlap of 50%. The accelerometric values are normalized by z-score.

4.1.2. Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set

The Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set [25] is an extended version of the UCI Human Activity Recognition Using Smartphones data set and consists of accelerometric and gyroscopic data for 12 activities – 6 main activities and 6 transitions between poses, collected from 30 people, with a sampling frequency of 50Hz. The raw data from the accelerometer are used for the experiment, and not

the predefined training and test sets with segmented and processed data. Only the 6 main activities are used in the experimental evaluation – "walking", "walking upstairs", "walking downstairs", "sitting", "standing" and "laying". The selection of only these activities is done to facilitate comparisons with the results of other studies using the first version of the data set in which only these activities are labeled. Data from people with id from 1 to 23 are used for training, those with id 24 and 25 for validation, and all others, from 26 to 30, for testing. The sensor data are segmented using a sliding window with a size of 2.56s and an overlap of 50%, the same as the window selected by the authors for the pre-segmented and processed data, which again allows comparison with the results in studies using these ready-made data. The accelerometric values are normalized by z-score.

4.1.3. PAMAP2

The PAMAP2 [24] data set consists of data for 18 activities – 12 basic and 6 optional, performed by 9 people. The data are collected from three inertial measurement units (IMUs) located on the wrist of the dominant arm, the chest and the ankle of the dominant leg respectively. Each block records at a frequency of 100Hz and consists of two accelerometers, one with a scale of ± 16g and another with a scale of ± 6g, one gyroscope and one magnetometer. In addition to the inertial data, heart rate data are also recorded. The data from the accelerometer with a scale of ± 16g located on the wrist of the dominant hand are used for the experiment.

Data from subjects 8 and 9 are not used in the experiment – subject 8 is excluded because of a left dominant hand and 9 because of the different performed activities. The following 8 activities are used in the experiment, selected to be present in the data of all other 7 subjects – "lying", "sitting", "standing", "walking", "ascending stairs", "descending stairs", "vacuum cleaning" and "ironing". Data from people with id from 1 to 5 are used for training, those from the person with id 6 for validation, and those from the person with id 7 for testing. The sensor data are segmented by a sliding window with a size of 5s and an overlap of 50%. The accelerometric values are normalized by z-score.

4.2. *Results on the WISDM Actitracker data set*

Due to the stochastic nature of the evolutionary algorithm the experimental evaluation on the WISDM Actitracker data set is repeated 10 times. One execution of the optimization procedure without subsequent full training of the best architecture takes on average about 3 hours and the full experiment of all 10 runs as well as the 10 training runs of each of the best architectures takes about 33 hours.

Table 2 shows the best CNN architectures for each of the 10 runs of the experimental evaluation of the evolutionary based architecture design. As can be seen none of the best architectures has a maximum depth defined as 10 convolutional layers and 5 fully connected layers. Most architectures have either 4 or 5 convolutional layers, the smallest number of convolution layers is 3 and the largest is 6. The most common numbers of fully connected layers are 2 and 3, the smallest number is 1 and the largest is 4. In most cases the convolutional layers are followed by a max-pooling layer. The most common number of elements in the fully connected layers is 500 and 750. For half of the fully connected layers, dropout regularization is applied, with the most common value of p being 0.3.

Table 2. The best CNN architectures from each run of the experimental evaluation on the WISDM Actitracker data set.

Run	Best architecture
1	[C(256,5), P(2), C(64,2), C(256,4), P(2), C(16,9), P(2), FC(1000), FC(250), SM(6)]
2	[C(256,5), P(2), C(256,3), C(16,6), P(2), C(16,8), C(256,7), P(2), FC(250), FC(250), Dropout(0.1), FC(100), SM(6)]
3	[C(16,9), P(2), C(64,7), P(2), C(64,7), P(2), C(16,6), P(2), FC(750), Dropout(0.3), FC(750), Dropout(0.2), FC(750), Dropout(0.3), FC(500), Dropout(0.1), SM(6)]
4	[C(256,5), P(2), C(128,3), C(256,8), P(2), C(32,6), P(2), C(32,2), FC(500), FC(500), SM(6)]
5	[C(256,3), C(64,6), P(2), C(32,5), C(16,7), P(2), C(64,9), P(2), FC(500), Dropout(0.5), FC(750), Dropout(0.4), FC(750), Dropout(0.4), SM(6)]
6	[C(32,6), P(2), C(32,9), P(2), C(32,9), P(2), FC(100), SM(6)]
7	[C(128,9), P(2), C(32,9), P(2), C(256,6), P(2), FC(750), FC(500), Dropout(0.5), SM(6)]
8	[C(64,6), C(256,5), P(2), C(32,3), P(2), C(256,5), P(2), C(32,3), P(2), C(64,3), P(2), FC(1000), FC(1000), SM(6)]

9	[C(256,7), P(2), C(32,6), P(2), C(16,8), P(2), C(128,8), P(2), FC(500), Dropout(0.1), FC(500), Dropout(0.2), FC(100), Dropout(0.3), SM(6)]
10	[C(64,9), P(2), C(64,8), P(2), C(32,8), P(2), C(64,7), C(128,5), P(2), FC(1000), Dropout(0.3), FC(100), FC(750), Dropout(0.4), SM(6)]

Table 3 shows the F_1 score and the accuracy averaged across all classes of the best CNN architecture for each of the 10 runs of the experimental evaluation. In order to calculate the evaluation metrics first the F_1 score and the accuracy for each class are averaged over all of the 10 training runs. Then the F_1 score and the accuracy thus obtained are averaged over all the classes. The evaluation of the F_1 score and the accuracy is calculated separately for the best CNN architecture of each experimental run.

As can be seen from the results in table 3 the evolved CNN architectures show good results which are close to some manually designed deep architectures using the same data set (Table 4).

Table 5 shows the performance metrics averaged across all 10 training runs of the best CNN architecture from the 6th run of the experiment. As can be seen the activities "jogging", "sitting", "standing" and "walking" are well recognized with precision, recall, specificity and F_1 score with almost all values above 0.95. On the other hand the model doesn't recognize "downstairs" and "upstairs" activities that well with an F_1 score just over 0.8. The mean F_1 score is 0.9194 ± 0.0711.

Figure 3 shows the mean across all 10 training runs confusion matrix of the best architecture from the 6th run of the experiment. It can be seen that the model does not make mistakes in the classification of Sitting. Because of their similarity the activities "downstairs" and "upstairs" are often misclassified as the other. The network also sometimes confuses the "jogging" activity with "walking".

Table 3. Mean F_1 score and accuracy of the best architecture from every run of the experiment on the WISDM Actitracker data set.

Run	Mean F_1 score ± σ	Mean accuracy ± σ
1	0.9126 ± 0.0889	0.9780 ± 0.0164
2	0.9027 ± 0.0954	0.9746 ± 0.0180
3	0.8724 ± 0.0989	0.9687 ± 0.0187
4	0.9123 ± 0.0918	0.9778 ± 0.0169
5	0.9010 ± 0.0821	0.9762 ± 0.0149
6	0.9194 ± 0.0779	0.9790 ± 0.0149
7	0.8891 ± 0.0933	0.9734 ± 0.0173
8	0.8999 ± 0.0935	0.9755 ± 0.0170
9	0.8905 ± 0.0909	0.9746 ± 0.0157
10	0.8836 ± 0.1116	0.9702 ± 0.0218

Table 4. Comparison with other deep learning results described in the literature using the WISDM Actitracker data set. Mean is the average accuracy of the best CNN architectures from all experimental runs. Top model is the accuracy of the best run.

Source	Model	Accuracy (%)
[1]	Deep Belief Network	98.23
[36]	CNN	96.88
[21]	CNN	98.2
The proposed automatic evolution based approach (mean)	CNN	97.48
The proposed automatic evolution based approach (top model)	CNN	97.90

Table 5. The performance metrics averaged across all 10 training runs of the best CNN architecture from the 6th run of the experiment on the WISDM Actitracker data set.

Activity	precision ± σ	recall ± σ	specificity ± σ	F_1 score ± σ	accuracy ± σ
Downstairs	0.7463 ± 0.0370	0.9245 ± 0.0295	0.9689 ± 0.0057	0.8254 ± 0.0295	0.9649 ± 0.0063
Jogging	0.9922 ± 0.0043	0.9329 ± 0.0184	0.9962 ± 0.0022	0.9615 ± 0.0095	0.9748 ± 0.0059
Sitting	0.9725 ± 0.0317	1 ± 0	0.9976 ± 0.0028	0.9858 ± 0.0165	0.9977 ± 0.0026
Standing	0.9974 ± 0.0052	0.9377 ± 0.0496	0.9999 ± 0.0003	0.9659 ± 0.0258	0.9968 ± 0.0024
Upstairs	0.8563 ± 0.0547	0.7772 ± 0.0321	0.9852 ± 0.0068	0.8136 ± 0.0304	0.9646 ± 0.0065
Walking	0.9532 ± 0.0041	0.9755 ± 0.0045	0.9746 ± 0.0024	0.9642 ± 0.0028	0.9749 ± 0.0019
average	0.9196	0.9246	0.9871	0.9194	0.9790
std	0.0906	0.0710	0.0119	0.0711	0.0136

4.3. *Results on the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set*

The experimental evaluation of the evolutionary based CNN design on the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set is repeated 10 times. One run of the optimization without subsequent full training of the best architecture takes an average of about 2.5 hours and the full experiment with all 10 runs as well as the 10 training runs of each of the best architectures takes about 30 hours.

Table 6 shows the best CNN architectures for each of the 10 runs of the experiment. As can be seen all architectures (with one exception) have either 4 or 5 convolutional layers. Half of the convolution layers are followed by a max-pooling layer. Most CNN architectures have one fully connected layer.

Dropout regularization is applied to about half of the fully connected layers with the most common value of *p* being 0.1. More than half of the fully connected layers have 500 or 750 elements.

Table 7 shows the F_1 score and the accuracy averaged across all classes of the best CNN architecture for each of the 10 runs of the experiment calculated according to the same procedure already described above. As can be seen the evolved CNN architectures using only the accelerometric data show good performance results: all evolved CNN perform significantly better than other deep architectures using the same data set (Table 8).

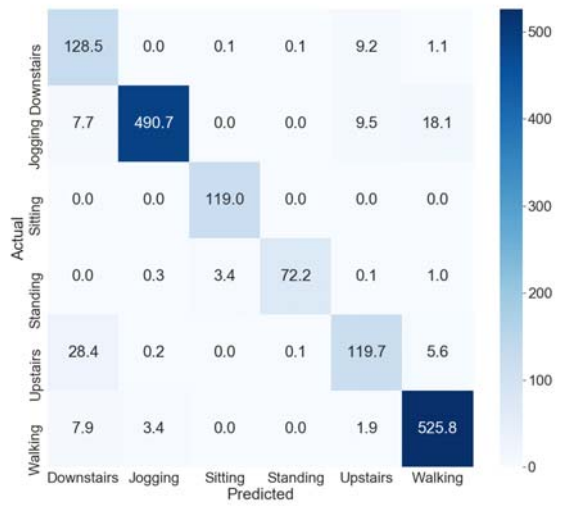

Fig. 3. The mean confusion matrix of the best CNN architecture from the 6th run of the experiment on the WISDM Actitracker data set.

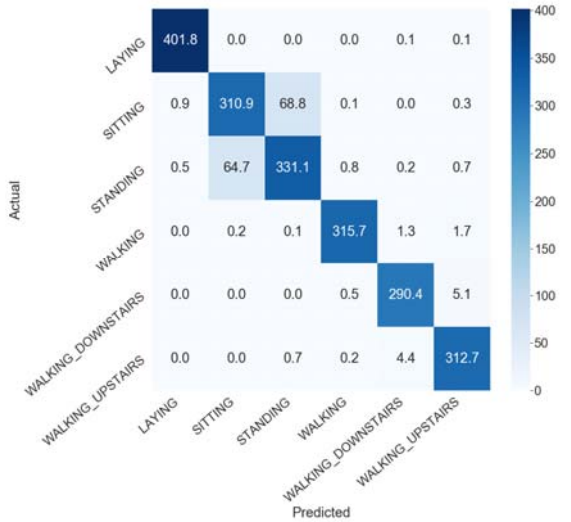

Fig. 4. The mean confusion matrix of the best architecture from the 10th run of the experiment on the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set.

Table 6. The best CNN architectures from each run of the experiment on the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set.

Run	Best architecture
1	[C(128,6), P(2), C(32,8), C(32,7), P(2), C(32,8), C(256,3), FC(500), FC(750), SM(6)]
2	[C(64,9), P(2), C(16,3), C(16,3), P(2), C(256,9), P(2), C(128,6), FC(100), Dropout(0.5), FC(1000), Dropout(0.1), SM(6)]
3	[C(256,9), C(128,9), P(2), C(256,4), P(2), C(64,6), FC(500), Dropout(0.1), SM(6)]
4	[C(64,7), C(16,2), P(2), C(32,6), P(2), C(256,7), P(2), FC(250), SM(6)]
5	[C(64,9), C(16,8), P(2), C(16,5), P(2), C(256,7), P(2), C(64,8), P(2), FC(500), SM(6)]
6	[C(256,7), C(256,5), P(2), C(256,3), C(256,8), FC(100), Dropout(0.1), SM(6)]
7	[C(256,8), P(2), C(256,8), P(2), C(128,4), P(2), C(128,6), C(256,2), FC(250), FC(100), Dropout(0.2), FC(500), SM(6)]
8	[C(256,5), C(32,7), C(16,2), P(2), C(32,3), P(2), C(256,7), P(2), FC(750), SM(6)]
9	[C(256,9), P(2), C(16,3), P(2), C(64,7), P(2), FC(1000), FC(750), SM(6)]
10	[C(64,9), C(32,7), C(64,9), P(2), C(16,5), P(2), C(256,6), FC(750), Dropout(0.4), SM(6)]

Table 7. Mean F_1 score and accuracy of the best CNN architecture from each run of the experiment on the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set.

Run	Mean F_1 score ± σ	Mean accuracy ± σ
1	0.9307 ± 0.0894	0.9751 ± 0.0332
2	0.9296 ± 0.0882	0.9748 ± 0.0328
3	0.9208 ± 0.0995	0.9717 ± 0.0365
4	0.9243 ± 0.0972	0.9728 ± 0.0362
5	0.9225 ± 0.1003	0.9722 ± 0.0373
6	0.9231 ± 0.0974	0.9725 ± 0.0357
7	0.9223 ± 0.1062	0.9719 ± 0.0393
8	0.9293 ± 0.0913	0.9746 ± 0.0340
9	0.9225 ± 0.1010	0.9721 ± 0.0375
10	0.9334 ± 0.0841	0.9761 ± 0.0313

Table 8. Comparison with other deep learning results described in the literature using the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set. Mean is the average accuracy of the best architectures from all experimental runs. Top model is the accuracy of the best run.

Source	Model	Accuracy (%)
[30]	2D CNN+Support Vector Machine	94.49
[26]	1D CNN	94.79
The proposed automatic evolution based approach (mean)	1D CNN	97.34
The proposed automatic evolution based approach (top model)	1D CNN	97.61

Table 9 shows the performance metrics averaged across all 10 training runs of the best architecture from the 10th run of the experimental evaluation. As can be seen "laying", "walking", "walking downstairs" and "walking upstairs" activities are very well recognized with values of the precision, recall, specificity and F_1 score above 0.97. For the activities "sitting" and "standing" the results are not that good with F_1 score of just over 0.8. The mean F_1 score is 0.9334 ± 0.0841.

Table 9. The performance metrics averaged across all 10 training runs of the best CNN architecture from the 10th run of the experiment on the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set.

Activity	precision ± σ	recall ± σ	specificity ± σ	F_1 score ± σ	accuracy ± σ
Laying	0.9965 ± 0.0023	0.9995 ± 0.0010	0.9992 ± 0.0005	0.9980 ± 0.0013	0.9992 ± 0.0005
Sitting	0.8280 ± 0.0209	0.8160 ± 0.0159	0.9626 ± 0.0059	0.8217 ± 0.0102	0.9361 ± 0.0042
Standing	0.8265 ± 0.0111	0.8319 ± 0.0261	0.9594 ± 0.0035	0.8289 ± 0.0138	0.9354 ± 0.0045
Walking	0.9950 ± 0.0058	0.9897 ± 0.0095	0.9991 ± 0.0011	0.9923 ± 0.0050	0.9977 ± 0.0015
Walking downstairs	0.9798 ± 0.0076	0.9811 ± 0.0077	0.9967 ± 0.0013	0.9804 ± 0.0036	0.9945 ± 0.0010
Walking upstairs	0.9754 ± 0.0059	0.9833 ± 0.0063	0.9956 ± 0.0011	0.9793 ± 0.0027	0.9938 ± 0.0008
average	0.9335	0.9336	0.9854	0.9334	0.9761
std	0.0828	0.0853	0.0190	0.0841	0.0313

Figure 4 shows the mean confusion matrix across the 10 training runs of the best architecture from the 10th run of the experiment. As can be seen the model often misclassifies the activities "sitting" and "standing".

4.4. *Results on the PAMAP2 data set*

The experimental evaluation on the PAMAP2 data set is repeated 10 times. One execution of the optimization of the architecture without subsequent full training of the best architecture takes on average about 1.7 hours and the full experiment with all 10 runs as well as the 10 training runs of the best architectures takes about 20 hours.

Table 10 shows the best CNN architectures for each of the 10 experimental runs. Most CNN architectures have 3, 4 or 6 convolution layers, the most common number of fully-connected layers is 2 or 4. In about half of the cases the convolutional layers are followed by a max-pooling layer. The most common number of elements in the fully connected layers is 100. Dropout regularization is applied in most of the fully connected layers with the most common value of p being 0.4 and 0.5.

Table 11 shows the F_1 score and the accuracy averaged across all classes of the best CNN architecture for each of the 10 runs of the experiment, calculated according to the already described procedure. As can be seen the evolved CNN architectures show decent results.

Table 12 shows the performance metrics averaged across all 10 training runs of the best CNN architecture from the 5th run of the experiment. As can be seen the model recognizes the "walking" activity well. The activity "ironing" is also relatively well recognized. However, the model has great difficulty in recognizing the "lying" activity. The mean F_1 score is 0.7387 ± 0.1999.

Figure 5 shows the average confusion matrix across the 10 training runs of the best CNN architecture from the 5th run of the experimental evaluation. As can be seen in most cases the model confuses the activity "lying" misclassifying it mainly as "sitting" and sometimes as the "standing" activity. The model also often misclassifies the activity "ascending stairs" as "descending stairs", as well as "vacuum cleaning" as "ironing" and sometimes confuses "standing" with the "lying" and "sitting" activities, as well as "descending stairs" with the "ascending stairs" and "standing" activities.

Table 10. The best CNN architectures from each run of the experiment on the PAMAP2 data set.

Run	Best architecture
1	[C(128,2), P(2), C(32,3), C(32,3), C(64,3), FC(100), Dropout(0.4), FC(750), Dropout(0.3), FC(1000), Dropout(0.2), FC(1000), Dropout(0.1), SM(8)]
2	[C(64,4), P(2), C(128,4), FC(100), Dropout(0.3), FC(100), Dropout(0.4), FC(750), Dropout(0.5), FC(100), SM(8)]
3	[C(16,2), C(256,7), P(2), C(16,9), C(16,9), C(128,6), P(2), C(32,6), P(2), FC(100), Dropout(0.3), SM(8)]
4	[C(128,2), C(256,3), C(256,2), C(256,2), C(128,2), P(2), C(128,7), P(2), FC(250), FC(250), FC(250), FC(250), Dropout(0.3), SM(8)]
5	[C(16,8), P(2), C(16,5), C(32,9), C(32,9), P(2), FC(100), Dropout(0.5), FC(500), Dropout(0.4), SM(8)]
6	[C(16,9), C(16,4), C(128,4), P(2), C(16,4), C(32,9), P(2), C(16,2), FC(250), Dropout(0.5), FC(750), SM(8)]
7	[C(64,2), P(2), C(16,6), C(256,2), P(2), C(256,2), C(16,7), C(16,7), FC(100), Dropout(0.2), FC(500), Dropout(0.3), FC(1000), Dropout(0.5), SM(8)]
8	[C(64,3), P(2), C(64,3), P(2), C(16,8), FC(100), Dropout(0.5), SM(8)]
9	[C(32,3), P(2), C(16,4), P(2), C(32,7), P(2), FC(100), Dropout(0.4), FC(100), Dropout(0.4), FC(750), SM(8)]
10	[C(64,9), P(2), C(16,4), P(2), C(16,8), C(64,8), P(2), FC(100), Dropout(0.5), FC(250), Dropout(0.4), SM(8)]

Table 11. The average F_1 score and the accuracy of the best CNN architecture for each of the 10 runs of the experiment on the PAMAP2 data set.

Run	Mean F_1 score ± σ	Mean accuracy ± σ
1	0.7184 ± 0.1836	0.9404 ± 0.0285
2	0.6829 ± 0.1693	0.9310 ± 0.0266
3	0.6867 ± 0.2293	0.9289 ± 0.0441
4	0.6536 ± 0.2261	0.9267 ± 0.0310
5	0.7387 ± 0.1999	0.9432 ± 0.0345
6	0.6965 ± 0.2113	0.9369 ± 0.0338
7	0.7117 ± 0.2295	0.9363 ± 0.0412
8	0.7008 ± 0.1885	0.9344 ± 0.0346
9	0.6985 ± 0.2362	0.9345 ± 0.0404
10	0.6819 ± 0.1798	0.9272 ± 0.0356

Table 12. The performance metrics of the best CNN architecture from the 5th run of the experiment on the PAMAP2 data set.

Activity	precision ± σ	recall ± σ	specificity ± σ	F_1 score ± σ	accuracy ± σ
ascending stairs	0.8199 ± 0.0697	0.7169 ± 0.1219	0.9807 ± 0.0122	0.7548 ± 0.0638	0.9541 ± 0.0089
descending stairs	0.6895 ± 0.1073	0.8130 ± 0.0639	0.9719 ± 0.0132	0.7375 ± 0.0441	0.9615 ± 0.0099
ironing	0.8800 ± 0.0262	0.8856 ± 0.0409	0.9756 ± 0.0059	0.8823 ± 0.0279	0.9605 ± 0.0089
lying	0.4157 ± 0.3793	0.2980 ± 0.4052	0.9844 ± 0.0094	0.3127 ± 0.3982	0.8849 ± 0.0586
sitting	0.5244 ± 0.2795	0.8980 ± 0.0183	0.8989 ± 0.0754	0.6185 ± 0.2115	0.8989 ± 0.0694
standing	0.7774 ± 0.1048	0.8417 ± 0.0942	0.9521 ± 0.0448	0.7994 ± 0.0748	0.9359 ± 0.0350
vacuum cleaning	0.9355 ± 0.0391	0.7779 ± 0.0384	0.9922 ± 0.0051	0.8482 ± 0.0222	0.9661 ± 0.0048
walking	0.9574 ± 0.0331	0.9558 ± 0.0260	0.9901 ± 0.0080	0.9559 ± 0.0161	0.9838 ± 0.0060
average	0.7500	0.7734	0.9682	0.7387	0.9432
std	0.1950	0.2059	0.0307	0.1999	0.0345

The best evolved CNN architectures for the WISDM Actitracker data set, the Smartphone-Based Recognition of Human Activities and Postural Transitions data set and the PAMAP2 data set are shown in fig. 6, fig. 7 and fig. 8 respectively.

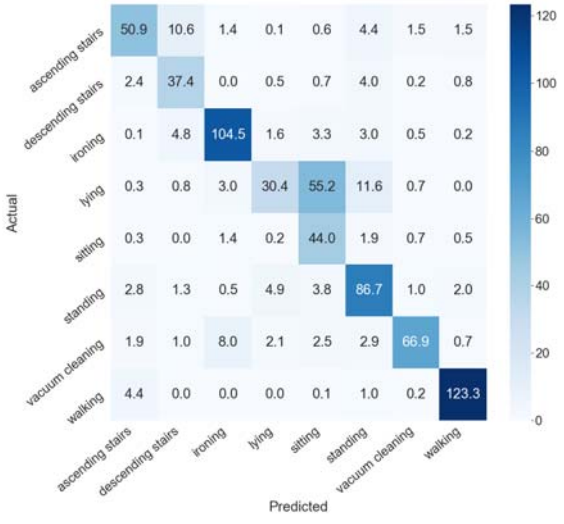

Fig. 5. The average confusion matrix of the best architecture from the 5th run of the experiment on the PAMAP2 data set.

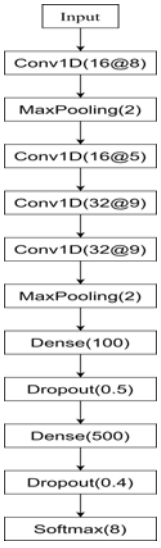

Fig. 6. The best CNN architecture for HAR obtained by the evolutionary based approach on the PAMAP2 data set.

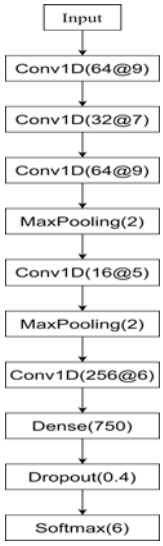

Fig. 7. The best CNN architecture for HAR obtained by the evolutionary based approach on the Smartphone-Based Recognition of Human Activities and Postural Transitions data set.

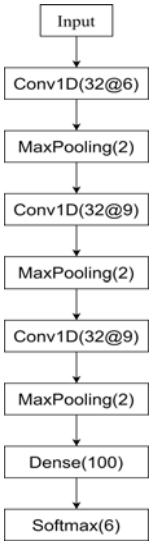

Fig. 8. The best CNN architecture for HAR obtained by the evolutionary based approach on the WISDM Actitracker data set.

4.5. *Additional experimental testing*

In addition to the experiments already described above, two additional experiments are performed for the evaluation of the suggested evolutionary based approach for CNN design for HAR – random search and evolutionary search using only mutation. The WISDM Actitracker data set is used in these additional experiments. The division of the data into a training, validation and test part, as well as the data processing are identical to the main experiment using this data set as described above.

4.5.1. Random search

The results obtained by the evolutionary based approach or CNN design for HAR are compared with the results obtained by random search. Because the evaluation of CNN architectures during the optimization process is the most computationally intensive part, for comparison purposes the same number of architectures are generated and evaluated during the random search as in the evolutionary search: 20 * (50 + 1) = 1020.

The generated architectures have a random length of the convolutional part (from 1 to 10) and the fully connected part (from 1 to 5), and the parameters of each layer are randomly generated using the same possible discrete values as in the above described experimental setup for the evolutionary based approach (Table 1). The architectures are trained for 5 epochs on the training set and then evaluated on the validation set. The random search is repeated 10 times. The best CNN architecture obtained from each run is trained 10 different times on a combination of the training and validation set for 100 epochs and its performance is evaluated on the test set.

Table 13 shows the results of the random search CNN design. The random search achieved F_1 score and accuracy averaged across the 10 experimental runs of 0.8838 ± 0.0254 and 0.9706 ± 0.0061 respectively. For comparison, the average F_1 score and accuracy of the relevant experimental evaluation of the evolutionary based CNN design are 0.8984 ± 0.0146 and 0.9748 ± 0.0033, respectively. The mean values of the random based and the evolutionary based approaches are close, slightly higher for the evolutionary method, but the evolutionary based approach shows more consistent results between the runs.

Table 13. The average F_1 score and accuracy of the best CNN architecture for each of the 10 runs of the random search experiment.

Run	Mean F_1 score ± σ	Mean accuracy ± σ
1	0.9123 ± 0.0872	0.9781 ± 0.0158
2	0.9070 ± 0.0929	0.9760 ± 0.0173
3	0.9072 ± 0.0920	0.9760 ± 0.0173
4	0.8620 ± 0.1180	0.9639 ± 0.0246
5	0.8969 ± 0.0993	0.9737 ± 0.0195
6	0.8765 ± 0.1183	0.9671 ± 0.0242
7	0.8402 ± 0.1017	0.9624 ± 0.0199
8	0.9072 ± 0.0912	0.9769 ± 0.0166
9	0.8668 ± 0.1236	0.9666 ± 0.0255
10	0.8621 ± 0.1128	0.9650 ± 0.0249

4.5.2. No crossover (only mutation)

Additional experimental evaluation is made to test if the use of crossover in the evolution procedure leads to better results than only utilizing mutation for the modification of the solutions. All other settings are the same as in the above described experimental setup of the evolutionary based CNN design.

Table 14 shows the results obtained for the evolutionary based CNN design with only mutation and without crossover. Mean F_1 score and accuracy across the 10 experimental runs are 0.8861 ± 0.0243 and 0.9715 ± 0.0062, respectively. The mean values do not differ much from those obtained with both crossover and mutation where the values are slightly higher: 0.8984 ± 0.0146 and 0.9748 ± 0.0033, but the evolution using crossover showed more consistent results between the runs.

Table 14. The average F_1 score and accuracy of the best architecture for each of the 10 runs of the no crossover experiment.

Run	Mean F_1 score ± σ	Mean accuracy ± σ
1	0.8985 ± 0.0906	0.9749 ± 0.0170
2	0.8792 ± 0.1083	0.9703 ± 0.0204
3	0.8964 ± 0.1034	0.9740 ± 0.0193
4	0.8746 ± 0.0987	0.9690 ± 0.0192
5	0.9087 ± 0.0934	0.9757 ± 0.0179
6	0.8986 ± 0.1025	0.9750 ± 0.0183
7	0.9151 ± 0.0802	0.9786 ± 0.0148
8	0.8857 ± 0.1055	0.9725 ± 0.0194
9	0.8749 ± 0.0987	0.9681 ± 0.0206
10	0.8293 ± 0.1197	0.9565 ± 0.0168

5. Conclusion

Using an inappropriate convolutional neural network for human activity recognition can lead to a significant deterioration in overall performance. Manual design of CNN architectures is a slow and laborious process that usually requires significant specialized knowledge. Automating the search for appropriate CNN architectures can greatly facilitate the optimization process and allow for wider use of deep networks by both the scientific community and the IT industry. Recently, a number of different options have been proposed for the automatic optimization of convolutional architectures such as the use of reinforcement learning as well as various metaheuristic approaches.

In the present work an evolutionary based approach is suggested to automatically optimize a 1D CNN architecture for recognizing human activities. The evolution of the CNN architectures starts with a population of simple architectures that are gradually elaborated by adding new layers. As the process of evolutionary based optimization of the CNN architecture takes considerable time, several measures are applied in order to speed up the optimization procedure – a relatively small number of hyperparameters is optimized; a small number of discrete values are defined for each hyperparameter; a small population is used in the evolutionary procedure and the CNN architecture evolution is applied for a small number of generations as well as the networks are trained for a small number of epochs for their fitness assessment. With all these limitations, an implementation of the optimization algorithm using one GPU took between 1.7 and 3 hours, depending on the data set.

The performance results of the experimental evaluation of the evolutionary based CNN design for the WISDM Actitracker data set is close to the performance of other deep architectures using the same dataset, while on the Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set the results are significantly better compared to other deep architectures. In addition the proposed evolutionary based approach showed more consistent results than random search and evolutionary search with only mutation thus providing a reliable approach for CNN design for HAR using different data sets from various inertial sensors.

Acknowledgments

This work is supported by the European Regional Development Fund and the Operational Program "Science and Education for Smart Growth" under contract UNITe № BG05M2OP001-1.001-0004 (2018-2023).

References

[1] Alsheikh, M. A., Selim, A., Niyato, D., Doyle, L., Lin, S., & Tan, H. P. (2016, March). Deep activity recognition models with triaxial accelerometers. In Workshops at the Thirtieth AAAI Conference on Artificial Intelligence.

[2] Anaraki, A. K., Ayati, M., & Kazemi, F. (2019). Magnetic resonance imaging-based brain tumor grades classification and grading via convolutional neural networks and genetic algorithms. Biocybernetics and Biomedical Engineering, 39(1), 63-74.

[3] Assunção, F., Lourenço, N., Machado, P., & Ribeiro, B. (2018, April). Evolving the topology of large scale deep neural networks. In European Conference on Genetic Programming (pp. 19-34). Springer, Cham.

[4] Bacanin, N., Bezdan, T., Tuba, E., Strumberger, I., & Tuba, M. (2020). Optimizing Convolutional Neural Network Hyperparameters by Enhanced Swarm Intelligence Metaheuristics. Algorithms, 13(3), 67.

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

[6] Baldominos, A., Saez, Y., & Isasi, P. (2018). Evolutionary convolutional neural networks: An application to handwriting recognition. Neurocomputing, 283, 38-52.

[7] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. The Journal of Machine Learning Research, 13(1), 281-305.

[8] Bibaeva, V. (2018, September). Using metaheuristics for hyper-parameter optimization of convolutional neural networks. In 2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP) (pp. 1-6). IEEE.

[9] Bochinski, E., Senst, T., & Sikora, T. (2017, September). Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In 2017 IEEE International Conference on Image Processing (ICIP) (pp. 3924-3928). IEEE.

[10] Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018, April). Efficient architecture search by network transformation. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

[11] De Rosa, G. H., Papa, J. P., & Yang, X. S. (2018). Handling dropout probability estimation in convolution neural networks using meta-heuristics. Soft Computing, 22(18), 6147-6156.

[12] Elsken, T., Metzen, J. H., & Hutter, F. (2017). Simple and efficient architecture search for convolutional neural networks. arXiv preprint arXiv:1711.04528.

[13] Fielding, B., & Zhang, L. (2018). Evolving image classification architectures with enhanced particle swarm optimisation. IEEE Access, 6, 68560-68575.

[14] Fielding, B., Lawrence, T., & Zhang, L. (2019, July). Evolving and Ensembling Deep CNN Architectures for Image Classification. In 2019 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

[15] Kwapisz, J. R., Weiss, G. M., & Moore, S. A. (2011). Activity recognition using cell phone accelerometers. ACM SigKDD Explorations Newsletter, 12(2), 74-82.

[16] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., & Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436.

[17] Loshchilov, I., & Hutter, F. (2016). CMA-ES for hyperparameter optimization of deep neural networks. arXiv preprint arXiv:1604.07269.

[18] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Dufy, N., & Hodjat, B. (2019). Evolving deep neural networks. In Artificial Intelligence in the Age of Neural Networks and Brain Computing (pp. 293-312). Academic Press.

[19] Mortazi, A., & Bagci, U. (2018, September). Automatically designing CNN architectures for medical image segmentation. In International Workshop on Machine Learning in Medical Imaging (pp. 98-106). Springer, Cham.
[20] Nweke, H. F., Teh, Y. W., Al-Garadi, M. A., & Alo, U. R. (2018). Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. Expert Systems with Applications, 105, 233-261.
[21] Ravi, D., Wong, C., Lo, B., & Yang, G. Z. (2016, June). Deep learning for human activity recognition: A resource efficient implementation on low-power devices. In 2016 IEEE 13th international conference on wearable and implantable body sensor networks (BSN) (pp. 71-76). IEEE.
[22] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019, July). Regularized evolution for image classifier architecture search. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, pp. 4780-4789).
[23] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le Q. V., & Kurakin, A. (2017, July). Large-scale evolution of image classifiers. In International Conference on Machine Learning (pp. 2902-2911). PMLR.
[24] Reiss, A., & Stricker, D. (2012, June). Introducing a new benchmarked dataset for activity monitoring. In 2012 16th International Symposium on Wearable Computers (pp. 108-109). IEEE.
[25] Reyes-Ortiz, J. L., Oneto, L., Samà, A., Parra, X., & Anguita, D. (2016). Transition-aware human activity recognition using smartphones. Neurocomputing, 171, 754-767.
[26] Ronao, C. A., & Cho, S. B. (2015, November). Deep convolutional neural networks for human activity recognition with smartphone sensors. In International Conference on Neural Information Processing (pp. 46-53). Springer, Cham.
[27] Suganuma, M., Shirakawa, S., & Nagao, T. (2017, July). A genetic programming approach to designing convolutional neural network architectures. In Proceedings of the genetic and evolutionary computation conference (pp. 497-504).
[28] Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2018). Automatically evolving CNN architectures based on blocks. arXiv preprint arXiv:1810.11875.
[29] Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019). Evolving deep convolutional neural networks for image classification. IEEE Transactions on Evolutionary Computation, 24(2), 394-407.
[30] Wagner, D., Kalischewski, K., Velten, J., & Kummert, A. (2017, September). Activity recognition using inertial sensors and a 2-D convolutional neural network. In 2017 10th International Workshop on Multidimensional (nD) Systems (nDS) (pp. 1-6). IEEE.
[31] Wang, B., Sun, Y., Xue, B., & Zhang, M. (2018, July). Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In 2018 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-8). IEEE.
[32] Wang, Y., Zhang, H., & Zhang, G. (2019). cPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. Swarm and Evolutionary Computation, 49, 114-123.
[33] Xie, L., & Yuille, A. (2017). Genetic CNN. In Proceedings of the IEEE international conference on computer vision (pp. 1379-1388).
[34] Xue, N., Triguero, I., Figueredo, G. P., & Landa-Silva, D. (2019, June). Evolving Deep CNN-LSTMs for Inventory Time Series Prediction. In 2019 IEEE Congress on Evolutionary Computation (CEC) (pp. 1517-1524). IEEE.
[35] Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S. H., & Patton, R. M. (2015, November). Optimizing deep learning hyper-parameters through an evolutionary algorithm. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (pp. 1-5).
[36] Zeng, M., Nguyen, L. T., Yu, B., Mengshoel, O. J., Zhu, J., Wu, P., & Zhang, J. (2014, November). Convolutional neural networks for human activity recognition using mobile sensors. In 6th International Conference on Mobile Computing, Applications and Services (pp. 197-205). IEEE.
[37] Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.