DE GRUYTER OPEN
RIGA TECHNICAL UNIVERSITY
Information Technology and Management Science
ISSN 2255-9094 (online)
ISSN 2255-9086 (print)
December 2017, vol. 20, pp. 40–47
doi: 10.1515/itms-2017-0007
https://www.degruyter.com/view/j/itms

Deep Convolutional Neural Networks: Structure, Feature Extraction and Training

Ivars Namatēvs
Riga Technical University, Latvia

***Abstract* – Deep convolutional neural networks (CNNs) are aimed at processing data that have a known network like topology. They are widely used to recognise objects in images and diagnose patterns in time series data as well as in sensor data classification. The aim of the paper is to present theoretical and practical aspects of deep CNNs in terms of convolution operation, typical layers and basic methods to be used for training and learning. Some practical applications are included for signal and image classification. Finally, the present paper describes the proposed block structure of CNN for classifying crucial features from 3D sensor data.**

***Keywords* – Convolution layers, convolution operation, deep convolutional neural networks, feature extraction.**

I. Introduction

Deep learning has recently given new power that allows building artificial intelligence (AI) systems that were not possible a few years ago.

Today, AI is an explosion technology that solves the tasks which require a huge amount of calculation power executed by computers. On the other hand, there are problems which can be solved by people, like recognising drawing in image, understanding spoken words or considering the direction and obstacles of traffic on roads, based on their intuition and experience. This means that with massively-parallel processing systems and introduction of complex algorithms these solutions can be figured out much faster and with a higher accuracy. The computing infrastructure is based on a hierarchy of perceptions. Each computing layer is characterised in terms of its relation to concepts where the essential layer consists of simple concepts. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach *deep learning* covering several aspects of machine learning [1].

There are two fundamental approaches in the field of AI. The first approach is based on knowledge engineering systems, logic programming and logical reasoning. The second approach covers microscopic biological models [2]. Artificial neural networks (ANNs) and genetic algorithms are the prime examples of this later approach. The field of ANNs was initially configured as an attempt to emulate the way that the brain performs a particular task, by regarding the brain as a highly complex, nonlinear, parallel information processing system [3], [4].

The most notable features of ANNs are: an extensive interconnection grid of simple processing units, adjustment of the grid parameters or weights in order to carry out tasks or adapt itself to its environment through the learning process. In mathematical terms, an ANN can be seen as a direct graph where each node implements a *neuron model* [3].

Deep convolutional neural networks (CNNs) are a specialised kind of ANNs that use convolution in place of general matrix multiplication in at least one of their layers [1]. In contrast to simple neural networks that have one or several hidden layers, CNNs consist of many layers. Such a feature allows them to compactly represent highly nonlinear and varying functions [5]. CNNs involve many connections, and the architecture is typically comprised of different types of layers, including convolution, pooling and fully-connected layers, and realise form of regularisation [6]. In order to learn complicated features and functions that can represent high-level abstractions (e.g., in vision, language, and other AI-level tasks), CNNs would need deep architectures. Deep architectures, and CNNs, consist of a large number of neurons and multiple levels of latent calculations of non-linearity. According to Bengio [7], each level of architecture of CNN represents features at a different level of abstraction defined as a composition of lower-level features.

CNNs have recently shown remarkable success in image recognition [8], [9], sentence and text classification [10], multivariate time series data analysis [11], medicine [12], time series physiological signals [13], electric machine fault diagnosis [14], ultrasonic signal classification [15] and biological image classification [16]. Deep learning techniques have recently been used by many companies, such as *Adobe*, *Apple*, *Baidu*, *Facebook*, *Google*, *IBM*, *Microsoft*, *NEC*, *Netflix*, and *NVIDIA* [17].

The rest of the article is organised as follows. In section II, a short historical overview is given referring to the evolution of convolutional networks. In section III, the definition of convolution operation is provided. Section IV covers the basic principles of CNNs, which are necessary to understand in order to elaborate a deep neural network approach. In section V, the main principles of CNN learning and training are given. In section VI, comprehensible deep CNN system architecture consisting of the proposed convolutional layers for signal data classification is brought out. Conclusions and proposals for future research are formulated in Section VII.

II. Brief History of CNNs

CNNs are biologically inspired by the structure of mammals' visual cortices and the operation of their vision system as presented in David Hubel and Torsten Wiesel's model [18]. Based on their seminal model, the computational deep learning can be recognised as part of computational intelligence

paradigm. The basic idea comes out from their model – using computing infrastructure, algorithms and labelled data for learning can simulate the neocortex's large array of neurons in an artificial "neural network".

Their model helped design understanding of many aspects of brain functions, especially, the primary visual cortex (PVC). Goodfellow et al. [1], [19] outline three properties of PVC to design a convolutional network: first, PVC has a 2D structure and is organised as a spatial map. CNNs capture this property by having features defined in terms of 2D maps. Second, PVC contains many simple cells that are characterised by a linear function of the image in a small, spatially localised receptive field. The detector units of a convolutional network are designed to copy these properties of simple cells. Third, PVC also contains many complex cells and these cells respond to features. Complex cells are immutable to a small change in the position of their feature and remaining unchanged regardless of changes of their conditions of measurement than simple cells. This inspires the pooling of CNNs.

The first as we regard as contemporary convolutional network has been proposed by Fukushima in 1980 and is called Neocognitron [20]. It perhaps was the first ANN that deserved the attribute 'deep' and the first to incorporate the neurophysiological model of PVC. Neocognitron is very similar to the architecture of modern, contest-winning, purely supervised, feedforward, gradient-based deep network learning with alternating convolutional and downsampling layers [21].

In 1986, Rumelhart et al. [22] proposed a backpropagation network to train a neural network with one or two hidden layers. In 1989, LeCun [23] demonstrated techniques, using a hierarchy of shift invariant local feature detectors for image recognition. In 1989, backpropagation was applied [24] to Neocognitron-like, weight-sharing, convolutional neural layers with adaptive connections. In 1991, Robinson and Fallside [25] intended a recurrent neural network (RNN) for speech recognition. In the same year, Bengio et al. [26] suggested for speech recognition multilayer perceptron (MLP). However, this wave of using deep neural networks which started in 1980 as the connectionist approach ended around 1995.

The current deep learning renaissance began in 2006 when Hinto et al. [27] demonstrated that a neural network could outperform the Gaussian or radial basis function (RBF) kernel on the MNIST benchmark. The year 2006 also saw early graphic processing units (GPUs) based CNN implementation up to 4 times faster than computing processing unit (CPU) CNNs [28] and compared earlier GPU implementations of standard feedforward neural networks (FNNs) with a reported speed-up factor of 20 [29]. GPUs or graphics cards, digital signal processing (DSP), field-programmable gate arrays (FPGA) and other silicon architectures have become critical components of computational resources for training and evaluation when executing the idiosyncratic patterns of deep CNNs.

Today, deep CNNs are used for many practical applications due to seminal ideas of Yann LeCun of convolutional networks, Geofrrey Hinton's exploring deep learning methods, Jeff Hawkins' creating the memory prediction framework theory of the brain and the described hierarchical temporal memory (HTM). Maximilian Riesenhuber and Tomas Poggio worked on the hierarchical model of object recognition (HMAX) which was based on Poggio's development of computational model of brain function to build intelligent machines vision that could mimic human performance. Finally, Sepp Hochreiter and Jürgen Schmidhuber worked on recurrent neural networks (RNN) and Long Short-Term Memory (LSTM) RNN.

It is worth mentioning some global conferences to be contributed to CNN development. The most noticeable are:

- Annual Conference on Neuron Information Processing Systems (NIPS);
- The world's biggest and most important GPU developer conference (GPU);
- International Conference on Machine Learning (ICML);
- The conference on Computer Vision and Pattern Recognition (CVPR);
- International Conference on Computer Vision (ICCV).

III. The Convolution Operation

In its most general form, the concept of *convolution* is a mathematical operation of two functions that produces a new function. This new function reflects to which extent the original functions match if their graphs are aligned with each other.

The convolution theorem states that under certain conditions the Fourier transform of a convolution is a point-wise product of Fourier transforms. In other words, convolution in one domain (e.g., time domain) equals point-wise multiplication in the other domain (e.g., frequency domain).

In one dimension, the convolution between two functions is defined as follows:

$$g(x) = f(x) \odot h(x) = \int_{-\infty}^{\infty} f(s)\, h(x-s)\ \mathrm{d}s, \tag{1}$$

where $f(x)$ and $g(x)$ are two functions;

s is a dummy variable of integration (takes values 0 or 1).

In two dimensions, the convolution between two functions is defined as follows:

$$g(x,y) = f(x,y)\ \odot h(x,y) = \\ = \iint_{-\infty}^{\infty} f(s,t)\, h(x-s, y-t)\ \mathrm{d}s\ \mathrm{d}t. \tag{2}$$

The convolution operation is typically denoted with an asterisk * and might not to be confused with multiplication.

For example, in one-dimensional applications we have a signal time domain, $x(t)$ and a frequency domain, $w(a)$, based on the convolutional theorem, the convolution operation is [1]:

$$s(t) = (x * w)(t), \tag{3}$$

where x, the first argument is referred to as the input;

w, the second argument is referred to as the kernel;

$s(t)$ output is referred to as the *feature map* or *kernel map.*

In computer applications, the time series data will be discretized and the time index t can then take only integer

values. Thus, the *discrete convolution* can be defined as follows:

$$s(t) = (x * w)(t) = \sum_{-\infty}^{\infty} x(a)w(t-a). \quad (4)$$

In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm [1]. These multidimensional arrays are referred as *tensors*.

If there is a two dimensional space, for example, image I as an input, the two-dimensional kernel K has to be used. The convolution for two dimensions is as follows:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \quad (5)$$

If we assume that there are fewer variations in the range of valid values of m compared to n based on assumption that convolution is commutative, we can equivalently write (5) as follows:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \quad (6)$$

In case m increases, the index into the input increases, but the index into the kernel decreases, it means we have flipped the kernel relative to the input. If the kernel is not flipped we use the related function called the *cross-correlation*:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \quad (7)$$

In the context of machine learning, the algorithm will learn the appropriate values of the kernel in the appropriate place [1]. In machine learning convolution is not used alone but simultaneously with the combination of other functions. Based on the principles of convolution CNNs operates.

IV. Convolutional Neural Networks

From the above-mentioned considerations, we can clearly recognise that the topology of CNN comparing with other traditional neural networks is different. The latter uses a matrix product **AB** that is produced by multiplying two matrices n x m, where n is a matrix of parameters and m is a parameter describing the interaction between each input unit and each output unit [1].

The main benefit of using CNNs with respect to traditional fully-connected neural networks is the reduced number of parameters to be learned [30]. The CNN topology is based on three main concepts, namely: local receptive fields, shared weights and spatial or temporal sampling [31]. It means that CNNs are typically comprised of different types of layers called convolutional layers, whereas each convolutional layer is made of small kernels that allow extracting high-level features in an effective way. The last convolutional layer is fed to fully-connected layers. As it has been stated before, if CNNs are the reduced number of parameters to be learned they caused to have much fewer connections and easier to train [8]. Consequently, this particular kind of neural networks assumes that we wish to learn *filters* in a data-driven fashion, as a means to extract features describing the inputs [6].

The standard model of CNN has a structure consisting of the input layer, alternating convolutional layers, pooling or subsampling layers and non-linear layers. The latter consists of a small number of fully-connected layers, but the final layer is often a sofmax classifier [32]. Accordingly with a complex layer terminology, one convolutional net or convolutional layer is composed of convolutional stage (e.g., affine transformation), detector stage (e.g., rectified linear), and pooling stage [1]. This means that each convolutional layer has more than one stage. As a result, each stage of the convolutional layer can be set apart and every step of processing of it can be ruled in its own rights. Typically, convolutional layers are interspersed with sub-sampling layers to reduce computational time and gradually build up further *spatial* and *configural* invariance [7]. The basic layers of a CNN are listed below.

Input layer. The input is usually a multidimensional array of data where data are fed to the network [6]. Input data can be, i.e., image pixels or their transformation, patterns, time series or video signals.

Convolutional layers or convolutional stage. It is the main building block of CNN. The prime purpose of convolution is to extract distinct features from the input. Krig [33] outlines that these layers are comprised of a series of filters or learnable kernels which aim at extracting local features from the input, and each kernel is used to calculate a feature map or kernel map.

The first convolutional layer extracts low-level meaningful features such as edges, corners, textures and lines. Next convolutional layer extracts higher-level features, but the highest-level features are extracted in the last convolution layer [34]. Kernel size refers to the size of the filter, which convolves around the feature map while the amount by which the filter slides (sliding process) is the stride. It controls how the filter convolves around the feature map. Therefore, the filter convolves around the different layers of input feature map by sliding one unit each time [1].

Another essential feature of CNNs is *padding* that gives option to make input data wider with, e.g., elements $\mathbf{V}_{i,j,k}$. For example, if there is a need to control the size of the output and the kernel width W independently, the zero padding of input is used.

Non-linear layers or detector stage. The detector stage is used to detect each linear activation through nonlinear activation function. In other words, linear activation introduces the non-linearity into neural networks and allows learning more complex models [11].

There are several nonlinear activation functions. The standard way to model a neuron's output f as a function of its input x is with $f(x) = tanh(x)$, $sigmoid(x)$ or Rectified Linear Unit (ReLU) [32]. The last one is preferable because it makes training several times faster than its equivalents. Some authors adopt $sigmoid(\cdot)$ function at all activation stages due to its simplicity [11]. ReLU applies the function $y = \max(x, 0)$. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of

the convolutional layer. According to [8], using ReLU it is possible to speed up the training of CNNs by keeping up the gradient more or less constant at all network layers.

The pooling or downsampling, or subsampling layers. It reduces the resolution of the previous feature maps through compressing features and computational complexity of the network [35]. It adjusts the features robust to noise and disorder. Another purpose of the pooling layer is to make it robust to small variations for previously learned features. As a result, pooling ensures that the network focuses on the most important patterns.

In general, a pooling layer produces downsampled versions of the input map and reduces the dimensionality of the feature maps used by the following layers [6], [7].

Pooling splits the inputs into regions with the size of $R \times R$ to produce one output from each region. If a given input with a size of $W \times W$ is fed to the pooling layer, the output size P is obtained by [36]:

$$P = \left\lfloor \frac{W}{R} \right\rfloor. \tag{8}$$

Pooling can be *max pooling*, *average of a rectangular neighbourhood*, and *pooling by downsampling*.

The max pooling action addresses the maximum output within a rectangular neighbourhood. Max pooling outputs only the maximum number in each kernel, thus reducing the feature map size. Max pooling introduces invariance. For example, we have input feature of size 44 × 44, which is divided into 22 × 22 regions of size 2 × 2, i.e., we apply max pooling of size 2, i.e., output index (2, 2). For maximum pooling, the maximum value of the four values is selected. For average pooling, the average of the four values is selected. The result of averaging is a fraction that has been rounded to the nearest integer.

Each output map may combine convolution with multiple input maps. In general, we can write [37]:

$$\mathbf{x}_j^L = f\left(\sum_{i \in M_j} \mathbf{x}_j^{L-1} * \mathbf{k}_{ij}^L + b_j^L\right), \tag{9}$$

where L – the convolutional layer;
L−1 – the downsampling layer;
$\mathbf{x}^{L-1}$ – input features of $L-1$ convolutional layer;
$\mathbf{k}_{ij}$ – kernel maps of L convolutional layer;
b^L – additive bias of L convolutional layer;
M_j – represents a selection of input maps;
i-th – input;
j-th –output.

In general, the feature extraction using CNNs consists of multiple similar steps and each step is made of three cascading layers: convolution layer, activation layer and pooling function.

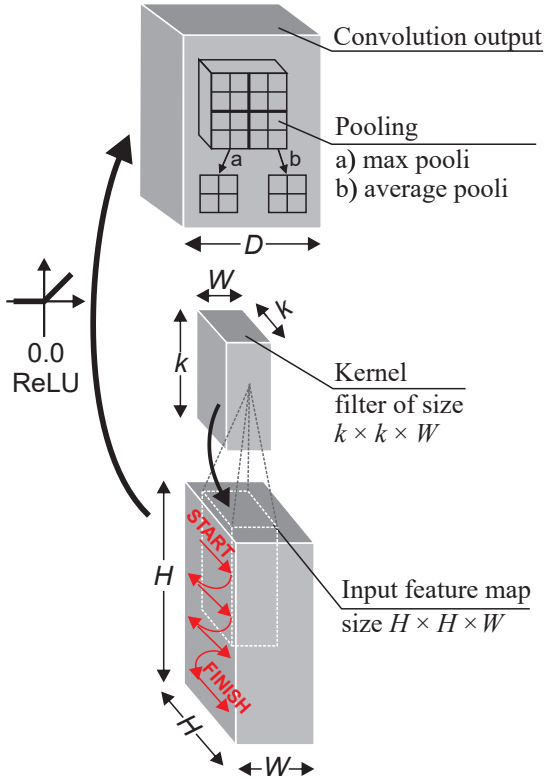
Figure 1 exhibits the process of 3D convolution used in CNNs.

The input of size $H \times H \times W$ is convoluted with p number of kernels, where the kernel size is $k \times k \times W$. One kernel convolving with an input feature map produces one output feature map, and p kernel produces independently p feature maps. Each kernel is moved starting from top-left corner of the input feature map to top-right corner element at a time. Then the kernel shifts one element downward, takes left-side position and moves towards right-side position. This process is finished when the kernel reaches the bottom-right position.

For example, for the case when we have input $H \times H = 44$ and $k \times k = 5$, there are 30 unique positions from the left to the right, and 30 unique positions from the top to the bottom that the kernel can take. Each feature in the convolution output will contain 30 × 30, i.e., $(H - k + 1) \times (H - k + 1) = (44 - 5 + 1) \times (44 - 5 + 1)$ elements. To create one element of one output feature $k \times k \times W$ operations are required.

From the above-mentioned considerations, it can be concluded that a new feature map is typically generated by sliding a filter over the input and computing the dot product (which is similar to the convolution operation), followed by a non-linear activation function to introduce non-linearity into the model [32].

For instance, one convolutional layer consists of the input feature map, the kernel and the convolution output. All units share the same weights (filters) in each feature map.



H – input heigh and width
W – input depth, kernel width
k – kernel high and width
s – kernel stride
n – number of kernels
D – #output feature maps

Fig. 1. The convolution process.

The advantage of sharing weights is the reduced number of parameters and the ability to detect the same feature regardless of its location in the inputs [36]. Max pooling or average pooling is used to convolution output to be an input for the next convolution step.

Fully connected layers. This is the last stage of topology of CNNs consisting of a generic multi-layer network. The last few

layers will be fully connected 1D layers to all activations in the previous layer [7]. From these layers it is possible to extract features to train another classifier. To specify how the network training penalizes the deviation between the predicted and the true labels, various loss functions can be used, e.g., softmax, sigmoid cross-entropy or Euclidean loss [6].

V. Training and Learning CNNs

Training deep architectures is a challenging task, and traditional methods that have proved effective when applied to uncomplicated neural network architectures are not as effective when applied to deep architecture. The training function means to use an overall algorithm that is used to train a neural network to recognise a certain input and map it to a certain output. The most expensive part of CNNs training is learning the features and accessibility to labelled data.

A learning function in deep neural networks requires computing the gradients of complicated functions and decides how those would be manipulated [1]. CNNs are usually trained by *backpropagation* (BP) and *Stochastic Gradient Descent* (SGD) to find weights and biases that minimise certain loss function in order to map the arbitrary inputs to the targeted outputs as closely as possible [3]. BP algorithm refers only to the method for computing the gradient, while SGD algorithm is used to perform learning using this gradient [1]. However, BP technique can make training gradient 10 million times faster relative to naive implementation techniques [22].

There are two central challenges in machine learning: *underfitting* and *overfitting* [38]. Overfitting occurs when the gap between the training error and test error is too large. Underfitting occurs when the model is not able to obtain a sufficiently low error on the training set. In CNNs, there are two primary ways to combat *overfitting*: *dropout* and *data augmentation*. Dropout is a cheap, powerful regularization strategy that can be seen as a process of constructing new inputs by multiplication by noise [8]. It is a method of adaptive reparametrization, motivated by the difficulty of training very deep neural network models. Data augmentation is to artificially enlarge the dataset using label-preserving transformations [12].

Another challenge in machine learning is: *regularisation* and *optimisation*. The optimisation perspective suggests that the weights should be large enough to propagate information successfully, but some regularisation concerns encourage making them smaller. Unless your training set contains tens of millions of examples or more, you should include some mild form of regularisation from the start [1].

According to [7], the parameter update includes: *feedforward pass*, *backpropagation pass* and *the gradient applied.* The aim of feedforward pass is to determine the predicted output CNN on input vector. Specifically, it computes feature maps from layer to layer and stage from stage until obtaining the output. The backpropagation pass means that backpropagation starts at the last layer of a neural network, recursively applies the chain rule to compute the gradients and go backwards to the inputs of a neural network.

Finally, to obtain optimal performance of CNNs we can tune three options: *regularisation*, *momentum* and *leaning rate*. Regularisation is the function to prevent overfitting of the data. Regularisation can be improved by adjusting a weight decay coefficient or by adding a regularisation strategy such as drop out or data augmentation [12]. Momentum is the function to control how fast or slow the network learn during training. Learning rate is the function to help in the convergence of the data.

VI. Architecture

In this section, some architecture and methodologies are considered for image and signal detection.

The architecture of CNN contains many (several) layers. Krizhevsky et al. [8] offer CNN that contains eight learned layers with weights. The first five are convolutional and the remaining three are fully-connected layers [39]. For example, the first convolutional layer consists of $224 \times 224 \times 3$ pixels input map and is convolved with 96 kernels of the size $11 \times 11 \times 3$, where the stride is 4 pixels. The second convolutional layer takes as input (response normalized and pooled) the output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The fully-connected layer has 4096 neurons. The training of the model has been done using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and the weight decay of 0.0005. An equal learning rate was used for all layers, which was adjusted manually through training. The learning rate was initialised at 0.01 and reduced three times prior to termination. NVIDIA GTX 580 3GB GPUs were used for training the network.

Archarya et al. [12] report using CNN for automated detection of myocardial infarction using the electrocardiogram (ECG) in the diagnosis of myocardial infarction. In pre-processing process, all ECG signals are segmented using the R-peak detection using Pan–Tompkins algorithm. To eliminate the outweigh before feeding the ECG segments and to address the problem of amplitude scaling into deep CNN for training and testing each segment should be normalized with z score [40]. The architecture of the proposed CNN consists of 11 layers where the last three are fully-connected layers. For example [41], the input layer (layer 0) is convolved with a kernel size 102 to form the first layer (layer 1). After which, a max pooling of size 2 is applied to every feature map. After performing the max pooling operation, the number of neurons reduces from 550×3 to 275×3. Finally, layer 10 is connected to the last layer with 2 output neurons. The training has been executed by using standard backpropagation with a batch size of 10 [40]. The regularisation, momentum and learning rate parameters are set to 0.2, 3×10^{-4}, and 0.7 respectively. A proportion of all ECG data has been 70 % for training, 20 % for validation and 10 % for testing of CNN. The final layer of the fully-connected network is a softmax layer with an output of X dimensional vector where X is the number of classes that we desire to have.

Ferreira and Giraldi [6] apply the application of CNN to granite image classification using learning discriminate features, instead of relying on feature engineering. Each image is converted to R, G and B colour channels and also converted to grey-level. These are used as input to the network. After the patching the images have been divided into tiles of interest, i.e., 28 × 28 images and 32 × 32 colour images. In the next step, different CNNs were used to recognise patterns of image.

The number of networks used was 4 and their architecture was as follows: *MNIST1* network consisted of one input layer, four convolutional layers, two pooling layers, one ReLU layer and one fully-connected layer. *MNIST2* network consisted of one input layer, five convolutional layers, three pooling layers, one ReLU layer and a final fully-connected layer. *MNIST3* network consists of one input layer, six convolutional layers, four pooling layers, one ReLU layer and a final fully-connected layer. *CIFAR* network consists of one input layer, five convolutional layers, three pooling layers, four ReLU layers and one fully-connected layer. These networks generated feature vectors with 500, 256, 256 and 64 dimensions, respectively. Finally, the image patch classification was done by choosing the 1st Nearest Neighbour (1NN) classifier, the tiny image blocks from input images.

The MNIST based networks were trained using a learning rate fixed on 0.001, SGD with momentum to 0.9 and weight decay to 0.0005 without dropout. The *CIFAR* was trained using SDG with momentum equal to 0.9 and the weight decay to 0.0001. Experiments were performed on a cluster node with 11 processors and 131 GB RAM and graphics card NVIDIA GeForce 210.

Zheng et al. [11] modify CNN and apply it to multivariate time series classification task (the input of multivariate time series classification is multiple 1D subsequences, but not 2D image pixels) separating multivariate time series into univariate ones. The feature learning is individually carried out on each univariate time series. The architecture consist of three input channels, two filter layers, two pooling layers, and two fully-connected layers. To update the parameters, the stochastic gradient descend method has been used instead. The reason for that was that SGD could converge faster for large scale data instead of full-batch version.

Wang et al. [13] focused on evaluating the efficacy of using CNN to construct a model of physiological signal anomaly detection and tested algorithm on eight physiological signals. The DEAP dataset, a dataset for emotion analysis using EEG, physiological, and video signals was used [42]. CNN transforms the raw unlabelled time series signals into a reduced set of features. Before the signals enter the model, the time series physiological signals were normalized. The deep CNN architecture contains two convolutional layers, two pooling layers and a multivariate Gaussian anomaly detection model.

VII. Implementation of CNN for 3D Sensor Data

At a high level, the block diagram of the proposed algorithm consists of data from sensors, pre-processing (normalization, filtering, linearization), CNN (feature extraction), classification (pattern recognition), results (classification). The algorithm performs the following steps to learn features:

1) Classifying sensor signals as data array into number of segments as labelled training data.
2) Filtering and normalizing segments of the raw sensor data (pre-processing step).
3) Extracting features using CNN.
4) Detecting patterns and creating classification.

Data are often corrupted by interfering noise. In order to decrease that noise we use reduction, label, scaling and normalization of the data in the pre-processing stage. In our case, we normalise the sensor data by subtracting the mean sensor data value and dividing it by the standard deviation.

For the feature extraction we use CNN. We take the sensor data as a second-order tensor matrix. The sensor data has c channels. Convolution operation is represented by $C(*)$ throughout the network.

The convolution layer_1 (CL_1). The labelled input sensor data are convoluted with K_n kernel maps with dimension of, $k_1 \times k_1 \times W_1$, respectively, one kernel map at a time, with kernel stride s_1 and zero padding. A bias b is added to each convolution operation between sensor data and a kernel map. The scored result then goes through the non-linear activation function ReLU_1 to generate CL_1. The dimensions of the resulting CL_1 are $[(W/s_1) \times (H/s_1) \times K_1]$. The number of kernel maps represents the depth, d_n of the convolution operation or number of extracted features that the network will extract after the convolution.

The convolution layer_2 (CL_2). The operation is exactly similar to the previous measure, if the input labelled sensor data are replaced by the CL_1 instead. The CL_1 is convoluted with kernel maps, k_2, with the kernel stride s_2 and the same zero padding as it has been for the first convolutional layer. A bias is added to each convolution operation. The result goes through the non-linear activation function ReLU_2 to generate the CL_2. The dimensions of the resulting CL_2 are $[(W/(s_1 \times s_2)) \times (H \times (s_1 \times s_2)) \times K_2]$. We must take into consideration that the total depth of convoluted layer is K_2 at this point, which is the number of extracted features that have from the original sensor data so far. Generally, convolution process goes through five convolutional layers because typically CNNs to be deep use 5 to 25 distinct layers of pattern recognition.

Finally, the CL_5 is fully connected to a hidden layer of r number of weight arrays or neurons. The dimension of each weight array is equal to $[W/(s_1 \times s_2 \times s_3 \times s_4 \times s_5) \times H/(s_1 \times s_2 \times s_3 \times s_4 \times s_5) \times K_5]$.

A bias is added to each product of CL_5 with a weight array, which then goes through the activation function ReLU_6. A softmax layer with m classifiers is used to yield the resulting final output.

VIII. Discussion

The aim of this study has been to show the basic theoretical concepts and applicability of CNNs for construction of the deep neural networks.

The study of the theoretical assumptions and some practical application of CNNs has shown that the main benefit of using CNNs with respect to standard fully-connected neural networks is the reduced number of parameters to be learned. Decreasing the number of the parameters leads to less noise during the training process. The reason is that the number of parameters depends on the kernel width. The wider the kernel width, the lager the number of parameters in the model.

On the other hand, CNNs normally need thousands or even millions of labelled data. The model parameters become larger if the weight decay parameters are decreased. Dropout rate should be decreased to avoid an increase in the number of iterations to converge.

Learning rate should be tuned optimally. Very high or very low rate will lead to optimisation problems and the decrease in effective capacity of the network. Increasing the number of hidden units increases the representation capacity of the model. Using zero padding before convolution keeps the representation size large.

Finally, it can be considered that the proposed algorithm diagram should be adapted using real sensor data as performance modelling and identification of classes.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning (Adaptive Competition and Machine Learning)*. The MIT Press, p. 779, 2016.

[2] T. Munakata, *Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More*, 2nd Edition. Springer-Verlag, London. p. 225, 2008.

[3] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From Architectures to Learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, Jan. 2008. https://doi.org/10.1007/s12065-007-0002-4

[4] A. Prieto, M. Atencia, and F. Sandoval, "Advances in Artificial Neural Networks and Machine Learning," *Neurocomputing*, vol. 121, pp. 1–4, Dec. 2013. https://doi.org/10.1016/j.neucom.2013.01.008

[5] M. Dalto, "Deep Neural Networks for Time Series Prediction with Application in Ultra-Short-Term Wind Forecasting," *IEEE*, pp. 1657–1663, 2015.

[6] A. Ferreira and G. Giraldi, "Convolutional Neural Network Approaches to Granite Tiles Classification," *Expert Systems with Applications*, vol. 84, pp. 1–11, Oct. 2017. https://doi.org/10.1016/j.eswa.2017.04.053

[7] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009. https://doi.org/10.1561/2200000006

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *In proceedings of Neural Networks (NIPS)*, Nevada, USA, pp. 1106–1114, 2012.

[9] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," *Published as a conference paper at ICLR*, Cornel University Library, 2015.

[10] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very Deep Convolutional Networks for Text Classification," *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, vol. 1, Long Papers, 2017. https://doi.org/10.18653/v1/e17-1104

[11] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks," *Lecture Notes in Computer Science*, pp. 298–310, 2014. https://doi.org/10.1007/978-3-319-08010-9_33

[12] U. R. Acharya, H. Fujita, S. L. Oh, Y. Hagiwara, J. H. Tan, and M. Adam, "Application of Deep Convolutional Neural Network for Automated Detection of Myocardial Infarction Using ECG Signals," *Information Sciences*, vol. 415–416, pp. 190–198, Nov. 2017. https://doi.org/10.1016/j.ins.2017.06.027

[13] K. Wang, Y. Zhao, Q. Xiong, M. Fan, G. Sun, L. Ma, and T. Liu, "Research on Healthy Anomaly Detection Model Based on Deep Learning from Multiple Time-Series Physiological Signals," *Scientific Programming*, vol. 2016, pp. 1–9, 2016. http://dx.doi.org/10.1155/2016/5642856

[14] R. Liu, G. Meng, B. Yang, C. Sun, and X. Chen, "Dislocated Time Series Convolutional Neural Architecture: An Intelligent Fault Diagnosis Approach for Electric Machine," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1310–1320, Jun. 2017. https://doi.org/10.1109/tii.2016.2645238

[15] M. Meng, Y. J. Chua, E. Wouterson, and C. P. K. Ong, "Ultrasonic Signal Classification and Imaging System for Composite Materials via Deep Convolutional Neural Networks," *Neurocomputing*, vol. 257, pp. 128–135, Sep. 2017. https://doi.org/10.1016/j.neucom.2016.11.066

[16] C. Affonso, A. L. D. Rossi, F. H. A. Vieira, and A. C. P. de L. F. de Carvalho, "Deep Learning for Biological Image Classification," *Expert Systems with Applications*, vol. 85, pp. 114–122, Nov. 2017. https://doi.org/10.1016/j.eswa.2017.05.039

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrana, and T. Darell, "Caffe: Convolutional Architecture for Fast Feature Embedding", Cornel University Library, Jun. 2014.

[18] D. H. Hubel and T. N. Wiesel, "Receptive Fields and Functional Architecture of Monkey Striate Cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, Mar. 1968. https://doi.org/10.1113/jphysiol.1968.sp008455

[19] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," S. Dasgupta and D. McAllester, eds., *ICML'13*, pp. 1319–1327, 2013.

[20] K. Fukushima, "Neocognitron. "A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[21] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015. https://doi.org/10.1016/j.neunet.2014.09.003

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986. https://doi.org/10.1038/323533a0

[23] Y. LeCun, "Generalization and Network Design Strategies", *Technical Report*, University of Toronto, 1989.

[24] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989. https://doi.org/10.1162/neco.1989.1.4.541

[25] T. Robinson and F. Fallside, "A Recurrent Error Propagation Network Speech Recognition System," *Computer Speech & Language*, vol. 5, no. 3, pp. 259–274, Jul. 1991. https://doi.org/10.1016/0885-2308(91)90010-n

[26] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Phonetically Motivated Acoustic Parameters for Continuous Speech Recognition Using Artificial Neural Networks," *Speech Communication*, vol. 11, no. 2–3, pp. 261–271, Jun. 1992. https://doi.org/10.1016/0167-6393(92)90020-8

[27] G. E. Hinton, "To Recognize Shapes, First Learn to Generate Images," *Technical Report UTML TR 2006-003*, University of Toronto, 2006.

[28] K. Chellapilla, S. Puri, and P. Simard, "High Performance Convolutional Neural Networks for Document Processing," *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), Université de Rennes 1, Suvisoft, 2006.

[29] K.-S. Oh and K. Jung, "GPU Implementation of Neural Networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, Jun. 2004. https://doi.org/10.1016/j.patcog.2004.01.013

[30] S.-H. Zhong, J. Wu, Y. Zhu, P. Liu, J. Jiang, and Y. Liu, "Visual Orientation Inhomogeneity Based Convolutional Neural Networks," *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov. 2016. https://doi.org/10.1109/ictai.2016.0079

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. https://doi.org/10.1109/5.726791

[32] S. Albelwi and A. Mahmood, "A Framework for Designing the Architectures of Deep Convolutional Neural Networks," *Entropy*, vol. 19, no. 6, p. 242, May 2017. https://doi.org/10.3390/e19060242

[33] S. Krig, *Computer Vision Metrics. Survey, Taxonomy and Analysis of Computer Vision, Visual Neuroscience, and Deep Learning*. Springer, p. 637, 2016. https://doi.org/10.1007/978-3-319-33762-3

[34] J. S. Ren, W. Wang, J. Wang, and S. Liao, "An Unsupervised Feature Learning Approach to Improve Automatic Incident Detection," *2012 15th International IEEE Conference on Intelligent Transportation Systems*, Sep. 2012. https://doi.org/10.1109/itsc.2012.6338621

[35] C. Affonso, A. D. Rossi, F. H. A. Vieira, and A. C. P. de L. F. de Carvalho, "Deep Learning for Biological Image Classification", *Expert Systems with Applications*, vol. 85, pp. 114–122, 2017.

[36] T. Chen, R. Y. He, and X. Wang, "A Gloss Composition and Context Clustering Based Distributed Word Sense Representation Model," *Entropy*, vol. 17, no. 9, pp. 6007–6024, Aug. 2015. https://doi.org/10.3390/e17096007

[37] J. Bouvrie, *Notes on Convolutional Neural Networks*, Nov. 2006 [Online]. Available: http://cogprints.org/5869/1/cnn_tutorial.pdf

[38] I. Song, H.-J. Kim, and P. B. Jeon, "Deep Learning for Real-Time Robust Facial Expression Recognition on a Smartphone," *2014 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2014. https://doi.org/10.1109/icce.2014.6776135

[39] H. Tabia and H. Laga, "Learning Shape Retrieval from Different Modalities," *Neurocomputing*, vol. 253, pp. 24–33, Aug. 2017. https://doi.org/10.1016/j.neucom.2017.01.101

[40] U. R. Acharya, H. Fujita, O. S. Lih, Y. Hagiwara, J. H. Tan, and M. Adam, "Automated Detection of Arrhythmias Using Different Intervals of Tachycardia ECG Segments with Convolutional Neural Network," *Information Sciences*, vol. 405, pp. 81–90, Sep. 2017. https://doi.org/10.1016/j.ins.2017.04.012

[41] U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, and H. Adeli, "Deep Convolutional Neural Network for the Automated Detection and Diagnosis of Seizure Using EEG Signals," *Computers in Biology and Medicine*, Sep. 2017. https://doi.org/10.1016/j.compbiomed.2017.09.017

[42] S. Guzel Aydin, T. Kaya, and H. Guler, "Wavelet-Based Study of Valence–Arousal Model of Emotions on EEG Signals with LabVIEW," *Brain Informatics*, vol. 3, no. 2, pp. 109–117, Jan. 2016. https://doi.org/10.1007/s40708-016-0031-9

Ivars Namatēvs holds *Mg. sc. ing.* from Riga Technical University and MBA degree from Riga Business School. His research interests include deep artificial intelligence, especially deep convolutional networks and data mining methods and their application, as well as genetic algorithms. The most important publications: I. Namatēvs. "Concept Analysis of Complex Adaptive Systems," *International Scientific Forum: Proceedings of XVI International Scientific Conference: Towards Smart, Sustainable and Inclusive Europe: Challenges for Future Development*. Riga, Latvia, 28–30 May 2015. Namatēvs, I., Aleksejeva, L., Poļaka, I. "Neural Network Modelling for Sports Performance Classification as a Complex Socio-Technical System," *Information Technology and Management Science,* vol. 19. pp. 45–52. 2016. Available from doi: 10.1515/itms-2016-0010. Namatēvs, I., "Exploring Model-Driven Domain Analysis for Software Engineering," *Proceedings of XVI International Scientific Conference.* Turība University, Riga, Latvia, 18 May 2017. Namatēvs, I., Aleksejeva, L. "Decision Algorithm for Heuristic Donor-Recipient Matching," *Mendel Soft Computing Journal*, vol. 23, No. 1, pp. 33–40, June 2017. ISSN:1803-3814.
E-mail: ivars@turiba.lv