Neural Computing and Applications (2021) 33:1453–1465
https://doi.org/10.1007/s00521-020-05008-0

ORIGINAL ARTICLE

Deep network in network

Hmidi Alaeddine[1] · Malek Jihene[1,2]

Received: 10 January 2020 / Accepted: 2 May 2020 / Published online: 24 May 2020

Abstract
The different CNN models use many layers that typically include a stack of linear convolution layers combined with pooling and normalization layers to extract the characteristics of the images. Unlike these models, and instead of using a linear filter for convolution, the network in network (NiN) model uses a multilayer perception (MLP), a nonlinear function, to replace the linear filter. This article presents a new deep network in network (DNIN) model based on the NiN structure, NiN drag a universal approximator, (MLP) with rectified linear unit (ReLU) to improve classification performance. The use of MLP leads to an increase in the density of the connection. This makes learning more difficult and time learning slower. In this article, instead of ReLU, we use the linear exponential unit (eLU) to solve the vanishing gradient problem that can occur when using ReLU and to speed up the learning process. In addition, a reduction in the convolution filters size by increasing the depth is used in order to reduce the number of parameters. Finally, a batch normalization layer is applied to reduce the saturation of the eLUs and the dropout layer is applied to avoid overfitting. The experimental results on the CIFAR-10 database show that the DNIN can reduce the complexity of implementation due to the reduction in the adjustable parameters. Also the reduction in the filters size shows an improvement in the recognition accuracy of the model.

Keywords Exponential linear unit (ELU) · Convolutional neural networks (CNNs) · Deep MLPconv · Image recognition · Network in Network (NiN)

1 Introduction

Convolutional neural networks (CNNs) have had great success in image recognition [1–11] and object detection [12–15]. They are a key element in the field of computer vision. Unlike traditional artificial neural networks (ANN), CNNs can directly process three-dimensional input data. They are organized in successive computation layers alternating between convolution and pooling [16–21] other types of deep neural networks (DNN) [22, 23], and CNNs are easy to form with back-propagation [24] because they have very clear connectivity in each convolutional layer [25]. The main parameters of CNNs are the weights of the linear convolution filters. To reduce the number of parameters, we suggest a strategy to diminish the linear convolution filters size and adopt the network depth. Reducing the convolution filters size has proved effective in image classification models, because it reduces the computation and the number of parameters used in the convolution layer operations while increasing the efficiency of the representation. Added to that, the use of multilayer perception [13] as a nonlinear function instead of using a linear convolution filter can improve the calculation layers as in NIN [5] and aims to increase the nonlinearity of the local patches in order to allow the abstraction of larger amounts of information within the receptive fields. Compared to CNN, the dense connection characteristic of MLP makes it possible to extract the local characteristics in the spatial domain and not to be able to extract them in the channel domain. This limits the performance of networks based on MLPs such as [5].

✉ Hmidi Alaeddine
alaeddine.hmidi@fsm.rnu.tn

Malek Jihene
jihene.malek@issatso.u-sousse.tn

[1] Faculty of Sciences of Monastir, Laboratory of Electronics and Microelectronics, LR99ES30, Monastir University, 5000 Monastir, Tunisia

[2] Higher Institute of Applied Sciences and Technology of Sousse, Sousse University, 4000 Sousse, Tunisia

Furthermore, MLP, in which a ReLU [26] is used as an activation function, allows the abstraction of information more representative of latent concepts. MLP is a fully connected network that causes an increase in the density of the connection. This causes a vanishing gradient problem and makes learning more difficult and the learning time slower. Different works [1–11] exploit the ReLU [26] to prevent the vanishing gradients problem since it activates above 0 and its partial derivative is 1. However, it has a potential drawback where the constant 0 will block the gradient across the inactivated ReLU [26] units, so that some units may never activate. Several proposed works have attempted to solve these challenges such as [27–30]. Reducing the size of the convolution filters and using eLU [29] may represent a solution to the challenges of NiN [5]. The choice of eLU [29] to replace ReLU [26] is due to its capacity to attenuate the vanishing gradients problem via the identity of positive values and improves the learning characteristics compared to other activation functions. ReLU [26] and eLUs [29] have negative values, allowing them to push medium unit activations closer to zero. Activations close to zero have a gradient similar to the natural gradient since the shape of the function is smooth, thus activating learning faster than the ReLU [26] where the neuron is disactivated. The eLUs [29] saturate in a negative value with smaller inputs and thus decrease the variation and the propagated information. They have negative values which make it possible to bring the average unit activation closer to zero, to reduce the calculation complexity, and thus improving the learning speed.

In this paper, based on [5], we propose to use an eLU [29] instead of ReLU [26] to accelerate the learning speed and to attenuate the vanishing gradients problem. In addition, we approach another important aspect of architecture, convolution/depth, where we increase the depth of NiN [5] by adding more convolutional layers, which is possible, thanks to the use of very small size convolution filters (3×3, 1, 1) instead of convolutional layers of size (5×5, 1, 2) in all the convolutional layers of [5]. This allows increasing the nonlinearity and reducing the number of parameters, hence the calculation complexity. Consequently, we obtain a more precise and deeper architecture NiN [5]. The resulting model is "Deep Network In Network" (DNIN).

The rest of this paper is organized as follows: In Sect. 2, a survey of related works is given. Section 3 bears on the strategy. Experimental evaluations and a comparative analysis are presented and discussed in Sect. 4. Section 5 is dedicated to the implementation details. The advantages and future work of the DNIN are reported in Sect. 6. The work is concluded in the last section.

2 Related works

Several techniques have been used to improve the performance of CNN: Increasing depth [31], increasing the width [19], modification of the convolution parameters [6, 32, 33] or pooling [34–42] change the activation function [27, 28, 43, 44] and reduce the number of parameters and resources. It should be noted that the depth and the width designate the capacity of the CNN, the depth of the network means the number of layers, and the width of the network refers to the number of units of each layer. The number of feature maps (channels) in each layer can represent the width of the CNNs. It is widely recognized that the depth of Deep convolutional neural network (DCNN) is one of the major differences between DCNNs and traditional artificial neural networks [17] [45]. The increase in the depth can make the formation of deep networks more difficult and slower training time and introduces several challenges such as the vanishing gradients problem, the reduction reuse characteristics during the forward propagation. To reduce this problem, several approaches are proposed such as residual block [6], batch normalization [46] and dropout [47].

In [6], He et al. presented a residual learning framework comprising a link around each of the two convolutional layers, adding both the original hijacked data and their results from the convolution function. In [27], the Maxout network provided a solution to the vanishing gradients problem. He tends to over-adjust the training database without using a regularization layer of the model. In [28], a Maxout network in Maxout network (MIM) model more complex than the maxout network [27] is proposed. This MIM model [28] integrates a number k of maxout units that are stacked in a MIM block. A MIM network with $k = 2$ produces the best result. In [30], the authors presented a new formulation, deeply supervised networks (DSN) to minimize classification errors while making the learning process of hidden layers direct and transparent.

Among the first CNNs for image classification, we find AlexNet [1], which won the ILSVRC [48] in 2012, and it is the first architecture to use the rectified linear unit (ReLU) for the activation function, aiming at improving the rate of convergence by attenuating the vanishing gradients problem. In 2013, Zeiler and Fergus introduced a multilayer deconvolutional neuron network [2] that reduced the ImageNet error rate from 16.4 to 11.7%. This reduction in the error rate is due to the modification of the topology of AlexNet [1] by reducing the convolution filters size of the first layer from (11×11, $S = 4$) to (7×7, $S = 2$) and increasing the number of convolutional kernel of the third, fourth and fifth of this convolutional layer. In 2014, several topologies with different numbers of layers ranging from

11 to 19 were proposed in [9]. The VGG-16 [3] has a homogeneous and regular structure of 16 layers. In the VGG-16 [3], the convolution filters (7 × 7 with a stride of 2) are replaced by three convolution filters of sizes (3 × 3, $S = 1$, $P = 1$) in order to increase the number of activation functions and to keep the same resulting dimension by reducing the number of parameters and the calculation complexity. It is noted that the width of VGG 16 [3] starts from 64 and increases two convolution layers by a factor of two. In 2014, GoogLeNet [4] was introduced by Christian Szegedy et al., the main target of this architecture was to reduce the cost of calculation and the number of parameters, while offering a high accuracy through the reduction in the dimension of the channel follows to exploit the 1 × 1 convolution layer before the 3 × 3 and 5 × 5 layers. In [4] and [5], a global average pooling layer is exploited at the last layer, instead of using a fully connected layer that is traditionally used in conventional CNNs to reduce the density of connections.

From these literatures, we consider all these previous approaches and the experiences already realized to propose a new approach. In our case, many attempts have been made to improve the original architecture of NiN [5] in order to obtain better precision where we use smaller convolution and stride sizes instead of the parameters of the convolution layers that are already in use. Another axis of improvement revolves around training and testing the networks in a dense way on the whole image CIFAR-10 and on several scales.

3 The proposed methods

3.1 Network in network (NiN)

Network in network [5] consists of several MLPconv layers which are stacked in a successive manner. Figure 1 illustrates the overall structure of the architecture.

The MLPconv layer consists of a linear convolution layer and a two-layer MLP with a ReLU used as an activation function. The calculation performed by the MLPconv layer is as follows:

$$\begin{aligned} f^{1}_{i,j,k_1} &= \max(\omega^{1T}_{k_1} x_{i,j} + b_{k_1}, 0) \\ &\cdots \\ f^{n}_{i,j,k_n} &= \max(\omega^{nT}_{k_n} f^{n-1}_{i,j} + b_{k_n}, 0) \end{aligned}.$$

Knowing that (i, j) represents the pixel index in the feature map, x_{ij} designates the input patch centered at the location (i, j), k is used to index the channels of the feature map and n denotes the number of layers. In [5], a global average pooling layer is used instead of fully connected layers that are traditionally used in CNNs. This solution generates a feature map for each category. The average of each feature map and the resulting vector will be directly introduced into the softmax layer. The advantage of this technique is that there is no parameter to optimize which avoids over-adjustments at this layer. Figure 2 shows an example of the fully connected layer and the global average pooling layer.

In terms of precision, the NIN [5] obtained an error rate equivalent to 10.41% using the dropout layer [47] and without data augmentation [5]. Using the data augmentation layer (translation and horizontal flipping), the NIN [5] obtained an error rate equivalent to 8.81%.

3.2 The structure of deep network in network

NiN [5] represents a major extension of the CNN. It uses a multilayer perception (MLP) as a nonlinear function, instead of using a linear convolutional filter. However, the MLP consists of fully connected layers (Fc). This limits parameter performance where many MLP parameters need to be calculated and stored. The MLP connection feature prevents the network from retrieving local entities in the channel domain although it can retrieve local entities from the spatial domain. As a solution, we propose different

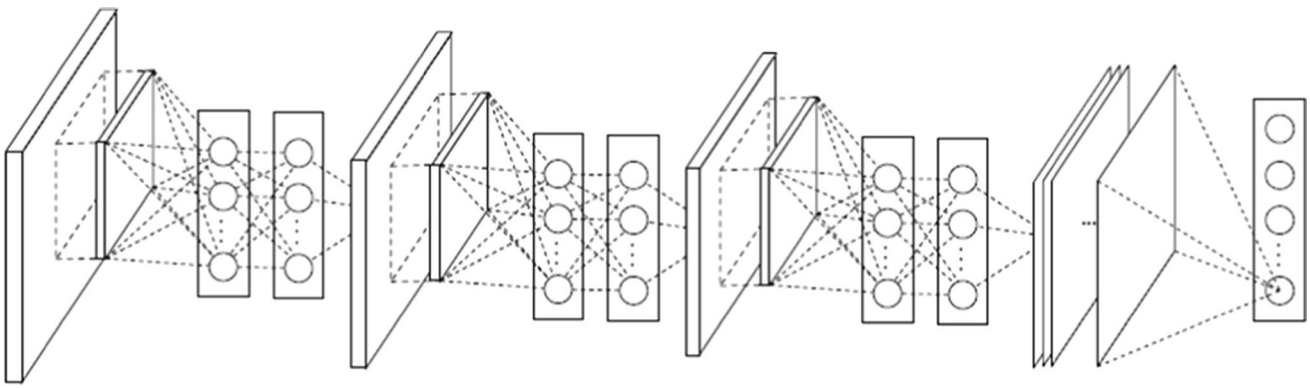

Fig. 1 Overall network in network structure (NIN)

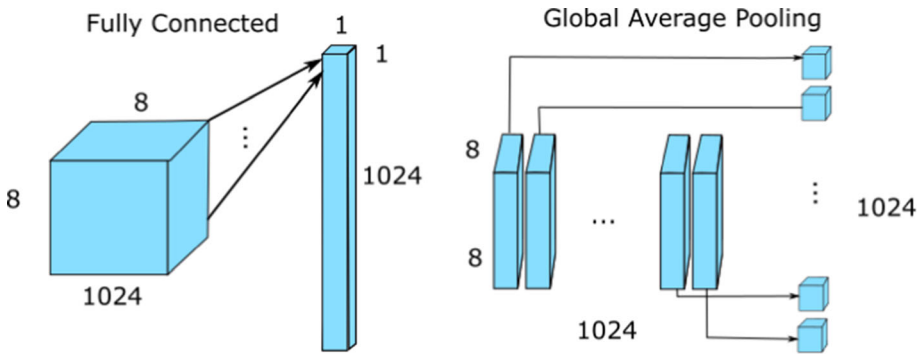

Fig. 2 Example of fully connected layer versus global average pooling layer

configurations of DNiN evaluated in this paper. All these configurations are designed according to the same parameters (height, width and number of convolution kernel). Our overall proposal is to go further in depth by performing an optimization in terms of parameter and accuracy. Since small filters have proven to be very effective in many works, including [1, 3, 6, 8, 9], we do not plan to use filters larger than 3×3. We use two filters with a very small size (3×3, 1, 1) instead of the filter size (5×5, 1, 2), and this type of filter is defined as the smallest size to capture the notion of left/right, up/down, center.

So, what did we gain by using a stack of two 3×3 convolution layers instead of a single 5×5 layer?

- First, we incorporate two nonlinear layers instead of one, which makes the decision function more discriminating.
- Second, we decrease the number of parameters from ($25\ C_{\text{in}}^2$ to $18\ C_{\text{in}}^2$) knowing that the number of input (C_{in}) and output (C_{out}) channels is the same.

Our configurations are captured in a fixed size RGB image equivalent to 32×32. The image is passed through a stack of layers that is constructed with variable and complex structures as shown in Fig. 3.

Another important concept in the design of our architecture is the pooling layer that allows big gains in computational power due to the reduction in the image spatial size. For this, it is common to periodically insert a pooling layer between a stack of successive convolution layers of a CNN architecture to reduce overlearning. The shape of the pooling layer used in [5] is a "max pooling" where the output is the maximum of the values of the input patch. In our architecture:

- First, an attempt was made to avoid the pooling layer in the proposed architecture due to the loss of information, but this involved overlearning and saturation in accuracy.
- Second, we tried to reduce the filter size to 2×2, which is the most common form. For this, we used two layers of size 2×2 with a stride of one.

For example, for a size image 32×32, the resulting image after a layer (3×3, 1) is $\frac{(32-3+1)}{1} = 30$ and after two layers (2×2, 1) is $\frac{((32-2+1)-2+1)}{1} = 30$, but a reduction in performance has been found in terms of accuracy. The model achieves an error rate of 17.8% for the CIFAR-10 dataset. Note that it is possible to use other pooling functions than the maximum, but max pooling layer was more efficient, because it increases more significantly the importance of strong activations. The general structure of "DNiN" is shown in Table 1.

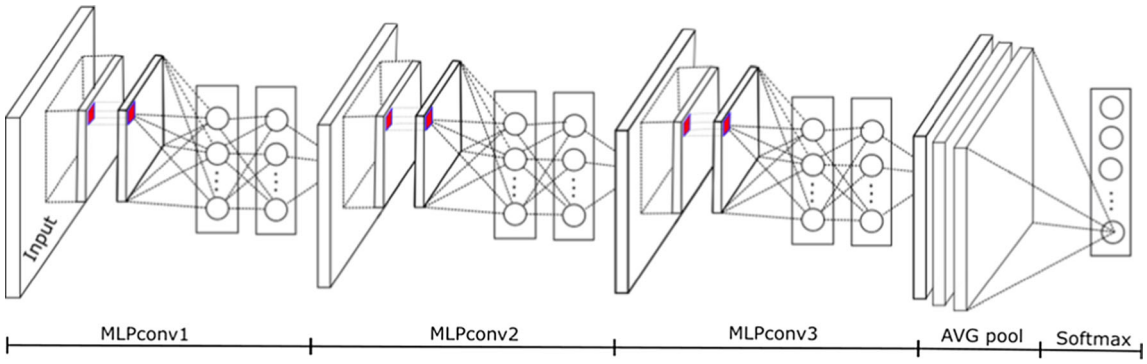

Fig. 3 Deep network in network

Table 1 Structure of networks in networks

Stucture	D Mlpconv 1	M.pool	D Mlpconv 2	M.pool	D MlPconv 3	A.Pool
Output size	32 × 32	16 × 16	16 × 16	8 × 8	8 × 8	1 × 1

3.3 Deep MLPConv

In conventional CNNs, the calculation in the convolution layer is based on the linear filter (Fig. 4a). However, it has been shown that a nonlinear filter is more complex as an MLP filter (Fig. 4b) produces more favorable results than a simple linear filter [5].

In NiN [5], an MLPconv layer is used as a convolution filter (Fig. 4b). This MLPconv layer is considered a normal linear convolution layer followed by a multilayer perception (with a depth of two layers). Compared to the original model [5], the size of the convolution layer and the activation function in the MLPconv layer have been modified. A stack of two convolutional layers of size 3 × 3 is used instead of a single 5 × 5 layer. The stride of these convolution layers is set at one pixel, padding as well. In addition, the eLU [29] is used as the activation function instead of ReLU [26]. The new convolution filter is named *Deep MLPconv*. Figure 4c illustrates the architecture of *Deep MLPconv* layer.

3.3.1 Type of layers and numbers of convolution kernels in Deep MLPconv

Let *Deep MLPConv (X)* be the *Deep MLPconv* layer, where × is a list of the layers used in the structure. For example, MLPconv (3, *R*) denotes the original structure of MLPconv in [5] with a convolution layer of size 3 × 3 and a ReLU [26] unit used as a nonlinear activation functions. Note that the number of MLP layers and nonlinear activation functions are always the same for all *Deep MLPconv* structures. All the configurations of the proposed *Deep MLPconv* layer are equipped with nonlinearity eLU [29]. It is only in the first "configuration (*A*)" that the rectified linear unit [26] is used. For other configurations, we add a new layer each time. Spatial pooling is done on a 3 × 3 window with a stride equivalent to 2. Two layers of max pooling layer are localized after the first two *Deep MLPconv* layers, and an average pooling layer is localized before the layer of softmax layer. It should be noted that the normalization of the local response (LRN) is not used in the various configurations.

- MLPconv (5, R): Original MLPconv layer in [5].
- MLPconv (5, E): MLPConv (5, R) with an exponential linear unit (eLU) instead of a rectified linear unit (ReLU).
- *Deep MLPconv* (3, R): two convolution layers of size 3 × 3 with a rectified linear unit (ReLU).
- *Deep MLPconv* (3, E): two convolution layers of size 3 × 3 with an exponential linear unit (eLU) instead of a rectified linear unit (ReLU).
- *Deep MLPconv* (3, E, BD): Deep MLPconv (3, E) with normalization and regularization layers.
- *Deep MLPconv* (3, E, BD, D): *Deep MLPconv* (3, E, BD) with increased data.

For all *Deep MLPconv* structures, the numbers of convolution kernel and MLPs 1 and MLPs 2 are the same. Table 2 describes the numbers of the kernel.

Figure 5 illustrates the architecture of *Deep MLPconv* (5, E).

3.3.2 Normalization and regularization layers in Deep MLPConv

As MLPs increase the number of parameters, a batch normalization [46] that produces a regularization effect is used, this layer requires a significant increase in data,

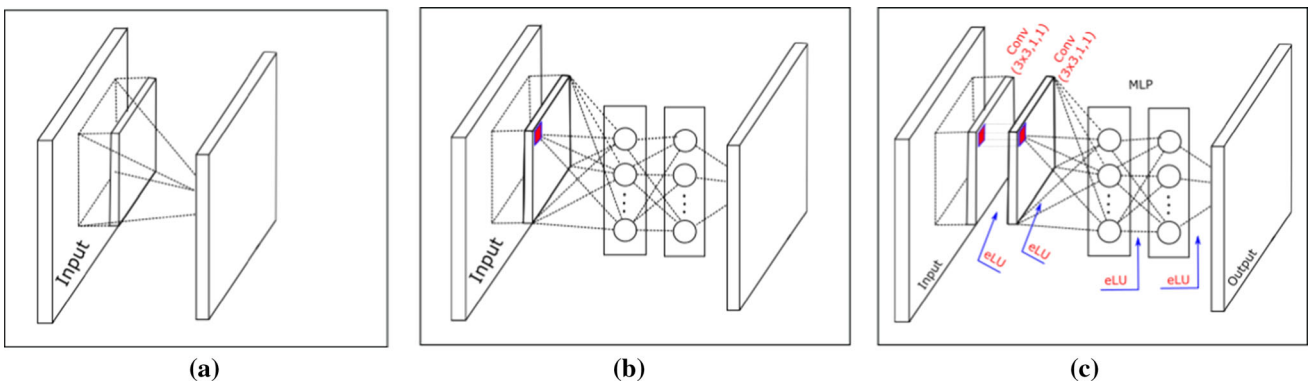

Fig. 4 **a** Linear convolution layer, **b** MLPconv layer, **c** Deep MLPconv layer

Table 2 Numbers of the kernel for Deep MLPConv

Couches	Conv	MLP-1	MLP-2
Nombres	192	160	96

which we would like to avoid, and this is not always possible. We add a Dropout layer [47] between the *Deep MLPconv* layers, "configuration (*D*)," as shown in Fig. 6. This layer produces a regularization effect to prevent the network from over-adjusting as represented.

Dropout [47] proves to be an effective technique for regularizing different networks. It reduces the error rate with almost 2%. This layer is a popular method proposed by Hinton et al. [47] to prevent neural networks from over-adjusting during training. Dropout [47] is adopted by many powerful architectures such as [1, 3, 5, 6, 8]. They eventually improve the generalization by randomly jumping a percentage of their connections. At the time of the test, all the neurons are used, but their outputs are multiplied by the equivalent probability 0.5. This layer introduces the regularization within the network.

4 Experimental results

We evaluate our configurations on a set of reference data: CIFAR-10. The dataset of CIFAR-10 is composed of 60,000 images grouped in 10 classes of images. These 60,000 images are separated between 50,000 total learning images and 10,000 test images. Each image is a 32 × 32 RGB image. The networks used for the databases consist of three *Deep MLPconv* structures. A maximum pooling layer follows the first two *Deep MLPconv* structures of all experiments. In what follows, we will refer to networks by their names (*A*–*E*) and their *Deep MLPconv* (*X*) structures. Note that all proposed configurations are summarized in Table 3.

Based on architecture [5], the "configuration (*A*)" of DNiN is designed by replacing each convolution layer of size 5 × 5 by two convolution layers of size 3 × 3 (which has the same receiving field) and keeping the same activation function. By comparing NiN [5] with DNiN's "configuration (*A*)," we observe that the classification error decreases with 0.82% as we increase nonlinear activation functions. The learning, test and loss curves for "configuration (*A*)" on CIFAR-10 are illustrated in Fig. 7.

4.1 eLU VS ReLU

Compared to the original architecture [5] and "configuration (*B*)," the nonlinear activation has been modified from ReLU [26] to eLU [29]. As it was demonstrated in [29] that the latter trained faster and achieved better results. On an experimental basis, eLUs [29] not only allow for faster learning, but also for better generalization performance than ReLU [26]. We formed our network with the eLU [29] inserted after convolution layers and MPLs. We no longer need to increase the number of training cycles compared to basic networks with ReLU [26] where we can notice that the "configuration (*B*)" exceeds the original NiN [5] in terms of accuracy for 70 K cycle instead of 162 K. The training, test and loss curves for "configuration (*B*)" on CIFAR-10 are shown in Fig. 8.

The accuracy test is 86.31% instead of 85.49%. The execution of the learning requires about 91 h. In "configuration (*C*)," eLU [29] was used instead of ReLU [26], and the modification allowed us to have an increase in accuracy as represented in Fig. 9. Using the same depth, the "configuration (*C*)" generates better classification accuracy than the "configuration (*A*)" but using a higher number of epochs.

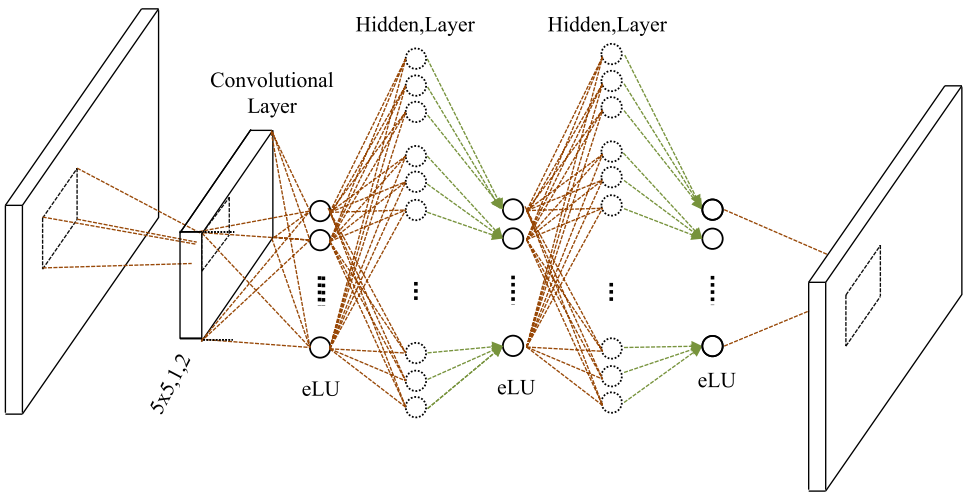

Fig. 5 Deep MLPconv (5, E)

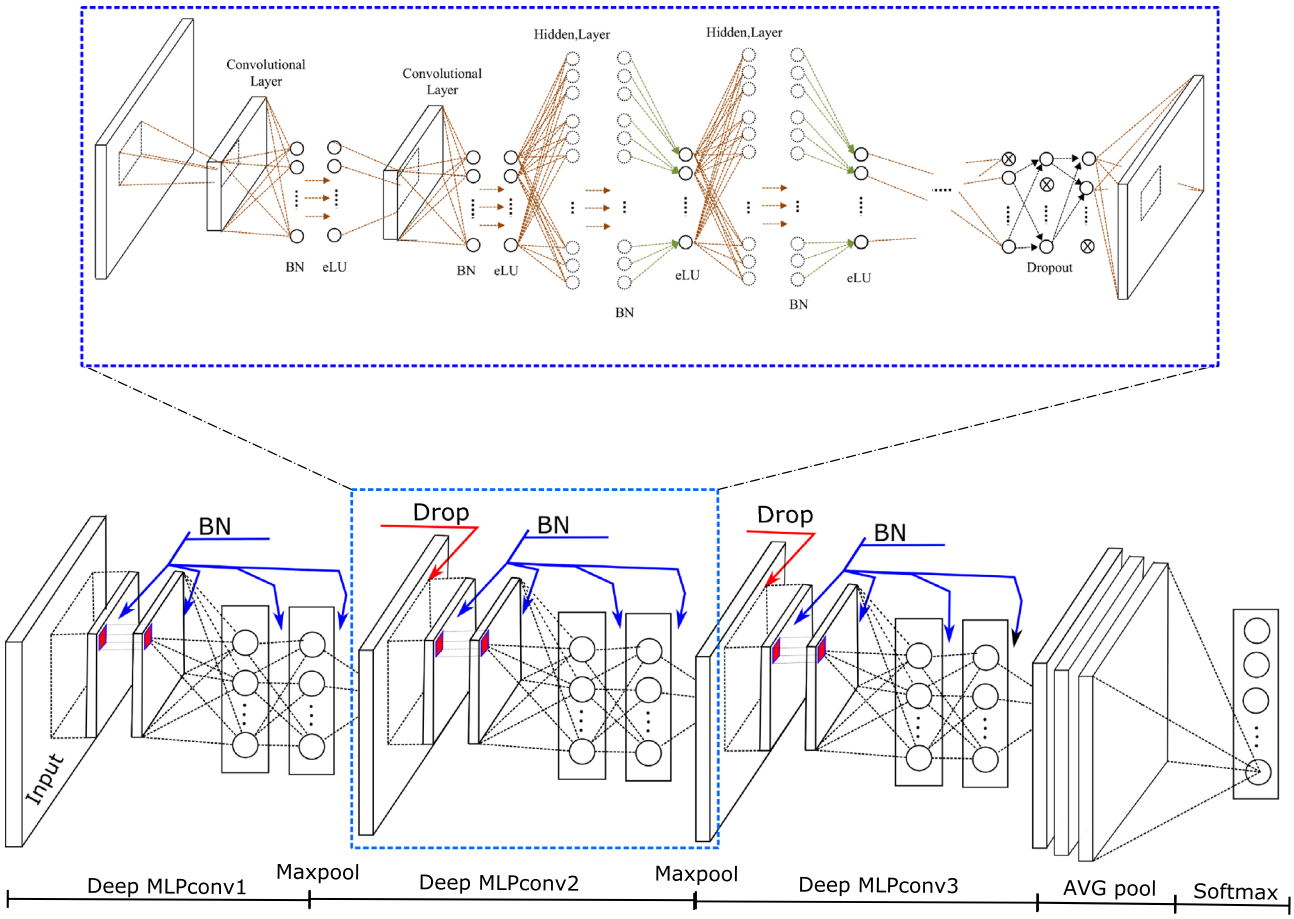

Fig. 6 Deep MLPconv (3, E, BD)

4.2 Normalization and regularization layers in DNiN

Studies show that a network with batch normalization [46] achieves greater accuracy than a network without batch normalization [46]. The exploitation of the batch normalization layer [46], which produces a regularization effect, requires a significant increase in data, which we would like to avoid and in our case is not possible because it plays an important role in increasing accuracy. We add a dropout layer [47] after the first two *Deep MLPConv (3, E, BD)* to prevent DNiN from overlearning. This solution "Configuration (*D*)" also provides improvements over other configurations (*A*, *B*, *C*) that do not use this layer.

4.3 The effect of data augmentation

Data augmentation [49] is a strategy that dramatically increases the diversity of data available for training models, without the need to collect new data, that is, creating modified copies of each instance in a single-instance database. Data augmentation [49] techniques such as cropping, padding and horizontal tilting are commonly used to form large neural networks. Applying this layer to the "configuration (*D*)" increases the accuracy.

The exploitation of the data augmentation layer "configuration (*E*)" leads to significantly better results than the learning on single images of the database. This confirms that increasing the data is a useful effect for reducing the classification test error.

4.4 The performances of different configurations (*A–E*)

In terms of parameters, our architecture exceeds the WRN-(28-10; 16-8; 40-4) [8], ResNet (110, 1202) [6] and ResNeXt-29 (8 × 64d, 16 × 64d) [9] architecture parameters despite their depth and width. For example, the wide models WRN-28-10 [8], and ResNeXt-29-16 × 64d [9], have, respectively, 33.18× and 61.90× more parameters DNiN. A comparison with architectures already completed is shown in Fig. 10.

Table 4 shows a comparison between the accuracy of different configurations with the size of the mini batch 128.

Table 3 Deep NiN configurations

Nom	*A*	*B*	*C*	*D*	*E*
Deep MLPConv (X)	Deep MLPConv (3, R)	Deep MLPConv (5, E)	Deep MLPConv (3, E)	Deep MLPConv (3, E, BD)	Deep MLPConv (3,E,BD,D)
Conv-1-1	3 × 3 × 192/st. 1/pad 1 (ReLU)	5 × 5 × 192/st. 1/pad 2 (eLU)	3 × 3 × 192/st. 1/pad 1 (eLU)	3 × 3 × 192/st. 1/pad 1 eLU/BN	
Conv-1-2	3 × 3 × 192/st. 1/pad 1 (ReLU)		3 × 3 × 192/st. 1/pad 1 (eLU)	3 × 3 × 192/st. 1/pad 1 eLU/BN	
(MLP)-MLP-1-1	1 × 1 × 160/st. 1/pad 0 (ReLU)	1 × 1 × 160/st. 1/pad 0 (eLU)	1 × 1 × 160/st. 1/pad 0 (eLU	1 × 1 × 160/st. 1/pad 0 eLU/BN	
(MLP)-MLP-1-2	1 × 1 × 96/st. 1/pad 0 (ReLU)	1 × 1 × 96/st. 1/pad 0 (eLU)	1 × 1 × 96/st. 1/pad 0 (eLU)	1 × 1 × 96/st. 1/pad 0 eLU/BN	
Max pool	3 × 3/st.2			3 × 3/st.2/Drop	
Conv-2-1	3 × 3 × 192/st. 1/pad 1 (ReLU)	5 × 5 × 192/st. 1/pad 2 (eLU)	3 × 3 × 192/st. 1/pad 1 (eLU)	3 × 3 × 192/st. 1/pad 1 eLU/BN	
Conv-2-2	3 × 3 × 192/st. 1/pad 1 (ReLU)		3 × 3 × 192/st. 1/pad 1 (eLU)	3 × 3 × 192/st. 1/pad 1 eLU/BN	
(MLP)-MLP-2-1	1 × 1 × 160/st. 1/pad 0 (ReLU)	1 × 1 × 160/st. 1/pad 0 (eLU)	1 × 1 × 160/st. 1/pad 0 (eLU)	1 × 1 × 160/st. 1/pad 0 eLU/BN	
(MLP)-MLP-2-2	1 × 1 × 96/st. 1/pad 0 (ReLU)	1 × 1 × 96/st. 1/pad 0 (eLU)	1 × 1 × 96/st. 1/pad 0 (eLU)	1 × 1 × 96/st. 1/pad 0 eLU/BN	
Max pool	3 × 3/st.2			3 × 3/st.2/Drop	
Conv-3-1	3 × 3 × 192/st. 1/pad 1 (ReLU)	3 × 3 × 192/st. 1/pad 1 (eLU)	3 × 3 × 192/st. 1/pad 1 (eLU)	3 × 3 × 192/st. 1/pad 1 eLU/BN	
Conv-3-2	3 × 3 × 192/st. 1/pad 1 (ReLU)		3 × 3 × 192/st. 1/pad 1 (eLU)	3 × 3 × 192/st. 1/pad 1 eLU/BN	
(MLP)-MLP-3-1	1 × 1 × 160/st. 1/pad 0 (ReLU)	1 × 1 × 160/st. 1/pad 0 (eLU)	1 × 1 × 160/st. 1/pad 0 (eLU)	1 × 1 × 160/st. 1/pad 0 eLU/BN	
(MLP)-MLP-3-2	1 × 1 × 96/st. 1/pad 0 (ReLU)	1 × 1 × 96/st. 1/pad 0 (eLU)	1 × 1 × 96/st. 1/pad 0 (eLU)	1 × 1 × 96/st. 1/pad 0 eLU/BN	
AVG pool 8 × 8					
Softmax					

Our results were obtained by calculating the average over five executions. It is noted that the increase in the number of *Deep MLPconv* structures up to four structures leads to an increase in the number of parameters and an accuracy equivalent to 90.44% using the dropout layers in the same locations and without using the batch normalization layers. DNiN with a number of *Deep MLPconv* equivalent to three generates the best performance. Using a higher or lower number of *Deep MLPconv* leads to a reduction in the accuracy of the classification.

The experimental results are presented in Table 4, on the CIFAR10 datasets, and the test accuracy rates for all models in [5] are 85.49% and 89.59%, respectively. The test error rates for all the proposed DNiN configurations are 86.31%, 87.29%, 88.25%, 90.63% and 92.54%, respectively. The results also demonstrate the effectiveness of the proposed idea. Furthermore, they show that the DNiN "configuration (*E*)" offers better results than the different DNiN configuration and different NiN models [5] in terms of classification accuracy based on the CIFAR-10 dataset.

The experimental results show that DNiN provides classification precision that allows it to have a well-localized location between several baselines. The importance of DNiN lies in its size, and small models are more suited to implementations on a chip (FPGA). Our architecture offers interesting performance with a number of layers that does not exceed 16 layers on the contrary to [6] which uses 110, [8] which uses 28 layers and [9] which uses 29 layers. Table 5 shows a comparison between our work and the state of the art on the CIFAR-10 database with/without the use of data augmentation. The results of our work are presented with the size of the mini batch 128. Our results

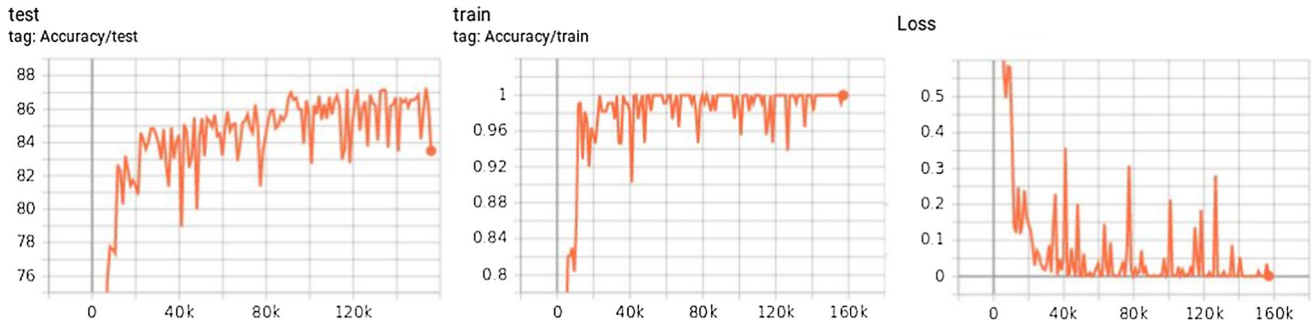

Fig. 7 Training, test and loss curves for configuration A on CIFAR-10

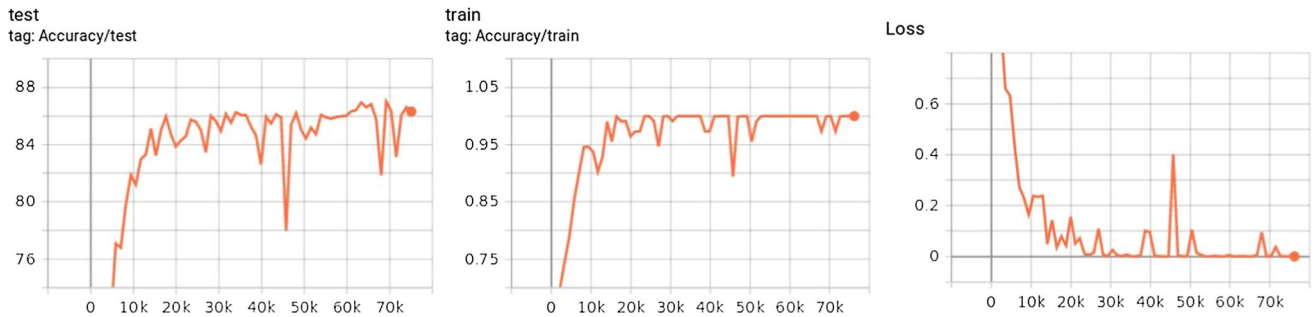

Fig. 8 Training, test and loss curves for configuration B on CIFAR-10

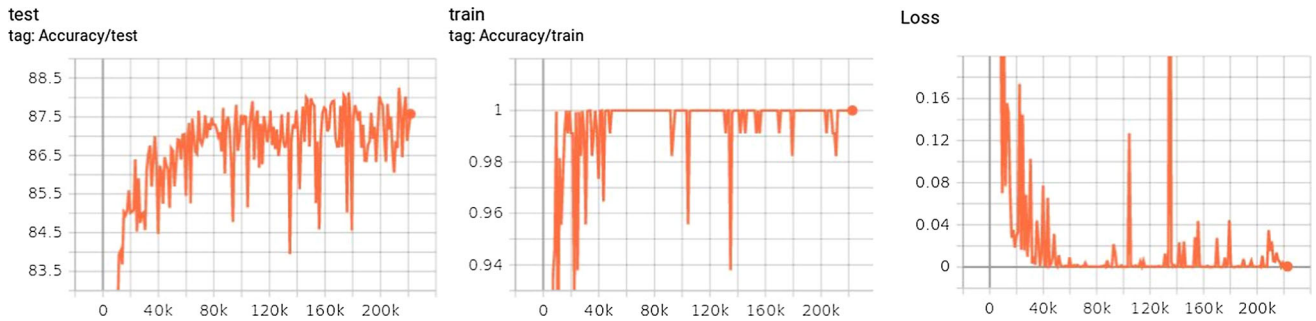

Fig. 9 Training, test and loss curves for configuration C on CIFAR-10

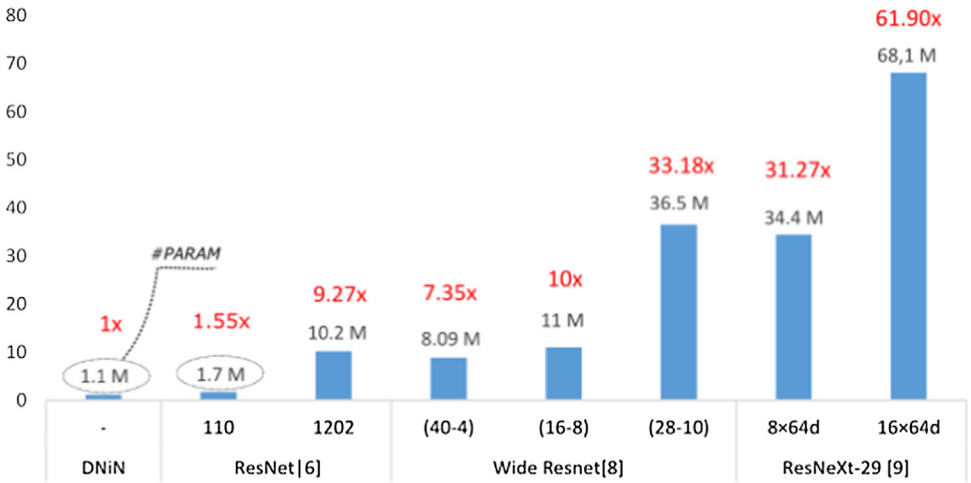

Fig. 10 CIFAR-10 parameters: a comparison with different architectures

Table 4 Precisions of different configurations (*A*–*E*) on CIFAR-10

Réf	Acc (%)
NiN [5]	85.49
	89.59
DNiN-A	86.31
DNiN-B	87.29
DNiN-C	88.25
DNiN-D	90.63
DNiN-E	92.54

were obtained by calculating the average of five executions.

4.5 Visualization of weights

We extract and visualize the weights of the first convolutional layer of our models formed for CIFAR-10. Figures 11 and 12 show the 192 × 3 × 3 convolution kernels learned by the first convolution layer on the 32 × 32 input images. Visualizing the weights of the first CONV layer is usually the most preferable as the filter weights deeper into the network because it directly looks at the raw pixel data.

5 Implementation details

We formed our configurations using a "Root Mean Square Propagation Algorithm" with a batch size equivalent to 128 and a weight decrease of 0.0001. We found that this small amount of weight loss was important for the model to learn. We initialized the weights in each layer from a normal to a random average distribution with a standard deviation of 0.01. We initialized the neural biases in all convolutional layers, as well as the MLP layers with the constant zero. In the implementation of the MLP, a 1 × 1 convolutional layer is considered a fully connected layer. The learning rate was initialized to 0.01 and divided by 10 twice before the end in epochs 81 and 122. We formed the network for about 200 cycles at most on the CIFAR-10 training set in a central processing unit (CPU). The python algorithm based on the deep learning library « TensorFlow » to classify and recognize images provides the implementation of the CPU. Table 6 describes the details of the CPU architecture.

Table 5 CIFAR-10 test error

Refs.	Method	Error (%)
No data augmentation		
[42]	Stochastic pooling	15.13
[27]	Maxout network (k = 2)	11.68
[5]	NIN	10.41
Our	DNiN	9.37
[30]	DSN	9.69
[28]	MIM (k = 2)	8.52 ± 0.20
Data augmentation		
[27]	Maxout network (k = 2)	9.38
[5]	NIN	8.81
[30]	DSN	8.22
Our	DNiN	7.46
[6]	ResNet	6.43
[8]	Wide Resnet (28, 10)	3.89
[9]	ResNeXt	3.58

6 Advantage and limitations

DNiN provides a very limited number of parameters that allows it to occupy an important position at the top of the works reported in the literature. There are two main reasons for this reduction: The first reason is the reduction in the size of the convolution filters, and the second reason is the use of the global mean grouping layer, which does not include any parameters to optimize. Added to that, DNiN delivers competitive test errors compared to the baseline with a very short number of layers compared to these works. The importance of DNiN also arises in its small size that makes it very suitable for integration as an image recognition system in applications of embedded systems. However, DNiN integrates disadvantages and limitations that reside mainly in the number of convolution kernel. DNIN uses a number of convolution kernels equivalent to 192 that negatively affects the number of parameters, the calculation complexity and the memory efficiency.

7 Conclusion

We are proposing a new model "Deep Network In Network" (DNIN) for the classification of images. The linear exponential unit (eLU) is used as an activation function to accelerate the learning of this network. In this network, a new nonlinear *Deep MLPconv* filter is used. This structure is based on very small convolution filters (3 × 3) and the exponential linear unit which accelerates learning. The use of this structure has proved beneficial for the accuracy of the classification. Our results confirm once again the importance of the depth and the choice of the activation function to improve the classification accuracy and show that our DNiN model obtains the acceptable results compared to the other architectures tested on the datasets, CIFAR-10 by considerably improving the main

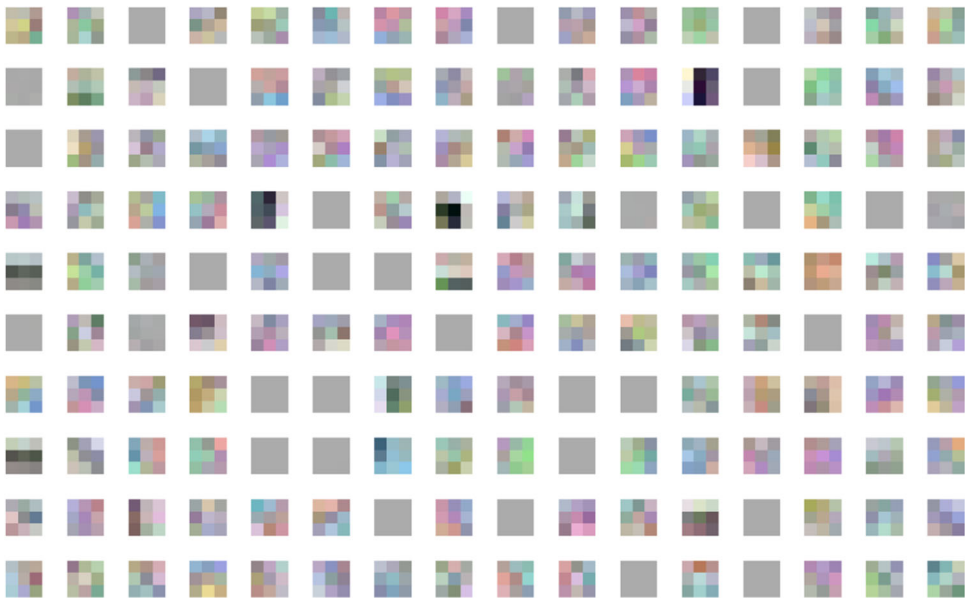

Fig. 11 192 convolution cores of size 3 × 3 learned by the first convolution layer on the 32 × 32 input images for the "configuration (*D*)"

Fig. 12 192 convolution cores of size 3 × 3 learned by the first convolution layer on the 32 × 32 input images for the "configuration (*D*)"

Table 6 CPU architecture

Processor	Name	Memory	Core count	Core clock	Memory bandwidth	Memory clock
CPU	Intel Xeon Processor E5-2620 v4	64 GB DDR4-2400	8 cores, 16 threads	2.1 GHz	68.3 GB/s	2133 MHz

performances (precision, parameter) and learning speed. Our future work will focus on optimizing classification performance. Working to reduce the number of convolutional kernels while maintaining or increasing the same accuracy can be an important point for research and reflection, and due to the important role it plays in reducing the number of parameters and the complexity of calculation in addition to reducing learning speed.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105
2. Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: Computer vision–ECCV 2014. Springer, pp 818–833
3. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. In: CoRR. http://arxiv.org/abs/1409.1556
4. Szegedy C et al (2015) Going deeper with convolutions. In: 2015 IEEE conference on computer vision and pattern recognition (CVPR), Boston, MA USA, pp 1–9. https://doi.org/10.1109/cvpr.2015.7298594
5. Lin M, Chen Q, Yan S (2014) Network in network. In: International conference on learning representations. http://arxiv.org/abs/1312.4400
6. He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. In: CoRR. http://arxiv.org/abs/1512.03385
7. He K, Zhang X, Ren S, Sun J (2016) Identity mappings in deep residual networks. In: Leibe B, Matas J, Sebe N, Welling M (eds) Computer vision – ECCV 2016. ECCV 2016. Lecture notes in computer science, vol 9908. Springer, Cham, pp 630–645. https://doi.org/10.1007/978-3-319-46493-0_38
8. Zagoruyko S, Komodakis N (2016) Wide residual networks. Sergey Zagoruyko Nikos Komodakis: Wide Residual Networks, pp 87.1–87.12 https://doi.org/10.5244/C.30.87
9. Xie S, Girshick R, Dollár P, Tu Z, He K (2016) Aggregated residual transformations for deep neural networks. arXiv:1611.05431
10. Huang G, Sun Y, Liu Z, Sedra D, Weinberger KQ (2016) Deep networks with stochastic depth, vol $9908, pp 646–661. https://doi.org/10.1007/978-3-319-46493-0_39
11. Huang G, Liu Z, van der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks
12. Sermanet P, Eigen D, Zhang X, Mathieu M, Fergus R, LeCun Y (2014) OverFeat: integrated recognition, localization and detection using convolutional networks. In: ICLR, 2014
13. Cao J, Pang Y, Li X (2016) Pedestrian detection inspired by appearance constancy and shape symmetry. In: Proceedings of IEEE international conference on computer vision and pattern recognition, pp 1316–1324
14. Gong M, Zhao J, Liu J, Miao Q, Jiao L (2016) Change detection in synthetic aperture radar images based on deep neural networks. IEEE Trans Neural Netw Learn Syst 27(1):125–138
15. Liu J, Gong M, Qin K, Zhang P (2018) A deep convolutional coupling network for change detection based on heterogeneous optical and radar images. IEEE Trans Neural Netw Learn Syst 29(3):545–559
16. Bengio Y (2009) Learning deep architectures for AI. Found Trends Mach Learn 2(1):127–131
17. Chang C-H (2015) Deep and shallow architecture of multilayer neural networks. IEEE Trans Neural Netw Learn Syst 26(10):2477–2486
18. Gong M, Liu J, Li H, Cai Q, Su L (2015) A multi objective sparse feature learning model for deep neural networks. IEEE Trans Neural Netw Learn Syst 26(12):3263–3277
19. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
20. Liu J, Gong M, Zhao J, Li H, Jiao L (2016) Difference representation learning using stacked restricted Boltzmann machines for change detection in SAR images. Soft Comput 20(12):4645–4657
21. Zhang P, Gong M, Su L, Liu J, Li Z (2016) Change detection based on deep feature representation and mapping transformation for multi-spatial-resolution remote sensing images. Photogram Remote Sens 116:24–41
22. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
23. Schmidhuber J (2014) Deep learning in neural networks: an overview. Neural Networks 61:85–117
24. Rumelhart D, Hinton G, Williams R (1986) Learning representations by back propagating error. Nature 323:533–536
25. Agostinelli F, Hoffman M, Sadowski P, Baldi P (2014) Learning activation functions to improve deep neural networks. In: CoRR. http://arxiv.org/abs/1412.6830
26. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML 2010), pp 807–814
27. Goodfellow IJ, Warde-Farley D, Mirza M, Courville AC, Bengio Y (2013) Maxout networks. In: Proceedings of the 30th international conference on machine learning (ICML 2013), volume 28 of JMLR proceedings, pp 1319–1327. http://jmlr.org/
28. Liao Z, Carneiro G (2016) On the importance of normalisation layers in deep learning with piecewise linear activation units. arXiv:1508.00330
29. Clevert D-A, Unterthiner T, Hochreiter S (2016) Fast and accurate deep network learning by exponential linear units (ELUs) comments: published as a conference paper at ICLR 2016 subjects—learning (cs.LG), 2016
30. Lee C-Y, Xie S, Gallagher P, Zhang Z, Tu Z (2015) Deeply supervised nets. In: Proceedings of AISTATS 2015
31. Srivastava RK, Greff K, Schmidhuber J (2015) Highway networks. In: CoRR. http://arxiv.org/abs/1505.00387
32. Jarrett K, Kavukcuoglu K, Ranzato MA, LeCun Y (2009) What is the best multi-stage architecture for object recognition?. In: Proceedings of the IEEE international conference on computer vision, pp 2146–2153
33. LeCun Y, Kavukcuoglu K, Farabet C (2010) Convolutional networks and applications in vision. In: Proceedings of the IEEE international symposium on circuits and systems, pp 253–256
34. Chan T, Jia K, Gao S, Lu J, Zeng Z, Ma Y (2014) PCANet: a simple deep learning baseline for image classification? In: CoRR. http://arxiv.org/abs/1404.3606
35. Gong Y, Wang L, Guo R, Lazebnik S (2014) Multi-scale orderless pooling of deep
36. Graham B (2014) Fractional max-pooling. In: CoRR. https://arxiv.org/abs/1412.6071
37. He K, Zhang X, Ren S, Sun J (2014) Spatial pyramid pooling in deep convolutional networks for visual recognition. In: Proceedings of European conference on computer vision, pp 346–361
38. Lee C, Gallagher P, Tu Z (2015) Generalizing pooling functions in convolutional neural networks: mixed gated and tree. In: CoRR. http://arxiv.org/abs/1509.08985
39. Murray N, Perronnin F (2014) Generalized max pooling. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition, pp 2473–2480
40. Springenberg J, Dosovitskiy A, Brox TT, Riedmiller M (2014) Striving for simplicity: the all convolutional net. In: CoRR. http://arxiv.org/abs/1412.6806

41. Yoo D, Park S, Lee J, Kweon I (2015) Multi-scale pyramid pooling for deep convolutional representation. In: Proceedings of IEEE workshop computer vision and pattern recognition, pp 1–5
42. Zeiler MD, Fergus R (2013) Stochastic pooling for regularization of deep convolutional neural networks. In: CoRR. http://arxiv.org/abs/1301.3557
43. Chang J, Chen Y (2015) Batch-normalized maxout network in network. arXiv:1511.02583
44. Castaneda G, Morris P, Khoshgoftaar TM (2019) Evaluation of maxout activations in deep learning across several big data domains. J Big Data 6:72. https://doi.org/10.1186/s40537-019-0233-0
45. Shao L, Wu D, Li X (2014) Learning deep and wide: a spectral method for learning deep networks. IEEE Trans Neural Netw Learn Syst 25(12):2303–2308
46. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: CoRR, vol. abs/1502.03167
47. Srivastava Nitish, Hinton Geoffrey, Krizhevsky Alex, Sutskever Ilya, Salakhutdinov Ruslan (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958
48. Russakovsky O, Deng J, Su H et al (2015) ImageNet large scale visual recognition challenge. Int J Comput Vis (IJCV) 115(3):211–252
49. Shorten C, Khoshgoftaar TM (2019) A survey on image data augmentation for deep learning. J Big Data 6:60. https://doi.org/10.1186/s40537-019-0197-0