Deep Neural-Kernel Blocks

Siamak Mehrkanoon[†1]

[†] *Department of Data Science and Knowledge Engineering, Maastricht University, The Netherlands*

Abstract

This paper introduces novel deep architectures using the hybrid neural-kernel core model as the first building block. The proposed models follow a combination of a neural networks based architecture and a kernel based model enriched with pooling layers. In particular, in this context three kernel blocks with average, maxout and convolutional pooling layers are introduced and examined. We start with a simple merging layer which averages the output of the previous representation layers. The maxout layer on the other hand triggers competition among different representations of the input. Thanks to this pooling layer, not only the dimensionality of the output of multi-scale representations is reduced but also multiple sub-networks are formed within the same model. In the same context, the pointwise convolutional layer is also employed with the aim of projecting the multi-scale representations onto a new space. Experimental results show an improvement over the core deep hybrid model as well as kernel based models on several real-life datasets.

Key words: Deep learning, neural networks, kernel methods, pooling layer, competitive learning, dimensionality reduction

1. Introduction

Recent advances in models with deep architectures have given rise to powerful techniques with successful impact in revolutionizing many application fields by solving a variety of complex tasks in computer vision, processing and speech processing among others. Deep learning has emerged as a powerful representation learning technique that can learn multiple layers of representations of the raw input data [5]. Various deep learning architectures such as stacked denoising autoencoders [44, 16], convolutional neural networks [21, 20, 19], long short term memories [17], generative adversarial networks [12] among others have been proposed in the literature. Among different architectures, deep convolutional neural networks have propelled unprecedented advances in artificial intelligence including object recognition, speech recognition, and image captioning.

It should be noted that although these deep architectures have shown to be successful in achieving the state of the art performance, however for training such a complex network one requires huge amounts of labeled training data to fit millions of parameters by solving a costly nonlinear optimization problem. Inspired by the success of deep learning models in different application domains, literature has witnessed the design of various artificial neural network architectures using different networks topologies.

Artificial neural networks based models achieve good performance, but require a lot of parameter tweaking which is not well founded in theory. In contrast, support vector machines (SVM) and kernel based models are very well researched and have relatively few hyper-parameters to tweak [43, 37, 7]. Therefore, it appears worthwhile to pursue a hybrid approach incorporating the best of both worlds and exploring new synergies. Among research works conducted in this direction, one can for instance refer to convolutional kernel networks [25], deep gaussian processes [10], kernel methods for deep learning [8], convex deep learning models with normalized kernels [2]. The authors in [14] discussed probabilistic methods for multiclass classification problems. An overview of the conducted research works in the literature for finding synergies between artificial neural networks and kernel based models can be found in [4].

Recently, Mehrkanoon *et al.* in [31, 29] introduced a new line of research that combines the ANN and kernel models in order to achieve the best of two methodologies. To this end, they proposed the hybrid layers in a neural networks architecture using kernel functions. In particular, the deep hybrid model in [29] uses an explicit feature map based on random Fourier features to make the bridge between ANNs and kernel models. The authors showed that stacking several hybrid layers can potentially brings more flexibility to the model. Some theoretical analysis on the generalization properties of learning with random Fourier features can for instance be found in [36, 40].

It is the purpose of this paper to extend the models discussed in [31, 29] and introduce novel deep neural-kernel networks architectures. To this end, different choices of merging layers in the context of neural-kernel models will be explored. The proposed deep architectures are built using stacking of kernel blocks which enables the new representations of the data to be learned in multiple levels (blocks) corresponding to different scales of feature mapping.

In particular, three types of kernel blocks. i.e. average kernel block, maxout kernel block and convolutional kernel block

[1]Corresponding author.
E-mail address:siamak.mehrkanoon@maastrichtuniversity.nl, mehrkanoon2011@gmail.com

are introduced. In addition the developed models are able to learn an optimal combination of the multiple representations of the input in an end-to-end fashion. The maxout kernel block is composed of multiple linear transformations of the input followed by the maxout and an explicit feature mapping layers. The maxout layer imposes a local nonlinearity before the explicit feature maps are applied. As in [31, 29], here the random Fourier features are used for constructing the explicit feature maps. The other introduced kernel block is the average kernel block which is obtained by replacing the maxout layer with an average layer. The last kernel block discussed here is the convolutional kernel block which follows the architecture of maxout kernel block but instead a pointwise convolutional later is used for reducing the dimension of the concatenated multiple linear transformations of the input. As it will be shown later, the average kernel block is a special case of the introduced pointwise convolutional kernel block.

This paper is organized as follows. A brief overview of the related research works in the literature is given in section 2. In section 3, three novel deep neural-kernel architectures are presented. The experimental results are reported in Section 4 and the conclusions and future works are drawn in Section 5.

2. Related works

For the sake of comprehension, this section provides a brief review of previous studies on deep hybrid neural-kernel networks architectures that will serve as a starting point for more advanced models developed in this paper. Assume that $\mathcal{D} = \{x_1, ..., x_n\}$, where $\{x_i\}_{i=1}^n \in \mathbb{R}^d$ is the training set with labels $\{y_i\}_{i=1}^n$ and that the total number of classes is Q. Let us first recall the basic yet an important difference between a single module of an artificial neural network architecture and a kernel based model. As stated in [31] in classical artificial neural network architectures, one assumes that the nonlinear activation functions f is applied on the weighted sum of the given input instance. A single-layer ANN can be formulated as follows:

$$y(x) = f(Wx + b),$$

where $x \in \mathbb{R}^d$, $W \in \mathbb{R}^{d_h \times d}$ and b is the bias vector. Here f denotes the activation function. Commonly used activation functions in the literature are for instance sigmoid, hyperbolic tangent and more recently piecewise linear activation functions such as Rectified Linear Units (ReLU) [33], Leaky Rectified Linear Unit (LReLU) [23], Parametric Rectified Linear Unit (PReLU) [15], Randomized Leaky Rectified Linear Unit (RReLU) [46], Exponential Linear Unit (ELU) [9] and maxout units [13]. In contrast, in the kernel based models, the nonlinear feature map is directly applied on the given input instance. The target value is then estimated by means of a weighted sum of the projected sample. Furthermore, in kernel based approaches one often works with the primal-dual setting.

The choices of feature maps used in the primal space is either implicit or explicit. When implicit feature mapping is used the problem is solved in the dual thanks to the kernel trick. Alternatively one can also use an explicit feature map and solves the problem directly in the primal. For instance, Mehrkanoon et al. in [30, 28, 27] have discussed the use of both implicit and explicit feature mapping for addressing different problem statements ranging from semi-supervised learning to domain adaption. In a nutshell, in kernel based model one expresses the solution in the primal in terms of feature maps as follows:

$$y(x) = w^T \varphi(x) + b,$$

where $\varphi(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^h$ is the feature map and h is the dimension of the feature space (which could possibly be infinite dimensional). The data are first embedded into a feature space and the optimal solution is sought in that space. One can observe that a single module of kernel based model is linear in the weight vector w. Therefore, optimal values of model parameter w and b can be obtained by applying convex optimization based techniques.

Deep hybrid architectures that bridge the artificial neural networks and kernel based models in the primal level have been proposed in [31, 29]. An explicit feature mapping based on random Fourier features is used for the kernel based model counterpart. This in particular enables the hybrid model to be scalable to large scale data as well as facilitating the integration of the ANN and kernel based models. Other possible explicit feature maps obtained for instance by greedy based selection techniques [38] or Nyström methods [45] could also potentially be used in the formulation introduced in [31].

Random Fourier features are proposed in in the field of kernel methods by exploiting the classical Bochner's theorem in harmonic analysis (see [35]). The random Fourier features $\hat{\varphi}(x)$, for the sample x, are defined as $\hat{\varphi}(x) = \frac{1}{\sqrt{D}}[z_{\xi_1}(x), \ldots, z_{\xi_D}(x)]^T \in \mathbb{R}^D$. Here $z_\xi(x) = \cos(\xi^T x)$ and $\xi_1, \ldots, \xi_D \in \mathbb{R}^d$ are sampled from $p(\xi)$. For a Gaussian kernel, they are drawn from a normal distribution $\mathcal{N}(0, \sigma^2 I_d)$. In addition, a normalization factor $\frac{1}{\sqrt{D}}$ is used to reduce the variance of the estimation. Mehrkanoon et al. [31] proposed the basis for implementing hybrid neural-kernel networks using random Fourier feature maps and formulated a two layer hybrid model as follows (see also Fig. 1):

$$s = W_2 \hat{\varphi}(W_1 x + b_1) + b_2. \tag{1}$$

where $W_1 \in \mathbb{R}^{d_1 \times d}$ and $W_2 \in \mathbb{R}^{Q \times d_2}$ are weight matrices and the bias vectors are denoted by b_1 and b_2.

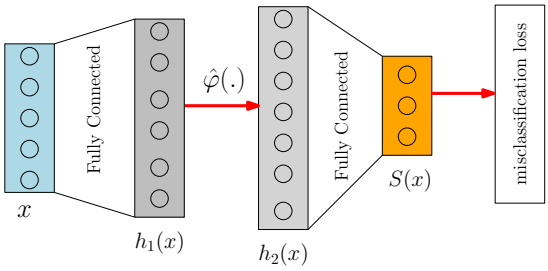

Figure 1: Two-layer hybrid neural-kernel network architecture [31].

An extension of the introduced model in [31] is also presented in [29] where the authors in particular showed how one could build a deep hybrid model.

Remark 1. Inspecting Fig. 1, one can realize that the introduced hybrid model can also be seen from a purely neural networks architecture point of view but with a new activation function that is imposed by an explicit feature map derived from a kernel based model. More precisely, in the classical neural networks architecture, the output of each neuron in the hidden layer is obtained by applying a nonlinear activation function only on that neuron independently from other neurons. As opposed to classical neural networks, in the proposed hybrid models [31, 29] the output of each neuron in the hidden layer is obtained by applying an explicit feature map that is a function of all the neurons. So in this case the output of each neuron is obtained by taking into account the exiting similarities between all of the neurons (see Fig. 2). Alternatively one could also design explicit feature maps that takes into account only nearby neurons when calculating the output.

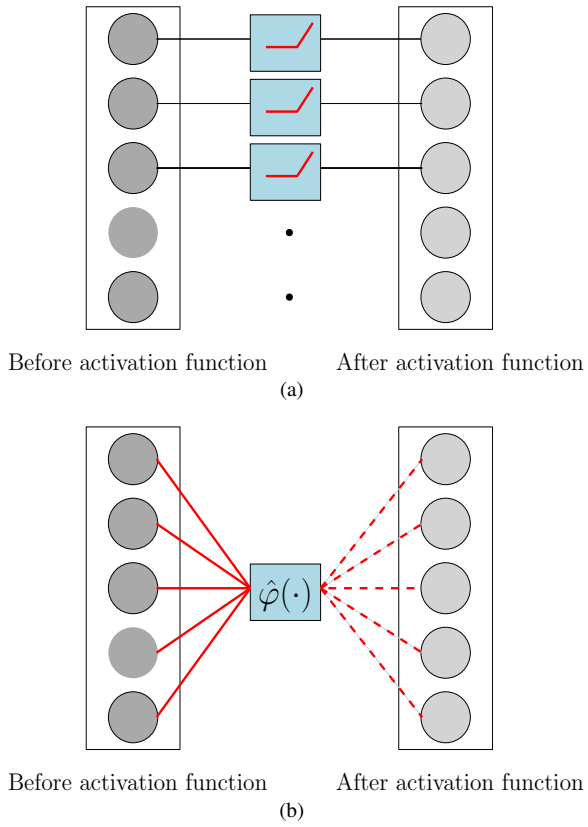

Figure 2: (a) Activation functions in classical neural networks architecture. (b) explicit feature map in a hybrid model.

3. Proposed deep neural-kernel architectures

This section introduces three deep neural-kernel architectures obtained by staking several kernel blocks. In what follows, the maxout, average and convolutional kernel blocks are discussed which will serve as the first building blocks for the development of the corresponding deep models.

3.1. The maxout kernel block

As it has been shown in the literature, competitive activation units have been successfully employed in deep neural networks. The breakthrough performance of deep learning can be sought in using piece-wise linear functions such as ReLU [33, 11], maxout [13] and their variants. With the aim of this activation units the whole input space is divided into several sub-regions. Subsequently one can form sub-networks where they can be jointly trained for solving simpler sub-problems.

In particular, ReLU activation units allow the division of the input space into two regions whereas maxout activation units can for instance divide the space in as many regions as the number of inputs [13]. Therefore thanks to the possibility of building large number of sub-networks that are trained jointly, one can more effectively estimate the exponentially complex functions. The maxout activation function computes the maximum of a set of linear functions, and has the property that it can approximate any convex function of the input and it has achieved state-of-the-art performance on multiple machine learning benchmarks [13]. Designing activation functions that enable fast training of accurate deep neural networks is still an active area of research. For instance, authors in [39] replaced the max function with a probabilistic max function. Other researchers proposed to learn the activation function during training, i.e. adaptive activation function where piecewise linear activation function is learned independently for each neuron using gradient descent, and can represent both convex and non-convex functions of the input, see[1].

Here we introduce the maxout kernel block which will subsequently be used to build deep maxout neural-kernel networks. In the maxout kernel block the input data passes through multiple linear transformations followed by maxout units before reaching the explicit feature maps. A maxout unit takes as input the output of multiple linear functions and returns the largest. Incorporating multiple linear functions of the input increases the expressive power of maxout units, allowing them to approximate arbitrary convex functions. The maxout layer learns not just the relationship between hidden units, but also the activation function of each hidden unit.

Given the input representation $h^{(\ell-1)}$ in the $(\ell-1)$-th layer, the weight matrices and the bias vectors $V_k^{(\ell)} \in \mathbb{R}^{d_\ell \times d}$ and $b_k^{(\ell)} \in \mathbb{R}^{d_\ell}$ of the (ℓ)-th layer for $k = 1, \ldots, m$ respectively, the (ℓ)-th maxout kernel block is introduced as follows:

$$\begin{aligned} h^{(\ell)}_{\text{maxout}} &= \max_{k\in\{1,\ldots,m\}} \; V_k^{(\ell)} h^{(\ell-1)} + b_k^{(\ell)}, \\ h^{(\ell)} &= \hat{\varphi}_{(\ell)}(h^{(\ell)}_{\text{maxout}}). \end{aligned} \tag{2}$$

Here $h^{(0)}$ is the original input x. A deep maxout neural-kernel network can then be constructed by means of stacking the maxout kernel blocks as shown in Fig. 3 and formulated as follows:

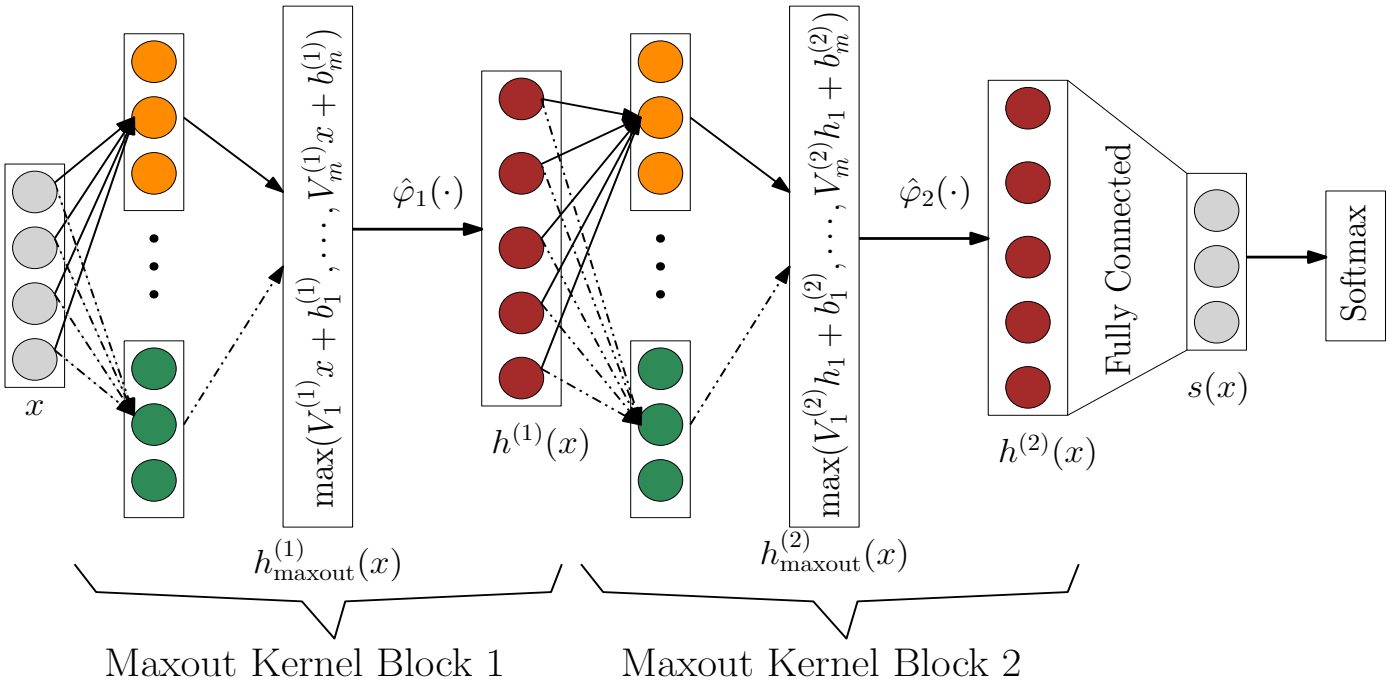

Figure 3: Deep neural-kernel network architecture with maxout kernel blocks.

$$
\begin{aligned}
h^{(1)}_{\text{maxout}} &= \max_{k\in\{1,\ldots,m\}} \quad V_k^{(1)}x + b_k^{(1)}, \\
h^{(1)} &= \hat{\varphi}_{(1)}(h^{(1)}_{\text{maxout}}), \\
h^{(2)}_{\text{maxout}} &= \max_{k\in\{1,\ldots,m\}} \quad V_k^{(2)}h^{(1)} + b_k^{(2)}, \\
h^{(2)} &= \hat{\varphi}_{(2)}(h^{(2)}_{\text{maxout}}), \\
s(x) &= Wh^{(2)} + b, \qquad (3)
\end{aligned}
$$

where here the weight matrix and the bias vector of the last layer are denoted by $W \in \mathbb{R}^{Q\times d_2}$ and $b \in \mathbb{R}^Q$. We denote all the trainable model parameters as Θ. The optimization problem corresponding to the proposed model can be formulated as follows:

$$\min_{\Theta} J(\Theta) = \gamma\Omega(\theta) + \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i). \qquad (4)$$

Here $L(\cdot,\cdot)$ and $\Omega(\Theta)$ are the regularization term and cross-entropy loss. The role of the regularization term is to avoid overfitting. Some of the possible regularization techniques that can be employed are for instance imposing penalties on layer parameters or layer activity, noise insertion and dropout. Another form of regularization is early stopping by monitoring the performance of the network on the validation set. The regularization parameter γ controls the relative importance given to the regularization term. After obtaining the model parameters Θ, the score variable for the test point x_{test} can be computed by replacing x in (3) with x_{test}. The final class label for the test point x_{test} is computed as follows:

$$\hat{y}_{\text{test}} = \underset{\ell=1,\ldots,Q}{\text{argmax}}(s_{\text{test}}). \qquad (5)$$

The pseudocode of our approach is described in Algorithm 1. Here in order to avoid over fitting the network is trained with an early stooping criterion. In this way, after maximum twenty epochs if no improvement on the validation set is achieved then the training will be stopped.

Algorithm 1: Deep Maxout Neural-Kernel Networks

Input: Training data set $\mathcal{D}$, regularization constants $\gamma, \in \mathbb{R}^+$, Kernel parameter, learning rate η and test set $\mathcal{D}^{\text{test}} = \{x_i\}_{i=1}^{n_{\text{test}}}$.

Output: Class label for the test instances $\mathcal{D}^{\text{test}}$.

```
Initialize the model parameters Θ
while the stopping criterion is not satisfied do
    Evaluate the gradient of the cost function w.r.t model parameters.
    Make a gradient step and update the model parameters: Θ^new = Θ − η∇_Θ J(Θ).
    Θ ← Θ_new.
return W
Compute the score variable s, for the test instances in D^test
Predict the test labels using (5).
```

3.2. The average kernel block

One of the approaches for fusing multiple representations is using the average layer which gives equal weight to each of the representations. Here we introduce the average kernel block by replacing the maxout layer discussed in section 3.1 with an average layer. Given the input representation $h^{(\ell-1)}$ in the $(\ell-1)$-th layer, the weight matrices and the bias vectors $V_k^{(\ell)} \in \mathbb{R}^{d_\ell\times d}$

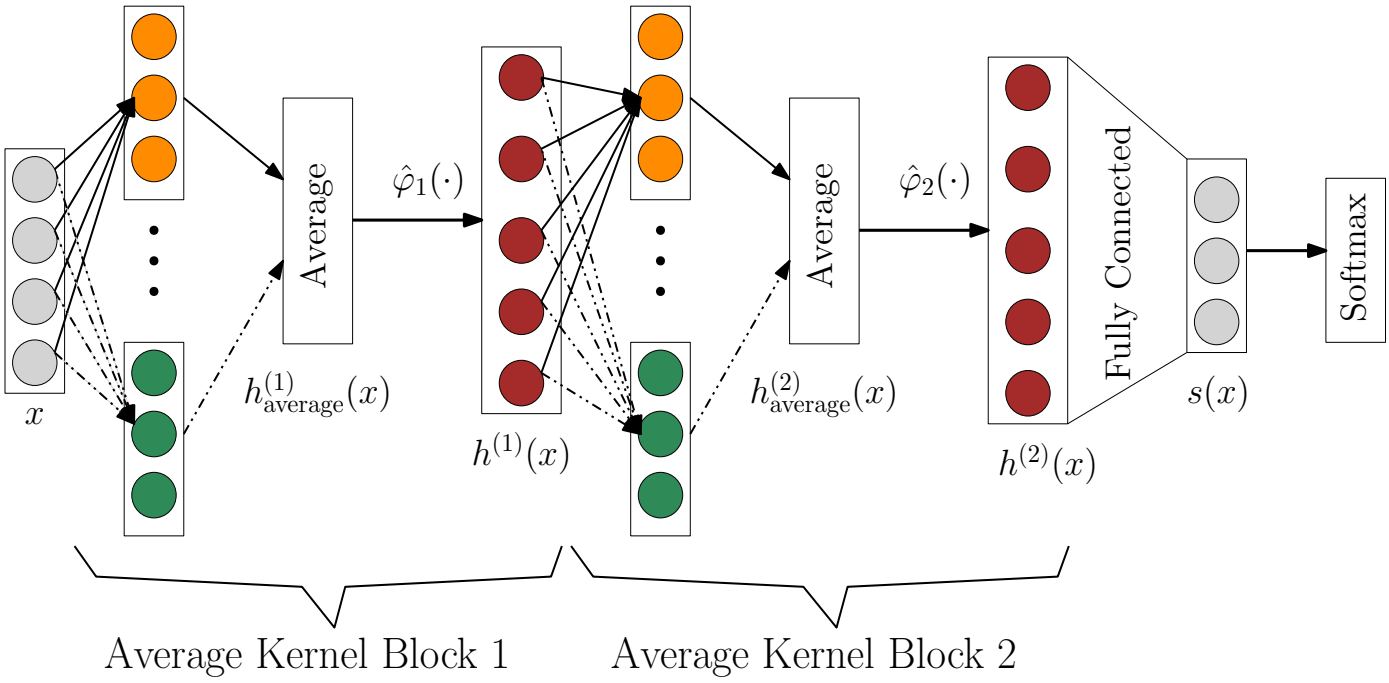

Figure 4: Deep neural-kernel network architecture with average kernel blocks

and $b_k^{(\ell)} \in \mathbb{R}^{d_\ell}$ of the (ℓ)-th layer for $k = 1, \ldots, m$ respectively, the (ℓ)-th average kernel block is formulated as follows:

$$\begin{aligned} h^{(\ell)}_{\text{average}} &= \text{mean}\{V_1^{(\ell)} h^{(\ell-1)} + b_1^{(\ell)}, \ldots, V_m^{(\ell)} h^{(\ell-1)} + b_m^{(\ell)}\}, \\ h^{(\ell)} &= \hat{\varphi}_{(\ell)}(h^{(\ell)}_{\text{average}}). \end{aligned} \tag{6}$$

In particular, $h^{(0)}$ is the original input data x. A deep average neural-kernel network can then be built using stacking the average kernel blocks as shown in Fig. 4 and formulated as follows:

$$\begin{aligned} h^{(1)}_{\text{average}} &= \text{mean}\{V_1^{(1)} x + b_1^{(1)}, \ldots, V_m^{(1)} x + b_m^{(1)}\}, \\ h^{(1)} &= \hat{\varphi}_{(1)}(h^{(1)}_{\text{average}}), \\ h^{(2)}_{\text{average}} &= \text{mean}\{V_1^{(2)} h^{(1)} + b_1^{(2)}, \ldots, V_m^{(2)} h^{(1)} + b_m^{(2)}\}, \\ h^{(2)} &= \hat{\varphi}_{(2)}(h^{(2)}_{\text{average}}), \\ s(x) &= W h^{(2)} + b, \end{aligned} \tag{7}$$

where $W \in \mathbb{R}^{Q \times d_2}$ and $b \in \mathbb{R}^Q$ are the weight matrix and the bias vector of the last fully connected layer. The dimensions of the hidden layers are defined as previously and the same loss function and the optimization algorithm is applied for learning the model parameters (see section 3.1).

3.3. The convolutional kernel block

Here we introduce a model that learns new representation of the data from multiple linear transformations of the input data and leverages the fused learned features to predict the desired target. To this end, a convolutional kernel block is designed which follows the same architecture as the one in the maxout kernel block but with a different type of feature pooling. In particular, here a pointwise convolutional layer is employed with the aim of projecting the multiple representations onto a new space.

A pointwise convolution is a (1×1)-convolutional layer (see [22]) which applies the filters along the depth dimension to reduce or increase the dimensionality of the concatenated features. Here, in order to have a comparable number of hidden units among the three proposed kernel blocks, we only use one filter in the convolutional layer. However, in practice one could apply and learn more number of filters provided that enough training data is available to avoid overfitting.

Given the input representation $h^{(\ell-1)}$ in the $(\ell - 1)$-th layer, the (1×1)-convolution operations, with one filter, applied on the concatenated linear transformations of $h^{(\ell-1)}$ can be written in a matrix multiplication form as follows:

$$Z^{(\ell)} R^{(\ell)} 1_m, \tag{8}$$

where $Z^{(\ell)} = \left[V_1^{(\ell)} h^{(\ell-1)} + b_1^{(\ell)}, \ldots, V_m^{(\ell)} h^{(\ell-1)} + b_m^{(\ell)} \right]_{d_\ell \times m}$ denotes the concatenated linear transformations of $h^{(\ell-1)}$. In particular, $h^{(0)}$ is the original input data x. R is a diagonal matrix of size $m \times m$ whose diagonal elements are the filter parameter of the ℓ-th convolutional layer, in our case only one parameter i.e. $r^{(\ell)}$. A vector of all ones with size m is denoted by 1_m.

From Eq. (8), one can note the main difference between the proposed average and convolutional kernel blocks. If one sets the filter parameter r to $\frac{1}{m}$, then the convolutional kernel block reduces to the average kernel block. In the numerical experiments we will observe how this one additional parameter that is being learned influences the test performance of the model. The

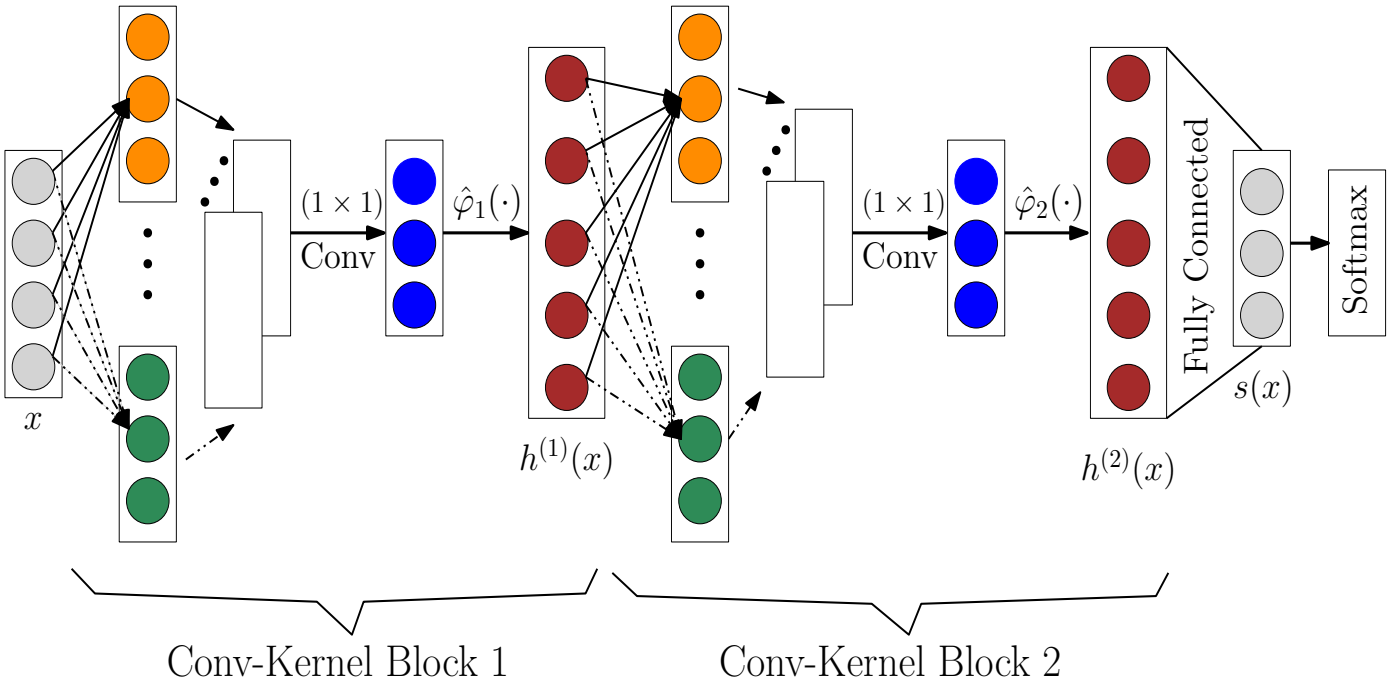

Figure 5: Deep neural-kernel network architecture with convolutional kernel blocks

(ℓ)-th convolutional kernel block is then formulated as follows:

$$\begin{aligned} h^{(\ell)}_{\text{conv}} &= Z^{(\ell)} R 1_m, \\ h^{(\ell)} &= \hat{\varphi}_{(\ell)}(h^{(\ell)}_{\text{conv}}). \end{aligned} \tag{9}$$

A deep neural-kernel network can then be built using stacking the convolutional kernel blocks as shown in Fig. 5 and formulated as follows:

$$\begin{aligned} h^{(1)}_{\text{Conv}} &= Z^{(1)} R 1_m, \\ h^{(1)} &= \hat{\varphi}_{(1)}(h^{(1)}_{\text{Conv}}), \\ h^{(2)}_{\text{Conv}} &= Z^{(2)} R 1_m, \\ h^{(2)} &= \hat{\varphi}_{(2)}(h^{(2)}_{\text{Conv}}), \\ s(x) &= W h^{(2)} + b, \end{aligned} \tag{10}$$

where $W \in \mathbb{R}^{Q \times d_2}$ and $b \in \mathbb{R}^Q$ are the weight matrix and the bias vector of the last fully connected layer. The dimensions of the hidden layers are defined as previously and the same loss function and the optimization algorithm is applied for learning the model parameters (see section 3.1).

4. Experimental results and discussion

In this section several experimental results on real-life datasets taken from UCI machine learning repository [3] as well as image benchmark datasets are presented and discussed. The given UCI dataset is randomly partitioned to 80% training and 20% test sets respectively. We compare the performance of the proposed deep models with respect to each other as well as with that of the TROP-ELM [32], LS-SVM [41] and deep hybrid model [29]. The parameters of the proposed models are the dimension of the fully connected layers and the explicit feature maps as well as the variance of the normal distribution from which one constructs the random Fourier features. Here, following the lines of [6], we use the random search scheme for model selection. In our experiments, the dimensions of the middle layers are sought in the range of $[100, 700]$. We also set the regularization parameter $\gamma = 0.0001$ for all the experiments. Furthermore, for these datasets two neural-kernel blocks with four linear transformations (see Fig. 3) are used in the experiments.

The proposed deep architectures can potentially be trained using different strategies. One for instance can use block wise training scheme i.e. learning the parameters of the first block and freeze and use it when learning the parameters of the second block. The other possibility is to learn all the model parameters jointly in an end-to-end fashion which is adopted in this paper.

Different regularization techniques such as batch normalization, dropout, imposing various penalties on each block parameters can potentially be applied in order to avoid overfitting and have a good generalization performance. Here, in all the experiments an early stopping criterion which keeps track of the validation loss is applied. In particular, after twenty epochs of no improvement on the validation set the training is stopped.

The obtained empirical results of the proposed models and those of TROP-ELM [32], LS-SVM [41] and the hybrid model [29] are reported in Table 1. The Gaussian kernel is used in both TROP-ELM and LS-SVM models. In most of the cases studied here the proposed deep models improve the accuracy over other discussed models in [32, 41, 29]. In some of the cases such as the Titanic, Covertype and Motor datasets a significant improvement is achieved. Among the proposed deep models,

the accuracy of the maxout neural-kernel model is comparable with that of pointwise convolutional neural-kernel model and better than that of average neural-kernel model. In general, one can expect that when the underlying non-linearity of the data is complex, the proposed deep architectures can potentially obtain a better decision boundary in the expense of slightly more training computation times. The training computation times of the proposed deep models and that of the deep hybrid model [29] for all the studied datasets are depicted in Fig. 6. The training and validation loss as well as the accuracy of the proposed models and the deep hybrid model [29] for the Motor and Covertype datasets are also illustrated in Fig. 7. The same stopping criterion is applied to all the models, but as it can be seen from Fig. 7 some models stopped earlier than the others. In general, a considerable gap between the training and validation losses of the proposed deep models with those of the deep hybrid model [29] can be observed. This indicates that one should expect a better generalization performance on the test set under the assumption that both training and test data exhibit the same distribution.

The learned weights of the deep maxout neural-kernel networks with $m = 4$ for the Motor dataset is also shown in Fig. 8. It is interesting to note the difference between the magnitude and patterns of the weights corresponding to $V_i^{(1)}$ and $V_i^{(2)}$ (for $i = 1, \ldots, 4$) in different blocks which can indicate how much emphasis the network is giving to each transformation.

Furthermore, the t-SNE visualization [24] of the obtained projections of the hidden layers as well as the score variables using the proposed deep maxout neural-kernel architecture with $m = 4$ for the Motor dataset are also depicted in Fig. 9. One can observe that as the data flows through the network, its representation changes. Ideally, one would expect that the representation obtained in the deeper layer can have a better chance of forming more separable clusters corresponding to the exciting classes.

Next we evaluate the proposed deep maxout neural-kernel block and convolutional neural-kernel block models on four real-life benchmark datasets CIFAR-10 [18], CIFAR-100 [18], SVHN [34] and MNIST [5]. The architectures used for these datasets follow a pipeline which is composed of a combination of convolutional neural networks for the purpose of feature learning and the proposed models for classification. In particular, the employed CNN networks consist of three stacked convolutional layers followed by a spatial pooling layer which downsamples the input image by a factor of two. The resulting representation is then fed to the proposed models and the model parameters of CNN and the proposed models are jointly learned in an end-to-end fashion. For these datasets one neural-kernel block with three linear transformations is used in the experiments. The size of the feature maps (number of kernels per convolutional layer) increases as one goes deeper in the network and here in our experiments we set it to 32, 64 and 128.

The MNIST dataset consists of hand written digits 0-9 which are of size 28×28. There are $60,000$ training images and $10,000$ testing images in total. As the preprocessing step, the images are scaled to [0,1] before feeding them into the networks. The obtained test errors of the proposed models together with those of other models reported in the literature [47, 42] are tabulated in Table 2. The CIFAR-10 dataset is composed of color images with 50,000 training and 10,000 testing images. Each image is an RGB image of size 32x32. The size of the local receptive field as well as the weight decay are tunned using the validation set. As in [13], here we apply the global contrast normalization and ZCA whitening for preprocessing the images. A comparison with other models in [13, 47, 42] is reported in Table 3. The size and format of the CIFAR-100 dataset is the same as those of CIFAR-10 dataset, but instead it contains 100 classes. Thus the number of images in each class is only one tenth of the CIFAR-10 dataset. For CIFAR-100 we use the same setting as the CIFAR-10 dataset. The obtained test errors of various models are tabulated in Table 4. The SVHN dataset is composed of total number of $630,420$ color images of size 32×32. The task of this data set is to classify the digit located at the center of each image. We follow the training and testing procedure that is used in Goodfellow et al. [13] . In addition, as in [13] a local contrast normalization is applied as the preprocessing step. The obtained results are compared with previous works that adopted convolutional structures, and are shown in Table 5. From the reported results in Table 2-5, one can observe that the proposed models have achieved comparable and sometimes better performance with respect to the other convolutional and SVM based models. It should be noted that in these experiments we have not fully explored dropout, weight constraints, hidden unit sparsity, or exhausting search for model selection.

In what follows, we also analyze the influence of the number of neural-kernel blocks and the number of linear transformations within each block on the performance of the proposed deep maxout neural-kernel block architecture. The obtained accuracy as well as the training and validation loss corresponding to different combinations of the number of blocks and linear transformations for the Magic dataset are shown in Fig. 10 and Fig. 11 respectively. In general, we can observe that as one increases the number of blocks, the complexity of the model increases and therefore the training loss converges faster with less number of epochs. It should also be noted that the more complex the model becomes, the higher the chance the model overfits the data. The optimal combinations of these two parameters, i.e. the number of blocks and linear transformations, could in principle be tuned using a validation dataset.

5. Conclusions

In this paper three novel neural-kernel blocks with average, maxout and pointwise convolutional layers are introduced. In addition their corresponding deep architectures are developed using a stacking strategy. The proposed neural-kernel blocks enrich the classical hybrid neural-kernel model with integrating pooling layers in the new architectures. In this context, thanks to the employed pooling layers the multi-scale representations of the data are combined in a competitive way which lead to an improvement on the generalization performance of the model. In particular, we also showed that the introduced

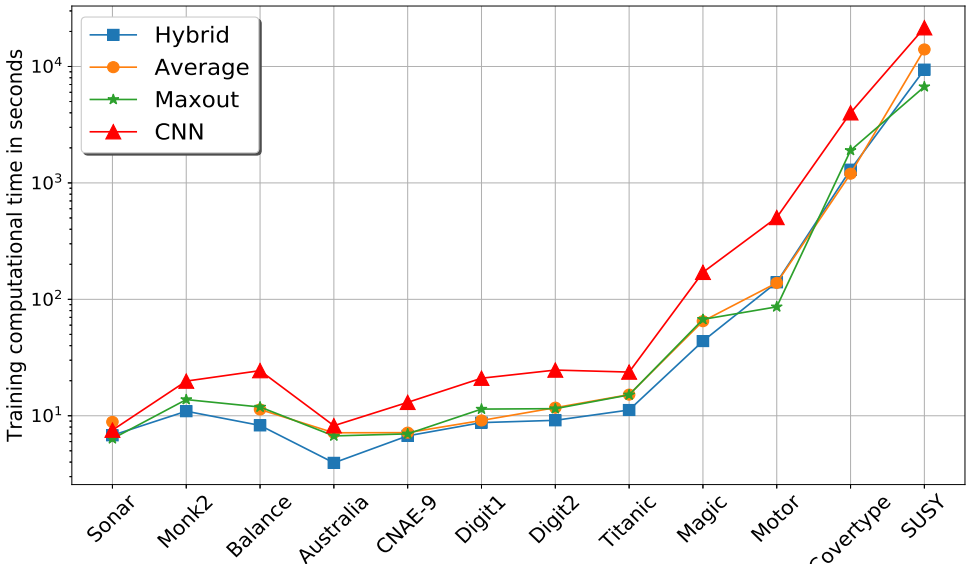

Figure 6: The training computation times of the proposed deep neural-kernel architectures and the deep hybrid model [29] for the studied datasets.

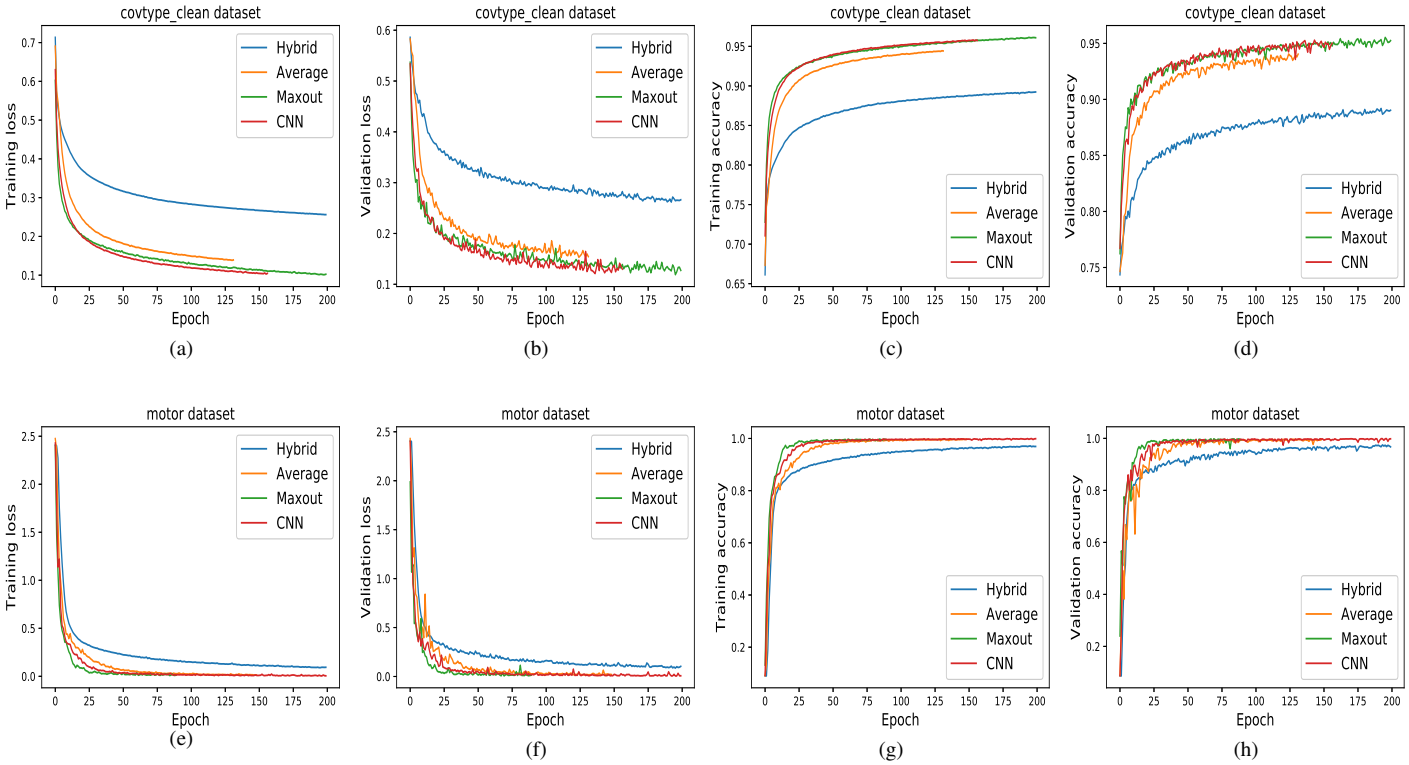

Figure 7: (a,b,c,d) The validation and training loss as well as the accuracy of the proposed deep neural-kernel architectures and the classical deep hybrid neural-kernel model [29] for the Covertype dataset. (e,f,g,h) The validation and training loss as well as the accuracy of the proposed deep neural-kernel architectures and the classical deep hybrid neural-kernel model [29] for the Motor dataset.

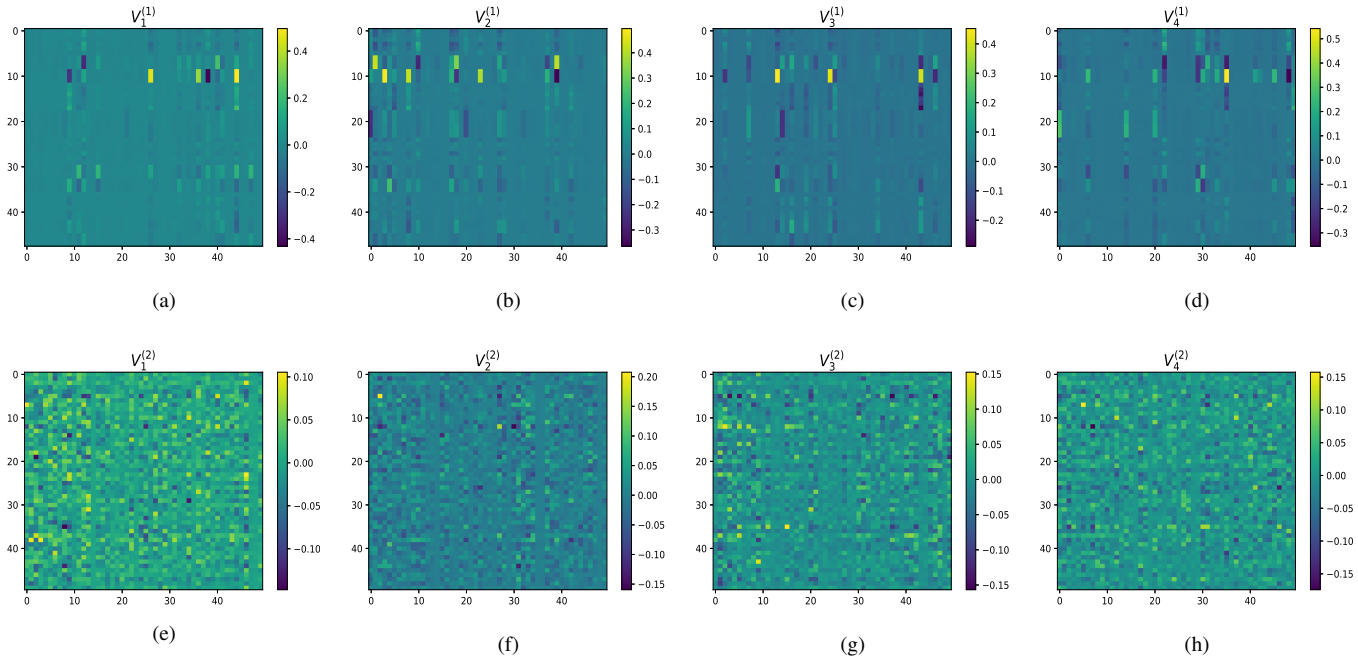

Figure 8: Motor dataset. (a,b,c,d) The learned weights of the maxout kernel block 1. (e,f,g,h) The learned weights of the maxout kernel block 2. (See Fig. 3)

Table 1: The average accuracy of the proposed deep neural kernel architectures and deep hybrid model [29] on several real-life datasets.

							Proposed Deep Neural-Kernel architectures	
Dataset	N	d	Q	**deep hybrid [29]**	**TROP-ELM [32]**	**LS-SVM [41]**	**Average**	**Maxout**	**CNN**
Sonar	208	60	2	0.77 ± 0.04	0.75 ± 0.03	0.70 ± 0.03	0.77 ± 0.02	0.78 ± 0.01	0.78 ± 0.01
Monk2	432	6	2	1.00 ± 0.00	0.96 ± 0.02	0.95 ± 0.02	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
Balance	625	4	3	0.97 ± 0.02	0.93 ± 0.03	0.94 ± 0.01	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
Australian	690	14	2	0.87 ± 0.01	0.88 ± 0.02	0.83 ± 0.02	0.88 ± 0.01	0.90 ± 0.01	0.88 ± 0.02
CNAE-9	1080	856	9	0.94 ± 0.02	0.89 ± 0.03	0.93 ± 0.01	0.94 ± 0.01	0.95 ± 0.01	0.94 ± 0.01
Digit-MultiF1	2000	240	10	0.98 ± 0.01	0.95 ± 0.02	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.02	0.98 ± 0.01
Digit-MultiF2	2000	216	10	0.97 ± 0.02	0.96 ± 0.01	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.02	0.99 ± 0.01
Titanic	2201	3	2	0.78 ± 0.02	0.80 ± 0.02	0.78 ± 0.02	0.81 ± 0.01	0.84 ± 0.01	0.83 ± 0.01
Magic	19,020	10	2	0.86 ± 0.01	0.85 ± 0.02	0.84 ± 0.01	0.86 ± 0.01	0.87 ± 0.02	0.88 ± 0.01
Motor	58,509	49	11	0.96 ± 0.01	0.89 ± 0.02	N.A	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
Covertype	581,012	54	3	0.88 ± 0.02	0.80 ± 0.03	N.A	0.94 ± 0.01	0.95 ± 0.01	0.94 ± 0.01
SUSY	5,000,000	18	2	0.81 ± 0.01	0.76 ± 0.02	N.A	0.81 ± 0.02	0.82 ± 0.01	0.81 ± 0.01

Note: "N.A" stands for Not Applicable due the large size of the dataset.

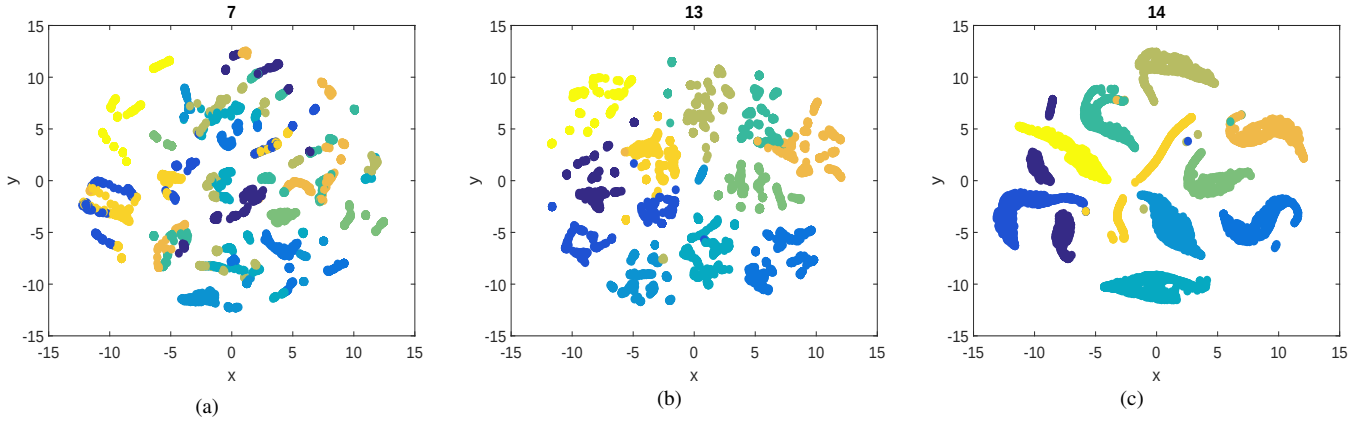

Figure 9: (a,b,c) t-SNE projections of the test data in the hidden layers $h^{(1)}, h^{(2)}$ and the obtained score variables $S(x)$ for the Motor dataset.

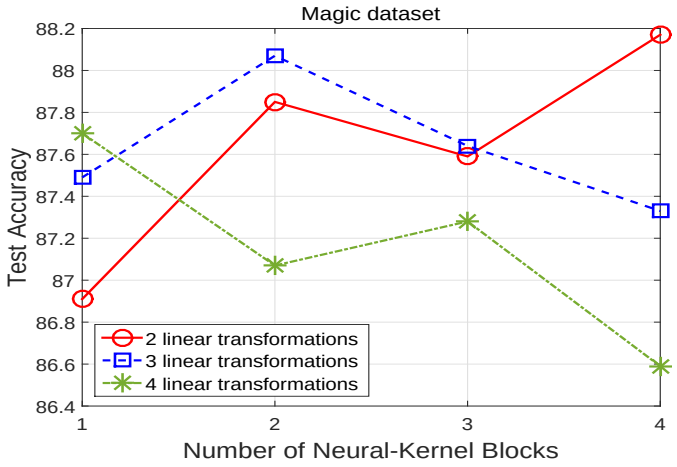

Figure 10: The obtained test accuracy of the proposed deep maxout kernel block architecture with different combinations of the number of blocks and linear transformations within each block for the Magic datasets.

Table 2: Test set error rates of various networks for MNIST dataset

Method	Test error (%)
CNN + Maxout Kernel Blocks	0.48%
CNN + Conv-Kernel Blocks	0.51%
2-Layer CNN + 2-Layer NN [47]	0.53%
Conv. maxout + Dropout [47]	0.45%
Stochastic Pooling [47]	0.47%
DLSVM [42]	0.87%

Table 3: Test set error rates of various networks for CIFAR-10 dataset

Method	Test error (%)
CNN + Maxout Kernel Blocks	11.52%
CNN + Conv-Kernel Blocks	11.61%
Conv. maxout + Dropout [13]	11.68%
Stochastic Pooling [47]	15.13%
DLSVM [42]	11.90%

Table 4: Test set error rates of various networks for CIFAR-100 dataset

Method	Test error (%)
CNN + Maxout Kernel Blocks	38.77%
CNN + Conv-Kernel Blocks	39.21%
Learned Pooling [26]	43.71%
Stochastic Pooling [47]	42.51%
Conv. maxout + Dropout [13]	38.57%

Table 5: Test set error rates of various networks for SVHN dataset

Method	Test error (%)
CNN + Maxout Kernel Blocks	2.41%
CNN + Conv-Kernel Blocks	2.56%
Stochastic Pooling [47]	2.80%
Conv. maxout + Dropout [13]	2.47%

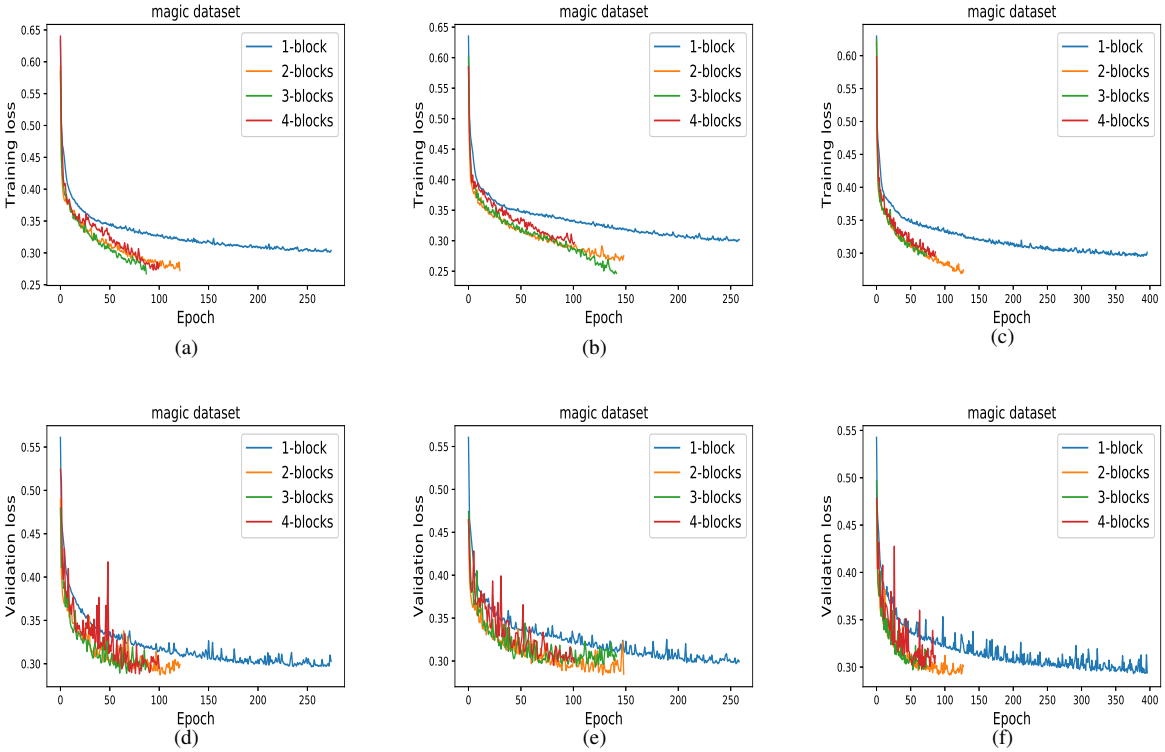

Figure 11: The training and validation loss of the proposed deep maxout kernel blocks architecture with different combinations of the number of blocks and linear transformations within each block for the Magic dataset. (a,d) two linear transformations is used within each block. (b,e) three linear transformations is used within each block. (c,f) four linear transformations is used within each block.

average kernel block is a special case of the pointwise convolutional kernel block. The proposed deep architectures are compared with respect to each other, the classical deep hybrid model as well as kernel based models. The validity and applicability of the proposed models are shown on several real-life benchmark datasets. Our future work is devoted to exploring different strategies for extending the proposed framework to semi-supervised and transfer learning.

Acknowledgments

This work was partially supported by the Postdoctoral Fellowship of the Research Foundation-Flanders (FWO: 12Z1318N). Siamak Mehrkanoon is an assistant professor at the Department of Data Science and Knowledge Engineering, Maastricht University, The Netherlands.

References

[1] Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P., 2014. Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830 .

[2] Aslan, Ö., Zhang, X., Schuurmans, D., 2014. Convex deep learning via normalized kernels, in: Advances in Neural Information Processing Systems, pp. 3275–3283.

[3] Asuncion, A., Newman, D.J., 2007. UCI machine learning repository.

[4] Belanche, L., Costa-jussa, M., 2017. Bridging deep and kernel methods, in: in: Proceedings of the 25th European Symposium on Artificial Neural Networks, Computational Intelligence and machine Learning (ESANN), 2017, pp. 1-10., pp. 1–10.

[5] Bengio, Y., Courville, A., Vincent, P., 2013. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence 35, 1798–1828.

[6] Bergstra, J., Bengio, Y., 2012. Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 281–305.

[7] Bishop, C.M., Nasrabadi, N.M., 2006. Pattern recognition and machine learning. volume 1. Springer New York.

[8] Cho, Y., Saul, L.K., 2009. Kernel methods for deep learning, in: Advances in neural information processing systems, pp. 342–350.

[9] Clevert, D.A., Unterthiner, T., Hochreiter, S., 2015. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 .

[10] Damianou, A., Lawrence, N., 2013. Deep gaussian processes, in: Artificial Intelligence and Statistics, pp. 207–215.

[11] Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323.

[12] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets, in: Advances in neural information processing systems, pp. 2672–2680.

[13] Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y., 2013. Maxout networks. arXiv preprint arXiv:1302.4389 .

[14] Gritsenko, A., Eirola, E., Schupp, D., Ratner, E., Lendasse, A., 2016. Probabilistic methods for multiclass classification problems, in: Proceedings of ELM-2015 Volume 2. Springer, pp. 385–397.

[15] He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034.

[16] Hinton, G.E., Salakhutdinov, R.R., 2006. Reducing the dimensionality of data with neural networks. science 313, 504–507.

[17] Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural

computation 9, 1735–1780.
[18] Krizhevsky, A., Hinton, G., 2009. Learning multiple layers of features from tiny images. Technical Report. Master's thesis, Department of Computer Science, University of Toronto.
[19] Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.
[20] Lawrence, S., Giles, C.L., Tsoi, A.C., Back, A.D., 1997. Face recognition: A convolutional neural-network approach. IEEE transactions on neural networks 8, 98–113.
[21] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324.
[22] Lin, M., Chen, Q., Yan, S., 2013. Network in network. arXiv preprint arXiv:1312.4400 .
[23] Maas, A.L., Hannun, A.Y., Ng, A.Y., 2013. Rectifier nonlinearities improve neural network acoustic models, in: Proc. icml, p. 3.
[24] Van der Maaten, L., Hinton, G., 2008. Visualizing data using t-sne. Journal of Machine Learning Research 9, 2579–2605.
[25] Mairal, J., Koniusz, P., Harchaoui, Z., Schmid, C., 2014. Convolutional kernel networks, in: Advances in Neural Information Processing Systems, pp. 2627–2635.
[26] Malinowski, M., Fritz, M., 2013. Learnable pooling regions for image classification. arXiv preprint arXiv:1301.3516 .
[27] Mehrkanoon, S., Suykens, J.A.K., 2014. Large scale semi-supervised learning using KSC based model, in: Proc. of International Joint Conference onNeural Networks (IJCNN), pp. 4152–4159.
[28] Mehrkanoon, S., Suykens, J.A.K., 2016. Scalable semi-supervised kernel spectral learning using random Fourier features, in: IEEE Symposium Series on Computational Intelligence (SSCI), IEEE. pp. 1–8.
[29] Mehrkanoon, S., Suykens, J.A.K., 2018a. Deep hybrid neural-kernel networks using random Fourier features. Neurocomputing 298, 46–54.
[30] Mehrkanoon, S., Suykens, J.A.K., 2018b. Regularized semipaired kernel CCA for domain adaptation. IEEE transactions on neural networks and learning systems 29, 3199–3213.
[31] Mehrkanoon, S., Zell, A., Suykens, J.A.K., 2017. Scalable hybrid deep neural kernel networks, in: in: Proceedings of the 25th European Symposium on Artificial Neural Networks, Computational Intelligence and machine Learning (ESANN), 2017, pp. 17-22., pp. 17–22.
[32] Miche, Y., Van Heeswijk, M., Bas, P., Simula, O., Lendasse, A., 2011. TROP-ELM: A double-regularized ELM using LARS and Tikhonov regularization. Neurocomputing 74, 2413–2421.
[33] Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th international conference on machine learning (ICML-10), pp. 807–814.
[34] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y., 2011. Reading digits in natural images with unsupervised feature learning, in: NIPS workshop on deep learning and unsupervised feature learning, p. 5.
[35] Rahimi, A., Recht, B., 2007. Random features for large-scale kernel machines, in: Advances in neural information processing systems, pp. 1177–1184.
[36] Rudi, A., Rosasco, L., 2017. Generalization properties of learning with random features, in: Advances in Neural Information Processing Systems, pp. 3215–3225.
[37] Shawe-Taylor, J., Cristianini, N., 2004. Kernel methods for pattern analysis. Cambridge university press.
[38] Smola, A.J., Schölkopf, B., 2000. Sparse greedy matrix approximation for machine learning. 17th ICML, Stanford, 2000 , 911–918.
[39] Springenberg, J.T., Riedmiller, M., 2013. Improving deep neural networks with probabilistic maxout units. arXiv preprint arXiv:1312.6116 .
[40] Sriperumbudur, B., Szabó, Z., 2015. Optimal rates for random fourier features, in: Advances in Neural Information Processing Systems, pp. 1144–1152.
[41] Suykens, J.A.K., Van Gestel, T., De Brabanter, J., De Moor, B., Vandewalle, J., 2002. Least squares support vector machines. Singapore: World Scientific Pub. Co.
[42] Tang, Y., 2013. Deep learning using linear support vector machines. arXiv preprint arXiv:1306.0239 .
[43] Vapnik, V., 1998. Statistical learning theory. Wiley.
[44] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A., 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research 11, 3371–3408.
[45] Williams, C., Seeger, M., 2001. Using the Nyström method to speed up kernel machines, in: Proceedings of the 14th Annual Conference on Neural Information Processing Systems, pp. 682–688.
[46] Xu, B., Wang, N., Chen, T., Li, M., 2015. Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853 .
[47] Zeiler, M.D., Fergus, R., 2013. Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:1301.3557 .