Enhancing deep neural networks via Multiple Kernel Learning

Ivano Lauriola[a,b], Claudio Gallicchio[c], Fabio Aiolli[a]

[a] *University of Padova - Department of Mathematics*
Via Trieste, 63, 35121 Padova - Italy
[b] *Fondazione Bruno Kessler*
Via Sommarive, 18, 38123 Trento - Italy
[c] *University of Pisa - Department of Computer Science*
Largo B. Pontecorvo, 3, 56127 Pisa - Italy

Abstract

Deep neural networks and Multiple Kernel Learning are representation learning methodologies of widespread use and increasing success. While the former aims at learning representations through a hierarchy of features of increasing complexity, the latter provides a principled approach for the combination of base representations. In this paper, we introduce a general framework in which the internal representations computed by a deep neural network are optimally combined by means of Multiple Kernel Learning. The resulting ensemble methodology is instantiated for Multi-layer Perceptrons architectures (both fully trained and with random-weights), and for Convolutional Neural Networks. Experimental results on several benchmark datasets concretely show the advantages and potentialities of the proposed approach.
Keywords: Deep Neural Networks, Deep Learning, Multiple Kernel

Email addresses: `ivano.lauriola@phd.unipd.it` (Ivano Lauriola), `gallicch@di.unipi.it` (Claudio Gallicchio), `aiolli@math.unipd.it` (Fabio Aiolli)

Learning, Ensemble Learning

1. Introduction

In recent years, the interest in studying deep neural network architectures [1] is literally exploding. This interest is mainly driven by recent technological achievements [2], and by the great success in real-world problems, with relevant examples of application domains provided, e.g., by image processing [3, 4], speech recognition [5] and bioinformatics [6]. From a computational point of view, the processing carried out by deep neural models exploits the sequential composition of multiple non-linear hidden layers, enabling the development of progressively more abstract representations of the entities involved in the addressed learning problem. Typically, in this kind of learning models, only the representation computed in the highest hidden layer is used to feed an output level, while the intermediate representations of previous layers are considered just as preparatory for the creation of progressively more complex concepts that are useful for the final output computation.

However, despite the empirical performance of deep architectures on large-scale problems, there are no guarantees about the quality of the last hidden representation. The learning process may fail, or it could not provide an optimal representation, especially in the case of small- and medium-sized problems. Inspired by recent results on Multiple Kernel Learning (MKL) [7, 8], we claim that the intermediate hidden representations can provide different points of view of the main task. Moreover, they contain useful information in addition to prepare the next hidden representation, and their principled combination can improve the quality of the representation, and the performance of the system.

In this paper, we empirically show that, focusing on medium-sized problems (i.e., featured by a number of training examples ranging from several hundreds to tens of thousands), intermediate representations of a deep Feed-forward Neural Network (FFNN) can be exploited to enrich the representation built on the latter hidden layer, improving the effectiveness of the network. Most importantly, we propose an ensemble method to optimally combine the hidden layers' representations by means of the MKL paradigm. The combination mechanism optimizes a quality measure of the final representation rather than an empirical loss as in the case of common neural networks. In this case, this quality measure is the margin between the classes. The proposed method, named KerNET, is experimentally evaluated on two well-known neural architectures, namely the Multi-layer Perceptron (MLP), and the Convolutional Neural Network (CNN). Moreover, while typical approaches in the context of deep FFNNs involve the use of stochastic gradient descent-based algorithms to tune the weights on the connections between all the layers, recently a randomized approach to the design of neural architectures is gauging increasing interest [9, 10]. In this context, a successful methodology consists in the use of MLP-like architectures in which all the internal weights are left untrained after initialization, leaving the training algorithms to operate only on the connections pointing to the output layer.

While this approach routes back to early works in the area of neural networks [11, 12], in the last years it has been popularized under several forms and names, including Random Vector Functional Link [13], Extreme Learning Machine [14], No-Prop [15], and Stochastic Configuration Network [16]. Abstracting from the peculiarities of the different forms reported in literature, we refer to the untrained, random setting of internal weights in a MLP-like architecture as Random-Weights Neural Network (RWNN).

The empirical analysis provided in this paper aims at showing the concrete benefit of the proposed KerNET framework adopted in synergy with MLPs, RWNNs and CNNs, and assessed against several baselines on medium-sized benchmark datasets from several real-world domains including bio-medical and digit recognition.

This paper is organized as follows. Background and preliminary concepts on MKL and related approaches exploiting kernel methods in synergy with deep FFNNs are recalled in Section 2. The motivation and design of the proposed framework are presented and discussed in Section 3. Section 4 describes the experimental assessment carried out to evaluate the KerNET method against several baselines. Finally, Section 5 draws conclusions.

2. Background

In this section we briefly recall the fundamental concepts and the literature background related to the proposed KerNET framework. Specifically, Section 2.1 gives a basic background on Multiple Kernel Learning, while Section 2.2 summarizes previous work about the synergy between kernel methods and deep neural networks.

2.1. Multiple Kernel Learning

Kernel machines constitute a framework of machine learning algorithms widely used in the literature. A kernel machine comprises two parts [17]. The first element is a general-purpose learning algorithm whose solution is expressed by dot-products between training examples. The second element is the kernel function which defines the implicit representation of training data. A kernel function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a positive semi-definite function which corresponds to the dot-product of the input vectors in a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{K}$. In other words there exists a function $\phi : \mathcal{X} \rightarrow \mathcal{K}$ which maps data from the input space $\mathcal{X}$ to the kernel space $\mathcal{K}$ such that $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle$, with $\boldsymbol{x}, \boldsymbol{z} \in \mathcal{X}$.

The most important step when dealing with kernel methods is the choice of the kernel function. Usually a validation procedure is used to select the most suitable kernel for a given task from a set of predefined functions. Several methods have been recently proposed to learn the kernel function from data directly. A popular kernel learning paradigm is Multiple Kernel Learning (MKL), whose purpose is to learn the kernel as a principled combination of base (or weak) kernels [7, 8]. In this paper, convex combinations of base kernels have been considered, i.e. linear non-negative combinations whose weights vector has a fixed 1-norm, that is

$$k_{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{z}) = \sum_r \mu_r k_r(\boldsymbol{x}, \boldsymbol{z}) = \sum_r \mu_r \langle \phi_r(\boldsymbol{x}), \phi_r(\boldsymbol{x}) \rangle, \quad \mu_r \geq 0, \| \boldsymbol{\mu} \|_1 = 1, \quad (1)$$

where k_r is the kernel function based on the r-th mapping ϕ_r, and $\boldsymbol{\mu}$ is the weight vector that the algorithm learns. Any type of kernel function can be virtually combined, such as polynomial or RBF kernels for real-valued data [8], conjunctive/disjunctive kernels for boolean data [18], and sub-structure based kernels for strings and graphs datasets [19]. In this work, kernels generated from deep neural networks have been used. The detailed procedure is reported and discussed in Section 3.

Other methods exist to combine or to aggregate different information, such as the popular Canonical Correlation Analysis (CCA), which aim to find a linear combination of two or more [20] matrices (representations) that have maximum correlation with each other. Differently from those methods, MKL tries to maximize a quality measure leveraging all facets of the available views rather than maximizing a correlation between all input kernels.

2.2. Related work

Despite the widespread use and the great attention in the machine learning community, theoretically grounded approaches to effectively construct and regularize deep neural networks still need to be explored [21]. To fill this gap, in this paper, we propose a novel approach stemming from the idea of combining the expressive power of neural networks with the robustness of theoretically grounded kernel methods. In this section we give a brief overview of the main contributions in the recent literature that showed the potentiality of neural networks - kernel synergy.

A first way to combine neural networks and kernel methods consists of pipelining the operation of the two components, using the kernel machine on the top of the neural network. As a primer example, authors of [22] have proposed an ensemble method for multimodal sentiment analysis. In that work, features extracted from audio and video sources by means of CNNs are combined through MKL, leading to a significant improvement with respect to standard architectures. Other ensemble systems comprising neural networks and simple SVMs have been also widely used in real-world applications, as is the case of neuroimaging [23] , fault detection [24], and intrusion detection [25]. Even in these cases, the network deals with the feature extraction, and the SVM performs the classification.

Different newsworthy techniques apply some kernelized layers on the top of a classical deep neural network, as is the case of Deep Fried convnet [26]. The Deep Fried convnet replaces the fully connected layers on the top of a CNN with an Adaptive Fastfood transform, which is a generalization of the Fastfood transform for approximating kernels [27]. Another similar method has been proposed in [28]. It consists of a Gaussian kernel applied to the top of a neural network, whose parameters are learned during the optimization of the network. Kernel Activation Functions (KAFs) represent a further example of cooperation between neural networks and kernel methods [29]. Briefly, in the KAF setting the activation functions are modeled as kernels. The KAF and the network parameters are usually learned simultaneously. Usually, fixed kernel families are used for this purpose, such as the popular RBF. Furthermore, authors in [30] have proposed a method to learn these functions directly from data, exploiting the MKL framework. Kernel approaches have been also explored for the purposes of neural networks pre-training. A relevant instance is given in the context of graph processing by the work in [31], in which graph kernels are used to pre-train a siamese network for graph classification, showing promising results. Finally, we note that the concept (and benefit) of combining different representations computed by neural networks has been explored in a recent contribution [32]. In this case, authors have extracted different feature sets by means of multiple neural networks, and then they have defined a joint representation which is integrated into the framework of Conditional Random Field for sequence labeling tasks.

Differently from all the above mentioned works, in this paper the kernel approach is used to combine the internal representations developed by the different layers of an individual deep neural network architecture.

3. The KerNET framework

Deep neural networks rely on a stacked sequence of non-linear mappings which provide an increasingly complex representation of input data. The computation performed by such model can be mathematically described in terms of a non-linear function $\phi_{net} : \mathcal{X} \to \mathcal{Y}$ that maps the input data from the input space $\mathcal{X}$ into the output space $\mathcal{Y}$. In the case of a general deep FFNN, the function ϕ_{net} can be defined as a composition of base mapping functions:

$$\phi_{net}(\boldsymbol{x}) = \psi_{out} \circ \psi_l \circ \cdots \circ \psi_2 \circ \psi_1(\boldsymbol{x}), \tag{2}$$

where ψ_1 maps the input into the 1-st hidden layer, and each successive ψ_r operates the non-linear transformation from hidden layer $r-1$ to the hidden layer r, i.e: $\psi_r(\boldsymbol{x}) = g(\boldsymbol{W}_r\boldsymbol{x}+\boldsymbol{b})$, where $\boldsymbol{W}_r$ is the weight matrix, $\boldsymbol{b}$ is the bias vector and g is a non-linear elementwise function. This notion can be easily extended to more complex types of layers, such as convolutional layers, where the matrix multiplication is replaced by the convolution operator. Finally, ψ_{out} maps the representation from the highest layer to the output. The symbol l refers to the number of hidden layers in the architecture.

Let ϕ_{net} be a FFNN with $\psi_1, \dots, \psi_l$ base mappings. Based on the definition of the network described in eq. 2, the hidden representation of an example $\boldsymbol{x}$ at the hidden layer r can be defined as the composition of the first r base mapping functions, i.e.:

$$\phi_r(\boldsymbol{x}) = \psi_r \circ \psi_{r-1} \circ \cdots \circ \psi_1(\boldsymbol{x}). \tag{3}$$

The same representation can be recursively and efficiently computed as:

$$\phi_r(\boldsymbol{x}) = \begin{cases} \psi_r \circ \phi_{r-1}(\boldsymbol{x}) & r > 0 \\ \boldsymbol{x} & r = 0 \end{cases}$$

Conventionally, $\phi_0(\boldsymbol{x}) = \boldsymbol{x}$ denotes the input representation. This process determines a pipelined computation, as graphically illustrated in Figure 1 in the context of the above introduced notation. In particular, arrows in the upper part indicate the operation of the base mapping functions implemented by the successive hidden layers in the neural architecture, i.e., $\psi_1, \dots, \psi_l$, followed by the output mapping ψ_{out}. Such mapping functions are parametrized by the weight matrices $\boldsymbol{W}_1, \dots, \boldsymbol{W}_l, \boldsymbol{W}_{out}$. The computation performed through the successive applications of the base mappings determines the development of internal neural representations at the hidden layers, indicated in the lower part of Figure 1, i.e. $\phi_1, \dots, \phi_l$, and where the initial (i.e., input) representation ϕ_0 is indicated as well.

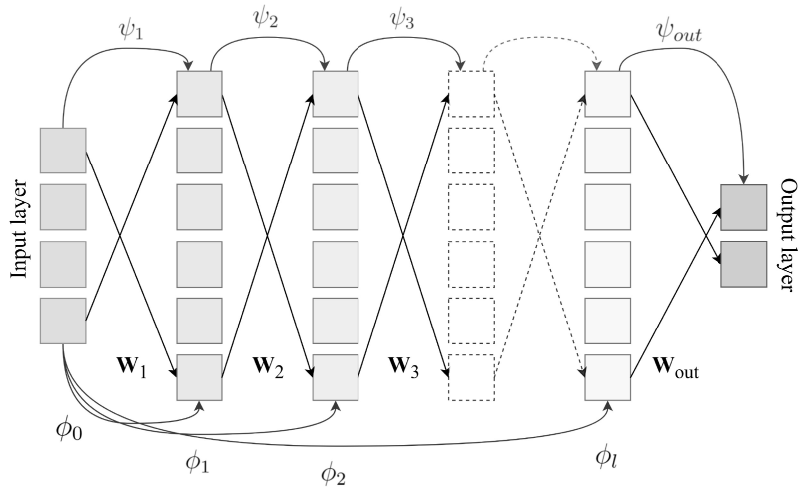

Figure 1: The architecture of a neural network described as a composition of non-linear functions that relates the input to the output layer. ϕ_r is the inner representation at the r-th hidden layer, developed through a stacked sequence of non-linear transformations $\psi_1, \dots, \psi_r$. $\boldsymbol{W}$ are the weights matrices which define the transformations.

Typically, deep FFNNs perform the classification by using only the representation computed in the last hidden layer of the architecture, i.e. $\phi_l(\boldsymbol{x})$. However, the previous intermediate representations, typically neglected at the stage of output computation, provide a rich pool of different viewpoints on the input data that we aim at fruitfully exploit.

Inspired by the extensive literature on MKL (see Section 2.1), we claim that the combination between all of the hidden representation simultaneously rather than the last one can improve the performance and the generalization capability of a neural network. To this end, we propose the KerNET framework, a simple yet effective general ensemble to combine all of the hidden representation that a neural network learns via MKL.

The KerNET framework generally comprises two distinct phases. At first, a deep FFNN is trained. This phase possibly includes a model selection step to select the best hyper-parameters configuration. Then, each intermediate representation ϕ_r, including the input one, ϕ_0, is used to build a base kernel $k_r(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi_r(\boldsymbol{x}), \phi_r(\boldsymbol{z}) \rangle$. Finally, base kernels are combined by means of a MKL algorithm according to the eq. 1.

The resulting representation relies on a richer feature space, where the first layers are involved in the classification step as well as the last one, according to their contribution in the combination. The weights vector $\boldsymbol{\mu}$ that the MKL algorithm learns defines the contribution of base representations, and it is selected by optimizing a quality criteria of the resulting representation. The pseudo code of the ensemble is shown in alg.1.

Section 3.1 minutely describes the instances of the KerNET ensemble with

```
Algorithm 1: KerNET
  Input:
  $\boldsymbol{X}, \boldsymbol{y}$: training examples and their labels
  Output:
  $k_{\boldsymbol{\mu}}$: the combined kernelized representation.
1 begin
2     a model selection phase is applied to select the best hyper-parameter
        configuration
3     the base networks $\phi_{net}$ is trained according to the best hyper-parameter
        configuration selected in the previous step
4     intermediate representations $\phi_0, \phi_1, \dots$ are extracted from $\phi_{net}$, where
        $\phi_r(\boldsymbol{x})$ defines the representation of the example $\boldsymbol{x}$ in the hidden layer $r$.
5     hidden representations are translated to base kernels
        $k_i(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi_i(\boldsymbol{x}), \phi_i(\boldsymbol{z}) \rangle$
6     the combined kernel is learned via MKL: $k_{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{z}) = \sum_r \mu_r k_r(\boldsymbol{x}, \boldsymbol{z})$
7 end
8 return $k_{\boldsymbol{\mu}}$
```

the considered architectures, i.e. MLP, RWNN, and CNN, and the extraction process of the intermediate representations. Then, Section 3.2 explains the mechanism used to combine the representations.

3.1. Instances of the framework

The KerNET framework can be virtually applied to any deep FFNN. In this work, two main architectures have been considered, namely the MLP (both fully trained and in a RWNN setting) and the CNN. In the fully trained MLP scenario, the network's weights are learned using stochastic gradient descent, with a model selection procedure to select the best hyper-parameter configuration. Then, the intermediate representations developed by the hidden layers can be easily extracted from the network. Each hidden layer r

contains a feature vector $\phi_r(\boldsymbol{x})$ associated to the input example $\boldsymbol{x}$ (see eq. 3), which can be easily used for the kernel computation. Figure 1 describes this process. The same pipeline can be applied to the RWNN, where training is restricted only to the output layer, and the translation process from intermediate representations to kernels does not change.

In the case of CNN, there are some small clarifications to be done with respect to the MLP/RWNN setup. The output of a convolution layer is not a single vector as is the case of the previous architectures, but it is a tensor whose shape depends on the dimensions of the input image times the number of filters. As such, after the training of the network, intermediate representations are flattened to build kernels. This flattening operation, which removes the structural and the spatial information of an image, is not a limitation. Indeed, flattening layers are usually applied to the network after a stacked sequence of convolutions. Furthermore, CNNs include a variety of different layers besides convolutions, such as Pooling, Dropout, and Dense layers. The case of Dense layers reflects the same translation process from representations to kernels described for the MLP. However, we decided to not consider the intermediate outputs from Pooling and Dropout layers to not inject further intricacy. In accordance with the notation previously defined, we denote as $\phi_r(\boldsymbol{x})$ the intermediate representation developed at the r-th convolutional or dense layer of the network. The KerNET process with CNN as base network is depicted in Figure 2.

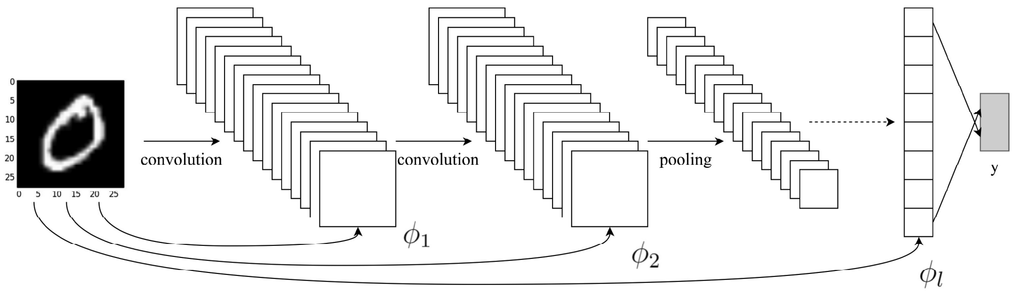

Figure 2: A CNN consists of a stacked sequence of Convolution, Pooling, and Dense layers. The output of convolutions and dense layers define intermediate representations ϕ_r. As in the case of a deep FFNN illustrated in Figure 1, the intermediate representations are combined in the KerNET framework via MKL.

3.2. The combining mechanism

As introduced in Section 3, a MKL algorithm is used to find the optimal convex combination of base representations. Among the plethora of MKL algorithms available in the literature, EasyMKL [8] has been considered in this work. The algorithm has the advantage of being an efficient and scalable MKL algorithm, which can be easily applied to medium-sized datasets. Moreover, the algorithm has recently proven its effectiveness on several tasks and real-world problems, such as hyper-parameters selection [33, 19], Alzheimer's disease detection [34], and bio-medical entity recognition [35]. Briefly, the algorithm is a binary classifier which learns the convex kernels combination that maximizes an approximation of the distance between the convex hulls of the positive and negative classes.

However, the MKL is not the only paradigm to combine these base representations. A similar combination mechanism can be provided by a further neural architecture which substitutes the KerNET ensemble, and that can be studied as baselines for comparison. This network reflects the main deep structure of the former networks, i.e. it consists of a stacked sequence of Dense layers, or Convolutions and Pooling. The main difference with classical architectures is that all of the layers are simultaneously connected to the output. This corresponds to a "virtual" dense layer placed before the output, and that consists of the concatenation of all previous layers, including the input one. In this way, the deep architecture with representations of increasing complexity is preserved, but the output computation, i.e. the final prediction, is allowed to exploit the information that comes from the whole network. This architecture is depicted in Figure 3.

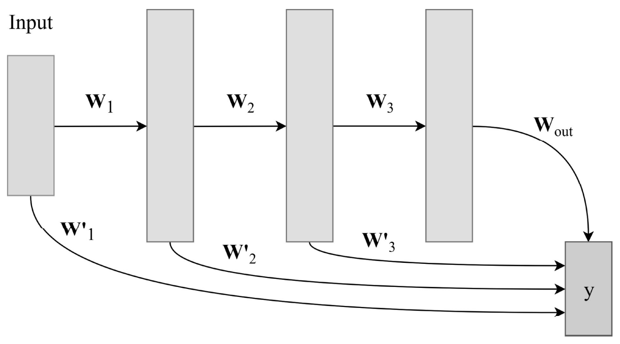

Figure 3: A neural network architecture to combine intermediate representations.

A similar architectural design has been proposed in the case of Fully Convolutional Networks (FCN) [36] for semantic segmentation. There, intermediate representations have been combined to improve the overall results. However, there are some differences between the (baseline) architecture in Figure 3, and the FCN. Firstly, FCN combines layers by element-wise addition rather than concatenation. Secondly, they combine pooling layers rather than flatten convolutions. Finally, while the FCN approach has been introduced originally for tasks of semantic segmentation nature, here we consider the case of general classification tasks.

In the remainder of the paper, the architectural setup described in Figure 3 is considered for comparative assessment of KerNET effectiveness. More specifically, we refer this architecture as MLP-ALL, RWNN-ALL, and CNN-ALL, when instantiated with MLP, RWNN, and CNN respectively. Despite the highlighted similarity, it is worth noticing that the MKL step is not limited to combine the intermediate representations, but it finds a combination which maximizes a quality criteria (the margin between classes in this case) of the resulting representation rather than an empirical loss. For completeness, we evaluated both methodologies, neural combination and MKL. Furthermore, we find it useful to notice that the concept of maximizing the margin between classes while combining internal representations brings interesting similarities to the operation of specialized loss functions, such as the contrastive loss [37]. Note, however, that in this work we do not consider the systematic evaluation of loss functions for multiple reasons. Firstly, the modular architecture of KerNET allows us to use different MKL algorithms, objective functions, and quality criteria in addition to the margin. Secondly, the MKL optimizes a global measure, computed on all examples simultaneously, whereas loss functions (typically) act on batches of examples.

4. Experimental assessment

This section describes the various aspects of the experimental analysis conducted in this paper. Specifically, the adopted datasets are introduced in Section 4.1. The implementation details, the validation procedure, and the configuration of the networks are presented in Section 4.2. The setup of the experiments under an empirical comparison perspective is described and discussed in Section 4.3. Numerical results are reported in Section 4.4. Finally, an analysis of the contribution of each layer is given in Section 4.5.

4.1. Datasets

Several benchmark datasets have been used to evaluate the KerNET method. The datasets have been selected with different characteristics to better analyze the behaviour of the proposed ensemble in different stressed conditions. Small- and medium- sized, binary and (single-label) multi-class datasets have been considered, with 528 up to 60000 training examples, and up to 11 classes. These datasets are splice, dna, madelon, satimage, usps, pendigits, and MNIST, and they are freely available on online repositories[1]. The details of these datasets are shown in Table 1. The first datasets have been used to evaluate the effectiveness of the KerNET framework when instantiated with MLP and RWNN models. Conversely, the MNIST dataset has been considered to assess the impact of KerNET with CNNs. Overall, the

[1]`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

adopted datasets enable to outline a broad-spectrum experimental analysis, comparatively conducted under a variety of cases.

In our experiments, a preprocessing phase has been used, where features have been rescaled in range $[0, 1]$, and examples are then normalized, i.e. $\|\cdot\|_2^2 = 1$. The accuracy score has been used to evaluate the performances.

dataset	**# training**	**# test**	**# features**	**# classes**
splice	1000	2175	60	2
dna	2000	1186	180	3
madelon	2000	600	500	2
satimage	4435	2000	36	6
usps	7291	2007	256	10
pendigits	7494	3498	16	10
MNIST	60000	10000	784	10

Table 1: Summary of the characteristics of the adopted datasets.

4.2. Networks settings and model selection

As introduced in the previous sections, three different realizations of the KerNET framework have been analyzed, using MLPs, RWNNs, and CNNs as base networks.

In all the experiments, training data has been split into training (90%) and validation (10%) sets. The validation set has been used to select the best hyper-parameters configuration, and to early stop the training procedure by observing the validation loss. The hyper-parameters of the MLP architecture are the number of hidden layers $l \in \{0, 1, \dots, 10\}$, and the number of neurons per layer $d \cdot m$, where m is the number of input features and $d \in \{0.5, 1, 1.5, 2, 3, 5\}$. In the case of RWNN, the hidden layers' weights have been initialized randomly from a uniform distribution in $[-1, 1]$, and then the corresponding weight matrices have been rescaled to control their Frobenius norm ω. In particular, we considered settings where the same value of ω has been used for all the hidden layers, exploring values of $\omega \in \{0.1, 0.2, 0.5, 1, 2, 3\}$. The value of ω has been selected on the validation set. On the other side, the CNN consists of stacked convolution layers with 32 3×3 filters. A 2×2 MaxPooling has been introduced every two convolutions. A dense layer with 64 neurons has been placed on top of the network. The number of convolution layers, from 1 up to 6, has been selected in validation. Other hyper-parameters have been selected with a preliminary experimentation phase. The loss function applied is the categorical cross-entropy. The activations functions are the sigmoid for MLPs, ReLU for CNNs, and tanh for RWNNs. Softmax activations have been applied to the output layer for MLPs and CNNs, whereas linear activations have been used for the output of RWNNs. As pertains the learning algorithms, for MLPs and CNNs we applied gradient descent with Adam [38] optimization (using Adam default configuration from the Keras library[2]). The output weights matrix of RWNNs has been learned in closed-form via pseudo-inversion [39]. The winner-take-all strategy has been used to elect the final class. For each hyper-parameter configuration, 5 networks guesses have been trained, on which average results (and standard deviations) have been computed. For

[2]`https://keras.io`

the purposes of model selection, the best hyper-parameter setting for each model under consideration has been selected as the one achieving the highest average accuracy on the validation set.

After the validation of the base network, weak kernels have been extracted from intermediate layers, and the EasyMKL algorithm has been applied to learn the optimal kernels combination. EasyMKL has a further hyper-parameter, i.e. λ, which represents a trade off between the maximization of the margin ($\lambda = 0$), and the maximization of the distance between the centroids ($\lambda = 1$). The value of λ has been selected by means of a 5-fold cross-validation from the set $\{0, 0.1, 0.2, 0.5, 0.9, 1\}$. A Support Vector Machine (SVM) has been used as base learner with the learned kernels combination. The C value has been selected in $\{10^i, i = -3 \dots 3\}$. In the case of multi-class problems, the one-vs-one decomposition strategy has been used.

Note that, despite the number of hyper-parameters, the model selection of the ensemble is not excessively expensive. The hyper-parameters of the network are selected without considering the MKL step. The complexity of this model selection phase is consistent with classical neural networks. In the MKL phase, the grid search of the two hyper-parameters (λ and C) can be simplified, for instance by considering a different base learner rather than the popular SVM. Authors in [8] replaced the SVM with the KOMD [40] algorithm, whose hyper-parameter λ is shared with EasyMKL. Moreover, as stated in [33], the combination weights of the MKL algorithm can be effectively learned by using sub-sampling strategies, without a significant decrease in accuracy and making our approach scalable with large datasets. Although the KerNET framework is certainly compatible with the above mentioned simplification schemes, this aspect is left out of the scopes of this paper, which focuses on analyzing the general properties of the approach.

4.3. Empirical comparison

The empirical comparison conducted in this paper relies on three different points. Firstly, the effectiveness of combining all of the intermediate representations of the neural network has been evaluated, showing that a rich pool of representations of increasing expressiveness can cooperate for improving the final classification. Then, the KerNET framework has been evaluated, showing that the combination mechanism and the quality of the final representation plays a crucial role in the resulting representation, and thus the MKL framework can be effectively applied to this purpose. Finally, the robustness of the proposed method with respect to the dimension of the network is analyzed, showing that the ensemble is able to adapt its combination to avoid overfitting. Having said that, the methods and baselines that have been considered in this evaluation are:

- NET: the base network where the output level is fed only by the last hidden layer in the architecture, i.e. MLP, RWNN, and CNN.
- NET-ALL: the extended architecture described in the Figure 3. This model helps understanding the effectiveness of combining multiple intermediate representations rather than using a single one.

- NET-SVM: The base network, but with the output level implemented by an SVM. This further baseline allows us to understand how much the SVM contributes in the MKL setting rather than the combination of neural representations. This baseline can be seen as a special case of the KerNET method, where the latter hidden representation is the only one with a positive weight.

- KerNET: the proposed ensemble, where the EasyMKL algorithm has been used to learn the optimal combination of the intermediate neural representations.

4.4. Results

Table 2 reports the test accuracy obtained when using MLP as base network. As it can be seen, compared to the basic network setup, while MLP-ALL and MLP-SVM do not generally lead to significant performance improvements, the KerNET approach results in performance enhancement on all the considered tasks. Overall, in this setting KerNET achieved the highest test accuracy in 5 over 6 tasks. Note that this performance improvement has only a moderate computational cost. In particular, just to give an idea on the additional cost for the inference phase (i.e., on the test set), the times required[3] are $1.36'$ (MLP), $2.42'$ (MLP-ALL) $0.71'$ (MLP-SVM), $1.92'$ (KerNET) for Splice, and $1.75'$ (MLP), $3.30'$ (MLP-ALL), $3.68'$ (MLP-SVM), and $5.10'$ (KerNET) for Satimage. This latency is mainly due to the

[3]Evaluated on a system equipped by an Intel(R) Xeon(R) CPU E5-2603 0 @ 1.80GHz.

prediction through the MKL algorithm and to MKL implementation[4].

A similar trend is observed when considering RWNN as base network, as indicated by the results reported in Table 3. Also in this case, indeed, we can observe that KerNET is the only approach that generally leads to performance enhancement with respect to the base network, globally resulting in the highest test accuracy in 4 over the 6 considered cases. In this context, KerNET results indicate that a principled combination of the internal neural representations developed in successive layers of a deep architecture is advantageous also in the absence of (or prior to) training of the hidden layers' connections. Finally, comparing the results in Table 3 with those in Table 2, we can see that the performance of RWNN is generally lower than that of MLP. While this is not surprising, given that RWNNs use untrained hidden layers (and hence are featured by a significantly smaller number of trainable weights), we also observe that the performance gap is greatly reduced when RWNNs are used together with KerNET. This side consideration allows us to highlight the further merit of KerNET as an effective way to introduce adaptivity in the context of RWNNs, providing a promising trade-off between efficiency of training algorithms and accuracy.

The advantage of the KerNET framework is confirmed also when adopted in conjunction with CNNs, as is it shown by the test accuracy reported in Table 4 for MNIST. Complementing the already discussed results, values in

[4]`https://github.com/IvanoLauriola/MKLpy`

dataset	MLP	MLP-ALL	MLP-SVM	KerNET
splice	$89.77_{\pm0.54}$	$85.46_{\pm0.19}$	$85.43_{\pm0.14}$	$\mathbf{90.07}_{\pm0.21}$
dna	$94.25_{\pm0.12}$	$93.96_{\pm0.29}$	$94.22_{\pm0.10}$	$\mathbf{95.11}_{\pm0.00}$
madelon	$50.00_{\pm0.00}$	$\mathbf{57.47}_{\pm1.02}$	$50.00_{\pm0.00}$	$55.73_{\pm0.08}$
satimage	$87.50_{\pm0.93}$	$82.58_{\pm1.17}$	$78.84_{\pm0.23}$	$\mathbf{88.63}_{\pm0.15}$
usps	$93.71_{\pm0.43}$	$93.04_{\pm0.13}$	$91.46_{\pm0.28}$	$\mathbf{93.97}_{\pm0.29}$
pendigits	$95.73_{\pm0.35}$	$95.85_{\pm0.24}$	$93.18_{\pm0.78}$	$\mathbf{96.71}_{\pm0.34}$

Table 2: Average test accuracy scores and standard deviation of MLP and derived methods computed on 5 runs. The best result is highlighted in bold font for each task.

dataset	RWNN	RWNN-ALL	RWNN-SVM	KerNET
splice	$84.05_{\pm0.23}$	$82.78_{\pm1.24}$	$83.83_{\pm0.78}$	$\mathbf{85.35}_{\pm0.16}$
dna	$93.61_{\pm0.27}$	$89.61_{\pm1.20}$	$92.28_{\pm0.39}$	$\mathbf{94.47}_{\pm0.38}$
madelon	$50.17_{\pm1.98}$	$51.50_{\pm3.17}$	$56.90_{\pm1.01}$	$\mathbf{58.43}_{\pm0.86}$
satimage	$75.07_{\pm0.46}$	$75.74_{\pm4.35}$	$74.13_{\pm0.38}$	$\mathbf{80.58}_{\pm0.22}$
usps	$88.64_{\pm0.23}$	$58.70_{\pm0.24}$	$\mathbf{93.43}_{\pm0.45}$	$92.18_{\pm0.14}$
pendigits	$92.34_{\pm0.97}$	$87.39_{\pm2.45}$	$\mathbf{95.38}_{\pm0.18}$	$91.63_{\pm0.23}$

Table 3: Average test accuracy scores and standard deviation of RWNN and derived methods computed on 5 runs. The best result is highlighted in bold font for each task.

Table 4 suggest that the introduced approach is generally favourable also in cases in which the performance of the base networks is already very good.

dataset	CNN	CNN-ALL	CNN-SVM	KerNET
MNIST	$99.08_{\pm0.11}$	$99.08_{\pm0.10}$	$99.22_{\pm0.07}$	$\mathbf{99.32}_{\pm0.07}$

Table 4: Average test accuracy scores and standard deviation of CNN and derived methods computed on 5 runs. The best result is highlighted in bold font for each task.

From a general perspective, the results in Tables 2, 3 and 4 provide an empirical evidence of the fact that the aggregation of multiple intermediate neural representations can improve the network performance in some cases, but the way in which the combination is optimized is the key aspect. In other words, the mere quantity of information is not sufficient without control.

One of the desired key points of the proposed ensemble approach is the ability to learn a final representation that reflects the complexity of the problem at hand. When dealing with increasingly complex and deeper neural architectures, the algorithm should be virtually able to adapt the combination weights towards simpler representations, avoiding to fall in overfitting behavior. In order to empirically test the proposed approach against this possibility, we run a further set of ad-hoc designed experiments. Specifically, we considered the KerNET framework instantiated with a MLP architecture of increasing depth, with a number of hidden layers up to 10. To assess the impact of the proposed methodology, we conducted this analysis in comparison to the results achieved by the base MLP network. The experimental conditions were as in previous experiments, with 5 runs performed for each hyper-parameters configuration, and the same approach for model selection (including the same ranges of explored values for the hyper-parameters). In this phase, we focused on a selection of tasks (i.e., splice, dna, usps) chosen to be representative of the variety of considered cases, covering smaller to larger values of training sets, number of input features and number of target classes. The test accuracy achieved by KerNET and base MLP is plotted in Figure 4 for increasing depth of the neural architecture.

As expected, the number of hidden layers is a crucial hyper-parameter for a deep neural network, especially in the case of medium-sized datasets, where the number of learnable parameters should be carefully controlled. Results in Figure 4 clearly show an overfitting behavior of the base MLP

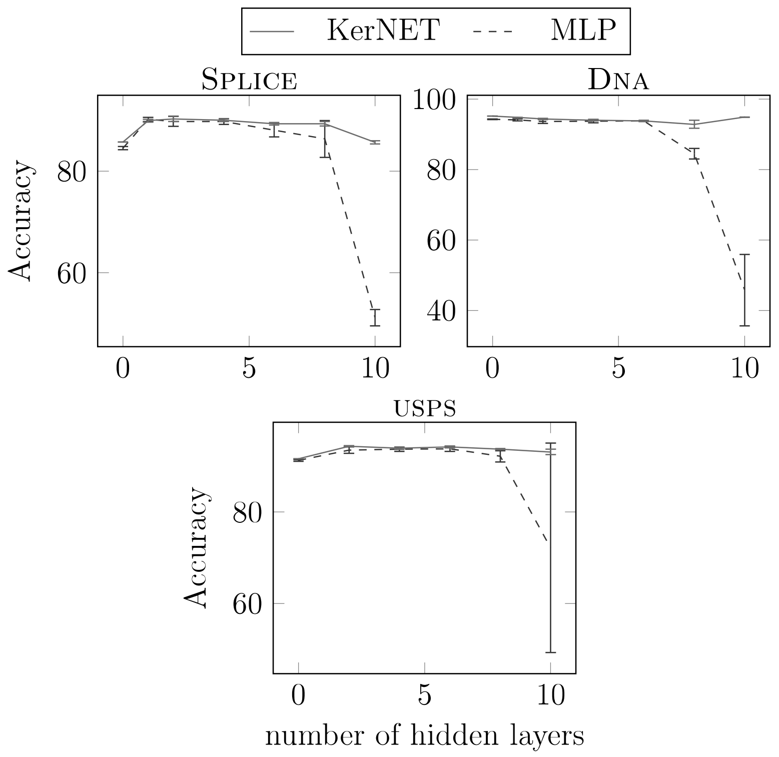

Figure 4: Average test accuracy scores and standard deviation of MLP and KerNET methods computed on 5 runs while increasing the number of hidden layers on selected benchmark datasets.

networks for increasing depth of the architecture, appreciable after 6 layers in all the considered cases. On the other hand, we observe that the KerNET approach is able to drastically reduce the sensibility to the number of layers, appropriately exploiting the intermediate representations to learn a robust combination of layers' activations. This results in the ability to effectively counterbalance the increasing complexity of the base model, avoiding overfitting and achieving higher performance in all the explored settings.

4.5. Evaluation of combination weights

As described in the previous sections, a major advantage of the proposed KerNET method is its ability to emphasize the information coming from different layers, and to select the most suitable abstraction level by learning the weights vector $\boldsymbol{\mu}$ (see equation (1)), which modulates the contribution of each layer. Figure 5 depicts the weights distribution $\boldsymbol{\mu}$ computed by the MKL algorithm for a selection of datasets, which are splice, dna, satimage, and usps. The architecture used in this experiment is the MLP with 8 hidden layer. This experimental setup, including the choice of the datasets and of the MLP architecture, allows us to observe the weights vector distribution in two distinct cases.

What is evident from Figure 5, is that the system perceives the complexity of the task, and it is able to correctly distribute the weights that modulate the representation in a suitable way. In particular, we observe that in the case of splice and dna, where the MLP network is extremely large, the KerNET system gives stronger weights to the representations developed in the lower levels of the architecture (i.e., to the input - where the relation with the output is linear - and to the first hidden layers), preventing overfitting. On the other hand, the more complex satimage and usps tasks leverage deep hidden representations computed by large networks, and the KerNET ensemble gives stronger weights to higher levels representations. Specifically, for satimage the combination weights tend to progressively focus more on the higher layers in the neural architecture, whereas for usps the weights

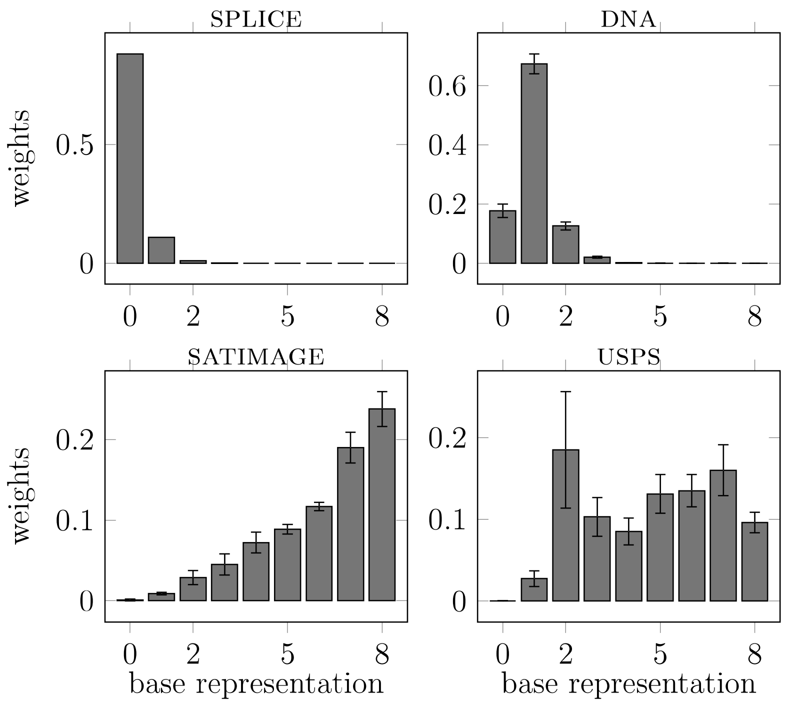

Figure 5: Weights distribution computed by the proposed system (0 = input representation). For multiclass datasets, the average distribution obtained by decomposed tasks is shown.

are spread through the whole network (and especially on layers from 2 to 8), showing that in this case different types of features can be equally important.

5. Conclusions and future work

This paper describes KerNET, a general framework to enrich the representation of a deep neural network by ensembling information developed within the successive layers of the hierarchical architecture. The combination is performed by applying the Multiple Kernel Learning framework which optimally aggregates the hidden representations according to a quality criteria based on maximizing the margin between classes in the feature space. The proposed methodology has been applied to different feed-forward neural models, including Multi-layer Perceptrons, Random-Weights Neural Networks and Convolutional Neural Networks. A wide empirical assessment has been carried out, considering several tasks of different nature. The results of our experimental analysis showed that the principled combination of intermediate representations provided by the KerNET framework is able to adapt to the characteristics of the learning task at hand and of the adopted neural network architecture. Remarkably, our analysis pointed out that the introduced framework has the effect of enhancing the performance of the base models both in terms of accuracy score and in terms of robustness against overfitting in deeper architectural settings. The performance advantages of the introduced framework have been observed for heterogeneous neural network setups, highlighting its potentialities in terms of versatility in the area of deep Neural Networks.

While the introduced KerNET approach already gives an effective way to the design of the output computation in deep networks, it paves the way to further developments along with different research directions. First, exploiting the proposed method, we foresee novel theoretically-grounded approaches to drive the construction of deep models with optimal trade-off between goodness of performance and minimality of the architectures. Moreover, note that the proposed approach could be easily extended to cope with virtually any kind of deep neural network. In this sense, we plan to investigate applications to Deep Recurrent Neural Networks [41]. Finally, extensive applications to large-scale problems are also planned in the near future.

Acknowledgements

The work of I. Lauriola is supported by grant CR30I1_162758 of the Swiss National Science Foundation.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press Cambridge, 2016.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning., in: OSDI, Vol. 16, 2016, pp. 265–283.

[3] J. Wei, Y. Zhang, Y. Xia, M3net: A multi-model, multi-size, and multi-view deep neural network for brain magnetic resonance image segmentation, Pattern Recognition 91 (2019) 366–378.

[4] X. Bu, Y. Wu, Z. Gao, Y. Jia, Deep convolutional network with locality and sparsity constraints for texture classification, Pattern Recognition 91 (2019) 34–46.

[5] T. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. Mohamed, G. Dahl, B. Ramabhadran, Deep convolutional neural networks for large-scale speech tasks, Neural Networks 64 (2015) 39–48.

[6] S. Min, B. Lee, S. Yoon, Deep learning in bioinformatics, Briefings in bioinformatics 18 (5) (2017) 851–869.

[7] M. Gönen, E. Alpaydın, Multiple kernel learning algorithms, Journal of machine learning research 12 (Jul) (2011) 2211–2268.

[8] F. Aiolli, M. Donini, Easymkl: a scalable multiple kernel learning algorithm, Neurocomputing 169 (2015) 215–224.

[9] S. Scardapane, D. Wang, Randomness in neural networks: an overview, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 7 (2).

[10] C. Gallicchio, J. D. Martin-Guerrero, A. Micheli, E. Soria-Olivas, Randomized machine learning approaches: Recent developments and challenges, in: ESANN, i6doc.com, 2017, pp. 77–86.

[11] M. Minsky, S. Papert, Perceptrons, MIT press, 1969.

[12] W. F. Schmidt, M. A. Kraaijveld, R. P. Duin, Feedforward neural networks with random weights, in: 11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems, IEEE, 1992, pp. 1–4.

[13] Y.-H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (5) (1992) 76–79.

[14] G. Huang, G.-B. Huang, S. Song, K. You, Trends in extreme learning machines: A review, Neural Networks 61 (2015) 32–48.

[15] B. Widrow, A. Greenblatt, Y. Kim, D. Park, The no-prop algorithm: A new learning algorithm for multilayer neural networks, Neural Networks 37 (2013) 182–188.

[16] D. Wang, M. Li, Stochastic configuration networks: Fundamentals and algorithms, IEEE transactions on cybernetics 47 (10) (2017) 3466–3479.

[17] J. Shawe-Taylor, N. Cristianini, et al., Kernel methods for pattern analysis, Cambridge university press, 2004.

[18] I. Lauriola, M. Polato, F. Aiolli, Radius-margin ratio optimization for dot-product boolean kernel learning, in: International conference on artificial neural networks, Springer, 2017, pp. 183–191.

[19] F. Aiolli, M. Donini, N. Navarin, A. Sperduti, Multiple graph-kernel learning, in: 2015 IEEE Symposium Series on Computational Intelligence, IEEE, 2015, pp. 1607–1614.

[20] J. Yin, S. Sun, Multiview uncorrelated locality preserving projection, IEEE transactions on neural networks and learning systems.

[21] P. Angelov, A. Sperduti, Challenges in deep learning, in: Proceedings of ESANN, 2016, pp. 489–495.

[22] S. Poria, H. Peng, A. Hussain, N. Howard, E. Cambria, Ensemble application of convolutional neural networks and multiple kernel learning for multimodal sentiment analysis, Neurocomputing 261 (2017) 217–230.

[23] S. Campese, I. Lauriola, C. Scarpazza, G. Sartori, F. Aiolli, Psychiatric disorder classification with 3d convolutional neural networks, INNS Big Data and Deep Learning conference, 2019.

[24] D. Thukaram, H. Khincha, H. Vijaynarasimha, Artificial neural network and support vector machine approach for locating faults in radial distribution systems, IEEE Transactions on Power Delivery 20 (2) (2005) 710–721.

[25] S. Mukkamala, G. Janoski, A. Sung, Intrusion detection using neural networks and support vector machines, in: IJCNN'02, Vol. 2, IEEE, 2002, pp. 1702–1707.

[26] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, Z. Wang, Deep fried convnets, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1476–1483.

[27] Q. Le, T. Sarlós, A. Smola, Fastfood-approximating kernel expansions in loglinear time, in: Proceedings of the international conference on machine learning, Vol. 85, 2013.

[28] A. G. Wilson, Z. Hu, R. Salakhutdinov, E. P. Xing, Deep kernel learning, in: Artificial Intelligence and Statistics, 2016, pp. 370–378.

[29] S. Scardapane, S. Van Vaerenbergh, S. Totaro, A. Uncini, Kafnets: Kernel-based non-parametric activation functions for neural networks, Neural Networks 110 (2019) 19–32.

[30] S. Scardapane, E. Nieddu, D. Firmani, P. Merialdo, Multikernel activation functions: formulation and a case study, arXiv preprint arXiv:1901.10232.

[31] N. Navarin, D. V. Tran, A. Sperduti, Pre-training graph neural networks with kernels, arXiv preprint arXiv:1811.06930.

[32] X. Sun, S. Sun, M. Yin, H. Yang, Hybrid neural conditional random fields for multi-view sequence labeling, Knowledge-Based Systems (2019) 105151.

[33] M. Donini, N. Navarin, I. Lauriola, F. Aiolli, F. Costa, Fast hyperparameter selection for graph kernels via subsampling and multiple kernel learning, in: ESANN, 2017.

[34] M. Donini, J. M. Monteiro, M. Pontil, J. Shawe-Taylor, J. Mourao-Miranda, A multimodal multiple kernel learning approach to alzheimer's disease detection, in: 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2016, pp. 1–6.

[35] I. Lauriola, R. Sella, F. Aiolli, A. Lavelli, F. Rinaldi, Learning representations for biomedical named entity recognition, in: 2nd workshop

on Natural Language for Artificial Intelligence (NL4AI 2018), 2018, pp. 83–94.

[36] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[37] R. Hadsell, S. Chopra, Y. LeCun, Dimensionality reduction by learning an invariant mapping, in: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), Vol. 2, IEEE, 2006, pp. 1735–1742.

[38] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[39] G.-B. Huang, D. H. Wang, Y. Lan, Extreme learning machines: a survey, International journal of machine learning and cybernetics 2 (2) (2011) 107–122.

[40] F. Aiolli, G. Da San Martino, A. Sperduti, A kernel method for the optimization of the margin distribution, in: International Conference on Artificial Neural Networks, Springer, 2008, pp. 305–314.

[41] C. Gallicchio, A. Micheli, L. Pedrelli, Deep reservoir computing: a critical experimental analysis, Neurocomputing 268 (2017) 87–99.