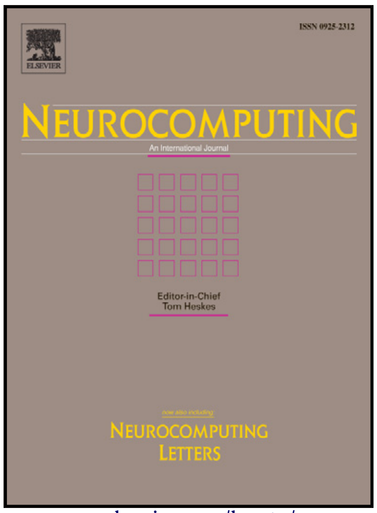


www.elsevier.com/locate/neucom

Deep Extreme Learning Machines: Supervised Autoencoding Architecture for Classification

Migel D. Tissera, Mark D. McDonnell

PII: S0925-2312(15)01132-7
DOI: http://dx.doi.org/10.1016/j.neucom.2015.03.110
Reference: NEUCOM15908

To appear in: *Neurocomputing*

Received date: 29 October 2014
Revised date: 17 March 2015
Accepted date: 17 March 2015

Deep Extreme Learning Machines: Supervised Autoencoding Architecture for Classification

Migel D. Tissera, Mark D. McDonnell

Computational and Theoretical Neuroscience Laboratory,
Institute for Telecommunications Research,
School of Information Technology and Mathematical Sciences
University of South Australia, Mawson Lakes, SA 5095, Australia.

Abstract

We present a method for synthesising deep neural networks using Extreme Learning Machines (ELMs) as a stack of supervised autoencoders. We test the method using standard benchmark datasets for multi-class image classification (MNIST, CIFAR-10 and Google Streetview House Numbers (SVHN)), and show that the classification error rate can progressively improve with inclusion of additional autoencoding ELM modules in a stack. Moreover, we found that the method can correctly classify up to 99.19% of MNIST test images, which surpasses the best error rates reported for standard 3-layer ELMs or previous deep ELM approaches when applied to MNIST. The approach simultaneously offers a significantly faster training algorithm to achieve its best performance (in the order of five minutes on a four-core CPU for MNIST) relative to a single ELM with the same total number of hidden units as the deep ELM, hence offering the best of both worlds: lower error rates and fast implementation.

Keywords: Extreme Learning Machine, Supervised learning, Autoencoder, Classifier, MNIST, CIFAR, SVHN

1. Introduction

In recent years several hardware platforms optimised for neural network implementation have been developed. These implementations range from

Email address: `migel.tissera@mymail.unisa.edu.au` (Migel D. Tissera)

massively-parallel custom-built System-on-Chip (SoC) silicon microprocessor arrays (e.g. SpiNNaker [1]), to analog VLSI processors directly emulating the ion channels of the neurons as leakage currents in CMOS subthreshold region (e.g. Neurogrid [2]). The emergence of these platforms has been accompanied by a parallel effort to develop algorithms which mimic the computational capability of the human brain, particularly in developing synthesised (engineered) neural networks. These algorithms are now utilised for both investigating brain function in computational neuroscience (for example, models of controlling eye position and working memory [3]), and for implementing computing systems in machine learning. In machine learning, an emerging algorithm is the Extreme Learning Machine (ELM) [4, 5, 6], which is known to be relatively fast to train in comparison with iterative training methods, and performs with similar accuracy to Support Vector Machines (SVMs) [7]. This current paper is motivated by recent work that has aimed to produce neuromorphic implementations of ELM [8] and related methods [9, 10], based on hardware that simulates 'spiking neurons'. See [11] for further discussion of neuromorphic implementations. One potential limitation of hardware implementations, or implementations on resources-constrained platforms, is the number of hidden units available for concurrent activation. Our focus is on developing an ELM algorithm that enables the number of hidden units that need to be concurrently activated to be reduced, as well as offering even faster training times, whilst maintaining good performance.

The neural network architecture of standard existing ELM approaches is a three layer feedforward structure. The first layer is the input layer, the second—the hidden layer—is activated by weighted projections of the input to non-linear sigmoid neurons, and the third and final layer is the output, consisting of units with *linear* input-output characteristics (see Figure 1). In ELM, the connection weights between the input and the hidden layer neurons are randomly specified and remain untrained [4, 5]. For example, the input layer connection weights may be uniformly distributed with values between -0.5 and +0.5. This is analogous with neurobiology, in the sense that a negative connection weight inhibits a neuron's activity, and a positive weight excites neuronal activity. After projecting the input to the hidden layer, each hidden-unit's non-linear sigmoid function generates responses. Then using training data, the connection weights between the hidden and the output layer is trained in a single pass by mathematical optimisation. Only this connection weight matrix is altered during training. It is calculated by a least squares regression method such as the Moore-Penrose pseudoinverse

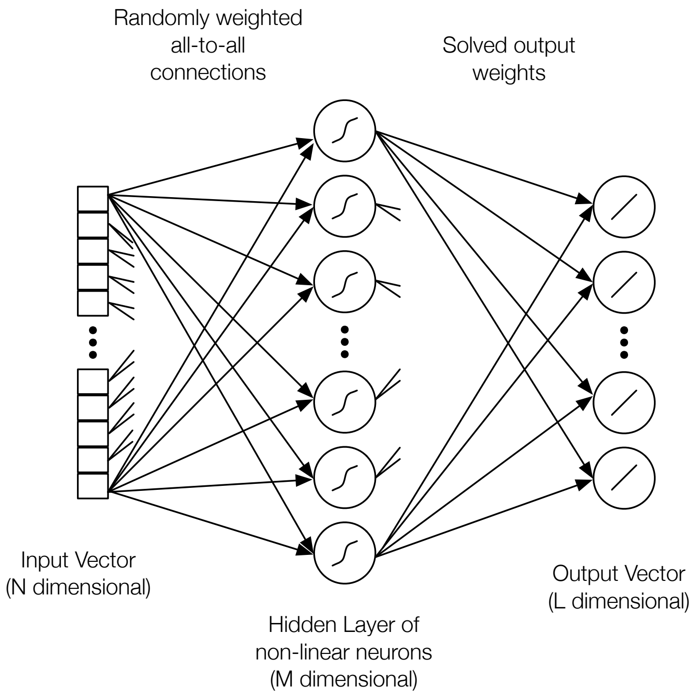

Randomly weighted all-to-all connections
Solved output weights
Input Vector (N dimensional)
Hidden Layer of non-linear neurons (M dimensional)
Output Vector (L dimensional)

Figure 1: The network layout of an Extreme Learning Machine (ELM). The first layer is the input layer, and the second—the hidden layer—is activated by weighted projections of the input to non-linear sigmoid units. The third and final layer is the output and consists of units with linear input-output characteristics.

[12].

The methodology used in the above approach can be summarised as follows:

1. Using random and fixed weights, project an input layer to a hidden layer of sigmoidal units.
2. Using training data, numerically solve for the output weights between the hidden layer and the output layer by calculating the pseudoinverse of the matrix product of the hidden layer values for all training data, and the corresponding desired output responses.

This class of methods has been referred to as Linear Solutions of Higher Dimensional Interlayer (LSHDI) networks [11]. It is a significant deviation from classical artificial neural network training methods. In classical artificial neural networks, the input weights are iteratively trained, rather than computing the output weights only, in a single batch. This interesting property can significantly enhance the efficiency of training since the full and final solution is obtained by mathematical optimisation of a convex function, in one single step. LSHDI methods also can solve significant problems in computational neuroscience simulations of real neurobiological neurons [3]. Although widely accepted and very capable models exist at the single neuron level to mimic neurobiology, until the emergence of LSHDI, there had been no widely applicable method to synthesise (train) a network to solve multiple tasks [13]. This class of methods are now emerging as the core of a generic neural compiler for creating silicon neural systems [1].

One drawback of classical ELMs is the number of neurons in its single hidden layer are typically very large and hence training the network can be computationally impractical, given a large dataset (the algorithm order of complexity for solving for the output weights is $\mathrm{O}(KM^2)$, where K is the number of training points and M is the number of hidden units [14]). It also makes use of batch training, meaning that the network is trained using the entire dataset at once, which usually requires large memory and processing power. In [15, 11, 16] on-line training methods (as opposed to batch training) have been proposed to overcome this, but due to the large number of neurons typically used in the single hidden layer, the training time still largely depends on the size of the network, retaining an $\mathrm{O}(M^2)$ implementation complexity.

In this paper we introduce a different way to address the problem of large hidden layer sizes. Our approach takes inspiration from biology and the recent advances in deep learning architectures [17, 18, 19]. We show that by constructing a deep ELM network as a stack of *supervised* autoencoder ELM modules, and training module by module, the network training time and memory usage can be significantly improved, whilst simultaneously boosting classification error-rate performance *above what can be achieved using a single ELM with the same total number of hidden units.*

There have been several previous approaches to multi-layered ELM networks. Two approaches result in a similar deep network architecture to ours: (i) [20], uses unsupervised autoencoding of hidden-layer activations as a method for constructing a deep network; (ii) [21] introduces a 'random shifts and kernalization' method to define the input to each hidden layer in the network. Another relevant approach is that of [22], which splits the input variables amongst a cascade of multiple ELM modules, with modules beyond the first module also receiving responses from the previous module. In the

Discussion (Section 4) we describe how our approach fundamentally differs from these networks.

The advances made by our algorithm are a result two key factors:

1. selection of the untrained input weights using our recently introduced weight shaping method known as *constrained receptive field ELM* (RF-C-ELM) [14] (which builds upon the Constrained ELM (C-ELM) method of [23]), rather than selecting these weights from a random distribution;
2. we train each ELM module in the stack to both autoencode its input *and* classify it, and then feed both the autoencoding and the classification vectors into the subsequent module.

As we shall show, training ELM modules in the stack using both the training data and the classification results of the previous module leads to an iteratively improved classification of test data with each subsequent module. This enhancement occurs simultaneously with a reduction in the training order of complexity for the same total number of hidden units. Thus our method offers the 'best of both worlds': enhanced classification rates and enhanced runtime complexity.

2. Methodology

In this section, we introduce the methods that we use to construct our deep ELM network.

2.1. Deep ELM architecture and unsupervised training component

We start by describing the flow of an input vector through the network. The input layer takes as input a test or training vector, $\mathbf{X} \in \mathbb{R}^{N\times 1}$. This input layer is connected to the first ELM module's hidden layer (of size M) by an all-to-all weight matrix $\mathbf{W}_{p1} \in \mathbb{R}^{M\times N}$. We write the input vector to this layer as $\mathbf{H}_1 = \mathbf{W}_{p1}\mathbf{X} \in \mathbb{R}^{M\times 1}$. The output of the i–th hidden layer unit is given by the logistic sigmoid function,

$$f[\mathbf{H}_{1,i}] = \frac{1}{1 + \exp\left(-\mathbf{H}_{1,i}\right)}. \quad (1)$$

An output weights matrix $\mathbf{W}_{o1}$ is then multiplied by the hidden-layer responses to produce an approximation of the input, $\hat{\mathbf{X}} \in \mathbb{R}^{N\times 1}$. In order to use K training vectors to train the output weights to perform this *autoencoding*, we form a matrix $\mathbf{A} \in \mathbb{R}^{M\times K}$ in which each column contains the output of the hidden layer $f[\mathbf{H}_1]$ at one training point. Then using the training data itself as another matrix $\mathbf{Y} \in \mathbb{R}^{N\times K}$ in which each column contains training vectors, we solve for $\mathbf{W}_{o1} \in \mathbb{R}^{N\times M}$ that minimises the mean square error between

$$\mathbf{Y}_{\text{predicted}} := \mathbf{W}_{o1}\mathbf{A} \tag{2}$$

and the set of original training images, $\mathbf{Y}$. Similar to supervised training of an ELM carried out elsewhere [5, 20, 14], we solve this problem numerically by finding the solution to the following set of NM linear equations in NM unknown variables comprised from the elements of $\mathbf{W}_{o1}$,

$$\mathbf{Y}\mathbf{A}^\top = \mathbf{W}_{o1}(\mathbf{A}\mathbf{A}^\top + c\mathbf{I}), \tag{3}$$

where c is a regularisation parameter that can be optimised as a hyper-parameter, and $\mathbf{I}$ is the $M \times M$ identity matrix.

This trained weight matrix $\mathbf{W}_{o1}$ converts an input instance, $\mathbf{X}$, into a new vector $\hat{\mathbf{X}}_1 = \mathbf{W}_{o1}f[\mathbf{H}_1]$, which is the autoencoded version of the original data.

We next construct the second module of the deep network. Using a new projection weight matrix $\mathbf{W}_{p2} \in \mathbb{R}^{Q\times N}$, we connect $\hat{\mathbf{X}}$ to the second hidden layer, whose input is $\mathbf{H}_2 \in \mathbb{R}^{Q\times 1}$, where Q is not necessary the same as the first hidden layer size, M. Then the output weights of the second module, $\mathbf{W}_{o2}$, are trained in the same fashion as those of the first module to produce a new auto encoded response, $\hat{\mathbf{X}}_2 := \mathbf{W}_{o2}f[\mathbf{H}_2]$.

The entire process can be repeated numerous times, to form a V-module deep ELM network.

The above explanation provides a procedural description of the sequence of steps required during training. After the entire network has been trained, however, it is natural to combine the two weight matrices $\mathbf{W}_{o1}$ and $\mathbf{W}_{p2}$ to form one single weight matrix $\mathbf{W}_{h12} \in \mathbb{R}^{Q\times M}$, which fully connects the first two hidden layers, i.e.

$$\mathbf{W}_{h12} = \mathbf{W}_{p2}\mathbf{W}_{o1}, \tag{4}$$

and similar for subsequent modules. This is only possible because both the input and output layers of an ELM module are linear.

The approach described above is illustrated in Figure 2.

We next describe the supervised aspect of the autoencoding.

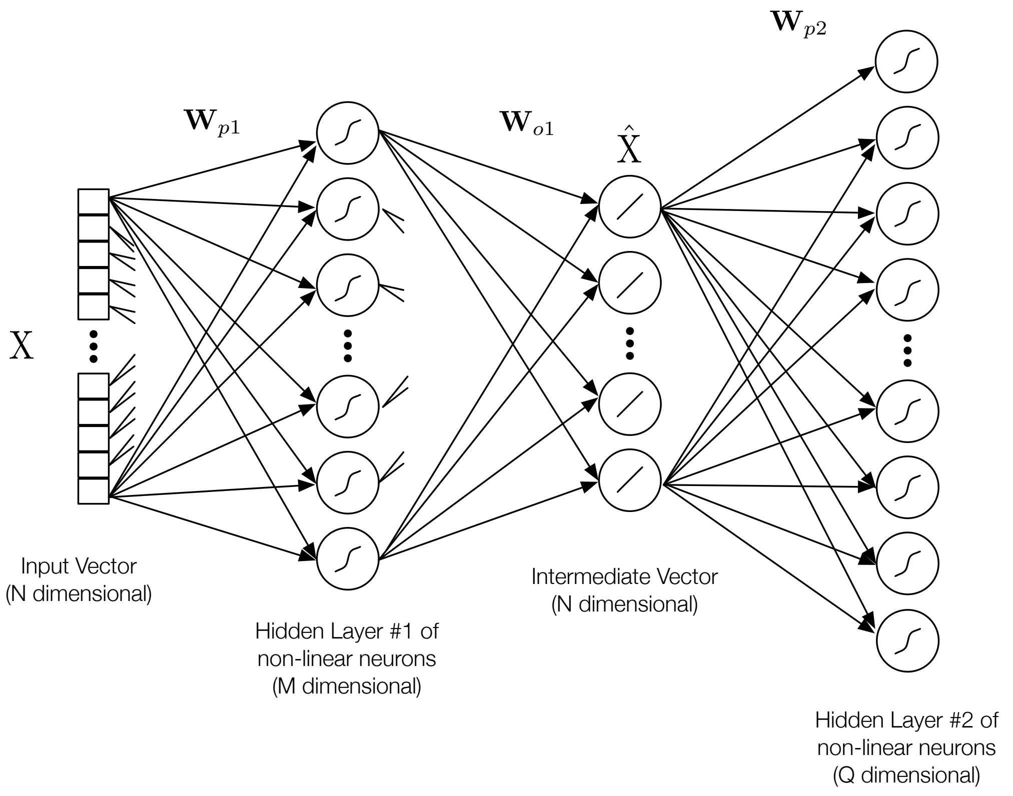


Figure 2: The first two hidden-layers of our supervised deep extreme learning machine network are shown. From left-to-right in feedforward: an N dimensional input vector, $\mathbf{X}$ is projected to the M dimensional first hidden-layer using a random weight matrix $\mathbf{W}_{p1}$. After transformation by the sigmoidal hidden units, the result is multiplied by the output weights to produce a vector $\hat{\mathbf{X}}$ of the same dimensionality as the input. Then the vector $\hat{\mathbf{X}}$ is projected to a Q dimensional second hidden-layer using a random weight matrix $\mathbf{W}_{p2}$. The output weight matrices (e.g. $\mathbf{W}_{o1}$) are obtained using training data by solving Equation (3). We refer to each 3-layer ELM as a *module*; each module contains a single hidden layer, and thus the stack of modules is a deep neural network. Additional modules can be added indefinitely, with a readout of a classification available from each intermediate layer.

2.2. Deep ELM supervised autoencoding: 'label pixels'

In the above description, no supervision or use of training data labels was mentioned, but we now describe how these aspects can be introduced using the same general method. The approach is inspired by that used in *discriminative restricted Boltzmann machines* [24] (see also [25]). Regular restricted Boltzmann machines are generative, and train both forward weights from an input layer to a hidden layer, and backward weights from the hidden layer to the input layer. It has been shown that providing additional elements in the input layer with values that correspond to training labels during training, but are initially zero during testing, enables 'generation' of predictions of the labels of test data in these additional elements following repeated propagation of inputs to the hidden layer and back to the input layer. This method has been shown to provide good classification results on standard benchmarks [24].

Inspired by this idea, one simple approach that we use that is suitable for classification of images is to embed the training labels as pixel values in the otherwise unused borders of training images. More generally if unused borders are not available, or the data is not images, it is trivial to instead simply expand the dimension of the input data by the number of classes, just as in discriminative restricted Boltzmann machines. For the example of the MNIST dataset [26] we can take the first 10 pixels in the first row of the training images, and reset them to zero (if they are not already at zero, which most are). Then we use the label of the respective image, to set the corresponding pixel to 1. For example, for label '0', pixel 1 is set to one; for label 9, pixel 10 is set to one (note that we preprocess all image data in MNIST by rescaling pixel values to the interval $[0, 1]$). The addition of 'label pixels' is illustrated in Figure 3.

We use the resulting combined "images and labels" training set as the target matrix, $\mathbf{Y}$, used to train each autoencoding ELM module in our network. Note that the first module receives as input the images only, without the supervised 'label pixels.' Hence, the matrix $\mathbf{Y}$ contains the labelled training images, but the input to the network's first module, $\mathbf{X}$, is the raw images. In the classification step (i.e. at the output layer of each module), we use the highest valued pixel from amongst the label pixels as the classified output. As in discriminative restricted Boltzmann machines, the values of these pixels tend to converge towards the actual labels of the test data with repeated generation; the architectural difference is that we use a feedforward network to achieve this. For the more general case, we simply use the elements of the prediction vector corresponding to the labels.

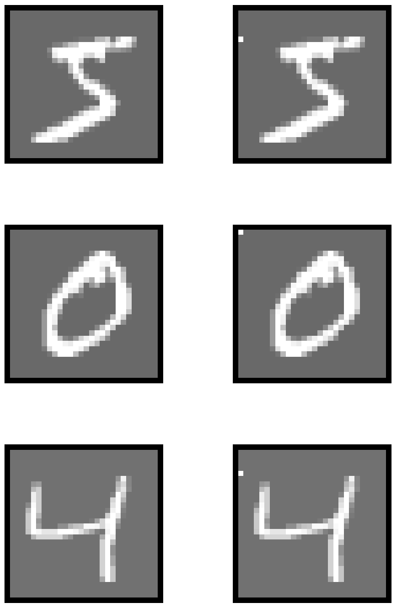


Figure 3: Illustration of one way to introduce supervised 'label pixels' to training data images. Left hand side: Examples of the original MNIST Training images. Right hand side: Combined "image and label" training data. Top Right: Note the 6^{th} pixel has been set to 1 to indicate the label 5, and similar for the other examples. During testing, the autoencoded test images progress from having all the label pixels equal to zero initially to having label pixels generated that represent classifications of the image itself, similar to discriminative restricted Boltzmann machines [24].

2.3. *RF-C-ELM input-weight shaping method*

Our Deep ELM method both significantly improves the network training time relative to a single ELM with the same total number of hidden units (as shown in Section 3 below), and can improve the classification accuracy on well known datasets relative to any other single-module ELM method. Achieving the latter relies on using a method introduced recently elsewhere [14] that extends the approach of [23]. We now briefly overview this method.

In the standard ELM approach, the input weights are initialised randomly, for example uniformly distributed between -0.5 and +0.5. This works reasonably well, but in [27, 23], it was shown to be advantageous to explicitly compute the input weights based on supervised use of the training data.

Additionally, it was shown in [14] that classification performance when using ELMs can be significantly enhanced by restricting the number of input pixels or features that provide input to an ELM hidden layer. In other words, it is advantageous to ensure the input weights vector is sparse. For images, we have found that limiting the non-zero weights to each hidden unit to randomly sized and located rectangular patches, analogous to receptive fields in the mammalian visual system, gives better performance than random selection of which weights are set to zero.

Finally, we have shown in [14] that combining the Constrained ELM (C-ELM) method of [23] with the receptive-fields ELM (RF-ELM) method introduced in [14], is superior to either method alone. This RF-C-ELM method is readily carried out by first calculating the input C-ELM weights according to the algorithm of [23], and then applying receptive field masks to those weights as described in [14].

2.4. Input weights for the label pixels

We found that the best classification performance resulted when we reserved a fraction of the hidden units in each ELM module to receive weighted input only from the label pixels, and all other hidden units to receive weighted input only from the remaining pixels. Since there are only 10 label pixels, we did not use the RF-C-ELM method to choose the weights from label pixels to the hidden units associated with label pixels. Instead, we simply used zero-mean bipolar random weights for these weights and RF-C-ELM for all remaining weights.

3. Experiments

We firstly describe three image classification tasks that we tested our method on, and then present results on each of these benchmarks.

3.1. Classification Databases

3.1.1. MNIST

The main dataset we focus on is MNIST [26], which consists of a total of 70000 handwritten digits (60000 for training and 10000 for testing) that have been converted into 28×28 pixel images. Although originally binary images, the standard dataset is now greyscale due to resizing with interpolation. Hence, the input layer to our deep ELMs is of size $N = 784$. Since we perform autoencoding, the size of the output layer is the same as the input layer, i.e. $N = 784$ in all input and output layers.

3.1.2. CIFAR-10

Like MNIST, the CIFAR-10 image classification task has 10 classes of image [28]. Unlike MNIST, the images are in RGB format, and are of size 32×32 pixels. We treat each of the three channels as adding additional input pixels, and hence have $N = 32 \times 32 \times 3 = 3072$. There are 50000 training images and 10000 test images.

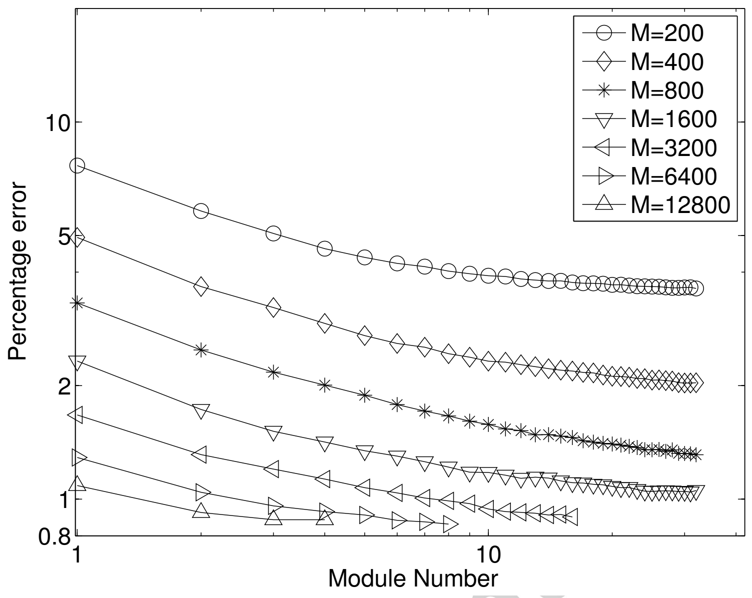
3.1.3. SVHN

SVHN is the Google Streetview House Number database [29]. Like MNIST, the 10 classes correspond to ten numerical digits. Unlike MNIST and like CIFAR-10, each image is RGB and of size 32×32 pixels. We converted each RGB image to greyscale and thus have $N = 1024$. Although the full database has 26032 test images and over 600000 training images, we used only the smaller 'hard' set of 73257 training images.

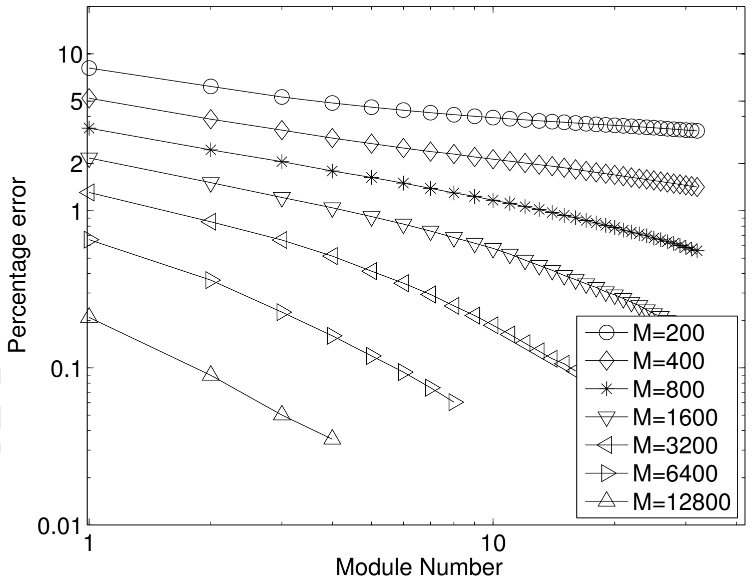
3.2. Results

To begin, Figure 4 shows results for classification error rates for MNIST. Figure 4(a) shows the error rate as the number of modules increases. The most important aspects that be observed are that (i) the error rate generally decreases as the number off modules increases from the single ELM case; (ii) adding more modules for a given value of M can readily surpass the performance of a single ELM with larger M. However, after some point adding more modules leads to diminished returns in the error rate. Increasing the number of modules above the values shown in Figure 4(a) does not enhance performance significantly. This is consistent with data in Figure 4(b), which shows the error rate when the data used for training is classified by the trained deep ELM—note that the error rate for the training data significantly surpasses that on the test data for $M = 12800$, as well as for smaller M as the number of modules increases.

Figure 5 shows the same data from Figure 4(a), but with a rescaling where the error rate is plotted against the total number of hidden units (summed over all modules) rather than than the number of modules. Although the



(a) Test dataset, RF-C-ELM



(b) Training dataset, RF-C-ELM

Figure 4: **Classification error rates on MNIST for deep ELMs for increasing network depth.** All data was calculated as the average over 10 repeats for each condition. Subfigure (a) are results from the 10000 test images. Subfigure (b) are results from applying the trained deep ELM to the 60000 images used for the training. Each trace is for a different number of hidden units in each module, M (each module contains a single hidden layer).

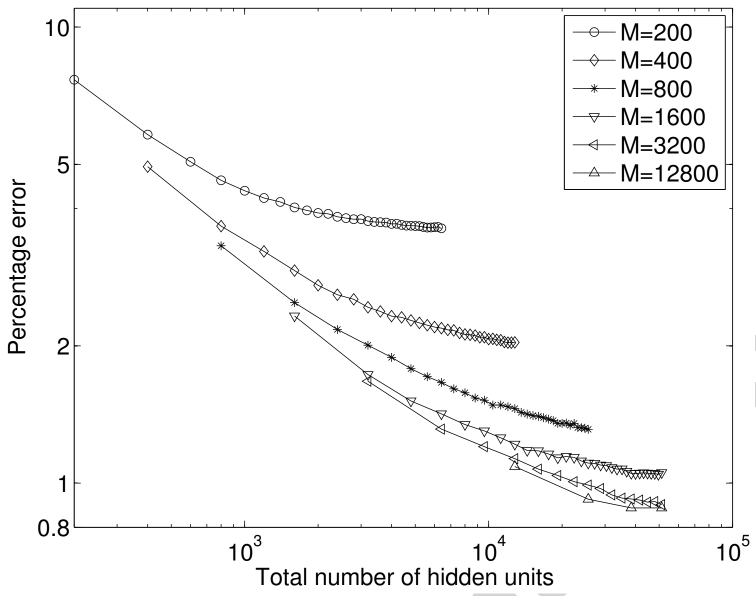


Figure 5: **Classification error rates on MNIST for deep ELMs for increasing total number of hidden units.** The figure shows the same data as in Figure 4, but the X-axis is scaled by the total number of hidden units in each deep ELM. Each trace is for a different number of hidden units in each module, M.

best performance is achieved with fewer modules with more hidden units per module, it is clear there is a very small performance loss in comparison with using more modules and fewer hidden units per module. In fact, we found that the best error rate can actually be achieved with fewer M per module and more modules. This is evidenced in Table 1, which summarises the best results we obtained for the data shown in Figs 4 and 5. The best classification performance we obtained was 99.19% correct on the MNIST test data, which occurred for $M = 6400$ with 8 modules. However, this corresponds to only one error fewer than for $M = 3200$ with 16 modules and $M = 12800$ with 3 modules, and therefore given the random nature of the input layer weights, it can be concluded that smaller M and more modules can achieve best performance.

Given the $O(M^2)$ algorithm for training each layer, these results suggest the deep ELM is advantageous for situations where training time is considered to be an important factor.

To evaluate this quantitatively, Figure 6(a) shows how the mean time

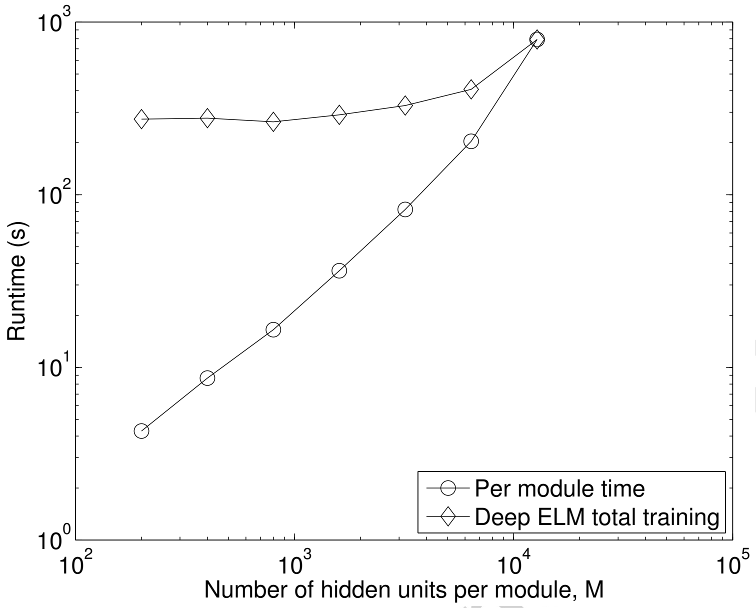
M	Best Result	Num modules for best	Total hidden units for best
200	96.56 %	30	6000
400	98.08 %	31	12400
800	98.80 %	31	24800
1600	99.05 %	25	40000
3200	99.18 %	16	51200
6400	99.19 %	8	51200
12800	99.18 %	3	38400

Table 1: Summary of best classification performance rates (percent of correctly classified test images) on MNIST, for our deep ELM. Data was obtained from a maximum of 32 modules for $M \leq 1600$ or a maximum of 51200 total hidden units for $M \geq 3200$. The best result was obtained from ten repeated generations of all RF-C-ELM input layer weights in each deep ELM.

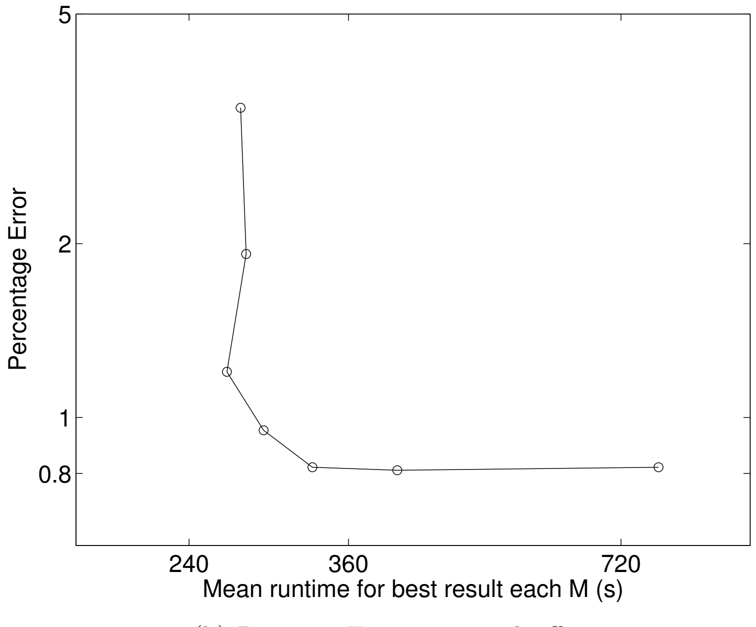
required to train a deep ELM varies with the number of hidden units per module. When the total runtime is divided by the number of layers, the result increases approximately linearly on the log-log axes as the number of hidden units per module, M, increases. This is due to the $O(M^2)$ algorithm required to find the output weights for each module. This subfigure also shows that the total runtime required for deep ELMs with the same total number of hidden units (12800) increases as the number of hidden units per module, M, increases, despite needing to calculate weights for fewer modules as M increases. This result illustrates the $O(VM^2)$ run-time dependency of a V-layer deep ELM. Figure 6(b) further reinforces that good close to optimal performance for the method can be obtained for a deep ELM with smaller M at a significantly save on run-time.

As a comparison to illustrate that deep ELMs can be advantageous on a range of data sets, Figure 7 shows results for the error rate against the total number of hidden units for deep ELMs for the CIFAR-10 and SVHN datasets. In each case, adding more layers generally decreases the error rate up to a point where more layers offers diminishing returns.

Finally, in order to demonstrate that the supervision component of the deep ELM is essential for improved performance with increasing numbers of layers, we show in Figure 8 the error rate against increasing number of modules, when the label pixels are set to zero in all modules for training. In contrast with the data shown in Figure 6, performance does not improve with added modules.

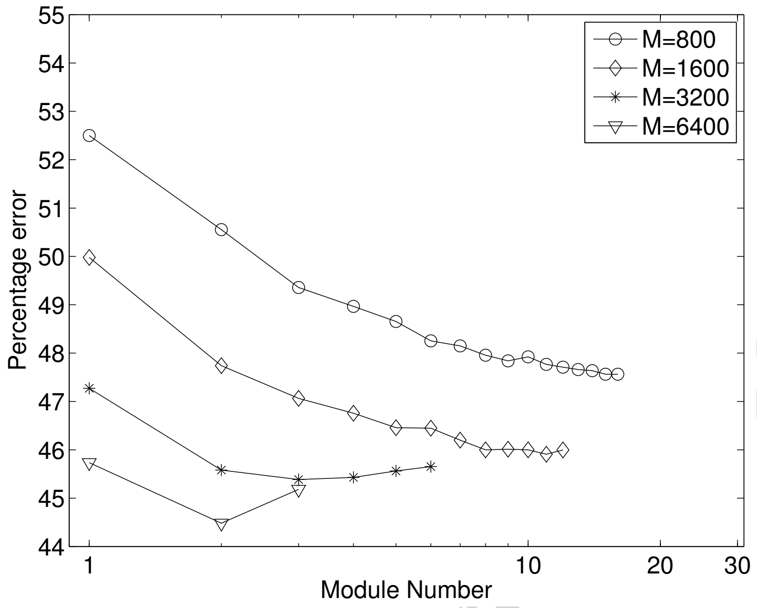


(a) Varying M

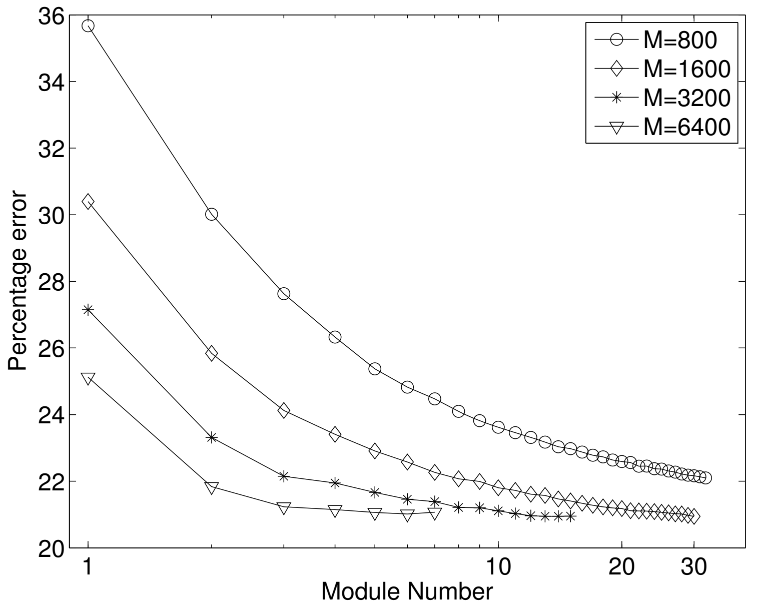


(b) Runtime-Error rate tradeoff

Figure 6: **Indicative run times for deep ELM.** Both subfigures show the training time required prior to producing the data in Figure 4. The total training time is defined as the total time to train a total of 12800 hidden units using between 1 and 64 layers of size from 12800 to 200. We also show in subfigure (a) the total runtime divided by the number of modules. Subfigure (b) illustrates that the runtime required for near optimal error rates can be much smaller when using multiple layers of smaller size.



(a) CIFAR-10



(b) SVHN

Figure 7: **Deep ELM classification error rates on CIFAR-10 and SVHN test data as the total number of hidden units per module varies.** All data was calculated as the average over 3 repeats. Each trace is for a different number of hidden units in each module, M, and each data point is the error rate after $1, 2, \ldots$ modules. In each case the percentage error shown is calculated after each individual module in a deep ELM composed from multiple modules.

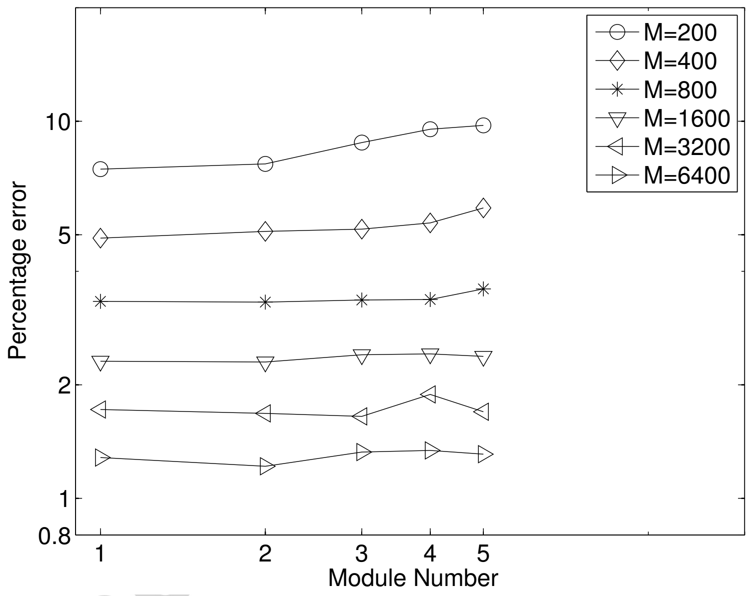


Figure 8: **Classification error rates on MNIST for *unsupervised* deep ELM, as the number of layers increases.** The figure shows that the removal of supervision from the deep ELM by setting all label pixels to zero does not lead to improved performance as the number of modules increases.

4. Discussion

In previous work, a deep ELM structure that exploits autoencoding was proposed [20]. In that method, the initial step is to train an ELM using the training data as the target, without using any labels. Then, the transpose of these trained autoencoding output weights replace the input weights in the ELM. Then, the hidden-layer activations are trained in a similar autoencoding fashion multiple times, before finally projecting into a larger hidden layer to train as the classifier output.

This process is quite different from our autoencoding deep network in three significant ways:

1. we retain untrained input weights in multiple separate ELM modules that combine to form the deep ELM;
2. we autoencode the input to each module's input layer, rather than autoencoding a hidden layer response;
3. we use the label data to train each module's output weights, thus providing supervision to the training.

The deep ELM approach of [21] is similar to ours in that it combines the classification output from one ELM module with a representation of the input data. However, in [21], the original input data is used, whereas we use an autoencoding. Moreover, the method of combining is different; in [21], the output of each ELM module is randomly projected to the dimension of the input data, and then added to it. In contrast, we randomly project the 'label pixel' outputs from an ELM module to an independent set of hidden units in the next ELM module. Moreover, we use the RF-C-ELM method to specify the remaining input layer weights from input representation to hidden layer in each module.

We have found (see Section 3) that the method we propose achieves up to 99.19% correct classification on MNIST test data, which compares favourably with the best results reported for previous deep ELM methods (i.e. 99.03% in [20] and 92.01% in [21]), and the best result for a single ELM without use of data augmentation (99.10% reported in [14]). The performance we achieved in this paper for CIFAR-10 is about 56% correct classification, which although better than the $\sim$43% in [21], remains significantly lower than non-ELM methods that make use of convolutions (amongst other differences) to achieve correct classification rates greater than 90% [30]. Thus, it seems likely that although a deep ELM approach can achieve close to state-of-the-art performance on a relatively easy to classify database like MNIST, substantial innovations that expand the basic principles will be needed to begin to close the gap to state-of-the-art non-ELM methods. One step in this direction has been made by adding a convolutional stage prior to ELM classification [31], which enables correct classification rates on MNIST of greater than 99.5%, on CIFAR-10 of greater than 75% and on SVHN of greater than 96%. Similarly, preprocessing of CIFAR-10 prior to ELM classification has led to reported results in excess of 65% correct classification [32]. In future work, it will be interesting to combine such convolutional and preprocessing approaches with the deep ELM approach described in this paper.

In other previous work, a deep ELM structure was proposed where the input variables are split into disjoint sets, and fed forward into multiple ELM modules/layers [22]. The modules are combined sequentially by feeding the output of module i as an additional input to module $i + 1$. While we also feedforward the response of module i in our own network, we do not split the data; instead, the first layer received all input variables, and subsequent layers receive the autoencoding response of the previous layer, along with its classification response.

Our approach enables a classification to be made for every module of the deep network, with generally improved classification for subsequent modules. There is scope for future work to exploit this feature. For example, the output classification vector for each stage can be assessed in terms of confidence in whether the correct prediction is made. If, for example, one of the prediction vectors shows one class predictions as having a value close to its target value (unity in our approach) and the rest as close to zero, then confidence in a correct prediction will be high, and propagation of data through remaining layers may be unnecessary. There is therefore potential for faster test data classification than would be the case if propagation through all L modules is carried out.

In summary, using our method, we have reduced the network training time compared with a single hidden-layer network of equivalent hidden neurons. This becomes apparent as the number of hidden-layer neurons increases. This method of constructing deep neural networks is potentially favourable for hardware or online implementations, due to the reduced number of hidden units per layer, which could be a benefit to IC designs of limited size. The expected loss of performance could potentially be offset through the use of time-multiplexing that reuses the same circuitry repeatedly to compute the responses of each layer in our deep ELM.

Acknowledgments

Mark D. McDonnell's contribution was supported by the Australian Research Council under ARC grant DP1093425 (including an Australian Research Fellowship).

5. References

[1] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, pp. 652–665, 2014.

[2] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed analog-digital multi chip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, pp. 699–716, 2014.

[3] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems.* The MIT Press, 2003.

[4] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme Learning Machine: Theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.

[5] G.-B. Huang, D. H. Wang, and Y. Lan, "Extreme Learning Machines: A Survey," *International Journal of Machine Learning and Cybernetics*, vol. 2, pp. 107–122, 2011.

[6] E. Cambria and G.-B. Huang, "Extreme learning machines," *IEEE Intelligent Systems*, vol. 28, pp. 30–31, 2013.

[7] G.-B. Huang, "An insight into extreme learning machines: Random neurons, random features and kernels," *Cognitive Computation*, vol. 6, pp. 376–390, 2014.

[8] A. Basu, S. Shuo, H. Zhou, M. H. Lim, and G.-B. Huang, "Silicon spiking neurons for hardware implementation of extreme learning machines," *Neurocomputing*, vol. 102, pp. 125–134, 2013.

[9] F. Galluppi, S. Davies, S. Furber, T. Stewart, and C. Eliasmith, "Real time on-chip implementation of dynamical systems with spiking neurons," *The International Joint Conference on Neural Networks IJCNN*, pp. 1–8, 2012.

[10] S. Choudhary, S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," *In Proc. International Conference on Artificial Neural Networks, Lecture Notes in Computer Science (LNCS), Springer, Heidelberg*, vol. 7552, pp. 121–128, 2012.

[11] J. Tapson and A. van Schaik, "Learning the pseudoinverse solution to network weights," *Neural Networks*, vol. 45, pp. 94–100, 2013.

[12] R. Penrose, "A generalized inverse for matrices," *Mathematical Proceedings of the Cambidge Philosophical Society*, vol. 51, pp. 406–413, 1955.

[13] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A Large-Scale Model of the Functioning Brain," *Science*, vol. 338, pp. 1202–1205, 2012.

[14] M. D. McDonnell, M. D. Tissera, A. van Schaik, and J. Tapson, "Fast, simple and accurate handwritten digit classification using extreme learning machines with shaped input-weights," 2014, arXiv:1412.8307.

[15] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, pp. 1411–1423, 2006.

[16] B. Widrow, A. Greenblatt, Y. Kim, and D. Park, "The No-Prop algorithm: A new learning algorithm for multilayer neural networks," *Neural Networks*, vol. 37, pp. 182–188, 2013.

[17] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[18] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[19] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[20] L. L. C. Kasun, H. Zhou, and G.-B. Huang, "Representational Learning with ELMs for Big Data," *IEEE Intelligent Systems*, vol. 28, pp. 31–34, 2013.

[21] W. Yu, F. Zhuang, Q. He, and Z. Shi, "Learning deep representations via extreme learning machines," *Neurocomputing*, vol. 149, pp. 308–315, 2015.

[22] H.-G. Han, L.-D. Wang, and J.-F. Qiao, "Hierarchical extreme learning machine for feedforward neural network," *Neurocomputing*, vol. 128, pp. 128–135, 2014.

[23] W. Zhu, J. Miao, and L. Qing, "Constrained extreme learning machine: a novel highly discriminative random feedforward neural network," in *Proc. International Joint Conference on Neural Networks (IJCNN), 6-11 July, Beijing, China*, 2014, pp. 800–807.

[24] H. Larochelle and Y. Bengio, "Classification using discriminative restricted boltzmann machines," in *Proceedings of the 25th international conference on Machine learning - ICML '08, Helsinki, Finland*, 2008, pp. 536–543.

[25] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio, "Learning algorithms for the classification restricted Boltzmann machine," *Journal of Machine Learning Research*, vol. 13, pp. 643–669, 2012.

[26] Y. LeCun, C. Cortes, and C. J. C. Burges, "The MNIST database of handwritten digits," Accessed August 2014, `http://yann.lecun.com/exdb/mnist/`.

[27] J. Tapson, P. de Chazal, and A. van Schaik, "Explicit computation of input weights in Extreme Learning Machines," *Proceedings of ELM2014, Accepted*, vol. arXiv:1406.2889, 2014.

[28] A. Krizhevsky, "Learning multiple layers of features from tiny images," Ph.D. dissertation, Masters Thesis, Dept of CS, University of Toronto. See `http://www.cs.toronto.edu/ kriz/cifar.html`), 2009.

[29] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011, NIPS Workshop on Deep Learning and Unsupervised Feature Learning. See `http://ufldl.stanford.edu/housenumbers`.

[30] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proc. 18th International Conference on Artificial Intelligence and Statistics (AISTATS), San Diego, CA, USA. JMLR: W&CP*, vol. 38, 2015.

[31] M. D. McDonnell and T. Vladusich, "Enhanced image classification with a fast-learning shallow convolutional neural network," 2015, arxiv.org:1503.04596.

[32] W. Zhu, J. Miao, and L. Qing, "Constrained extreme learning machines: A study on classification cases," 2015, arXiv:1501.06115.