Transfer Learning Using Rotated Image Data to Improve Deep Neural Network Performance

Telmo Amaral[1], Luís M. Silva[12], Luís A. Alexandre[3], Chetak Kandaswamy[1], Joaquim Marques de Sá[14], and Jorge M. Santos[15]

[1]Instituto de Engenharia Biomédica (INEB), Universidade do Porto, Portugal
[2]Departamento de Matemática, Universidade de Aveiro, Portugal (lmas@ua.pt)
[3]Instituto de Telecomunicações, Universidade da Beira Interior, Portugal
[4]Dep. de Eng. Electrotécnica e de Computadores, Fac. de Eng. da Univ. do Porto, Portugal
[5]Dep. de Matemática, Instituto Superior de Engenharia do Instituto Politécnico do Porto, Portugal

Abstract In this work we explore the idea that, in the presence of a small training set of images, it could be beneficial to use that set itself to obtain a transformed training set (by performing a random rotation on each sample), train a source network using the transformed data, then retrain the source network using the original data. Applying this transfer learning technique to three different types of character data, we achieve average relative improvements between 6% and 16% in the classification test error. Furthermore, we show that it is possible to achieve relative improvements between 8% and 42% in cases where the amount of original training samples is very limited (30 samples per class), by introducing not just one rotation but several random rotations per sample.

Keywords: Transfer Learning; Deep Learning; Stacked Auto-Encoders.

1 Introduction

Deep architectures, such as neural networks with two or more hidden layers, are a class of networks that comprise several levels of non-linear operations, each expressed in terms of parameters that can be learned [1]. Until 2006, attempts to train deep architectures generally resulted in poorer performance but a breakthrough took place with the introduction by Hinton et al. [7] of the deep belief network, whose hidden layers are initially treated as restricted Boltzmann machines (RBMs) and pre-trained, one at a time, in an unsupervised greedy approach. This pre-training procedure was soon generalised to rely on machines easier to train than RBMs, such as auto-encoders [8].

The goal of transfer learning (TL) is to reuse knowledge associated with a source problem to improve the learning required by a target problem [9]. The source and target problems may be, for example, classification tasks that differ as to the data distributions, or that involve different sets of classes. A common approach to TL is that of transferring representations that were learned from one problem to another problem.

In this paper we investigate if, in the presence of a small training set, it is possible to use that set itself to obtain a transformed training set (by performing for example a random rotation on each sample), train a source network using the transformed data, then retrain that network using the original data, to achieve lower classification errors than would be possible by using only the original data. We explore this idea using three types of character image data. We achieved significant improvements in the classification error by fully training a source network using slightly rotated versions of the original training samples, then fine-tuning that network again using the original samples. For very small amounts of training data, it was possible to further improve performance by introducing more than one rotation per sample.

Deep architectures have been used recently in TL settings, as discussed in reviews by Bengio et al. [2, Sec. 2.4] and Deng and Yu [5, Ch. 11]. For example, Glorot et al. [6] pre-trained stacked denoising auto-encoders using unlabelled data from multiple domains, thus learning a generic representation that could be used to train SVMs for sentiment classification on a specific domain. This differs from our work, as the target network was not obtained by fully retraining a source network and no data transformations (in fact no image data) were involved. In the field of character recognition, Ciresan et al. [4] trained convolutional neural networks (CNNs) on either digits or Chinese characters and retrained them to recognise uppercase Latin letters. Again, no data transformations were involved.

Affine and elastic transformations have been used extensively to increase the amount data available to train neural networks, as in the work of Ciresan et al. [3] with very large (but shallow) multi-layer perceptrons trained through parallelisation. Simard et al. [10] suggest the use of distorted data as good practice in the supervised training of CNNs. These two works did not involve networks pre-trained without supervision, or transfer learning. More generally, existing work with deep architectures does not address the use of transformed image data as a means to obtain an artificial problem from which knowledge can be gathered and transferred to the original problem, to improve performance.

2 Stacked Auto-encoders

The auto-encoder (AE) is a simple network that tries to produce at its output what is presented at the input. The basic AE is in fact a simple neural network with one hidden layer and one output layer, subject to two restrictions: the number of output neurons is equal to the number of inputs; and the weight matrix of the output layer is the transposed of the weight matrix of the hidden layer (that is, the weights are clamped). The values of the hidden layer neurons are called the *encoding*, whereas the values of the output neurons are called the *decoding*. Unsupervised learning of the weights and biases of AEs can be achieved by gradient descent, based on a training set of input vectors.

Consider a network designed for classification, with a layer of inputs, two or more hidden layers, and a softmax output layer with as many units as classes. The hidden layers of such a network can be *pre-trained* one at a time in an

Algorithm 1 Transfer Learning approach.

Given design sets $X_{ds.tra}$ and $X_{ds.ori}$, and test set $X_{ts.ori}$,

1. Randomly initialise a classifier network;
2. Pre-train the network using $X_{ds.tra}$ (ignoring labels);
3. Fine-tune the network using $X_{ds.tra}$;
4. Fine-tune the network using $X_{ds.ori}$;
5. Test the network using $X_{ts.ori}$, obtaining classification error ε.

unsupervised way. Each hidden layer is "unfolded" to form an AE. Once that AE has learned to reconstruct its own input, its output layer is no longer needed and its hidden layer becomes the input to the next hidden layer of the network. The next hidden layer is in turn pre-trained as an individual AE and the process is repeated until there are no more hidden layers. A deep network pre-trained in this fashion is termed a stacked auto-encoder (SAE).

The goal of unsupervised pre-training is to bring the network's hidden weights and biases to a region of the parameter space that constitutes a better starting point than random initialisation, for a subsequent supervised training stage. In this context, the supervised training stage is usually called *fine-tuning* and can be achieved by conventional gradient descent, based on a training set of input vectors paired with class labels. The output layer weights are randomly initialised and learned only in the fine-tuning stage.

3 Transfer Learning Based Approach

We used a TL approach where the involved problems differed only in terms of the data distribution. Let $X_{ds.ori}$ be an original design set containing $n_{ds.ori}$ data samples. We assume that each data sample contains not only an input vector (representing for example an image) but also the corresponding class label, and the design set contains both training and validation data. Let $X_{ds.tra}$ be a design set containing $n_{ds.ori}$ transformed data samples, obtained by doing a transformation (such as a random rotation, in the case of image data) on each data sample from $X_{ds.ori}$. Let $X_{ts.ori}$ be a test set containing $n_{ts.ori}$ original data samples.

Given $X_{ds.tra}$, $X_{ds.ori}$ and $X_{ts.ori}$, we can use TL to design and test a classifier, by applying Algorithm 1. Specifically, in steps 2 and 3 the initialised network is trained (first without supervision, then with supervision) to classify transformed data and, in step 4, the resulting network is retrained (with supervision) to classify original data. The idea is to transfer knowledge from an artificially created source problem to the original target problem. Algorithm 1 can be trivially modified by omitting step 3, so that the source network is only pre-trained, instead pre-trained and fine-tuned.

We compared the performance of classifiers obtained via the TL approach described above with the performance of classifiers obtained via the baseline (BL) method defined in Algorithm 2. In this BL approach, a classifier is pre-trained

Algorithm 2 Baseline approach.

Given design set X_{ds} and test set $X_{ts.ori}$,

1. Randomly initialise a classifier network;
2. Pre-train the network using X_{ds} (ignoring labels);
3. Fine-tune the network using X_{ds};
4. Test the network using $X_{ts.ori}$, obtaining classification error ε.

and fine-tuned in a conventional way, using data from a single distribution, then tested on original data. When design set X_{ds} is the original design set $X_{ds.ori}$, the last two steps of the BL approach perform the same operations as the last two steps of the TL approach: fine-tuning and testing using only original data. This yields a test error that can be directly compared with the test error obtained with TL. Alternatively, when X_{ds} is a transformed design set $X_{ds.tra}$, the BL approach is equivalent to TL without step 4: pre-training and fine-tuning using transformed data and testing *directly* on original data, without retraining on original data. As seen later in Section 5, this helped us to determine whether TL was really beneficial, or if simply transforming design data was enough to improve performance on original test data.

4 Data and Hyper-parameters

The data used in this work consisted of grey-level images of handwritten digits, typewritten (synthesised) digits, and lowercase letters, all containing 28×28=784 pixels. For each data type, we prepared a test set $X_{ts.ori}$ containing $n_{ts.ori}$ original samples and a design set $X_{ds.ori.full}$ containing $n_{ds.ori.full}$ original samples. We use the subscript $full$ because, in practice, only randomly picked subsets of $X_{ds.ori.full}$ were used in the experiments. Table 1 shows the numbers of samples available from each data type, as well as the number of classes involved, c. All data originated from the MNIST-basic set prepared by the LISA lab[1] and the Chars74K set prepared by Microsoft Research India[2].

Table 1. Numbers of design and test samples available from each data type.

Data type	c	$n_{ds.ori.full}$	$n_{ts.ori}$
Handwritten digits	10	3000	50000
Typewritten digits	10	3000	7160
Typewritten letters	26	7800	18616

All the deep networks we used had an architecture with two hidden layers composed of 100 units each and an output layer appropriate to the number of classes being considered. Using an additional hidden layer did not have a significant effect on the observed validation errors.

[1] See http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations.

[2] See http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/.

Algorithm 3 Experimental procedure.

Given $X_{ds.ori.full}$, $X_{ds.tra}$ and an integer $k \geq 1$,

For each data type (handwritten digits, typewritten digits, and typewritten letters),

1. For each $n_{ds.ori}$ such that $\frac{n_{ds.ori}}{c} \in [30, 60, 90, 120, 150]$,
   (a) Obtain $X_{ds.ori}$ by randomly picking $n_{ds.ori}$ samples from $X_{ds.ori.full}$;
   (b) Obtain $X_{ds.tra}$ by creating k random rotations of each sample from $X_{ds.ori}$;
   (c) Run baseline approach using $X_{ds.ori}$ and $X_{ts.ori}$;
   (d) Run baseline approach using $X_{ds.tra}$ and $X_{ts.ori}$;
   (e) Run transfer learning approach using $X_{ds.tra}$, $X_{ds.ori}$ and $X_{ts.ori}$.

Hidden layers were pre-trained via online gradient descent, the cross-entropy cost function and a learning rate of 0.001, for a minimum of 15 epochs and then until the relative improvement in the validation error fell below 1%. Whole networks were fine-tuned via online gradient descent, the cross-entropy cost function and a learning rate of 0.1, until the validation error did not decrease for 50 epochs. These hyper-parameter values did not result from a thorough selection procedure, but we believe they yielded validation errors that were sufficiently low to enable the comparisons done in this work.

Our code was based on an implementation of SAEs originally developed by Hugo Larochelle[3]. All experiments ran on an Intel Core i7-950 and enough physical memory to prevent swapping. A pool of five parallel processes was used.

5 Experiments and Results

We followed the procedure shown in Algorithm 3. In step 1a, two thirds of the randomly picked $n_{ds.ori}$ design samples are assigned to training and one third is assigned to validation. The transformed $X_{ds.tra}$ obtained in step 1b contains $k \times n_{ds.ori}$ samples, since it is generated by creating k distinct randomly rotated versions of each sample from the original design set. In practice, actually two variants of $X_{ds.tra}$ were obtained in this step: $X_{ds.tra.030}$, by doing random rotations in the interval $[-30°, 30°]$; and $X_{ds.tra.180}$, by using the interval $[-180°, 180°]$ instead. In addition, in step 1e two variants of TL are tried: one using $X_{ds.tra}$ only for pre-training (this is later denoted as PT); the other using $X_{ds.tra}$ for both pre-training and fine-tuning (denoted as PT+FT).

5.1 Using a single rotation (k=1)

In a first series of experiments we set k to 1 in Algorithm 3. The experimental procedure was repeated 20 times. At each repetition, a new random number generator seed was used to pick $n_{ds.ori}$ design samples in step 1a, to rotate samples in step 1b, and to initialise the networks trained in steps 1c, 1d and 1e.

[3] See http://www.dmi.usherb.ca/~larocheh/mlpython/.

Table 2 shows the average classification error $\overline{\varepsilon}$ obtained using $X_{ts.ori}$, for the three data types and for different numbers $n_{ds.ori}/c$ of design samples per class. For each data type and value of $n_{ds.ori}/c$, the lowest mean error is underlined. The p-value for the Student's t-test is also reported in square brackets, to help assess if the errors obtained in that experiment were significantly lower than those obtained with the BL approach using $X_{ds.ori}$.

Table 2. Percent average classification test error $\overline{\varepsilon}$ (standard deviation) [p-value] obtained for different data types, approaches, design sets, and numbers $n_{ds.ori}/c$ of design samples per class.

Data	Approach and design sets	$n_{ds.ori}/c$ = 30	60	90	120	150
Handwritten digits	BL $X_{ds.ori}$	27.2 (02.4)	16.3 (01.6)	12.8 (01.0)	11.1 (00.7)	10.3 (00.6)
	BL $X_{ds.tra.030}$	33.5 (02.5) [<0.01]	18.3 (01.3) [<0.01]	13.9 (00.6) [<0.01]	12.4 (00.5) [<0.01]	11.2 (00.5) [<0.01]
	BL $X_{ds.tra.180}$	66.0 (04.7) [<0.01]	57.5 (03.3) [<0.01]	45.9 (03.6) [<0.01]	39.3 (02.4) [<0.01]	36.0 (02.2) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT	31.5 (03.4) [<0.01]	16.4 (01.4) [0.40]	12.5 (00.7) [0.15]	11.1 (00.5) [0.47]	10.2 (00.5) [0.33]
	TL $X_{ds.ori}$ after $X_{ds.tra.180}$ PT	32.8 (03.8) [<0.01]	19.2 (01.4) [<0.01]	12.8 (00.7) [0.45]	11.0 (00.4) [0.45]	10.4 (00.6) [0.34]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	22.7 (02.0) [<0.01]	13.5 (00.9) [<0.01]	10.8 (00.5) [<0.01]	9.9 (00.5) [<0.01]	9.1 (00.4) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.180}$ PT+FT	27.9 (02.4) [0.20]	17.2 (01.2) [0.02]	12.6 (00.6) [0.19]	11.0 (00.4) [0.33]	10.1 (00.5) [0.11]
Typewritten digits	BL $X_{ds.ori}$	12.4 (01.5)	8.6 (00.7)	7.1 (00.4)	6.1 (00.3)	5.5 (00.4)
	BL $X_{ds.tra.030}$	18.4 (02.9) [<0.01]	9.9 (00.8) [<0.01]	7.8 (00.6) [<0.01]	7.0 (00.4) [<0.01]	6.4 (00.4) [<0.01]
	BL $X_{ds.tra.180}$	55.7 (06.0) [<0.01]	36.3 (07.1) [<0.01]	27.6 (03.6) [<0.01]	23.5 (02.9) [<0.01]	21.3 (02.5) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT	14.1 (01.4) [<0.01]	8.8 (00.7) [0.17]	7.1 (00.6) [0.47]	6.1 (00.4) [0.44]	5.3 (00.4) [0.04]
	TL $X_{ds.ori}$ after $X_{ds.tra.180}$ PT	15.8 (01.7) [<0.01]	9.2 (00.9) [0.02]	6.9 (00.4) [0.13]	6.1 (00.5) [0.47]	5.6 (00.5) [0.37]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	11.0 (01.0) [<0.01]	7.4 (00.4) [<0.01]	5.9 (00.6) [<0.01]	5.3 (00.4) [<0.01]	4.8 (00.3) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.180}$ PT+FT	14.8 (01.4) [<0.01]	9.2 (00.7) [0.01]	7.1 (00.4) [0.48]	6.2 (00.4) [0.08]	5.6 (00.4) [0.32]
Typewritten letters	BL $X_{ds.ori}$	21.6 (00.7)	16.4 (00.7)	14.6 (00.4)	13.4 (00.3)	12.8 (00.3)
	BL $X_{ds.tra.030}$	26.4 (01.4) [<0.01]	19.7 (00.6) [<0.01]	17.6 (00.4) [<0.01]	16.2 (00.4) [<0.01]	15.7 (00.4) [<0.01]
	BL $X_{ds.tra.180}$	63.7 (04.1) [<0.01]	50.5 (03.2) [<0.01]	46.6 (02.1) [<0.01]	43.2 (02.1) [<0.01]	41.1 (01.9) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT	22.2 (01.1) [0.03]	16.2 (00.7) [0.19]	14.2 (00.4) [<0.01]	13.1 (00.4) [<0.01]	12.3 (00.3) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.180}$ PT	20.9 (00.9) [0.01]	15.7 (00.6) [<0.01]	13.8 (00.3) [<0.01]	12.7 (00.3) [<0.01]	12.1 (00.4) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	19.4 (01.1) [<0.01]	15.3 (00.6) [<0.01]	13.6 (00.5) [<0.01]	12.5 (00.3) [<0.01]	11.9 (00.4) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.180}$ PT+FT	21.4 (01.9) [0.36]	16.0 (00.5) [0.01]	14.1 (00.3) [<0.01]	13.1 (00.2) [<0.01]	12.5 (00.3) [<0.01]

As shown, for each data type, the BL approach was tried not only with $X_{ds.ori}$, but also with $X_{ds.tra.030}$ and $X_{ds.tra.180}$. The obtained results show that training a model with transformed data and directly testing it on original data invariably led to worse results than training and testing with original data.

For all data types, the results obtained with TL when transformed design data were used for both pre-training and fine-tuning (PT+FT) were generally better than the results obtained when transformed data were used only for pre-training (PT).

The average test error $\overline{\varepsilon}$ obtained with the BL approach and with TL when using transformed data for PT+FT is plotted in Fig. 1, for handwritten digits and typewritten letters. Training times were found to increase linearly with the amount of design data. Partial and total average training times are reported in Table 3, for the case of $n_{ds.ori}/c$=150 samples per class. Times are shown for the BL approach and for TL, when slightly rotated data ($X_{ds.tra.030}$) were used both to pre-train and to fine-tune (PT+FT). The table rows corresponding to k=5 and k=10 will be addressed later.

The results show that, for all data types and for all numbers of design samples per class, TL based on variant $X_{ds.tra.030}$ of the transformed data led to significantly lower errors than the BL approach. This improved accuracy had a price in terms of time needed to design the classifiers: total training times needed

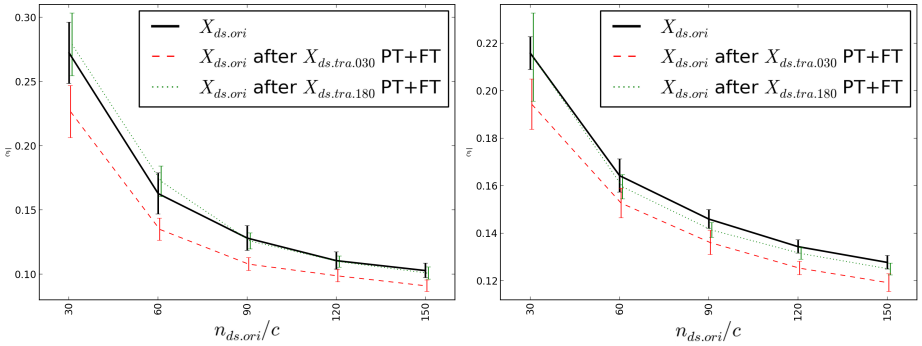

Figure 1. Average classification test error $\bar{\varepsilon}$ obtained with the BL and TL approaches, for each data type, for different numbers $n_{ds.ori}/c$ of original design samples per class. Left: handwritten digits; right: typewritten letters.

Table 3. Average time in seconds (standard deviation) needed to pre-train and fine-tune source and target models, for different data types, approaches, and values of k.

Data type	Approach an design sets	k	Source		Target	Total
			PT	FT	PT FT	
Handwritten digits	BL $X_{ds.ori}$				97 (15) 53 (14)	150 (19)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	1	83 (10)	64 (12)	102 (32)	248 (38)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	5	302 (48)	438 (172)	84 (20)	824 (170)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	10	514 (58)	770 (473)	98 (18)	1382 (491)
Typewritten digits	BL $X_{ds.ori}$				112 (12) 63 (26)	175 (28)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	1	88 (13)	67 (25)	97 (15)	252 (36)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	5	334 (51)	420 (229)	101 (22)	855 (245)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	10	581 (64)	662 (356)	124 (45)	1367 (369)
Typewritten letters	BL $X_{ds.ori}$				260 (29) 197 (68)	457 (79)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	1	212 (24)	217 (84)	402 (165)	832 (193)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	5	786 (57)	1664 (520)	352 (146)	2801 (507)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	10	1359 (94)	3558 (1288)	327 (123)	5243 (1274)

by TL were 50% to 100% longer than those needed by the BL approach. This was not surprising, as TL involves unsupervised and supervised training stages that yield a first classifier (steps 2 and 3 in Algorithm 1) followed by a supervised training stage that yields a second classifier (step 4), whereas the BL approach involves the unsupervised and supervised training of a single classifier (steps 2 and 3 in Algorithm 2).

For all data types and for all amounts of design samples per class, variant $X_{ds.tra.030}$ of the transformed data used for PT+FT always led to better results than variant $X_{ds.tra.180}$, as illustrated in Fig. 1. This indicates that it was better to restrict the random rotation of original images to a small range than to allow it to assume any value.

Fig. 2 (left) shows the average relative improvement in the test error ($\overline{\Delta\varepsilon_r}$) obtained over 20 repetitions when TL was applied instead of the BL approach, in experiments that used slightly rotated design data to pre-train and fine-tune the source classifier. The relative improvement was computed as $\Delta\varepsilon_r=(\varepsilon_{BL}-\varepsilon_{TL})/\varepsilon_{BL}$, where ε_{BL} and ε_{TL} are the test errors yielded by the BL approach and TL, re-

spectively. For all data types, the observed improvements in the average error were roughly constant across the different numbers of original design samples per class.

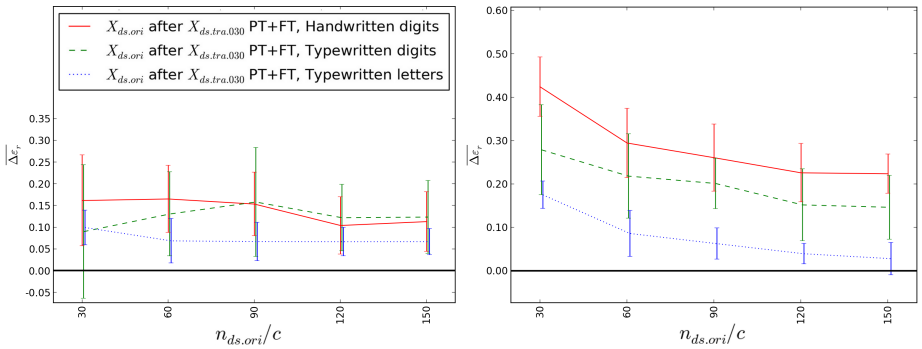

Figure 2. Average relative improvement in the classification test error $\overline{\Delta\varepsilon_r}$ yielded by TL (using slightly rotated design data for pre-training and fine-tuning), for different data types, for different amounts $n_{ds.ori}/c$ of original design samples per class. Left: for k=1. Right: for k=5.

5.2 Using several rotations (k>1)

In a second series of experiments, we used transformed design data obtained by creating several rotated versions of each original design sample, by using first k=5 and then k=10 in Algorithm 3. Steps 1c and 1d were skipped, because now we were not concerned with comparing the TL and BL approaches. Rather we wanted to compare TL results obtained using k>1 with the TL results previously obtained using k=1. In addition, when applying TL, we considered only cases where the transformed design data were obtained via small rotations ($X_{ds.tra.030}$) and used both to pre-train and to fine-tune the source model (PT+FT).

The experimental procedure was repeated 20 times for each value of k. The obtained results are shown in Table 4, together with results for k=1 reproduced from Table 2. For each data type and value of $n_{ds.ori}/c$, the two p-values shown for k=5 and k=10 were computed in relation to the results obtained with k=1. Some of the results shown in Table 4 are also plotted in Fig. 3.

For all data types, when only 30 original design samples per class were used, the errors yielded by TL were significantly lower when the transformed design set was formed by several rotations of each original sample (k=5 or k=10) than when the transformed set was obtained via a single rotation (k=1). In the case of typewritten digits, this benefit persisted for 60 samples per class and, in the case of handwritten digits, it was visible for any number of samples per class.

For handwritten and typewritten digit data, regardless of the amount of design data per class, the performances obtained with k=10 and k=5 were not distinguishable. With typewritten letters, for more than 90 samples per class,

Table 4. Percent average classification test error $\overline{\varepsilon}$ (standard deviation) [p-value] obtained for different data types, approaches, design sets, and numbers $n_{ds.ori}/c$ of design samples per class.

Data	Approach and design sets	k	$n_{ds.ori}/c$ = 30	60	90	120	150
H. digits	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	1	22.7 (02.0)	13.5 (00.9)	10.8 (00.5)	9.9 (00.5)	9.1 (00.4)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	5	15.6 (01.4) [<0.01]	11.4 (00.8) [<0.01]	9.4 (00.6) [<0.01]	8.5 (00.5) [<0.01]	8.0 (00.4) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	10	15.2 (02.0) [<0.01]	10.9 (00.5) [<0.01]	9.2 (00.5) [<0.01]	8.6 (00.5) [<0.01]	8.0 (00.4) [<0.01]
T. digits	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	1	11.0 (01.0)	7.4 (00.4)	5.9 (00.6)	5.3 (00.4)	4.8 (00.3)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	5	8.8 (00.7) [<0.01]	6.7 (00.6) [<0.01]	5.7 (00.5) [0.07]	5.1 (00.5) [0.12]	4.7 (00.2) [0.08]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	10	8.9 (00.7) [<0.01]	6.8 (00.5) [<0.01]	6.0 (00.7) [0.43]	5.3 (00.5) [0.49]	4.8 (00.4) [0.45]
T. letters	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	1	19.4 (01.1)	15.3 (00.6)	13.6 (00.5)	12.5 (00.3)	11.9 (00.4)
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	5	17.8 (00.7) [<0.01]	15.0 (00.5) [0.06]	13.7 (00.4) [0.33]	12.9 (00.2) [<0.01]	12.4 (00.3) [<0.01]
	TL $X_{ds.ori}$ after $X_{ds.tra.030}$ PT+FT	10	18.0 (00.7) [<0.01]	15.1 (00.5) [0.22]	14.0 (00.4) [0.01]	13.6 (00.4) [<0.01]	13.1 (00.3) [<0.01]

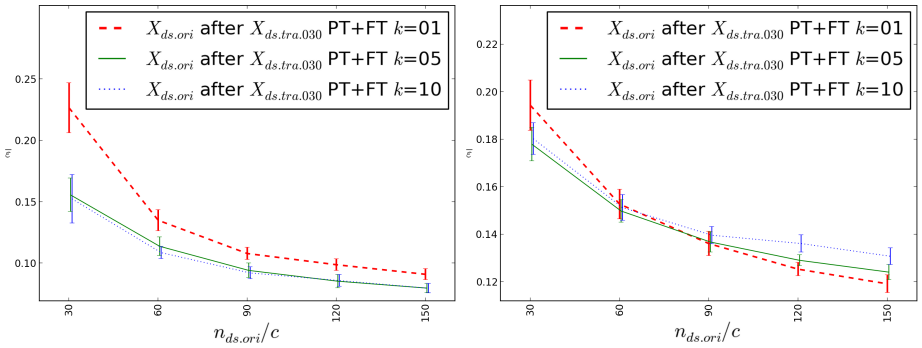

Figure 3. Average classification test error $\overline{\varepsilon}$ obtained with TL, for different values of k, for each data type, for different numbers $n_{ds.ori}/c$ of original design samples per class. Left: handwritten digits; right: typewritten letters.

the errors obtained with k=10 were actually higher than those obtained with k=5.

The effects discussed above can also be observed in Fig. 2 (right), which plots the improvements that the average TL errors shown in Table 4 for k=5 achieved in relation to the average BL errors shown in Table 2.

Average training times observed for $n_{ds.ori}/c$=150 are included in Table 3. The benefits obtained by using k=5 rotations per original design sample had a clear cost in terms of total training times, which were about three times longer than the times observed when k=1.

6 Conclusions and Future Directions

In this work we explored the idea that, in the presence of a small design set of image data, it could be beneficial to use that same set to obtain a transformed design set (by performing a random rotation on each original sample), train a source network using the transformed data, then retrain that network using the original data. For the three data types involved in our experiments, networks

designed via this TL approach yielded significantly lower errors than networks trained using only original (non-rotated) data. Relative improvements between 6% and 16% were observed in the average errors, at the expense of training times 50% to 100% longer.

In general, pre-training and fine-tuning a source network led to better results than just pre-training it. Restricting the rotations performed on the original design samples to a small range led to better results than freely rotating the samples. It would be interesting to study in finer detail the relationship between performance and the range of allowed rotation, and also try transformations other than rotation.

For small amounts of original design data, it was possible to further improve performance by including in the transformed data more than one randomly rotated version of each original sample. With k=5 rotations per original sample, relative improvements between 8% and 42% were observed in the average test error. This implied training times about three times longer than those associated with a single rotation.

References

[1] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[3] D. Ciresan, U. Meier, L. Gambardella, and J. Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.

[4] D. Ciresan, U. Meier, and J. Schmidhuber. Transfer learning for Latin and Chinese characters with deep neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2012.

[5] Li Deng and Dong Yu. Deep learning for signal and information processing. Microsoft Research monograph, 2013.

[6] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: a deep learning approach. In *International Conference on Machine Learning (ICML)*, pages 513–520, 2011.

[7] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[8] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *International Conference on Machine Learning (ICML)*, pages 473–480, 2007.

[9] S. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[10] P. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recognition (ICDAR)*, volume 3, pages 958–962, 2003.