Improving Performance on Problems with Few Labelled Data by Reusing Stacked Auto-Encoders

Telmo Amaral*, Chetak Kandaswamy*, Luís M. Silva*†, Luís A. Alexandre‡, Joaquim Marques de Sá*¶, Jorge M. Santos*§
*Instituto de Engenharia Biomédica (INEB), Universidade do Porto, Portugal. Email: tga@fe.up.pt
†Departamento de Matemática, Universidade de Aveiro, Portugal. Email: lmas@ua.pt
‡Instituto de Telecomunicações, Universidade da Beira Interior, Covilhã, Portugal
¶Dep. de Engenharia Electrotécnica e de Computadores, Fac. de Engenharia da Univ. do Porto, Portugal
§Departamento de Matemática, Instituto Superior de Engenharia do Instituto Politécnico do Porto, Portugal

***Abstract*—Deep architectures have been used in transfer learning applications, with the aim of improving the performance of networks designed for a given problem by reusing knowledge from another problem. In this work we addressed the transfer of knowledge between deep networks used as classifiers of digit and shape images, considering cases where only the set of class labels, or only the data distribution, changed from source to target problem. Our main goal was to study how the performance of knowledge transfer between such problems would be affected by varying the number of layers being retrained and the amount of data used in that retraining. Generally, reusing networks trained for a different label set led to better results than reusing networks trained for a different data distribution. In particular, reusing for less classes a network trained for more classes was beneficial for virtually any amount of training data. In all cases, retraining only one layer to save time consistently led to poorer performance. The results obtained when retraining for upright digits a network trained for rotated digits raise the hypothesis that transfer learning could be used to better deal with image classification problems in which only a small amount of labelled data is available for training.**

Keywords*—transfer learning; deep learning; artificial neural networks

I. 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]. The organisation of the mammal brain into processing stages that correspond to different levels of abstraction, as well as the way in which humans organise their ideas hierarchically, are among the main motivations for the use of such architectures. Nevertheless, until 2006, attempts to train deep architectures generally resulted in poorer performance than that achieved by shallow networks. A breakthrough took place with the introduction by Hinton et al. [8] 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 [10].

This work was financed by FEDER funds through the *Programa Operacional Factores de Competitividade* - COMPETE and by Portuguese funds through FCT - *Fundação para a Ciência e a Tecnologia* in the framework of the project PTDC/EIA-EIA/119004/2010.

The goal of transfer learning is to reuse knowledge associated with a source problem to improve the learning of the classification function associated with a target problem [11]. The source and target problems may differ, for example, as to the data distributions, or they may involve different sets of classes. A common approach to transfer learning is that of transferring representations that were learned from one problem onto another problem. Bruzzone and Marconcini [3], for example, have explored novel ways of transferring knowledge across support vector machines (SVMs) trained for different domains.

More recently, deep architectures have been used in transfer learning settings, as discussed by Bengio et al. [2, Section 2.4] and Deng and Yu [5, Chapter 11]. Glorot et al. [6], for example, used a deep network to learn highly generic feature representations from a source domain, then trained SVMs to conduct recognition on a different domain while using the same representations. Possibly the closest works to ours found in existing literature are the cross-lingual speech recognition experiments recently reported by Huang et al. [9] and by Heigold et al. [7], though with important differences: in both cases the problems at hand were not of image classification; the source and target problems differed in terms of both the set of class labels *and* the data distribution; and stacked auto-encoders were not employed. In addition, the work of Heigold et al. did not involve unsupervised pre-training of source networks or variations in the amount of data used to train target networks. Related work on character recognition, such as that of Ciresan et al. [4], typically involves changes in both the data distribution and the label set.

In this work, we explored the transfer of knowledge between deep networks designed to classify images of digits. We aimed to limit the differences between source and target problems by considering only two types of cases: either the set of class labels changed, while the underlying data distribution remained the same; or the label set remained fixed, while the data distribution changed. Our main goal was to study in a systematic way how the performance of transfer learning on such problems would be affected by varying the number of layers being retrained and, simultaneously, varying the amount of data available for retraining. In each experiment, we pre-trained without supervision and fine-tuned with supervision a network, using labelled data associated with the source problem; then we retrained with supervision selected layers from that network (while keeping the other layers untouched), using labelled data

available for the target problem, to obtain a new network. Our results are pertinent for situations in which few labelled data exist for the target problem (even if a large amount of unlabelled data happen to be available, for example to be used in unsupervised pre-training).

II. 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 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.

III. Transfer Learning

Let X be a random variable whose probability distribution is $P(X)$ and whose realisation values are data samples, each represented for example as a vector $\mathbf{x}$ of raw pixel values; and let Ω be a set of class labels, for example $\Omega{=}\{\text{odd}, \text{even}\}$. A classification problem involves learning a classification function that assigns data samples from $P(X)$ to labels in Ω. Given a source classification problem associated with data distribution $P_S(X)$ (or P_S for short) and label set Ω_S, and a target problem associated with distribution P_T and label set Ω_T, such that $P_S{\neq}P_T$ or $\Omega_S{\neq}\Omega_T$ or both, the aim of transfer learning is to help to learn the classification function associated with the target problem by using knowledge from the source problem. In other words, the aim is to learn a function that performs better than would be possible by learning "from scratch", i.e. using only knowledge from the target problem.

In all transfer learning experiments carried out in this work, the difference between the source problem and the target problem lay only on the label set or only on the probability distribution, i.e. either $\Omega_S{\neq}\Omega_T$ and $P_S{=}P_T$, or $\Omega_S{=}\Omega_T$ and $P_S{\neq}P_T$. Our approach consisted in pre-training and fine-tuning a network to classify data sampled from the source distribution into the source label set, then partially retraining that network to classify data sampled from the target distribution into the target label set. In order to partially retrain a source network, we selected one or more of its layers and fine-tuned them using labelled training data available for the target problem. When retraining a layer in this way, the weights and biases from the source network were used as initial values (except, of course, for the output layer in cases where the number of classes changed, implying the rebuilding of that layer from scratch). The main goal of this work was to study the impact, on the performance of transfer learning, of varying the amount of training data associated with the target problem as well as the number of retrained layers.

In a recent survey, Pan and Yang [11] define a number of transfer learning settings, depending on how the source and target problems differ. For each setting, they further define one or more possible approaches, depending on how the transfer of knowledge is achieved. In our work, the experiments in which the set of class labels changed correspond to an *inductive* setting, whereas the experiments in which the data distribution changed can be seen as a variant of *transductive* setting (though in our case only labelled data were used). The approach we took to both these settings was that of transferring sets of parameters associated with learned representations (i.e. the actual weights and biases from a network trained for the source problem).

IV. Experiments and Results

A. Data

Our experiments involved nine datasets of images, whose numbers of training, validation and test instances are shown in Table I. All datasets were balanced across classes, and all samples corresponded to a vector containing $28{\times}28{=}784$ real-valued pixel intensities. The first dataset consisted of samples from a distribution P_u of images representing upright handwritten digits in MNIST format[1], each assigned to a label from the set $\Omega_{09}{=}\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. A second set contained different samples from the same P_u distribution, each assigned to a label from the set $\Omega_{oe}{=}\{\text{odd}, \text{even}\}$. A third set consisted of samples from a distribution P_r of images representing handwritten digits in MNIST format subjected to a random rotation, each labelled using Ω_{09}. A fourth dataset contained samples from a distribution P_m of synthesised machine-printed digits, each labelled using Ω_{09}. This set was then split into two halves, one keeping the labels from Ω_{09}, the other labelled using Ω_{oe}. All digit data originated from the `mnist-basic` and `mnist-rot` sets provided by the LISA lab[2] and from the `Chars74K` set provided by Microsoft Research India[3]. A seventh dataset contained samples from a distribution P_c of canonical shapes, each assigned to a label from $\Omega_{02}{=}\{\text{ellipse}, \text{rectangle}, \text{triangle}\}$; the term canonical means that all ellipses were circles, all rectangles were squares, and all triangles were equilateral. An eighth dataset consisted of

[1] See http://yann.lecun.com/exdb/mnist/.

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

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

Table I. NUMBER OF INSTANCES AVAILABLE FOR EACH DATASET. ALSO SHOWN IS THE PERCENT AVERAGE CLASSIFICATION TEST ERROR ($\overline{\varepsilon}$) OBTAINED WITH A SOURCE NETWORK TRAINED USING EACH DATASET.

dataset		# Instances			$\overline{\varepsilon}$
Sampled from	Labelled using	Training	Validation	Test	
P_u	Ω_{09}	5000	1000	25000	5.9 (0.2)
P_u	Ω_{oe}	5000	1000	25000	3.4 (0.2)
P_r	Ω_{09}	5000	1000	25000	20.8 (0.5)
P_m	Ω_{09}	5000	1000	4160	2.3 (0.2)
P_m	Ω_{09}	2500	500	2080	3.4 (0.3)
P_m	Ω_{oe}	2500	500	2080	2.1 (0.2)
P_c	Ω_{02}	5000	1000	10000	5.7 (4.9)
P_n	Ω_{02}	5000	1000	10000	6.7 (4.5)
P_n	Ω_{01}	5000	1000	10000	4.5 (4.0)

samples from a distribution P_n of non-canonical (i.e. generic) shapes, each labelled using Ω_{02}. A final dataset contained samples from P_n, each labelled using Ω_{01}={round, with corners}, where "round" stands for elliptical shapes and "with corners" stands for rectangles and triangles. Shape data originated from the `Baby AI Shapes` dataset made available by the LISA lab[4].

B. Network Architecture

The networks we used in experiments with digit data had two hidden layers, with 100 units each, and an output layer appropriate to the number of classes being considered. The networks used with shape data had three hidden layers also with 100 units each. All hidden layers were pre-trained as auto-encoders via gradient descent, the cross-entropy cost function, and a learning rate of 0.001. Pre-training ran for a minimum of 15 epochs in the case of digit data, and for a minimum of 40 epoch when using shape data. The complete networks were fine-tuned via gradient descent, using the cross-entropy cost function and a learning rate of 0.1. Fine-tuning ran until the validation error did not decrease for 50 epochs, in the case of digit data, and for exactly 1000 epochs when using shape data. Our code for experiments with digit data was based on an implementation of stacked auto-encoders originally developed by Hugo Larochelle, using the MLPython library[5]; this code ran on an Intel Core i7-950 CPU, using a pool of six parallel processes and enough physical memory to prevent swapping. Code for experiments with shape data was based on the Theano library[6] and ran with the help of an Asus GTX 770 GPU.

C. Source Networks

Each of the datasets presented in Section IV-A was used to train a network to be used as source in subsequent experiments. These source networks were always trained using *all* the available training and validation data. Table I shows the average and standard deviation of the classification test error obtained with each network.

D. Changing the Set of Class Labels

In this section we address experiments whose source and target problems differed (only) in terms of the class label set. In a first set of experiments, we defined the target problem as that of classifying samples of handwritten upright digits (P_u) into ten classes (Ω_{09}={0, ..., 9}). We started by training a

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

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

[6]See http://deeplearning.net/software/theano/.

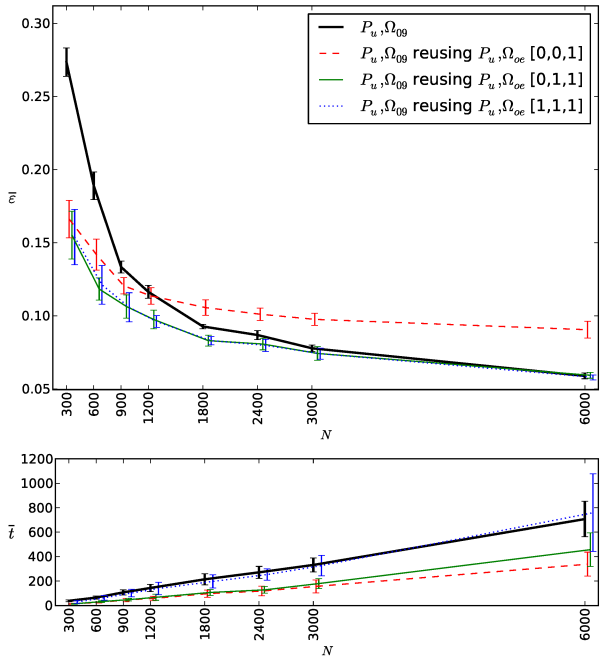

Figure 1. Results when classifying handwritten upright digits into ten classes by reusing a network trained for two classes: average classification test error ($\overline{\varepsilon}$) and average total training time ($\overline{t}$, in seconds), for different amounts (N) of data used to train the target network and for different numbers of retrained layers.

number of target networks "from scratch" (i.e. without reusing anything) for different numbers of training plus validation samples (N). The average classification test errors ($\overline{\varepsilon}$) obtained over 20 repetitions are shown in the first row of Table II and are also plotted as the thick black line in Fig. 1a, along with standard deviation bars.

We then trained another series of target networks, for the same values of N, this time reusing the source network trained to classify samples of upright digits into odd and even (Ω_{oe}). In other words, we aimed to classify into ten classes by reusing a source network trained for only two (though related) classes. Each target network was initialised using the weights and biases from the source network, then the two hidden layers were left untouched, while the output layer was retrained with supervision. This procedure is denoted as [0,0,1] in the second row of Table II, where the results are shown. Two additional rows of results are included: for the case where the first hidden layer was left untouched while both the second hidden layer and the output layer were retrained, denoted as [0,1,1]; and for the case where all three layers were retrained, denoted as [1,1,1]. These results are also plotted in Fig. 1a. In Table II, the lowest error obtained for each value of N is shown in boldface.

Our purpose was to investigate whether retraining only the output layer would be sufficient to successfully adapt the network to the new set of class labels, then try retraining increasingly more layers towards the input. It should be noted that the retraining procedure differed slightly depending on the type of layer it was applied to. When the output layer was retrained, it had to be rebuilt to match the new number of classes, then randomly initialised and fine-tuned. But, when a hidden layer was retrained, the weights and biases

Table II. CHANGING THE SET OF CLASS LABELS: PERCENT AVERAGE CLASSIFICATION TEST ERROR ($\bar{\varepsilon}$), FOR DIFFERENT AMOUNTS (N) OF DATA USED TO TRAIN THE TARGET NETWORK AND FOR DIFFERENT COMBINATIONS OF: TARGET DATA DISTRIBUTION (P_T); TARGET LABEL SET (Ω_T); SOURCE DISTRIBUTION (P_S); SOURCE LABEL SET (Ω_S); AND RETRAINED LAYERS. STANDARD DEVIATIONS SHOWN IN PARENTHESES.

P_T	Ω_T	P_S	Ω_S	Retrained layers	N = 150	300	450	600	900	1200	1500	1800	2400	3000	6000
P_u	Ω_{09}					27.3 (1.0)		18.9 (0.9)	13.3 (0.4)	11.6 (0.4)		9.3 (0.1)	8.7 (0.3)	7.8 (0.2)	5.9 (0.2)
P_u	Ω_{09}	P_u	Ω_{oe}	[0,0,1]		16.6 (1.3)		14.2 (1.1)	12.1 (0.6)	11.4 (0.6)		10.6 (0.5)	10.1 (0.4)	9.8 (0.4)	9.0 (0.6)
P_u	Ω_{09}	P_u	Ω_{oe}	[0,1,1]		15.5 (1.6)		**11.8** (0.8)	**10.6** (0.8)	9.7 (0.6)		**8.3** (0.4)	8.1 (0.4)	**7.4** (0.5)	5.9 (0.2)
P_u	Ω_{09}	P_u	Ω_{oe}	[1,1,1]		**15.4** (1.9)		12.1 (1.3)	**10.6** (1.0)	**9.6** (0.4)		**8.3** (0.3)	**8.0** (0.4)	**7.4** (0.4)	**5.8** (0.2)
P_m	Ω_{09}				29.5 (4.0)	11.6 (1.6)	8.7 (0.7)	7.3 (0.4)	6.3 (0.4)	5.2 (0.2)	4.8 (0.3)			**3.4** (0.3)	
P_m	Ω_{09}	P_m	Ω_{oe}	[0,0,1]	12.3 (1.4)	9.7 (0.8)	9.2 (1.5)	9.0 (0.8)	8.7 (1.3)	7.4 (1.0)	6.6 (0.7)			5.6 (0.4)	
P_m	Ω_{09}	P_m	Ω_{oe}	[0,1,1]	**11.3** (1.7)	9.0 (0.7)	8.1 (1.2)	7.8 (0.9)	6.7 (0.9)	5.6 (0.6)	5.3 (0.4)			3.9 (0.4)	
P_m	Ω_{09}	P_m	Ω_{oe}	[1,1,1]	**11.3** (1.5)	8.6 (0.6)	**7.6** (0.9)	**6.9** (0.9)	**5.7** (0.3)	**5.0** (0.4)	**4.7** (0.5)			**3.4** (0.2)	
P_u	Ω_{oe}					17.3 (0.7)		8.7 (0.5)	7.1 (0.5)	6.3 (0.5)		5.2 (0.3)	4.8 (0.2)	4.6 (0.3)	3.4 (0.2)
P_u	Ω_{oe}	P_u	Ω_{09}	[0,0,1]		9.4 (1.2)		7.5 (0.7)	6.9 (0.7)	6.8 (0.9)		6.1 (0.4)	7.6 (1.4)	5.6 (0.5)	6.4 (0.9)
P_u	Ω_{oe}	P_u	Ω_{09}	[0,1,1]		**8.5** (1.0)		**5.6** (0.8)	**4.7** (0.4)	**4.3** (0.3)		**4.2** (0.4)	**3.8** (0.3)	**3.6** (0.2)	3.1 (0.2)
P_u	Ω_{oe}	P_u	Ω_{09}	[1,1,1]		9.2 (1.0)		6.0 (0.9)	4.9 (0.2)	4.6 (0.5)		**4.2** (0.2)	**3.8** (0.1)	3.7 (0.2)	**3.0** (0.2)
P_m	Ω_{oe}				10.3 (0.0)	8.7 (0.6)	7.3 (1.4)	5.5 (0.6)	3.8 (0.2)	3.5 (0.4)	3.1 (0.3)			2.1 (0.2)	
P_m	Ω_{oe}	P_m	Ω_{09}	[0,0,1]	6.5 (1.9)	4.7 (1.0)	3.5 (0.9)	3.0 (0.5)	2.8 (0.4)	2.9 (0.3)	2.7 (0.3)			3.2 (0.6)	
P_m	Ω_{oe}	P_m	Ω_{09}	[0,1,1]	**6.4** (2.2)	4.6 (0.8)	**3.4** (0.6)	**2.7** (0.3)	**2.3** (0.3)	2.4 (0.3)	2.3 (0.3)			1.8 (0.4)	
P_m	Ω_{oe}	P_m	Ω_{09}	[1,1,1]	6.6 (2.0)	**4.5** (0.6)	**3.4** (0.6)	**2.7** (0.3)	2.4 (0.3)	**2.3** (0.3)	**2.2** (0.3)			**1.7** (0.2)	
P_n	Ω_{01}					22.5 (3.1)		15.3 (5.1)	15.0 (7.9)	7.9 (3.5)		9.8 (4.8)	9.2 (6.5)	8.4 (5.1)	4.5 (4.0)
P_n	Ω_{01}	P_n	Ω_{02}	[0,0,0,1]		5.9 (3.0)		5.7 (2.9)	5.4 (3.6)	5.4 (3.7)		4.8 (2.6)	4.8 (2.6)	4.6 (2.5)	4.5 (2.4)
P_n	Ω_{01}	P_n	Ω_{02}	[0,0,1,1]		5.7 (3.0)		**5.2** (2.7)	5.3 (3.8)	5.3 (4.1)		**4.4** (2.5)	**4.3** (2.3)	4.0 (2.2)	3.5 (1.7)
P_n	Ω_{01}	P_n	Ω_{02}	[0,1,1,1]		**5.4** (2.9)		5.3 (2.8)	5.5 (4.0)	5.5 (4.2)		4.5 (2.6)	4.6 (2.8)	4.2 (2.5)	3.5 (1.8)
P_n	Ω_{01}	P_n	Ω_{02}	[1,1,1,1]		5.4 (3.7)		5.2 (3.3)	**4.4** (3.3)	**4.4** (2.8)		4.8 (2.9)	4.5 (3.3)	**3.7** (2.3)	**3.0** (2.3)

inherited from the source were (re)used as initial values for the fine-tuning performed on the target. In addition, all target networks were obtained by reusing the *same* source network, trained using all the training and validation data available for the source problem.

Fig. 1 also shows the average total time ($\bar{t}$) taken to train each target network. In the case of baseline networks trained from scratch (thick black line), this total time corresponds to pre-training and fine-tuning. In the case of networks obtained through reuse, the total time corresponds only to the time taken to fine-tune the layers that we chose to retrain.

In a second set of experiments, we repeated the procedure described above, but using samples from synthetic machine-printed digits (P_m) instead of handwritten digits. The obtained results are shown in the second group of four rows in Table II. Finally, we repeated the two whole sets of experiments, but reversing the roles of the Ω_{09} and Ω_{oe} class label sets. In other words, we now aimed to classify only into two classes by reusing source networks trained for ten classes. The obtained results are shown in the third and fourth groups of four rows in Table II. The evolution of total training times shown in Fig. 1 for the first set of experiments is representative of what happened also in the subsequent three sets of experiments with digits.

A final set of experiments involved shape data, the target problem being that of classifying non-canonical shapes as either "rounded" or "with corners", and the source problem being that of classifying the same non-canonical shapes as ellipses, rectangles, or triangles.

Fig. 2 (left) shows the average relative improvement in the test error ($\overline{\Delta\varepsilon_r}$) obtained when transfer learning was used instead of training a network from scratch, along with standard deviation bars. This measure is plotted for different values of N, for the five sets of transfer learning experiment addressed in this section. For experiments with digit data, the shown results are for the case denoted as [0,1,1] in Table II. For experiments with shape data, the considered case is that denoted as [1,1,1,1].

E. Changing the Data Distribution

In this section we address experiments whose source and target problems differed (only) in terms of the data distributions. In a first set of experiments, we defined the target problem as that of classifying samples of handwritten rotated digits (P_r) into ten classes (Ω_{09}) and started by training a number of target networks from scratch for different values of N. The average classification test error $\overline{\varepsilon}$ obtained over 20 repetitions is shown in the first row of Table III.

We then trained another set of target networks for the same values of N, this time reusing a source network trained to classify samples of upright digits (P_u) into the same ten classes. In other words, we aimed to classify data associated with a higher generalisation error (as shown in the third row of Table I) by reusing a network trained on data associated with a lower error (first row in Table I). We first tried retraining only the first hidden layer, then also the second hidden layer, then all layers. The results are shown in the second, third and fourth rows of Table III. Our purpose was to assess if retraining only the layer closest to the network's input would be sufficient to successfully adapt the classification function to the new data distribution, then try retraining increasingly more layers towards the output.

In a second set of experiments, we aimed to classify handwritten digits (P_u) by reusing a network trained for machine-printed digits (P_m), thus preserving the idea of tackling data associated with a higher error (see the first row in Table I) by reusing network trained on data associated with a lower error (fourth row in Table I). The obtained results are shown in the second group of rows in Table III. Finally, we repeated the two whole sets of experiments, but reversing the roles of the data distributions associated with the source and target problems in each experiment. In other words, we now aimed to classify data associated with a lower error by reusing networks trained on data associated with a higher error. The obtained results are shown in the third and fourth groups of rows in Table III. The results corresponding to the third set of experiments are also plotted in Fig. 3. The evolution of total training times shown in this figure is representative of what happened also in the other three sets of experiments.

A final set of experiments involved shape data, the target

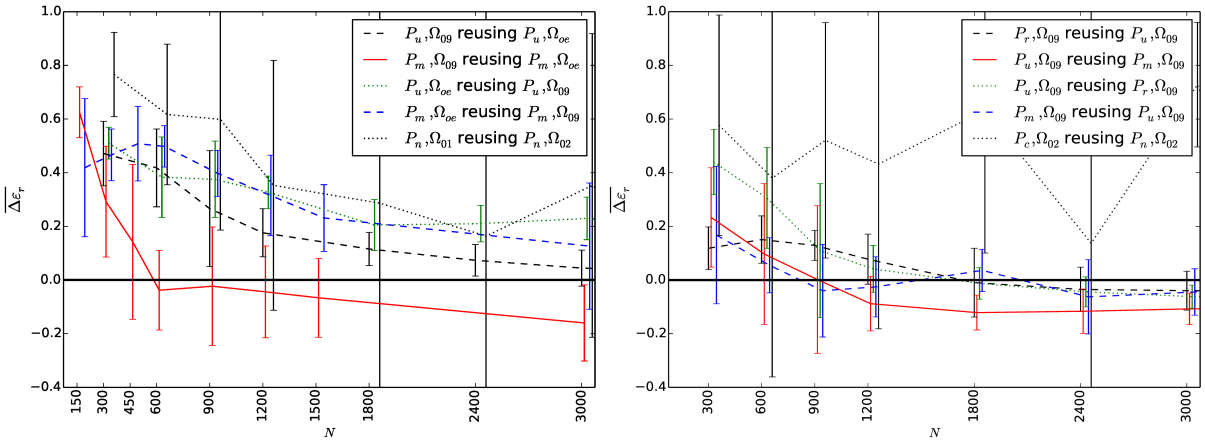

Figure 2. Relative improvement in the average classification test error ($\overline{\Delta\varepsilon_r}$), for different amounts N of data available for training the target network, for transfer learning experiments where the class label set changed (left) and the data distribution changed (right).

Table III. CHANGING THE DATA DISTRIBUTION: PERCENT AVERAGE CLASSIFICATION TEST ERROR ($\bar{\varepsilon}$), FOR DIFFERENT AMOUNTS (N) OF DATA USED TO TRAIN THE TARGET NETWORK AND FOR DIFFERENT COMBINATIONS OF: TARGET DATA DISTRIBUTION (P_T); TARGET LABEL SET (Ω_T); SOURCE DISTRIBUTION (P_S); SOURCE LABEL SET (Ω_S); AND RETRAINED LAYERS. STANDARD DEVIATIONS SHOWN IN PARENTHESES.

P_T	Ω_T	P_S	Ω_S	Retrained layers	N = 300	600	900	1200	1800	2400	3000	6000
P_r	Ω_{09}				63.8 (1.4)	55.2 (1.8)	46.0 (1.1)	38.4 (0.8)	**31.1** (0.9)	**28.4** (1.0)	**25.9** (0.6)	**20.8** (0.5)
P_r	Ω_{09}	P_u	Ω_{09}	[1,0,0]	60.3 (2.1)	52.9 (1.4)	48.0 (1.1)	45.3 (1.1)	42.4 (1.1)	39.2 (1.3)	36.8 (1.2)	30.8 (1.2)
P_r	Ω_{09}	P_u	Ω_{09}	[1,1,0]	**56.8** (2.2)	49.1 (2.6)	43.1 (0.8)	40.4 (0.5)	36.6 (0.5)	33.9 (0.7)	31.5 (0.7)	25.5 (0.5)
P_r	Ω_{09}	P_u	Ω_{09}	[1,1,1]	57.7 (1.8)	**48.0** (2.2)	**40.7** (1.1)	**36.5** (0.7)	32.5 (1.4)	30.1 (1.2)	27.5 (0.5)	21.8 (0.4)
P_u	Ω_{09}				27.3 (1.0)	18.9 (0.9)	**13.3** (0.4)	**11.6** (0.4)	**9.3** (0.1)	**8.7** (0.3)	**7.8** (0.2)	**5.9** (0.2)
P_u	Ω_{09}	P_m	Ω_{09}	[1,0,0]	25.8 (1.7)	19.3 (1.2)	15.5 (0.7)	13.9 (0.6)	11.9 (0.7)	10.7 (0.6)	9.7 (0.4)	7.6 (0.3)
P_u	Ω_{09}	P_m	Ω_{09}	[1,1,0]	23.4 (1.4)	**17.9** (1.2)	14.5 (1.1)	12.7 (0.5)	10.4 (0.3)	9.6 (0.3)	8.7 (0.3)	6.7 (0.2)
P_u	Ω_{09}	P_m	Ω_{09}	[1,1,1]	**22.4** (2.7)	18.2 (2.4)	14.6 (0.7)	12.9 (0.6)	10.5 (0.4)	9.7 (0.5)	8.6 (0.3)	6.5 (0.2)
P_u	Ω_{09}				27.3 (1.0)	18.9 (0.9)	13.3 (0.4)	11.6 (0.4)	**9.3** (0.1)	**8.7** (0.3)	**7.8** (0.2)	**5.9** (0.2)
P_u	Ω_{09}	P_r	Ω_{09}	[1,0,0]	28.7 (1.7)	21.2 (0.7)	18.1 (1.9)	15.3 (0.7)	13.2 (0.6)	11.8 (0.7)	10.8 (0.7)	8.5 (0.4)
P_u	Ω_{09}	P_r	Ω_{09}	[1,1,0]	21.5 (2.6)	16.0 (1.1)	14.0 (0.9)	12.5 (0.6)	10.5 (0.5)	9.7 (0.7)	8.9 (0.5)	6.9 (0.2)
P_u	Ω_{09}	P_r	Ω_{09}	[1,1,1]	**16.4** (1.1)	**14.1** (1.5)	**13.1** (1.2)	**11.3** (0.5)	9.5 (0.2)	9.1 (0.3)	8.3 (0.2)	6.4 (0.2)
P_m	Ω_{09}				11.9 (1.6)	7.6 (0.4)	**6.3** (0.4)	**5.2** (0.3)	4.7 (0.4)	**3.9** (0.4)	**3.5** (0.3)	**2.3** (0.2)
P_m	Ω_{09}	P_u	Ω_{09}	[1,0,0]	11.9 (1.0)	8.8 (1.0)	7.4 (0.6)	6.0 (0.5)	4.9 (0.5)	4.6 (0.5)	4.0 (0.3)	2.5 (0.2)
P_m	Ω_{09}	P_u	Ω_{09}	[1,1,0]	11.4 (1.7)	**7.3** (0.7)	6.4 (0.8)	5.3 (0.4)	**4.4** (0.2)	4.1 (0.3)	3.7 (0.4)	**2.3** (0.2)
P_m	Ω_{09}	P_u	Ω_{09}	[1,1,1]	**10.5** (0.5)	**7.3** (0.5)	6.8 (0.5)	5.5 (0.4)	4.6 (0.2)	4.1 (0.2)	3.6 (0.2)	**2.3** (0.2)
P_c	Ω_{02}				18.5 (12.5)	13.2 (10.1)	9.6 (6.0)	9.1 (4.7)	10.5 (6.1)	7.7 (5.7)	8.1 (4.2)	5.7 (4.9)
P_c	Ω_{02}	P_n	Ω_{02}	[1,0,0,0]	4.9 (3.1)	4.6 (3.5)	**3.6** (3.0)	**3.6** (3.0)	**2.8** (2.4)	**2.7** (1.7)	**2.0** (2.3)	1.8 (1.8)
P_c	Ω_{02}	P_n	Ω_{02}	[1,1,0,0]	4.6 (2.8)	4.4 (3.6)	4.0 (3.0)	4.0 (2.6)	3.6 (2.8)	3.2 (2.2)	3.7 (2.1)	**1.7** (1.4)
P_c	Ω_{02}	P_n	Ω_{02}	[1,1,1,0]	4.7 (4.5)	**3.9** (3.9)	4.7 (4.4)	4.3 (3.9)	4.1 (3.3)	3.8 (2.8)	4.6 (3.7)	2.9 (2.5)
P_c	Ω_{02}	P_n	Ω_{02}	[1,1,1,1]	**4.3** (4.0)	4.6 (4.2)	5.3 (4.4)	4.4 (4.5)	4.4 (3.1)	3.9 (2.8)	4.4 (3.1)	3.1 (2.7)

problem being that of classifying canonical shapes (i.e. circles, squares, and equilateral triangles) as either ellipses, rectangles, or triangles, while the source problem was that of classifying non-canonical shapes (i.e. generic ellipses, rectangles, and triangles) using the same set of three classes.

Fig. 2 (right) shows the relative improvement in the average test error ($\overline{\Delta\varepsilon_r}$) obtained when transfer learning was used instead of training a network from scratch. This error is plotted for the five sets of transfer learning experiments addressed in this section. For experiments with digit data, the shown results are for the case denoted as [1,1,1] in Table III. For experiments with shape data, the considered case is that denoted as [1,0,0,0].

V. DISCUSSION AND CONCLUSIONS

In this work we explored the transfer of knowledge between classification problems involving images of digits. In our experiments, the source problem and the target problem differed only in terms of the set of class labels or only in terms of the data distribution. Our goal was to study how the performance of transfer learning between such problems was affected by simultaneously varying the number of layers being retrained and the amount of data used to perform that retraining.

A comparison of Figs. 2 left and right shows that, in general, reusing source networks trained for a different label set led to higher improvements in the classification error than reusing networks trained for a different data distribution. This can also be observed by comparing the results shown in Figs. 1 and 3, as the sets of experiments they correspond to both dealt with the same target problem of classifying handwritten upright digits into ten classes (note how the baseline plots are the same).

Retraining only one layer led to the worst performance in all sets of experiments with digit data. So, even though retraining the output layer is unavoidable when class labels change and retraining the first hidden layer is (presumably) important when the data distribution changes, in practice more than one layer should be retrained for successful knowledge transfer to occur. In contrast, with shape data and a change in the data distribution, retraining only the input layer was sufficient to yield the best or nearly best results.

In transfer learning experiments involving a change of label set with digit data, retraining the second hidden layer and the output layer of the source networks resulted in the best performance while saving significant training time, in relation to fully training the target networks from scratch, as

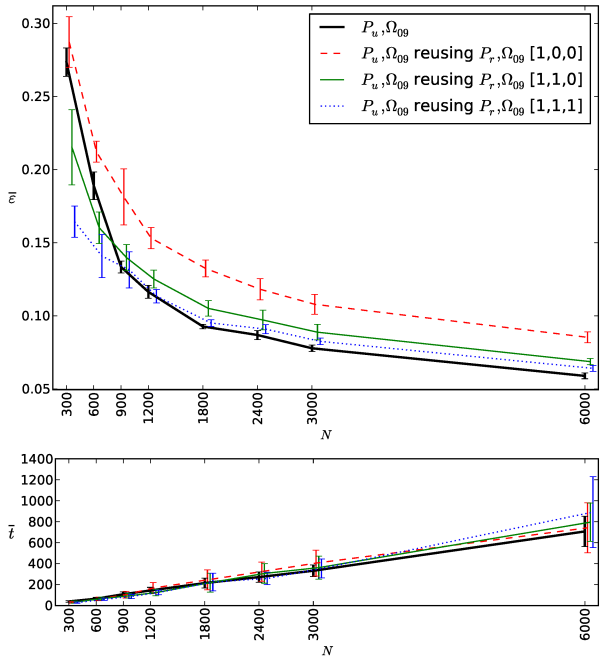

Figure 3. Results when classifying handwritten upright digits into ten classes by reusing a network trained for rotated digits: average classification test error ($\bar{\varepsilon}$) and average total training time ($\bar{t}$, in seconds), for different amounts (N) of data used to train the target network and for different numbers of retrained layers.

exemplified in Fig. 1. In the case of shape data, retraining the whole network was often necessary to obtain the best performance.

When the target problem was that of classifying digits into ten classes (first and second groups of rows in Table II) and only a limited amount of data was available for training, it was better to reuse a network trained for only two classes than to train a network from scratch for ten classes. In the case of machine-printed digits this advantage was only evident for very low values of N (up to 300 samples), but in the case of handwritten digits it was visible up to values around 2400 samples (Fig. 1). On the other hand, when the target problem involved classifying digits into only two classes (third and fourth groups in Table II), it was always better to reuse a network trained for ten classes than to train a network from scratch for two classes, regardless of the amount of data available for training. These conclusions can also be drawn from Fig. 2(left).

In experiments with shape data, which targeted three classes by reusing networks trained for only two classes, a relative improvement in the error was only clear for values of N up to 900 samples.

In transfer learning experiments involving a change of data distribution with digit data, no significant differences were observed between the times taken to fully train the target network from scratch and to retrain any number of layers from the source network, as exemplified in Fig. 3.

In experiments involving handwritten digits and machine-printed digits (second and fourth in the legend of Fig. 2 on the right), transfer learning was beneficial only when the data available for training the target network did not exceed 300 samples. In experiments involving only handwritten digits (first and third in the legend), this advantage held for higher values of N. Thus, when classifying upright digits reusing networks trained for rotated digits, we observed pronounced error improvements up to 600 samples; when classifying rotated digits reusing networks trained for upright digits, the improvements were more modest, but they persisted for higher values of N (up to 1200 samples).

In experiments that targeted canonical shapes by reusing networks trained with non-canonical shapes, the improvement in the error showed a tendency to remain constant regardless of the amount of data available for training. However, the observed standard deviations are very large and do not allow a definitive conclusion.

The results obtained when classifying handwritten upright digits by reusing networks trained for rotated digits (third in the legend of Fig. 2 on the right) are particularly interesting, as they raise the hypothesis that, in classification problems where only a small amount of labelled image data is available for training, it may be possible to take advantage of transfer learning to achieve improved performance. One possible approach would be to: a) perform random rotations (or another transformation) on each instance of the original data, in order to generate a larger amount of data; b) use the generated data to train a source network; and c) retrain that source network using the small amount of original images, to obtain a better network than would be possible by using only the original images. Future work should investigate this possibility, and also involve a wider variety of data types and larger architectures in experiments with digit data.

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] Lorenzo Bruzzone and Mattia Marconcini. Domain adaptation problems: a DASVM classification technique and a circular validation strategy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5): 770–787, 2010.

[4] Dan Claudiu Ciresan, Ueli Meier, and Jürgen 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. Heigold, V. Vanhoucke, A. Senior, P. Nguyen, M. Ranzato, M. Devin, and J. Dean. Multilingual acoustic models using distributed deep neural networks. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

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

[9] J.-T Huang, J. Li, D. Yu, L. Deng, and Y. Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[10] 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*, pages 473–480, 2007.

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