Transfer Learning for Latin and Chinese Characters with Deep Neural Networks

Dan C. Cireşan
IDSIA
USI-SUPSI
Manno, Switzerland, 6928
Email: dan@idsia.ch

Ueli Meier
IDSIA
USI-SUPSI
Manno, Switzerland, 6928
Email: ueli@idsia.ch

Jürgen Schmidhuber
IDSIA
USI-SUPSI
Manno, Switzerland, 6928
Email: juergen@idsia.ch

***Abstract*—We analyze transfer learning with Deep Neural Networks (DNN) on various character recognition tasks. DNN trained on digits are perfectly capable of recognizing uppercase letters with minimal retraining. They are on par with DNN fully trained on uppercase letters, but train much faster. DNN trained on Chinese characters easily recognize uppercase Latin letters. Learning Chinese characters is accelerated by first pretraining a DNN on a small subset of all classes and then continuing to train on all classes. Furthermore, pretrained nets consistently outperform randomly initialized nets on new tasks with few labeled data.**

I. Introduction

Knowing how to drive a car helps to learn more quickly how to drive a truck. Learning French is easier if you already know a Latin language. Learning the second language is easier than the first. Mathematics prepares students to study physics. Learning to get along with one's siblings may prepare one for getting along with others. Chess playing experience might make one a better strategic political or business thinker.

Such musings motivate our investigation of transfer learning, where new tasks and concepts are learned more quickly and accurately by exploiting past experience. In its most general form, transfer learning occurs when learning in one context enhances (positive transfer) or undermines (negative transfer) a related performance in another context. Transfer is a key ingredient of human learning: humans are often able to generalize correctly from a single training example. Unlike most machine learners, however, humans are trained on many different learning problems over their lifetime.

Although there is no generally accepted definition of transfer learning, many have found that neural nets (NN) pretrained on one task can learn new, related tasks more quickly. For example, [1] investigates if learning the n-th thing is any easier than learning the first, and concludes from experiments that methods leveraging knowledge from similar learning tasks outperform models trained on a single task. Similarly, multitask learning [2] can improve generalization capabilities by learning several tasks in parallel while using a shared representation. From an NN point of view this is most easily implemented by a classification network with multiple outputs, one per class in the classification task. A similar paradigm is self-taught learning, or transfer learning from unlabeled data [3]. This approach is especially promising if labeled data sets for the various learning tasks are not available.

Here we focus on NN-based classifiers and experimentally investigate the effect of transfer learning on well-defined problems where enough labeled training data are available. In particular, we train deep NN (DNN) on uppercase letters and digits from NIST SD 19 [4], as well as on Chinese characters provided by the Institute of Automation of the Chinese Academy of Sciences (CASIA [5]). Training a classifier on the Latin alphabet (up to 52 classes) is a rather straightforward problem, whereas training a classifier on the GB1 subset of Chinese characters (3755 classes) already poses a major challenge for any NN-based classifier that takes raw pixel intensities as its input.

DNN consist of many layers. All but the last layer can be interpreted as a general purpose feature extractor that maps an input into a fixed dimensional feature vector. Usually all the weights are randomly initialized and trained by back-propagation [6], [7], sometimes after unsupervised layer-wise pretraining [8], [9], [10]. Here we investigate if weights of all but the last layer, trained on a given task, can be reused on a different task. Instead of randomly initializing a net we start from an already trained feature extractor, transferring knowledge from an already learned task to a new task. For the new task only the last classification layer needs to be retrained, but if desired, any layer of the feature extractor might also be fine-tuned. As we will show, this strategy is especially promising for classification problems with many output classes, where good weight initialization is of crucial importance.

In what follows we will shortly describe the deep neural network architecture and give a detailed description of the various experiments we performed.

II. Deep Neural Network architecture

Our DNN [7]consists of a succession of convolutional and max-pooling layers. It is a hierarchical feature extractor that maps raw pixel intensities of the input image into a feature vector to be classified by a few, we generally use 2 or 3, fully connected layers. All adjustable parameters are jointly optimized, minimizing the misclassification error over the training set.

A. Convolutional layer

Each convolutional layer performs a 2D convolution of its M^{n-1} input maps with a filter of size K_x, K_y. The resulting activations of the M^n output maps are given by the sum of the M^{n-1} convolutional responses which are passed through a nonlinear activation function:

$$\mathbf{Y}_j^n = f(\sum_i \mathbf{Y}_i^{n-1} * \mathbf{W}_{ij}^n), \quad (1)$$

where n indicates the layer, $\mathbf{Y}$ is a map of size M_x, M_y, and $\mathbf{W}_{ij}$ is a filter of size K_x, K_y connecting input map i with output map j, and $*$ is the valid 2D convolution. That is, for an input map $\mathbf{Y}^{n-1}$ of size M_x^{n-1}, M_y^{n-1} and a filter $\mathbf{W}$ of size K_x, K_y the output map $\mathbf{Y}^n$ is of size $M_x^n = M_x^{n-1} - K_x + 1$, $M_y^n = M_y^{n-1} - K_y + 1$.

B. Max-pooling layer

The biggest architectural difference between our DNN and the CNN of [11] is the use of max-pooling layers [12], [13], [14] instead of sub-sampling layers. The output of a max-pooling layer is given by the maximum activation over non-overlapping rectangular regions of size K_x, K_y. Max-pooling creates slight position invariance over larger local regions and down-samples the input image by a factor of K_x and K_y along each direction. In the implementation of [15] such layers are missing, and instead of performing a pooling or averaging operation, nearby pixels are simply skipped prior to convolution.

C. Classification layer

Kernel sizes of convolutional filters and max-pooling rectangles are chosen such that either the output maps of the last convolutional layer are down-sampled to 1 pixel per map, or a fully connected layer combines the outputs of the last convolutional layer into a 1D feature vector. The last layer is always a fully connected layer with one output unit per class in the recognition task. We use a softmax activation function for the last layer such that each neuron's output activation can be interpreted as the probability of a particular input image belonging to that class.

D. Training procedure

During training a given dataset is continually deformed prior to each epoch of an online learning algorithm. Deformations are stochastic and applied to each image during training, using random but bounded values for translation, rotation and scaling. These values are drawn from a uniform distribution in a specified range, i.e. $\pm 10\%$ of the image size for translation, $0.9 - 1.1$ for scaling and $\pm 5^\circ$ for rotation. The final image is obtained using bilinear interpolation of the distorted input image. These distortions allow us to train DNN with many free parameters without overfitting and greatly improve generalization performance. All DNN are trained using on-line gradient descent with an annealed learning rate. Training stops when either the validation error becomes 0, the learning rate reaches its predefined minimum or there is no improvement on the validation set for 50 consecutive epochs. The undistorted, original training set is used as validation set.

III. Transfer learning

We start by fully training a net on a given task. After training stops, we keep the net with the smallest error on the validation dataset and change the number of neurons in the output layer to match the number of classes in the new classification task (e.g. if we train on digits before transferring to letters, the output layer size will grow from 10 to 26 neurons). The output layer weights are reinitialized randomly; the weights of remaining layers are not modified.

The net pretrained on the source task is then retrained on the destination task. We check performance by fixing all but the last n layers and retraining only the last n layers. To see how training additional layers influences performance, we begin by only training the last layer, then the last two, etc., until all layers are trained. Since max-pooling layers do not have weights, they are neither trained nor retrained (their fixed processing power resides in the maximum operator).

One way of assessing the performance of transfer learning is to compare error rates of nets with pretrained weights to those of randomly initialized nets. For all experiments we list recognition error rates of pre-trained and randomly initialized nets whose n top layers were trained.

IV. Experiments

All experiments are performed on a computer with i7-950 (3.33GHz), 16GB RAM and 4 x GTX 580. We use the GPU implementation of a DNN from [7]. This allows for performing all experiments with big and deep DNN on huge datasets within several days.

We experiment with Latin characters [4] and Chinese characters [5]. We test transfer learning within each dataset, but also from the Latin alphabet to Chinese characters. With so many different tasks, i.e. digits, lowercase letters, uppercase letters, letters (case insensitive), letters (case sensitive), Chinese characters, one can try transfer learning in many ways. Instead of trying them all, we select the hardest and most interesting ones. Here we consider a transfer learning problem hard if the number of classes or the complexity of the symbols increases. We perform transfer learning from digits to uppercase letters, from Chinese characters to uppercase Latin letters and from uppercase Latin letters to Chinese characters. In total there are 344307 digits for training and 58646 for testing; 69522 uppercase letters for training and 11941 for testing. For the Chinese character classification task there are 3755 different classes, which poses a major challenge for any supervised classifier, mainly because the dataset is huge (more than one Million samples equaling 2.5GB of data). We therefore evaluate the DNN on 1000 instead of all 3755 classes, resulting in 239121 characters for training and 59660 for testing. This is sufficient for a proof of concept that transfer learning between different datasets works. We also investigate the effect of pretraining a DNN on subsets of the classes,

i.e., we use subsets of 10 and 100 out of the 1000 classes to pretrain a DNN.

A. Latin characters: from digits to uppercase letters

For Latin characters the simplest symbols are the digits. Letters are generally more complex; there are also more classes, rendering the letter task even more difficult. We choose uppercase letters as destination task because the data has high quality (few mislabeled images) and less confusion between similar classes than lowercase letters. Since classification accuracy is not degraded by labeling errors and confused classes, results are more easily compared.

All characters from NIST SD 19 are scaled to fit a 20x20 pixel bounding box which is then placed in the middle of a 29x29 pixel image. The empty border around the actual character allows moderate distortions without falling outside the 29x29 box. As in [16], we use rotation of max. $\pm 15°$, scaling of max. $\pm 15\%$, and translation of max. $\pm 15\%$. For elastic distortions we use a Gaussian kernel with $\sigma = 6$ and an amplitude of 36 (see [15] for an explanation of these parameters).

In our previous work [16] on NIST SD 19 data we used relatively small and shallow nets. That was fine to train many nets and build a committee. Here the focus is on obtaining good results as quickly as possible, hence we only train one net, although we use a big and deep DNN this time. Its architecture is detailed in Table I. Filter sizes for convolutional and max pooling layers are chosen as small as possible (2 or 3) to get a deep net. The three stages of convolution and max-pooling layers (the feature extractors) are followed by a classifier formed by one fully connected layer with 200 neurons and the output layer. The learning rate starts at 0.001 and is annealed by a factor of 0.993 after each epoch.

TABLE I
8 LAYER DNN ARCHITECTURE USED FOR NIST SD 19.

Layer	Type	# maps & neurons	kernel
0	input	1 maps of 29x29 neurons	
1	convolutional	50 maps of 28x28 neurons	2x2
2	max pooling	50 maps of 14x14 neurons	2x2
3	convolutional	100 maps of 12x12 neurons	3x3
4	max pooling	100 maps of 6x6 neurons	2x2
5	convolutional	150 maps of 4x4 neurons	3x3
6	max pooling	150 maps of 2x2 neurons	2x2
7	fully connected	200 neurons	1x1
8	fully connected	10 or 26 neurons	1x1

A randomly initialized net fully trained on uppercase letters reaches a low error rate of 2.07% (Table II), and 0.32% if we consider the first two predictions. This indicates that 84.5% of the errors are due to confusions between similar classes. The error slowly increases if the first two convolutional layers (1 and 3) are not trained. It is worth noting that with random convolutional filters, in layer 1 and 3, very competitive results are obtained, an intriguing finding already noted and investigated elsewhere [17], [18], [19]. However, when no convolutional layer is trained the error spikes to almost twelve percent. Transferring the weights learned on the digit task to the uppercase letter task (second row in Table II) yields good results even if only the last two fully connected layers are retrained.

TABLE II
TEST ERRORS [%] FOR NETS PRETRAINED ON DIGITS AND TRANSFERED TO UPPERCASE LETTERS.

	First trained layer				
initialization	1	3	5	7	8
random	2.07	2.47	2.7	11.74	38.44
DIGITS	2.09	2.11	2.23	2.36	4.13

In addition, learning from pretrained nets is very fast compared to learning from randomly initialized nets. In Figure 1 test error rates [%] on uppercase letters are shown as a function of training time [s], for both randomly initialized (solid, blue) and pretrained (dotted, red) nets trained from the fifth (left) and seventh (right) layer onwards. The pretrained nets start from a much lower error rate, and if only the last two fully connected layers are (re-)trained (right), the randomly initialized net never manages to match the pretrained net even after 10000 s of training. If the last convolutional layer is also (re-)trained (left), the pretrained net is much better after 1000 s of training, but as training proceeds the difference between the two nets becomes smaller.

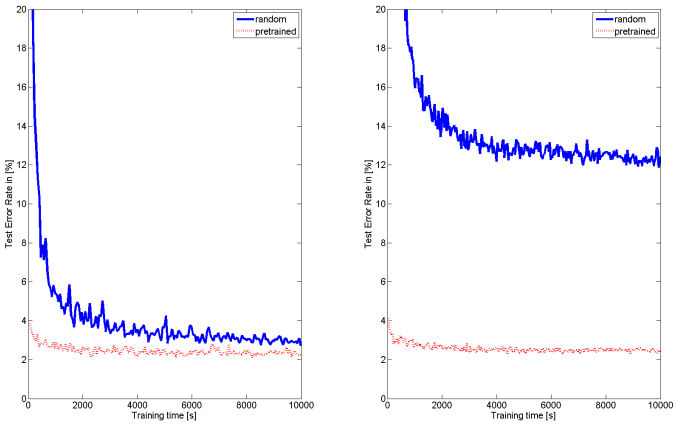

Fig. 1. Test error rates [%] on uppercase letters as a function of training time [s] for randomly initialized nets (solid, blue) and nets pretrained on digits (dotted, red). Both nets are (re-)trained on uppercase letters starting from the fifth (left) and seventh (right) layer, respectively.

We also check the performance of a fully trained small net that only has the classification layers (i.e. input layer followed directly by the fully connected layers: 7 and 8) of the DNN. When both fully connected layers are present (the net has one hidden layer with 200 neurons) the error is 4.03%, much higher than the corresponding 2.36% of the pretrained net. If only the output layer is used (i.e. a net with input layer followed by the output layer), the error goes up to 42.96%, which is much higher than 4.13%, and even higher then the random net's error. The take home message is that convolutional layers, even when not retrained, are essential for a low error rate on the destination task.

B. Learning uppercase letters from few samples per class

For classification tasks with a few thousand samples per class, the benefit of (unsupervised/supervised) pretraining is not easy to demonstrate. After sufficient training, a randomly initialized net will eventually become as good as the pretrained net. The benefits of unsupervised pretraining are most evident when the training set has only few labeled data samples. We therefore investigate how a randomly initialized net compares to a net pretrained on digits, (re-)training both nets on 10, 50, 100, 500 and 1000 samples per class, respectively. In Table III the error rates on the original uppercase test set (≈2500 samples per class) are listed for all experiments. As expected, the difference between the pretrained and randomly initialized net is the bigger the fewer samples are used. Using only 10 samples per class, the random net reaches an impressive error rate of 17.85% with and 34.60% without distortions, indicating that the distortions capture the writing style variations of uppercase letters very well. Nevertheless, retraining the pretrained net on only 10 samples per class results in a much lower error rate of 8.98% and 12.51% with and without distortions, still better than the error of the random net trained on distorted samples. The effects of distortions for the pretrained net are not as severe as for random nets. This indicates that the information extracted from the digit task as encoded in the network weights has been successfully transferred to the uppercase letter task by means of a much better network initialization.

TABLE III
TEST ERRORS [%] FOR NETS PRETRAINED ON DIGITS AND TRANSFERED TO UPPERCASE LETTERS. THE EXPERIMENTS ARE PERFORMED ON DISTORTED (DIST.) AND UNDISTORTED TRAINING SAMPLES.

	Number of training samples per class				
initialization	10	50	100	500	1000
random + dist.	17.85	5.06	4.07	2.55	2.38
DIGITS + dist.	8.98	4.96	3.77	2.56	2.52
random	34.60	15.17	10.49	5.15	4.42
DIGITS	12.51	6.54	5.03	3.30	3.12

C. Chinese characters to uppercase Latin letters

We continue the experiments with a more difficult problem, namely, transfer learning between two completely different datasets. From previous experiments [20] we know that 48x48 pixels are needed to represent the intricate details of the Chinese characters. We use a similar but smaller net than the one used for the competition, because we need to run many more experiments. As for the net used on Latin characters, we choose small filters to obtain a deep net (Table IV). Because the input size of a DNN is fixed, uppercase letters are scaled from 29x29 to 48x48 pixel.

The net fully trained on uppercase letters has a slightly lower error rate (1.89% Table V) than the one used in Subsection IV-A. This can be attributed to the deeper and bigger net. When the first layers are not trained, the error increases slightly and is twice as big as when only one convolutional layer (the seventh layer) is trained. If only the fully connected layers are trained, the error increases dramatically. We continue the experiments with nets pretrained on Chinese characters. Transfer learning works very well for this problem; the errors are lower than those of training random nets, even if the first three convolutional layers (1, 3 and 5) are kept fixed. It seems that filters trained on Chinese characters can be fully reused on Latin characters. This was expected, because although Chinese characters are more complex than Latin ones, they are written in the same way: a sequence of strokes. Even if the first nine layers are not trained and only the output layer is trained, a surprisingly low 3.35% test error rate is obtained.

TABLE IV
10 LAYER DNN ARCHITECTURE USED FOR CHINESE CHARACTERS.

Layer	Type	# maps & neurons	kernel
0	input	1 maps of 48x48 neurons	
1	convolutional	100 maps of 46x46 neurons	3x3
2	max pooling	100 maps of 23x23 neurons	2x2
3	convolutional	150 maps of 22x22 neurons	2x2
4	max pooling	150 maps of 11x11 neurons	2x2
5	convolutional	200 maps of 10x10 neurons	2x2
6	max pooling	200 maps of 5x5 neurons	2x2
7	convolutional	250 maps of 4x4 neurons	2x2
8	max pooling	250 maps of 2x2 neurons	2x2
9	fully connected	500 neurons	1x1
10	fully connected	10 or 26 or 100 or 1000 neurons	1x1

TABLE V
TEST ERRORS [%] FOR NETS PRETRAINED ON CHINESE CHARACTERS AND TRANSFERED TO UPPERCASE LETTERS.

	First trained layer					
initialization	1	3	5	7	8	9
random	1.89	2.04	2.28	3.53	10.45	44.59
CHINESE 1000	1.79	1.89	1.91	1.88	2.22	3.35

D. Chinese characters: speeding up training

Training big and deep DNN is very time-consuming even on GPUs, especially for classification tasks with many output classes, resulting in a huge training set. One way of speeding up training is to pretrain a DNN on a subset of the classes, i.e. only 1% or 10% of the classes. After pretraining, the net is retrained on the full problem, hoping that the initialization reduces training time.

When starting from a random initialization (first row in Table VI), the error rate is increasing rapidly with the number of untrained layers, reaching more than 20% when the first three convolutional layers are not trained. As soon as the net is pretrained on 1% of the classes (second row in Table VI), the results improve drastically: retraining the last three layers results in an error rate of 7.76%, almost three times lower than before. Pretraining on 10% of the classes greatly improves the results. Even training only the output layer yields 8.56%, and training of the first two convolutional layers becomes irrelevant.

In Figure 2 test error rates [%] on Chinese 1000 are shown as a function of training time [s], for a randomly initialized

TABLE VI
TEST ERRORS [%] FOR NETS PRETRAINED ON CHINESE-10 OR CHINESE-100 CHARACTERS OR UPPERCASE LETTERS AND TRANSFERRED TO CHINESE-1000.

	First trained layer					
initialization	1	3	5	7	8	9
random	4.84	6.28	9.49	20.82	90.05	96.86
CHINESE 10	4.79	5.02	5.53	7.76	15.37	28.15
CHINESE 100	4.77	4.91	4.83	5.51	6.89	8.56
uppercase	4.80	4.85	5.70	8.42	17.39	31.10

net (solid, blue) and nets pretrained on Chinese 10 (dotted, red) and Chinese 100 (dash-dotted, green). Training started from the fifth (left) and seventh (right) layer onwards. The pretrained nets reach a much lower error rate after shorter training time. If only the last two fully connected layers are (re-)trained (right), the randomly initialized net never reaches the pretrained nets, not even after 50000 s of training; here the difference between pretraining on 10 vs 100 classes is more severe. If the last convolutional layer is also (re-)trained (left), the pretrained nets are still much better, but the difference between random initialization and pretraining is smaller.

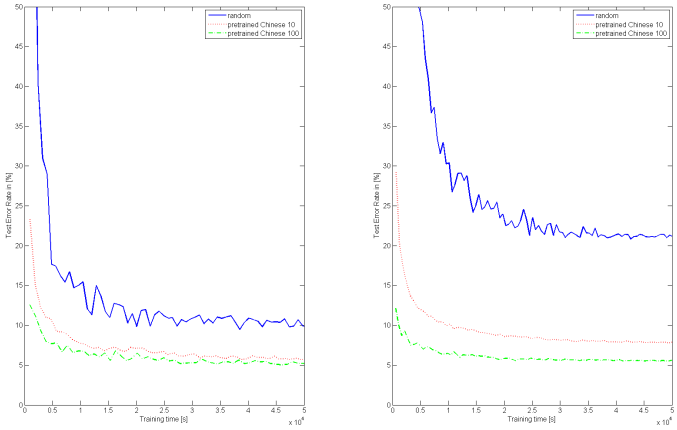

Fig. 2. Test error rates [%] on Chinese 1000 as a function of training time [s] for randomly initialized nets (solid) and nets pretrained on subsets of 10 (red, dotted) and 100 (green, dash-dotted) classes. All nets are (re-)trained on Chinese 1000 starting from the fifth (left) and seventh (right) layer, respectively.

E. Uppercase letters to Chinese characters

This is the most challenging task because Chinese characters (destination task) are much more complex than uppercase letters (source task). Moreover, resizing the letters from 29x29 to 48x48 made them have much thicker strokes than the Chinese characters. Despite these shortcomings, pretraining on uppercase letters is almost as good as pretraining on Chinese-10 (compare rows 2 and 4 in Table VI).

V. CONCLUSION

Transfer learning between different handwritten character recognition tasks is possible. In particular, transfer learning from Latin letters to Chinese characters works as well as pretraining a net with 1% of the classes of the Chinese training task, despite the lower apparent complexity of Latin letters. Advantages of transfer learning include: less training time is needed to obtain good results, and much better results are obtained when only few labeled samples per class are available for the destination task.

Unsupervised learning seems to be the most popular choice to pretrain deep neural networks, but here we show that deep networks can also be pretrained on either different labeled datasets or on subsets of the training set. Pretrained deep nets with frozen weights in the first n layers can serve as rather universal feature extractors. For new classification tasks, only the last layer needs to be re-initialized and retrained to accommodate for the changing number of output classes. Fine-tuning of the remaining layers is optional. Consequently, pretrained nets are much faster to train on new tasks. This is of particular importance for classification tasks with thousands of output classes (as often encountered in real world applications), where it is worthwhile to pretrain the shallow layers of a DNN on a much smaller subset of the whole training set and then only retrain the last two fully connected layers. Furthermore, nets pretrained on different tasks can be reused on new tasks, offering a highly practical alternative to fully unsupervised pre-training.

ACKNOWLEDGMENT

This work was partially supported by a FP7-ICT-2009-6 EU Grant under Project Code 270247: A Neuro-dynamic Framework for Cognitive Robotics: Scene Representations, Behavioral Sequences, and Learning.

REFERENCES

[1] S. Thrun, "Is learning the n-th thing any easier than learning the first?" in *Advances in Neural Information Processing Systems*. The MIT Press, 1996, pp. 640–646.

[2] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, pp. 41–75, 1997.

[3] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 759–766.

[4] P. J. Grother, "NIST special database 19 - Handprinted forms and characters database," National Institute of Standards and Thechnology (NIST), Tech. Rep., 1995.

[5] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang, "Chinese Handwriting Recognition Contest," in *Chinese Conference on Pattern Recognition*, 2010.

[6] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[7] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *International Joint Conference on Artificial Intelligence*, 2011, pp. 1237–1242.

[8] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, 2006.

[9] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Neural Information Processing Systems*, 2007.

[10] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.
[12] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nat. Neurosci.*, vol. 2, no. 11, pp. 1019–1025, 1999.
[13] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," in *Proc. of Computer Vision and Pattern Recognition Conference*, 2005.
[14] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International Conference on Artificial Neural Networks*, 2010.
[15] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Seventh International Conference on Document Analysis and Recognition*, 2003, pp. 958–963.
[16] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *International Conference on Document Analysis and Recognition*, 2011, pp. 1250–1254.
[17] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. International Conference on Computer Vision*, 2009.
[18] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *International Conference on Machine Learning*, 2011.
[19] A. M. Saxe, P. W. Koh, Z. Chen, M. Bh, B. Suresh, and A. Y. Ng, "On random weights and unsupervised feature learning," in *International Conference on Machine Learning*, 2011.
[20] C.-L. Liu, F. Yin, Q.-F. Wang, and D.-H. Wang, "ICDAR 2011 chinese handwriting recognition competition," in *11th International Conference on Document Analysis and Recognition*, 2011, pp. 1464–1469.