

## Deep, Big, Simple Neural Nets for Handwritten Digit Recognition

**Dan Claudiu Cireşan**

*dan@idsia.ch*

**Ueli Meier**

*ueli@idsia.ch*

**Luca Maria Gambardella**

*luca@idsia.ch*

**Jürgen Schmidhuber**

*juergen@idsia.ch*

*IDSIA, 6928 Manno-Lugano, Switzerland, and University of Lugano,  
6900 Lugano, Switzerland, and Scuola universitaria professionale  
della svizzera italiana, 6928 Manno-Lugano, Switzerland*

**Good old online backpropagation for plain multilayer perceptrons yields a very low 0.35% error rate on the MNIST handwritten digits benchmark. All we need to achieve this best result so far are many hidden layers, many neurons per layer, numerous deformed training images to avoid overfitting, and graphics cards to greatly speed up learning.**

### 1 Introduction ---

Automatic handwriting recognition is of academic and commercial interest. Current algorithms are already quite good at learning to recognize handwritten digits. Post offices use them to sort letters and banks to read personal checks. MNIST (LeCun, Bottou, Bengio, & Haffner, 1998) is the most widely used benchmark for isolated handwritten digit recognition. More than a decade ago, artificial neural networks called multilayer perceptrons (MLPs; Werbos, 1974; LeCun, 1985; Rumelhart, Hinton, & Williams, 1986) were among the first classifiers tested on MNIST. Most had few layers or few artificial neurons (units) per layer (LeCun et al., 1998), but apparently back then, they were the biggest feasible MLPs, trained when CPU cores were at least 20 times slower than today. A more recent MLP with a single hidden layer of 800 units achieved 0.70% error (Simard, Steinkraus, & Platt, 2003). However, more complex methods listed on the MNIST Web page always seemed to outperform MLPs, and the general trend went toward more and more complex variants of support vector machines (SVMs; Decoste & Schölkopf, 2002) and combinations of neural networks (NN) and SVMs (Lauer, Suen, & Bloch, 2007). Convolutional neural networks (CNNs) achieved a record-breaking 0.40% error rate (Simard et al., 2003) using novel

elastic training image deformations. Recent methods pretrain each hidden CNN layer one by one in an unsupervised fashion (this seems promising especially for small training sets), then use supervised learning to achieve a 0.39% error rate (Ranzato, Poultney, Chopra, & LeCun, 2006; Ranzato, Huang, Boureau, & LeCun, 2007). The biggest MLP so far (Salakhutdinov & Hinton, 2007) also was pretrained without supervision, then piped its output into another classifier to achieve an error of 1% without domain-specific knowledge. Deep MLPs initialized by unsupervised pretraining were also successfully applied to speech recognition (Mohamed, Dahl, & Hinton, 2009).

Are all these complexifications of plain MLPs really necessary? Can't one simply train really big plain MLPs on MNIST? One reason is that at first glance, deep MLPs do not seem to work better than shallow networks (Bengio, Lamblin, Popovici, & Larochelle, 2006). Training them is hard, as backpropagated gradients quickly vanish exponentially in the number of layers (Hochreiter, 1991; Hochreiter, Bengio, Frasconi, & Schmidhuber, 2001; Hinton, 2007), just as the first recurrent neural networks (Hochreiter & Schmidhuber, 1997). Indeed, previous deep networks successfully trained with backpropagation (BP) either had few free parameters due to weight sharing (LeCun et al., 1998; Simard et al., 2003) or used unsupervised, layer-wise pretraining (Hinton & Salakhutdinov, 2006; Bengio et al., 2006; Ranzato, Poultney, Chopra, & LeCun, 2006). But is it really true that deep BP-MLPs do not work at all, or do they just need more training time? How can this be tested considering that online BP for hundreds or thousands of epochs on large MLPs may take weeks or months on standard serial computers? But can't one parallelize it? On computer clusters, this is hard due to communication latencies between individual computers. Parallelization across training cases and weight updates for mini-batches (Nair & Hinton, 2009) might alleviate this problem, but still leaves the task of parallelizing fully online BP. Only GPUs are capable of such finely grained parallelism. Multithreading on a multicore processor is not easy either. We may speed up BP using streaming single instruction, multiple data extensions either manually or by setting appropriate compiler flags. The maximum theoretical speed-up under single precision floating point, however, is four, which is not enough. And MNIST is large: its 60,000 images take almost 50 MB, too much to fit in the L2/L3 cache of any current processor. This requires continually accessing data in considerably slower RAM. To summarize, currently it is next to impossible to train big MLPs on CPUs.

We will show how to overcome all these problems by training large, deep MLPs on graphics cards.

## 2 Data

---

MNIST consists of two data sets: one for training (60,000 images) and one for testing (10,000 images). Many studies divide the training set into two

sets consisting of 50,000 images for training and 10,000 for validation. Our network is trained on slightly deformed images, continually generated in online fashion; hence, we may use the whole un-deformed training set for validation without wasting training images. Pixel intensities of the original gray-scale images range from 0 (background) to 255 (maximum foreground intensity);  $28 \times 28 = 784$  pixels per image get mapped to real values  $\frac{\text{pixel intensity}}{127.5} - 1.0$  in  $[-1.0, 1.0]$  and are fed into the neural network input layer.

### 3 Architectures

---

We train five MLPs with two to nine hidden layers and varying numbers of hidden units. Mostly, but not always, the number of hidden units per layer decreases toward the output layer (see Table 1). There are 1.34 to 12.11 million free parameters (or weights, or synapses).

We use standard online BP (Russell & Norvig, 2002), without momentum, but with a variable learning rate that shrinks by a multiplicative constant after each epoch, from  $10^{-3}$  down to  $10^{-6}$ . Weights are initialized with a uniform random distribution in  $[-0.05, 0.05]$ . Each neuron's activation function is a scaled hyperbolic tangent:  $y(a) = A \tanh Ba$ , where  $A = 1.7159$  and  $B = 0.6666$  (LeCun et al., 1998).

### 4 Deforming Images to Get More Training Instances

---

So far, the best results on MNIST were obtained by deforming training images, thus greatly increasing their number. This allows for training networks with many weights, making them insensitive to in-class variability. We combine affine (rotation, scaling, and horizontal shearing) and elastic deformations, characterized by the following real-valued parameters:

- $\sigma$  and  $\alpha$ : For elastic distortions emulating uncontrolled oscillations of hand muscles (for details, see Simard et al., 2003).
- $\beta$ : a random angle from  $[-\beta, +\beta]$  describes either rotation or horizontal shearing. In case of shearing,  $\tan \beta$  defines the ratio between horizontal displacement and image height.
- $\gamma_x, \gamma_y$ : For horizontal and vertical scaling, randomly selected from  $[1 - \gamma/100, 1 + \gamma/100]$ .

At the beginning of every epoch, the entire MNIST training set gets deformed. Initial experiments with small networks suggested the following deformation parameters:  $\sigma = 5.0 - 6.0$ ,  $\alpha = 36.0 - 38.0$ ,  $\gamma = 15 - 20$ . Since digits 1 and 7 are similar, they get rotated or sheared less ( $\beta = 7.5^\circ$ ) than other digits ( $\beta = 15.0^\circ$ ).

Table 1: Error Rates on MNIST Test Set.

ID	Architecture (Number of Neurons in Each Layer)	Test Error for Best Validation (%)	Best Test Error (%)	Simulation Time (h)	Weights (Millions)
1	1000, 500, 10	0.49	0.44	23.4	1.34
2	1500, 1000, 500, 10	0.46	0.40	44.2	3.26
3	2000, 1500, 1000, 500, 10	0.41	0.39	66.7	6.69
4	2500, 2000, 1500, 1000, 500, 10	0.35	0.32	114.5	12.11
5	9 × 1000, 10	0.44	0.43	107.7	8.86




































 1 7	 7 1	 9 8	 5 9	 7 9	 3 5	 2 3
 4 9	 3 5	 9 7	 4 9	 9 4	 0 2	 3 5
 1 6	 9 4	 6 0	 0 6	 8 6	 7 9	 7 1
 4 9	 5 0	 3 5	 9 8	 7 9	 1 7	 6 1
 2 7	 5 8	 7 8	 1 6	 6 5	 9 4	 6 0

Figure 1: The 35 misclassified digits of the best network from Table 1, together with the two most likely predictions (bottom, from left to right) and the correct label according to MNIST (top, right).

5 Results

All simulations were performed on a computer with a Core2 Quad 9450 2.66 GHz processor, 3 GB of RAM, and a GTX280 graphics card. The GPU accelerates the deformation routine by a factor of 10 (only elastic deformations are GPU optimized); the forward propagation (FP) and BP routines are sped up by a factor of 40. Implementation details can be found in the appendix. We pick the trained MLP with the lowest validation error and evaluate it on the MNIST test set. Results are summarized in Table 1.

Most remarkable, the best network has an error rate of only 0.35% (35 out of 10,000 digits). This is significantly better than the best previously published results—0.39% by Ranzato et al. (2006) and 0.40% by Simard et al. (2003), both obtained by more complex methods. The 35 misclassified digits are shown in Figure 1. Many of them are ambiguous or uncharacteristic, with obviously missing parts or strange strokes. Interestingly, the second guess of the network is correct for 30 out of the 35 misclassified digits.

The best test error of this MLP is even lower (0.32%) and may be viewed as the maximum capacity of the network. Performance clearly profits from adding hidden layers and more units per layer. For example, network 5 has more but smaller hidden layers than network 4 (see Table 1).

Networks with 12 million weights can successfully be trained by plain gradient descent to achieve test errors below 1% after 20 to 30 epochs in less than 2 hours of training. How can networks with so many parameters generalize well on the unseen test set? The answer is that the continual

deformations of the training set generate a virtually infinite supply of training examples, and the network rarely sees any training image twice.

## 6 Conclusion

---

In recent decades the amount of raw computing power per euro has grown by a factor of 100 to 1000 per decade. Our results show that this ongoing hardware progress may be more important than advances in algorithms and software (although the future will belong to methods combining the best of both worlds). Current graphics cards (GPUs) are already more than 40 times faster than standard microprocessors when it comes to training big and deep neural networks by the ancient algorithm, online backpropagation (weight update rate up to  $5 \times 10^9$ /s and more than  $10^{15}$  per trained network). On the competitive MNIST handwriting benchmark, single-precision floating-point GPU-based neural nets surpass all previously reported results, including those obtained by much more complex methods involving specialized architectures, unsupervised pretraining, and combinations of machine learning classifiers, for example. Training sets of sufficient size to avoid overfitting are obtained by appropriately deforming images. Of course, the approach is not limited to handwriting and obviously holds great promise for many visual and other pattern recognition problems.

## Appendix: GPU Implementation

---

**A.1 Graphics Processing Unit.** Until 2007 the only way to program a GPU was to translate the problem-solving algorithm into a set of graphical operations. Despite being hard to code and difficult to debug, several GPU-based neural network implementations were developed when GPUs became faster than CPUs. Two-layer MLPs (Steinkraus, Buck, & Simard, 2005) and CNNs (Chellapilla, Puri, & Simard, 2006) have been previously implemented on GPUs. Although speed-ups were relatively modest, these studies showed how GPUs can be used for machine learning. More recent GPU-based CNNs trained in batch mode are two orders of magnitude faster than CPU-based CNNs (Scherer & Behnke, 2009).

The GPU code is written using CUDA (compute unified device architecture), a C-like general programming language. GPU speed and memory bandwidth are vastly superior to those of CPUs and crucial for fast MLP implementations. To fully understand our algorithm in terms of GPU and CUDA, visit the NVIDIA Web site (NVIDIA, 2009). According to CUDA terminology, the CPU is called host and the graphics card device or GPU.

**A.2 Deformations.** It takes 93 CPU seconds to deform the 60,000 MNIST training images, most of them (87) for elastic distortions. Only the most time-consuming part of the latter—convolution with a gaussian kernel (Simard et al., 2003)—is ported to the GPU. The MNIST training set is split into 600

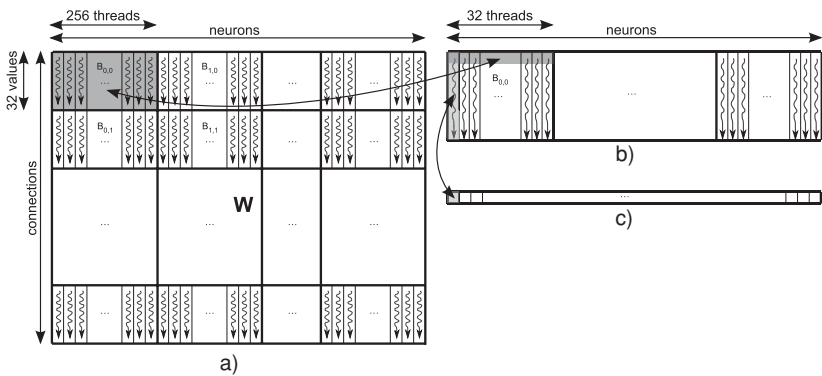


Figure 2: Forward propagation. (a) Mapping of kernel 1 grid onto the padded weight matrix. (b) Mapping the kernel 2 grid onto the partial dot products matrix. (c) Output of forward propagation.

sequentially processed batches. MNIST digits are scaled from the original  $28 \times 28$  pixels to  $29 \times 29$  pixels to get a proper center, which simplifies convolution. An image grid has  $290 \times 290$  cells, zero-padded to  $300 \times 300$ , thus avoiding margin effects when applying a gaussian convolution kernel of size  $21 \times 21$ . Our GPU program groups many threads into a block, where they share the same gaussian kernel and parts of the random field. The blocks contain  $21$  (the kernel size)  $\times 10$  threads, each computing a vertical strip of the convolution operation (see algorithm 1).

Generating the elastic displacement field takes only 3 seconds. Deforming the whole training set is more than 10 times faster, taking 9 instead of the original 93 seconds. Further optimization would be possible by porting all deformations onto the GPU and using the hardware’s interpolation capabilities to perform the final bilinear interpolation. We omitted this since deformations are already fast (deforming all images of one epoch takes only 5% to 15% of total computation time, depending on MLP size).

**A.3 Training Algorithm.** We closely follow the standard BP algorithm (Russell & Norvig, 2002), except that BP of deltas and weight updates are disentangled and performed sequentially. This allows for more parallelism within each routine.

*A.3.1 Forward Propagation.* The algorithm is divided into two kernels. The weight matrix  $W$  is partitioned as illustrated in Figure 2.

Each block of kernel 1 has 256 threads (see Figure 2a), each computing a partial dot product of 32 component vectors. The dot products are stored in a temporary matrix (see Figure 2b). This kernel has a very high throughput: average memory bandwidth is 115 GB/s. This is possible because many

Algorithm 1: Convolution Kernel for Elastic Distortion.

```

__global__ void ConvolveField(float *randomfield, int width, int height, float *kernel,
    float *outputfield, float elasticScale){
    float sum=0;
    const int stride_k=GET_STRIDE(GAUSSIAN_FIELD_SIZE,pitch_x>>2); //
        stride for gaussian kernel
    __shared__ float K[GAUSSIAN_FIELD_SIZE][stride_k]; //kernel (21 x 32
        values)
    __shared__ float R[GAUSSIAN_FIELD_SIZE+9][GAUSSIAN_FIELD_SIZE];
        //random field (30 x 21 values)
    __shared__ float s[10][GAUSSIAN_FIELD_SIZE]; //partial sums (10 x 21
        values)
    int stride_in=GET_STRIDE(width,pitch_x>>2); //random field stride as a
        multiple of 32
    int stride_out=GET_STRIDE(width-GAUSSIAN_FIELD_SIZE+1,pitch_x
        >>2); //output stride as a multiple of 32

    //loading gaussian kernel into K (21 x 21 values)
    K[0+threadIdx.y][threadIdx.x] = kernel[(0+threadIdx.y)*stride_k +
        threadIdx.x]; //rows 0..9
    K[10+threadIdx.y][threadIdx.x] = kernel[(10+threadIdx.y)*stride_k +
        threadIdx.x]; //rows 10..19
    if(threadIdx.y==0)
        K[20+threadIdx.y][threadIdx.x] = kernel[(20+threadIdx.y)*stride_k
            + threadIdx.x]; //row 20

    //loading randomfield into R
    //0..9 x 21 values
    R[0+threadIdx.y][threadIdx.x] = randomfield[(10*blockIdx.y+0+threadIdx.
        y)*stride_in + blockIdx.x + threadIdx.x];
    //10..19 x 21 values
    R[10+threadIdx.y][threadIdx.x] = randomfield[(10*blockIdx.y+10+threadIdx.
        y)*stride_in + blockIdx.x + threadIdx.x];
    //20..29 x 21 values
    R[20+threadIdx.y][threadIdx.x] = randomfield[(10*blockIdx.y+20+threadIdx.
        y)*stride_in + blockIdx.x + threadIdx.x];
    __syncthreads(); //wait until everything is read into shared memory

    //computing partial sums
    #pragma unroll 21 //GAUSSIAN_FIELD_SIZE
    for(int i=0;i<GAUSSIAN_FIELD_SIZE;i++){
        sum += R[threadIdx.y + i][threadIdx.x] * K[i][threadIdx.x];
    }
    s[threadIdx.y][threadIdx.x]=sum;
    __syncthreads();

    if(threadIdx.x==0){ //the first column of threads compute the final
        values of the convolutions
        #pragma unroll 20//GAUSSIAN_FIELD_SIZE-1
        for(int i=1;i<GAUSSIAN_FIELD_SIZE;i++){
            sum+=s[
                threadIdx.y][i];
            outputfield[(blockIdx.y*10+threadIdx.y)*stride_out + blockIdx
                .x] = sum * elasticScale;
        }
    }
}

```



relatively small blocks keep the GPU busy. Each block uses shared memory for storing the previous layer activations, which are simultaneously read by the first 32 threads of each block and then used by all 256 threads. After thread synchronization, the partial dot products are computed in parallel (see algorithm 2). The number of instructions is kept to a minimum by precomputing all common index parts.

The thread grid of kernel 2 (see Figure 2b) has only one row of blocks consisting of *warp* threads, since each thread has to compute a complete dot product (see Figure 2c) and then pipe it into the activation function. This kernel (see algorithm 2) is inefficient for layers with fewer than 1024 incoming connections per neuron, especially for the last layer, which has only 10 neurons—one for each digit. That is, its grid will have only one block, occupying only 3% of the GTX280 GPU.

*A.3.2 Backward Propagation.* This is similar to FP, but we need  $W^T$  for coalesced access. Instead of transposing the matrix, the computations are performed on patches of data read from device memory into shared memory, similar to the optimized matrix transposition algorithm of Ruetsch and Micikevicius (2009). Shared memory access is much faster without coalescing restrictions. Because we have to cope with layers of thousands of neurons, backpropagating deltas uses a reduction method implemented in two kernels communicating partial results via global memory (see algorithm 3).

The bidimensional grid of kernel 1 is divided into blocks of *warp* (32) threads. The kernel starts by reading a patch of  $32 \times 32$  values from  $W$ . The stride of the shared memory block is 33 (*warp* + 1), thus avoiding all bank conflicts and significantly improving speed. Next, 32 input delta values are read, and all memory locations that do not correspond to real neurons (because of vertical striding) are zero-padded to avoid branching in subsequent computations. The number of elements is fixed to *warp* size, and the computing loop is unrolled for further speed-ups. Before finishing, each thread writes its own partial dot product to global memory.

Kernel 2 completes BP of deltas by summing up partial deltas computed by the previous kernel. It multiplies the final result by the derivative of the activation function applied to the current neuron's state and writes the new delta to global memory.

*A.3.3 Weight Updating.* Algorithm 4 starts by reading the appropriate delta and precomputes all repetitive expressions. Then the first 16 threads read the states from global memory into shared memory. The “bias neuron” with constant activation 1.0 is dealt with by conditional statements, which could be avoided through expressions containing the conditions. Once threads are synchronized, each single thread updates 16 weights in a fixed unrolled loop.

Algorithm 2: Forward Propagation Kernels.

```

__global__ void MLP_FP_reduction_Kernel1(float *prevLN, float *W, float *
partialsum, unsigned int neurons, unsigned int prevneurons){
    const int threads=256;
    const int stride=GET_STRIDE(neurons,pitch_x>>2); //horizontal stride of
W matrix
    int X=blockIdx.x*threads + threadIdx.x; //precomputing expressions
    int Y=X+stride*blockIdx.y;
    int Z=blockIdx.y*pitch_y*stride + X;
    float sum=0.0f;
    __shared__ float output[pitch_y];
    if(blockIdx.y==0)
        if(threadIdx.x==0) output[0]=1.0f;
        else if(threadIdx.x<pitch_y) //there are only 32 values to read and
128 threads
            output[threadIdx.x] = threadIdx.x-1<prevneurons ?
                prevLN[threadIdx.x-1] : 0.0f;
        else;
    else if(threadIdx.x<pitch_y) //there are only 32 values to read and 128
threads
        output[threadIdx.x] = blockIdx.y*pitch_y+threadIdx.x-1<
            prevneurons ?
                prevLN[blockIdx.
                    y*pitch_y+
                    threadIdx.x
                    -1] : 0.0f;
    else;
    __syncthreads();
    if(X<neurons){//compute partial sums
        ##pragma unroll 32
        int size=0;
        if((blockIdx.y+1)*pitch_y>=prevneurons+1)
            size = prevneurons + 1 - blockIdx.y*pitch_y;
        else size=pitch_y;
        for (int ic=0; ic<size; ic++){
            sum += output[ic] * W[Z];
            Z+=stride;
        }
        partialsum[Y]=sum;
    }
}

__global__ void MLP_FP_reduction_Kernel2(float *currLN, float *partialsum,
unsigned int neurons, unsigned int size){
    float sum=0.0f;
    int idx = blockIdx.x*(pitch_x>>2) + threadIdx.x; //precomputed index
    unsigned int stride = GET_STRIDE(neurons,pitch_x>>2); //stride for
partialsum matrix

    if(idx>=neurons)return; //is this thread computing a true neuron?
    for (int i=0; i<size; i++) sum += partialsum[i*stride+idx]; //computing
the final dot product
    currLN[idx] = SIGMOIDF(sum); //applying activation
}

```

## Algorithm 3: Backpropagating Delta Kernels.

```

__global__ void backPropagateDeltasFC_A(float *indelta, float *weights, unsigned int
ncon, unsigned int nrneur, float *partial){
    const int px = pitch_x >> 2;
    unsigned int stride_x = GET_STRIDE(nrneur, px);
    unsigned int stride_y = GET_STRIDE(ncon, pitch_y);
    float outd = 0.0;
    int idx = blockIdx.x * px + threadIdx.x;
    int X = blockIdx.y * pitch_y * stride_x + idx;
    int Y = threadIdx.x;
    __shared__ float w[32*33]; //pitch_y and px should be equal ! +1 to avoid
        bank conflict!
    __shared__ float id[px]; //input delta
    #pragma unroll 32 //read the weight patch in shared memory
    for(int i=0; i<pitch_y; i++){ w[Y]=weights[X]; X+=stride_x; Y+=33;}
    //read the input delta patch in shared memory
    if(idx >= nrneur) id[threadIdx.x] = 0; //a fake input delta for inexistent indelta
    else id[threadIdx.x] = indelta[idx];
    __syncthreads(); //not needed for block with warp number of threads: implicit
        synchronization

    #pragma unroll 32 //compute partial results
    for(int i=0; i<px; i++) outd += w[threadIdx.x*33+i] * id[i];
    //write out the partial results
    partial[blockIdx.x * stride_y + blockIdx.y * pitch_y + threadIdx.x] = outd;
}

__global__ void backPropagateDeltasFC_B(float *outdelta, float *instates, unsigned int
ncon, unsigned int nrneur, float *partial){
    int px = pitch_x >> 2;
    unsigned int stride_x = GET_STRIDE(nrneur, px);
    unsigned int stride_y = GET_STRIDE(ncon, pitch_y);
    float outd = 0.0;
    int size = stride_x / px;
    int idx = blockIdx.x * pitch_y + threadIdx.x;
    if(idx == 0); //true only for block and thread 0
    else{
        for(int i=0; i<size; i++)
            outd += partial[i * stride_y + idx];
        outdelta[idx-1] = outd * DSIGMOIDF(instates[idx-1]); // -1
        BIAS ...
    }
}

```

## Algorithm 4: Weights Adjustment Kernel.

```

__global__ void adjustWeightsFC(float *states, float *deltas, float *weights, float eta,
    unsigned int ncon, unsigned int nrneur){
    const int pitch_y=16;
    const int threads=256;
    unsigned int px = pitch_x >> 2;
    unsigned int stride_x = GET_STRIDE(nrneur,px);
    float etadeletak = eta*deltas[blockIdx.x*threads+threadIdx.x],t;
    int b=blockIdx.y*stride_x*pitch_y + threads*blockIdx.x + threadIdx.x;
    __shared__ float st[pitch_y]; //for states
    int cond1 = blockIdx.y || threadIdx.x;
    int cond2 = (blockIdx.y+1)*pitch_y<=ncon;
    int size = cond2 * pitch_y + !cond2 * (ncon%pitch_y);
    if(threadIdx.x<pitch_y) st[threadIdx.x] = cond1 * states[blockIdx.y*
        pitch_y + threadIdx.x - 1] + !cond1;
    __syncthreads();

    if (blockIdx.x*threads + threadIdx.x < nrneur){
        #pragma unroll 16
        for (int j=0; j<16; j++){
            t=weights[b];
            t+= etadeletak * st[j];
            weights[b]=t;
            b+=stride_x;}}
    }

```

## Acknowledgments

Part of this work got started when D.C. was a Ph.D. student at University Politehnica of Timișoara. He thanks his Ph.D. advisor, Ștefan Holban, for his guidance and Răzvan Moșincat for providing a CPU framework for MNIST. This work was supported by Swiss CTI, Commission for Technology and Innovation, Project n. 9688.1 IFF: Intelligent Fill in Form, and by Lifeware S.A.

## References

- Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems*, 19 (pp. 153–160). Cambridge, MA: MIT Press.
- Chellapilla, K., Puri, S., & Simard, P. (2006). High performance convolutional neural networks for document processing. In *Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition*. N.p.
- Decoste, D., & Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, 46, 161–190.
- Hinton, G. (2007). To recognize shapes, first learn to generate images. In P. Cisek, T. Drew, & J. Kalaska (Eds.), *Computational neuroscience: Theoretical insights into brain function*. Burlington, MA: Elsevier.

- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313, 504–507.
- Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen*. Diploma thesis, Technische Universität München.
- Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In S. C. Kramer & J. F. Kolen (Eds.), *A field guide to dynamical recurrent neural networks*. Piscataway, NJ: IEEE Press.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term Memory. *Neural Computation*, 9, 1735–1780.
- Lauer, F., Suen, C., & Bloch, G. (2007). A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40, 1816–1824.
- LeCun, Y. (1985). Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva, Paris*, 85, 599–604.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 309–318.
- Mohamed, A., Dahl, G., & Hinton, G. E. (2009). Deep belief networks for phone recognition. In *Proc. of NIPS 2009 Workshop on Deep Learning for Speech Recognition and Related Applications*. N.p.
- Nair, V., & Hinton, G. E. (2009). Implicit mixtures of restricted Boltzmann machines. In D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), *Advances in neural information processing systems*, 21. Cambridge, MA: MIT Press.
- NVIDIA. (2009). *NVIDIA CUDA: Reference manual* (Version 2.3).
- Ranzato, M., Huang, F., Boureau, Y., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. San Mateo, CA: IEEE Computer Society Press.
- Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2006). Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems*, 19. Cambridge, MA: MIT Press.
- Ruetsch, G., & Micikevicius, P. (2009). *Optimizing matrix transpose in CUDA* (Tech. Rep.). Santa Clara, CA: NVIDIA.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Parallel distributed processing*. Cambridge, MA: MIT Press.
- Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
- Salakhutdinov, R., & Hinton, G. (2007). Learning a nonlinear embedding by preserving class neighborhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*. San Francisco: Morgan Kaufmann.
- Scherer, D., & Behnke, S. (2009). Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors. In *Proc. of NIPS 2009 Workshop on Large-Scale Machine Learning: Parallelism and Massive Datasets*. N.p.
- Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Intl. Conf. Document Analysis and Recognition* (pp. 958–962). San Mateo, CA: IEEE Computer Society Press.

- Steinkraus, D., Buck, I., & Simard, P. Y. (2005). GPUs for machine learning algorithms. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition* (pp. 1115–1120). San Mateo, CA: IEEE Computer Society Press.
- Werbos, P. J. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. Unpublished doctoral dissertation, Harvard University.

---

Received February 19, 2010; accepted May 25, 2010.

This article has been cited by:

1. Alexios Koutsoukas, Keith J. Monaghan, Xiaoli Li, Jun Huan. 2017. Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data. *Journal of Cheminformatics* 9:1. . [[Crossref](#)]
2. Jianshu Zhang, Jun Du, Shiliang Zhang, Dan Liu, Yulong Hu, Jinshui Hu, Si Wei, Lirong Dai. 2017. Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. *Pattern Recognition* 71, 196-206. [[Crossref](#)]
3. Huafeng Qin, Mounim A. El-Yacoubi. 2017. Deep Representation-Based Feature Extraction and Recovering for Finger-Vein Verification. *IEEE Transactions on Information Forensics and Security* 12:8, 1816-1829. [[Crossref](#)]
4. Waseem Rawat, Zenghui Wang. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*, ahead of print1-98. [[Abstract](#)] [[PDF](#)] [[PDF Plus](#)]
5. Fangyu Ding, Quansheng Ge, Dong Jiang, Jingying Fu, Mengmeng Hao. 2017. Understanding the dynamics of terrorism events with multiple-discipline datasets and machine learning approach. *PLOS ONE* 12:6, e0179057. [[Crossref](#)]
6. Ju-Chin Chen, Chao-Feng Liu. 2017. Deep net architectures for visual-based clothing image recognition on large database. *Soft Computing* 21:11, 2923-2939. [[Crossref](#)]
7. Hao Liu, Jiwen Lu, Jianjiang Feng, Jie Zhou. 2017. Group-aware deep feature learning for facial age estimation. *Pattern Recognition* 66, 82-94. [[Crossref](#)]
8. Claudio Gallicchio, Alessio Micheli. 2017. Echo State Property of Deep Reservoir Computing Networks. *Cognitive Computation* 9:3, 337-350. [[Crossref](#)]
9. Haixia Sun, Sikun Li. 2017. An optimization method for speech enhancement based on deep neural network. *IOP Conference Series: Earth and Environmental Science* 69, 012139. [[Crossref](#)]
10. Jozsef Suto, Stefan Oniga. 2017. Efficiency investigation of artificial neural networks in human activity recognition. *Journal of Ambient Intelligence and Humanized Computing* 37. . [[Crossref](#)]
11. Xueheng Qiu, Ye Ren, Ponnuthurai Nagarathnam Suganthan, Gehan A.J. Amaratunga. 2017. Empirical Mode Decomposition based ensemble deep learning for load demand time series forecasting. *Applied Soft Computing* 54, 246-255. [[Crossref](#)]
12. M. W. Spratling. 2017. A Hierarchical Predictive Coding Model of Object Recognition in Natural Images. *Cognitive Computation* 9:2, 151-167. [[Crossref](#)]
13. Hao Liu, Jiwen Lu, Jianjiang Feng, Jie Zhou. 2017. Learning Deep Sharable and Structural Detectors for Face Alignment. *IEEE Transactions on Image Processing* 26:4, 1666-1678. [[Crossref](#)]

14. Saikat Roy, Nibaran Das, Mahantapas Kundu, Mita Nasipuri. 2017. Handwritten isolated Bangla compound character recognition: A new benchmark using a novel deep learning approach. *Pattern Recognition Letters* **90**, 15-21. [[Crossref](#)]
15. Jeffrey S. Bowers. 2017. Grandmother cells and localist representations: a review of current thinking. *Language, Cognition and Neuroscience* **32**:3, 257-273. [[Crossref](#)]
16. Arash Samadi, Timothy P. Lillicrap, Douglas B. Tweed. 2017. Deep Learning with Dynamic Spiking Neurons and Fixed Feedback Weights. *Neural Computation* **29**:3, 578-602. [[Abstract](#)] [[Full Text](#)] [[PDF](#)] [[PDF Plus](#)]
17. Sandip Kumar Singh, Sandeep Kumar, J. P. Dwivedi. 2017. Compound fault prediction of rolling bearing using multimedia data. *Multimedia Tools and Applications* . [[Crossref](#)]
18. Osama Abdeljaber, Onur Avci, Serkan Kiranyaz, Moncef Gabbouj, Daniel J. Inman. 2017. Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks. *Journal of Sound and Vibration* **388**, 154-170. [[Crossref](#)]
19. Hubert Cecotti. 2017. Hierarchical k -Nearest Neighbor with GPUs and a High Performance Cluster: Application to Handwritten Character Recognition. *International Journal of Pattern Recognition and Artificial Intelligence* **31**:02, 1750005. [[Crossref](#)]
20. David Eriksson. 2017. A Principle for Describing and Verifying Brain Mechanisms Using Ongoing Activity. *Frontiers in Neural Circuits* **11**. . [[Crossref](#)]
21. Guy Van der Sande, Daniel Brunner, Miguel C. Soriano. 2017. Advances in photonic reservoir computing. *Nanophotonics* **6**:3. . [[Crossref](#)]
22. Glaucia C. Pereira. Genomics and Artificial Intelligence Working Together in Drug Discovery and Repositioning: The Advent of Adaptive Pharmacogenomics in Glioblastoma and Chronic Arterial Inflammation Therapies 253-281. [[Crossref](#)]
23. Jürgen Schmidhuber. Deep Learning 338-348. [[Crossref](#)]
24. Alan Mosca, George D. Magoulas. Learning Input Features Representations in Deep Learning 433-445. [[Crossref](#)]
25. Harrison B. Prosper. 2017. Deep Learning and Bayesian Methods. *EPJ Web of Conferences* **137**, 11007. [[Crossref](#)]
26. . References 573-600. [[Crossref](#)]
27. Mumtazimah Mohamad, Mokhairi Makhtar, Mohd Nordin Abd Rahman. The Reconstructed Heterogeneity to Enhance Ensemble Neural Network for Large Data 447-455. [[Crossref](#)]
28. Michael Otte. Collective Cognition and Sensing in Robotic Swarms via an Emergent Group-Mind 829-840. [[Crossref](#)]
29. Md Shopon, Nabeel Mohammed, Md Anowarul Abedin. Image augmentation by blocky artifact in Deep Convolutional Neural Network for handwritten digit recognition 1-6. [[Crossref](#)]



30. Michal Wróbel, Janusz T. Starczewski, Christian Napoli. Handwriting Recognition with Extraction of Letter Fragments 183-192. [[Crossref](#)]
31. Fangzheng Xue, Xuyang Chen, Xiumin Li. Real-Time Classification Through a Spiking Deep Belief Network with Intrinsic Plasticity 188-196. [[Crossref](#)]
32. ???. 2017. Single-Channel Speech Enhancement Based on Sparse Regressive Deep Neural Network. *Software Engineering and Applications* **06**:01, 8-19. [[Crossref](#)]
33. Jia Liu, Maoguo Gong, Jiaojiao Zhao, Hao Li, Licheng Jiao. 2016. Difference representation learning using stacked restricted Boltzmann machines for change detection in SAR images. *Soft Computing* **20**:12, 4645-4657. [[Crossref](#)]
34. Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, Shuo Feng. 2016. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing* **2016**:1. . [[Crossref](#)]
35. Dickson Neoh Tze How, Chu Kiong Loo, Khairul Salleh Mohamed Sahari. 2016. Behavior recognition for humanoid robots using long short-term memory. *International Journal of Advanced Robotic Systems* **13**:6, 172988141666336. [[Crossref](#)]
36. Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, Colin J. Akerman. 2016. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications* **7**, 13276. [[Crossref](#)]
37. Sebastien C. Wong, Adam Gatt, Victor Stamatescu, Mark D. McDonnell. Understanding Data Augmentation for Classification: When to Warp? 1-6. [[Crossref](#)]
38. Igor Sevo, Aleksandar Kelecevic. Clustered class-dependant training method for digit recognition classifiers 1-4. [[Crossref](#)]
39. Turker Ince, Serkan Kiranyaz, Levent Eren, Murat Askar, Moncef Gabbouj. 2016. Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks. *IEEE Transactions on Industrial Electronics* **63**:11, 7067-7075. [[Crossref](#)]
40. Peter U. Diehl, Guido Zarrella, Andrew Cassidy, Bruno U. Pedroni, Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware 1-8. [[Crossref](#)]
41. Oliver Dürr, Beate Sick. 2016. Single-Cell Phenotype Classification Using Deep Convolutional Neural Networks. *Journal of Biomolecular Screening* **21**:9, 998-1003. [[Crossref](#)]
42. Kaizhou Li, Hong Shi, Qinghua Hu. Complex convolution Kernel for deep networks 1-5. [[Crossref](#)]
43. Shenghao Tang, Tong Li, Xinghao Cheng. A novel remote sensing image change detection algorithm based on self-organizing feature map neural network model 1-7. [[Crossref](#)]
44. Lin Zhou, Yan Yan, Xingbin Qin, Chan Yuan, Dashun Que, Lei Wang. Deep learning-based classification of massive electrocardiography data 780-785. [[Crossref](#)]

45. Zhiqian Qi, Bo Wang, Yingjie Tian, Peng Zhang. 2016. When Ensemble Learning Meets Deep Learning: a New Deep Support Vector Machine for Classification. *Knowledge-Based Systems* **107**, 54-60. [[Crossref](#)]
46. Alexey Potapov, Vita Potapova, Maxim Peterson. 2016. A feasibility study of an autoencoder meta-model for improving generalization capabilities on training sets of small sizes. *Pattern Recognition Letters* **80**, 24-29. [[Crossref](#)]
47. Serkan Kiranyaz, Muhammad-Adeel Waris, Iftikhar Ahmad, Ridha Hamila, Moncef Gabbouj. Face segmentation in thumbnail images by data-adaptive convolutional segmentation networks 2306-2310. [[Crossref](#)]
48. Tomislav Hrkac, Karla Brkic, Slobodan Ribaric, Darijan Marcetic. Deep learning architectures for tattoo detection and de-identification 1-5. [[Crossref](#)]
49. Hubert Cecotti. Deep Random Vector Functional Link Network for handwritten character recognition 3628-3633. [[Crossref](#)]
50. Payam Farzan, Mahdi Izadi, Chandima Gomes, MohammadHesam Hesamian. 2016. Malaysia solar energy experience: Intelligent fault location algorithm for unbalanced radial distribution network including PV systems. *Journal of Renewable and Sustainable Energy* **8**:4, 045503. [[Crossref](#)]
51. Ziyi Li, Liyao Ma, Xiaolu Ke, Yong Wang. Handwritten digit recognition via active belief decision trees 7021-7025. [[Crossref](#)]
52. Wei Gao, Zhi-Hua Zhou. 2016. Dropout Rademacher complexity of deep neural networks. *Science China Information Sciences* **59**:7. . [[Crossref](#)]
53. Erol Gelenbe, Yongha Yin. Deep learning with random neural networks 1633-1638. [[Crossref](#)]
54. Anna Rakitianskaia, Eduan Bekker, Katherine M. Malan, Andries Engelbrecht. Analysis of error landscapes in multi-layered neural networks for classification 5270-5277. [[Crossref](#)]
55. Bing Han, Abhronil Sengupta, Kaushik Roy. On the energy benefits of spiking deep neural networks: A case study 971-976. [[Crossref](#)]
56. Liangpei Zhang, Lefei Zhang, Bo Du. 2016. Deep Learning for Remote Sensing Data: A Technical Tutorial on the State of the Art. *IEEE Geoscience and Remote Sensing Magazine* **4**:2, 22-40. [[Crossref](#)]
57. Bo-Kyeong Kim, Jihyeon Roh, Suh-Yeon Dong, Soo-Young Lee. 2016. Hierarchical committee of deep convolutional neural networks for robust facial expression recognition. *Journal on Multimodal User Interfaces* **10**:2, 173-189. [[Crossref](#)]
58. Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, Jose Miguel Hernandez-Lobato, Gu-Yeon Wei, David Brooks. Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators 267-278. [[Crossref](#)]

59. Hing Kai Chan, Tsan-Ming Choi, Xiaohang Yue. 2016. Guest Editorial Big Data Analytics: Risk and Operations Management for Industrial Applications. *IEEE Transactions on Industrial Informatics* 12:3, 1214-1218. [[Crossref](#)]
60. Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, Qi Tian. DisturbLabel: Regularizing CNN on the Loss Layer 4753-4762. [[Crossref](#)]
61. Valeriu Codreanu, Bob Dröge, David Williams, Burhan Yasar, Po Yang, Baoquan Liu, Feng Dong, Olarik Surinta, Lambert R.B. Schomaker, Jos B.T.M. Roerdink, Marco A. Wiering. 2016. Evaluating automatically parallelized versions of the support vector machine. *Concurrency and Computation: Practice and Experience* 28:7, 2274-2294. [[Crossref](#)]
62. Banafsheh Rekabdar, Monica Nicolescu, Mircea Nicolescu, Mohammad Taghi Saffar, Richard Kelley. 2016. A Scale and Translation Invariant Approach for Early Classification of Spatio-Temporal Patterns Using Spiking Neural Networks. *Neural Processing Letters* 43:2, 327-343. [[Crossref](#)]
63. Serkan Kiranyaz, Turker Ince, Moncef Gabbouj. 2016. Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks. *IEEE Transactions on Biomedical Engineering* 63:3, 664-675. [[Crossref](#)]
64. Renliang Weng, Jiwen Lu, Yap-Peng Tan, Jie Zhou. 2016. Learning Cascaded Deep Auto-encoder Networks for Face Alignment. *IEEE Transactions on Multimedia* 1-1. [[Crossref](#)]
65. Francisco Ortega-Zamorano, José M. Jerez, Iván Gómez, Leonardo Franco. Deep Neural Network Architecture Implementation on FPGAs Using a Layer Multiplexing Scheme 79-86. [[Crossref](#)]
66. Jürgen Schmidhuber. Deep Learning 1-11. [[Crossref](#)]
67. Maoguo Gong, Jiaojiao Zhao, Jia Liu, Qiguang Miao, Licheng Jiao. 2016. Change Detection in Synthetic Aperture Radar Images Based on Deep Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 27:1, 125-138. [[Crossref](#)]
68. Amy de Buitléir, Ronan Flynn, Michael Russell, Mark Daly. An Architecture for Pattern Recognition and Decision-Making 22-33. [[Crossref](#)]
69. Chuang Ding, Lei Xie, Pengcheng Zhu. 2015. Head motion synthesis from speech using deep neural networks. *Multimedia Tools and Applications* 74:22, 9871-9888. [[Crossref](#)]
70. Bo Wu, B. M. Wilamowski. An algorithm for visualization of big data in a two-dimensional space 000053-000058. [[Crossref](#)]
71. Li Li, Wang Yaonan, Zheng Yexin. A handwritten digit recognizer using ensemble method 469-473. [[Crossref](#)]
72. Takayuki Osogami, Makoto Otsuka. 2015. Seven neurons memorizing sequences of alphabetical images via spike-timing dependent plasticity. *Scientific Reports* 5:1. . [[Crossref](#)]

73. Anderson Tenorio Sergio, Teresa B. Ludermitr. Deep Learning for Wind Speed Forecasting in Northeastern Region of Brazil 322-327. [[Crossref](#)]
74. Carlo Cecati, Janusz Kolbusz, Pawel Rozycki, Pierluigi Siano, Bogdan M. Wilamowski. 2015. A Novel RBF Training Algorithm for Short-Term Electric Load Forecasting and Comparative Studies. *IEEE Transactions on Industrial Electronics* **62**:10, 6519-6529. [[Crossref](#)]
75. Daniel Soudry, Dotan Di Castro, Asaf Gal, Avinoam Kolodny, Shahar Kvatinsky. 2015. Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training. *IEEE Transactions on Neural Networks and Learning Systems* **26**:10, 2408-2421. [[Crossref](#)]
76. J. Lerouge, R. Herault, C. Chatelain, F. Jardin, R. Modzelewski. 2015. IODA: An input/output deep architecture for image labeling. *Pattern Recognition* **48**:9, 2847-2858. [[Crossref](#)]
77. Meijun Sun, Dong Zhang, Jinchang Ren, Zheng Wang, Jesse S. Jin. Brushstroke based sparse hybrid convolutional neural networks for author classification of Chinese ink-wash paintings 626-630. [[Crossref](#)]
78. Mark D. McDonnell, Migel D. Tissera, Tony Vladusich, André van Schaik, Jonathan Tapson. 2015. Fast, Simple and Accurate Handwritten Digit Classification by Training Shallow Neural Network Classifiers with the 'Extreme Learning Machine' Algorithm. *PLOS ONE* **10**:8, e0134254. [[Crossref](#)]
79. Serkan Kiranyaz, Turker Ince, Ridha Hamila, Moncef Gabbouj. Convolutional Neural Networks for patient-specific ECG classification 2608-2611. [[Crossref](#)]
80. Giacomo Indiveri, Shih-Chii Liu. 2015. Memory and Information Processing in Neuromorphic Systems. *Proceedings of the IEEE* **103**:8, 1379-1397. [[Crossref](#)]
81. Michiel Hermans, Joni Dambre, Peter Bienstman. 2015. Optoelectronic Systems Trained With Backpropagation Through Time. *IEEE Transactions on Neural Networks and Learning Systems* **26**:7, 1545-1550. [[Crossref](#)]
82. Mark D. McDonnell, Tony Vladusich. Enhanced image classification with a fast-learning shallow convolutional neural network 1-7. [[Crossref](#)]
83. Raqibul Hasan, Chris Yakopcic, Tarek M. Taha. Ex-situ training of dense memristor crossbar for neuromorphic applications 75-81. [[Crossref](#)]
84. Henning Puttnies, Vlado Altmann, Frank Golasowski, Dirk Timmermann. Cost-efficient universal approach for remote meter reading using web services and computer vision 1-6. [[Crossref](#)]
85. Shulan Pan, Yanwei Wang, Changsong Liu, Xiaoqing Ding. A discriminative cascade CNN model for offline handwritten digit recognition 501-504. [[Crossref](#)]
86. Pavlo Molchanov, Shalini Gupta, Kihwan Kim, Kari Pulli. Multi-sensor system for driver's hand-gesture recognition 1-8. [[Crossref](#)]
87. S. Elfving, E. Uchibe, K. Doya. 2015. Expected energy-based restricted Boltzmann machine for classification. *Neural Networks* **64**, 29-38. [[Crossref](#)]

88. David Menotti, Giovanni Chiachia, Allan Pinto, William Robson Schwartz, Helio Pedrini, Alexandre Xavier Falcao, Anderson Rocha. 2015. Deep Representations for Iris, Face, and Fingerprint Spoofing Detection. *IEEE Transactions on Information Forensics and Security* 10:4, 864-879. [[Crossref](#)]
89. Kayode Sanni, Guillaume Garreau, Jamal Lottier Molin, Andreas G. Andreou. FPGA implementation of a Deep Belief Network architecture for character recognition using stochastic computation 1-5. [[Crossref](#)]
90. Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61, 85-117. [[Crossref](#)]
91. Harri Valpola. From neural PCA to deep unsupervised learning 143-171. [[Crossref](#)]
92. Peter Tino, Lubica Benuskova, Alessandro Sperduti. Artificial Neural Network Models 455-471. [[Crossref](#)]
93. Hossein Ghodrati Noushahr, Samad Ahmadi, Arlene Casey. Fast Handwritten Digit Recognition with Multilayer Ensemble Extreme Learning Machine 77-89. [[Crossref](#)]
94. Mostafa Ali Shahin, Beena Ahmed, Kirrie J. Ballard. Classification of lexical stress patterns using deep neural network architecture 478-482. [[Crossref](#)]
95. J.M. Alonso-Weber, M.P. Sesmero, A. Sanchis. 2014. Combining additive input noise annealing and pattern transformations for improved handwritten character recognition. *Expert Systems with Applications* 41:18, 8180-8188. [[Crossref](#)]
96. Daniel Neil, Shih-Chii Liu. 2014. Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22:12, 2621-2628. [[Crossref](#)]
97. Sizhe Chen, Haipeng Wang. SAR target recognition based on deep learning 541-547. [[Crossref](#)]
98. Foteini Fotopoulou, George Economou. Graph Based Re-ranking Method with Application to Handwritten Digits 749-753. [[Crossref](#)]
99. . References 327-351. [[Crossref](#)]
100. Janusz A. Starzyk, Basawaraj. 2014. Memristor Crossbar Architecture for Synchronous Neural Networks. *IEEE Transactions on Circuits and Systems I: Regular Papers* 61:8, 2390-2401. [[Crossref](#)]
101. Raqibul Hasan, Tarek M. Taha. Enabling back propagation training of memristor crossbar neuromorphic processors 21-28. [[Crossref](#)]
102. Jiaojiao Zhao, Maoguo Gong, Jia Liu, Licheng Jiao. Deep learning to classify difference image for image change detection 411-417. [[Crossref](#)]
103. Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, Eugenio Culurciello. A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks 696-701. [[Crossref](#)]
104. Ryusuke Nebashi, Noboru Sakimura, Hiroaki Honjo, Ayuka Morioka, Yukihide Tsuji, Kunihiko Ishihara, Keiichi Tokutome, Sadahiko Miura, Shunsuke Fukami,

- Keizo Kinoshita, Takahiro Hanyu, Tetsuo Endoh, Naoki Kasai, Hideo Ohno, Tadahiko Sugibayashi. A delay circuit with 4-terminal magnetic-random-access-memory device for power-efficient time- domain signal processing 1588-1591. [[Crossref](#)]
105. P. Graff, F. Feroz, M. P. Hobson, A. Lasenby. 2014. SKYNET: an efficient and robust neural network training tool for machine learning in astronomy. *Monthly Notices of the Royal Astronomical Society* **441**:2, 1741-1759. [[Crossref](#)]
106. Madalina Cosmina Popescu, Lucian Mircea Sasu. Feature extraction, feature selection and machine learning for image classification: A case study 968-973. [[Crossref](#)]
107. Steven Skorheim, Peter Lonjers, Maxim Bazhenov. 2014. A Spiking Network Model of Decision Making Employing Rewarded STDP. *PLoS ONE* **9**:3, e90821. [[Crossref](#)]
108. Yong Lin, Eliza Yingzi Du, Zhi Zhou, N. Luke Thomas. 2014. An Efficient Parallel Approach for Sclera Vein Recognition. *IEEE Transactions on Information Forensics and Security* **9**:2, 147-157. [[Crossref](#)]
109. Yi Ding, Qi Feng, Tianjiang Wang, Xian Fu. 2014. A modular neural network architecture with concept. *Neurocomputing* **125**, 3-6. [[Crossref](#)]
110. Xue-Wen Chen, Xiaotong Lin. 2014. Big Data Deep Learning: Challenges and Perspectives. *IEEE Access* **2**, 514-525. [[Crossref](#)]
111. Louis K. Scheffer. Lessons from the neurons themselves 197-200. [[Crossref](#)]
112. Raqibul Hasan, Tarek M. Taha. On-chip static vs. dynamic routing for feed forward neural networks on multicore neuromorphic architectures 329-334. [[Crossref](#)]
113. Ying Zhang, Saizheng Zhang. Optimized Deep Learning Architectures with Fast Matrix Operation Kernels on Parallel Platform 71-78. [[Crossref](#)]
114. Yun Zhu, Yanqing Zhang, Yi Pan. Large-scale restricted boltzmann machines on single GPU 169-174. [[Crossref](#)]
115. Grégoire Montavon, Matthias Rupp, Vivekanand Gobre, Alvaro Vazquez-Mayagoitia, Katja Hansen, Alexandre Tkatchenko, Klaus-Robert Müller, O Anatole von Lilienfeld. 2013. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics* **15**:9, 095003. [[Crossref](#)]
116. Yoichi Nakaguro, Matthew N. Dailey, Sanparith Marukatat, Stanislav S. Makhanov. 2013. Defeating line-noise CAPTCHAs with multiple quadratic snakes. *Computers & Security* **37**, 91-110. [[Crossref](#)]
117. Alessandro Giusti, Dan C. Ciresan, Jonathan Masci, Luca M. Gambardella, Jurgen Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks 4034-4038. [[Crossref](#)]
118. Katja Hansen, Grégoire Montavon, Franziska Biegler, Siamac Fazli, Matthias Rupp, Matthias Scheffler, O. Anatole von Lilienfeld, Alexandre Tkatchenko, Klaus-Robert Müller. 2013. Assessment and Validation of Machine Learning Methods

- for Predicting Molecular Atomization Energies. *Journal of Chemical Theory and Computation* **9**:8, 3404-3419. [[Crossref](#)]
119. Y. Bengio, A. Courville, P. Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**:8, 1798-1828. [[Crossref](#)]
  120. Olarik Surinta, Lambert Schomaker, Marco Wiering. A Comparison of Feature and Pixel-Based Methods for Recognizing Handwritten Bangla Digits 165-169. [[Crossref](#)]
  121. O. Anatole von Lilienfeld. 2013. First principles view on chemical compound space: Gaining rigorous atomistic control of molecular properties. *International Journal of Quantum Chemistry* **113**:12, 1676-1689. [[Crossref](#)]
  122. Raimar Wagner, Markus Thom, Michael Gabb, Matthias Limmer, Roland Schweiger, Albrecht Rothermel. Convolutional Neural Networks for night-time animal orientation estimation 316-321. [[Crossref](#)]
  123. Christian J. Schuler, Harold Christopher Burger, Stefan Harmeling, Bernhard Scholkopf. A Machine Learning Approach for Non-blind Image Deconvolution 1067-1074. [[Crossref](#)]
  124. Li Deng, Xiao Li. 2013. Machine Learning Paradigms for Speech Recognition: An Overview. *IEEE Transactions on Audio, Speech, and Language Processing* **21**:5, 1060-1089. [[Crossref](#)]
  125. Damien Querlioz, Olivier Bichler, Philippe Dollfus, Christian Gamrat. 2013. Immunity to Device Variations in a Spiking Neural Network With Memristive Nanodevices. *IEEE Transactions on Nanotechnology* **12**:3, 288-295. [[Crossref](#)]
  126. Bernhard Nessler, Michael Pfeiffer, Lars Buesing, Wolfgang Maass. 2013. Bayesian Computation Emerges in Generic Cortical Microcircuits through Spike-Timing-Dependent Plasticity. *PLoS Computational Biology* **9**:4, e1003037. [[Crossref](#)]
  127. Sajjad Pourmohammad, Reza Soosahabi, Anthony S. Maida. An efficient character recognition scheme based on k-means clustering 1-6. [[Crossref](#)]
  128. Salvador Dura-Bernal, Thomas Wennekers, Susan L. Denham. 2012. Top-Down Feedback in an HMAX-Like Cortical Model of Object Perception Based on Hierarchical Bayesian Networks and Belief Propagation. *PLoS ONE* **7**:11, e48216. [[Crossref](#)]
  129. Mumtazimah Mohamad, Md Yazid Mohamad Saman, Muhammad Suzuri Hitam. A framework for multiprocessor neural networks systems 44-48. [[Crossref](#)]
  130. Dan Cireşan, Ueli Meier, Jonathan Masci, Jürgen Schmidhuber. 2012. Multi-column deep neural network for traffic sign classification. *Neural Networks* **32**, 333-338. [[Crossref](#)]
  131. Sascha Lange, Martin Riedmiller, Arne Voigtlander. Autonomous reinforcement learning on raw visual input data in a real world application 1-8. [[Crossref](#)]
  132. Dan C. Cireşan, Ueli Meier, Jürgen Schmidhuber. Transfer learning for Latin and Chinese characters with Deep Neural Networks 1-6. [[Crossref](#)]

133. D. Cireşan, U. Meier, J. Schmidhuber. Multi-column deep neural networks for image classification 3642-3649. [[Crossref](#)]
134. Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, Jürgen Schmidhuber. Deep Big Multilayer Perceptrons for Digit Recognition 581-598. [[Crossref](#)]
135. Renata F. P. Neves, Alberto N. G. Lopes Filho, Carlos A.B. Mello, Cleber Zanchettin. A SVM based off-line handwritten digit recognizer 510-515. [[Crossref](#)]
136. Cheng-Lin Liu, Fei Yin, Qiu-Feng Wang, Da-Han Wang. ICDAR 2011 Chinese Handwriting Recognition Competition 1464-1469. [[Crossref](#)]
137. Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, Jürgen Schmidhuber. Convolutional Neural Network Committees for Handwritten Character Classification 1135-1139. [[Crossref](#)]
138. Ueli Meier, Dan Claudiu Cireşan, Luca Maria Gambardella, Jürgen Schmidhuber. Better Digit Recognition with a Committee of Simple Neural Nets 1250-1254. [[Crossref](#)]
139. Dan Cireşan, Ueli Meier, Jonathan Masci, Jürgen Schmidhuber. A committee of neural networks for traffic sign classification 1918-1921. [[Crossref](#)]
140. Johannes Stallkamp, Marc Schlipsing, Jan Salmen, Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition 1453-1460. [[Crossref](#)]
141. Leo Pape, Faustino Gomez, Mark Ring, Jürgen Schmidhuber. Modular deep belief networks that do not forget 1191-1198. [[Crossref](#)]
142. Markus Thom, Roland Schweiger, Gunther Palm. Supervised Matrix Factorization with sparseness constraints and fast inference 973-979. [[Crossref](#)]