Gradient-Based Learning for Object Detection, Segmentation and Recognition

Yann LeCun Patrick Haffner Léon Bottou
Yoshua Bengio *

AT&T Shannon Lab, 100 Schulz Drive, Red Bank NJ 07701, USA,
yann@research.att.com http://www.research.att.com/~yann

*Yoshua Bengio is also with the Département d'Informatique et de Recherche Opérationelle, Université de Montréal, C.P. 6128 Succ. Centre-Ville, 2920 Chemin de la Tour, Montréal, Québec, Canada H3C 3J7.

Abstract

Finding an appropriate set of features is an essential problem in the design of visual recognition systems. This paper attempts to show that for recognizing simple deformable objects with high shape variability such as handwritten characters, it is possible, and even advantageous, to feed the system directly with minimally processed images and to rely on learning to extract the right set of features. Convolutional Neural Networks are shown to be particularly well suited to this task. We also show that these networks can be used to recognize multiple objects without requiring explicit segmentation of the objects from their surrounding. The second part of the paper presents the Graph Transformer Network model which extends the applicability of gradient-based learning to systems that use graphs to represents features, objects, and their combinations.

1 Gradient-Based Learning in Complex Systems

Until recently, the applicability of machine learning techniques to vision was been limited for three main reasons. First, vision tasks involve multiple steps such as detection, segmentation, feature extraction, and recognition, that require different strategies, not all of which are easily learnable. Second, the recognition of complex objects often involve combinatorial searches that are not easily learned. Third, objects are often represented by structured objects (such as valued graphs) or high-dimensional vectors that most traditional learning algorithms have a hard time dealing with.

Learning algorithms are generally restricted to a particular type of architecture. This often limits their applicability, since different perceptual tasks often require different recognition architectures. When learning techniques must be combined with prior knowledge to construct a recognition system, the prior knowledge is generally used to design an appropriate preprocessor or feature extractor, whose output is fed to a "generic" trainable recognizer. A more satisfying method would be to first construct an architecture that is appropriate to the task from a set of predetermined building blocks, and then to train this entire architecture by simultaneously optimizing the parameters of all the building blocks so as to maximize an overall performance measure.

More precisely, the most commonly accepted model of pattern recognition, is composed of a *segmenter* whose role is to extract objects of interest from their background, a hand-crafted *feature extractor* that gathers relevant information from the input and eliminates irrelevant variabilities, and a *classifier* which categorizes the resulting feature representations (generally vectors or strings of symbols) into categories. There are three major methods for classification: *template matching* ((Yuille et al. 1992)) matches the feature representation to a set of class templates; *generative methods* use a probability density model for each class, and pick the class with the highest likelihood of generating the feature representation; *discriminative models* compute a discriminant function that directly produces a score for each class. Generative ((Moghaddam and Pentland 1995)) and discriminative models ((Sung and Poggio 1994); (Papageorgiou et al. 1998)) are often estimated (learned) from training samples. In all of these approaches, the overall performance of the system is largely determined by the quality of the feature extractor and the segmentation technique.

The feature extractor usually relies on simplifying assumptions about the input data and can rarely take into account all the variability of the real world. The segmentation technique is either exhaustive, exploring a range of possible scale and positions, or hand-crafted, again using simplifying assumptions to restrict the number of candidate segments. Because simplifying assumptions are likely sources of unrecoverable losses of information, it may be preferable to feed the entire system with minimally processed inputs (e.g. "raw" pixel images), and train it from data so as to minimize an overall loss function (which maximizes a given performance measure). Keeping the preprocessing to a minimum ensures that no unrealistic assumption is made about the data. Unfortunately, that also requires to come up with a suitable learning architecture that can handle the high dimension of the input (number of pixels), the high degree of variability due to pose variations or geometric distortions among other things, and the necessarily complex non-linear relation between the input and the output. What is proposed here is a method for learning the features and the classification function simultaneously in a supervised way. Many authors, most recently Zhu and Mumford ((Zhu and Mumford)), have proposed unsupervised methods for learning appropriate features for natural images. These methods have the advantage of producing generic features, but the disadvantage of not being able to tune the features to the particular task at hand.

Gradient-Based Learning provides a framework in which to build such a system. The learning machine computes a function $Y^p = F(Z^p, W)$ where Z^p is the p-th input pattern, and W represents the collection of adjustable parameters in the system. The output Y^p contains scores or probabilities for each category. A loss function $E^p = \mathcal{D}(D^p, F(W, Z^p))$, measures the discrepancy between D^p, the "correct" output for pattern Z^p, and the output produced by the system. The average loss function $E_{train}(W)$ is the average of the errors E^p over a set of labeled examples called the training set $\{(Z^1, D^1), ...(Z^P, D^P)\}$. In the simplest setting, the learning problem consists in finding the value of W that minimizes $E_{train}(W)$.

Making the loss function *differentiable* with respect to W ensures that efficient, gradient-based non-linear optimization methods can be used to find a minimum. To ensure global differentiability, the system is built as a feed-forward network of modules. In the simplest case, each module computes a function $X_n = F_n(W_n, X_{n-1})$, where X_n is an object (a vector in the simplest case) representing the output of the module, W_n is the vector of tunable (trainable) parameters in the module (a subset of W), and X_{n-1} is the module's input (as well as the previous module's output). The input X_0 to the first module is the system's input pattern Z^p.

The main idea of Gradient-Based Learning, which is a simple extension of the well-known back-propagation neural network learning algorithm, is that the objective function can be efficiently minimized through gradient descent (or other more sophisticated non-linear optimization methods) because the gradient of E with respect to W can be efficiently computed with a backward recurrence through the network of modules. If the partial derivative of E^p with respect to X_n is known, then the partial derivatives of E^p with respect to W_n and X_{n-1} can be computed using the following backward recurrence:

$$\begin{aligned} \frac{\partial E^p}{\partial W_n} &= \frac{\partial F_n}{\partial W}(W_n, X_{n-1}) \frac{\partial E^p}{\partial X_n} \\ \frac{\partial E^p}{\partial X_{n-1}} &= \frac{\partial F_n}{\partial X}(W_n, X_{n-1}) \frac{\partial E^p}{\partial X_n} \end{aligned} \tag{1}$$

where $\frac{\partial F_n}{\partial W}(W_n, X_{n-1})$ is the Jacobian of F_n with respect to W evaluated at the point (W_n, X_{n-1}), and $\frac{\partial F_n}{\partial X}(W_n, X_{n-1})$ is the Jacobian of F_n with respect to X. The first equation computes some terms of the gradient of $E^p(W)$, while the second equation propagates the partial gradients backward. The idea can be trivially extended to any network of functional modules. A completely rigorous derivation of the gradient propagation procedure in the general case can be done using Lagrange functions ((LeCun 1988); (Bottou and Gallinari 1991)).

As we show in the following, a large variety of modules have been devised that allow Gradient-Based Learning to be applied to invariant object recognition, multiple object recognition, and recognition of objects that are best represented as graphs.

2 Shape Recognition with Convolutional Neural Networks

Traditional multi-layer neural networks are a special case of the above where the states X_n are fixed-sized vectors, and where the modules are alternated layers of matrix multiplications (the weights)

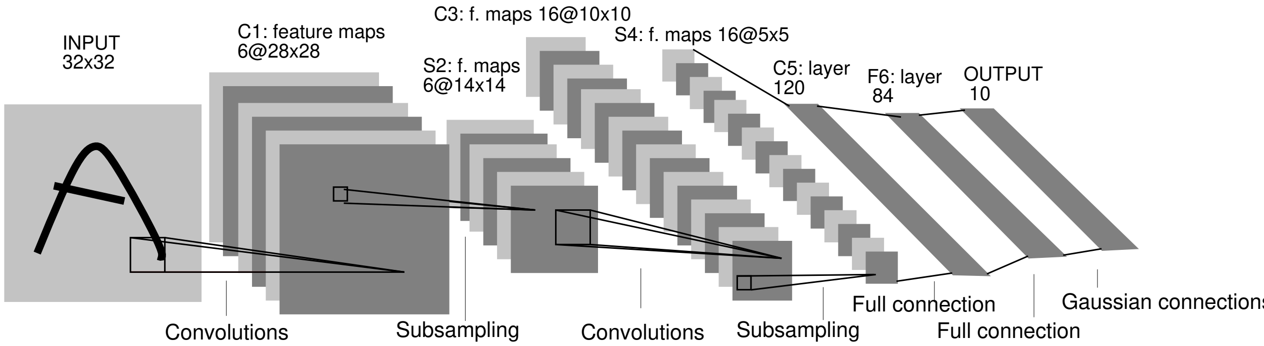

Figure 1: Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

and component-wise sigmoid functions (the units). Traditional multilayer neural nets where all the units in a layer are connected to all the units in the next layer can be used to recognize raw (roughly size-normalized and centered) images, but there are problems.

Firstly, typical images are large, often with several hundred variables (pixels). A fully-connected network with, say 100 units in the first layer, would already contain several 10,000 weights. Such a large number of parameters increases the capacity of the system and therefore requires a larger training set. But the main deficiency of unstructured nets is that they have no built-in invariance with respect to translations, scale, or geometric distortions of the inputs. Images of objects can be approximately size-normalized and centered, but no such preprocessing can be perfect. This, combined with intrinsic within-class variability, will cause variations in the position of distinctive features in input objects. In principle, a fully-connected network of sufficient size could learn to produce outputs that are invariant with respect to such variations. However, learning such a task would probably result in multiple units with similar weight patterns positioned at various locations in the input so as to detect distinctive features wherever they appear on the input. Learning these weight configurations requires a very large number of training instances to cover the space of possible variations. In convolutional networks, described below, the robustness to geometric distortions is automatically obtained by forcing the replication of weight configurations across space.

Secondly, a deficiency of fully-connected architectures is that the topology of the input is entirely ignored. The input variables can be presented in any (fixed) order without affecting the outcome of the training. On the contrary, images have a strong 2D local structure: variables (pixels) that are spatially nearby are highly correlated. Local correlations are the reasons for the well-known advantages of extracting and combining *local* features before recognizing spatial or temporal objects, because configurations of neighboring variables can be classified into a small number of relevant categories (e.g. edges, corners...). *Convolutional Networks* force the extraction of local features by restricting the receptive fields of hidden units to be local.

2.1 Convolutional Networks

Convolutional Networks combine three architectural ideas to ensure some degree of shift, scale, and distortion invariance: *local receptive fields*, *shared weights* (or weight replication), and spatial *subsampling*. A typical convolutional network for recognizing shapes, dubbed LeNet-5, is shown in figure 1. The input plane receives images of objects that are approximately size-normalized and centered. Each unit in a layer receives inputs from a set of units located in a small neighborhood in the previous layer. The idea of connecting units to local receptive fields on the input goes back to the early 60s, and was

largely inspired by Hubel and Wiesel's discovery of locally-sensitive, orientation-selective neurons in the cat's visual system ((Hubel and Wiesel 1962)). Local connections have been used many times in neural models of visual learning ((Fukushima 1975); (LeCun 1986); (LeCun et al. 1989); (Mozer 1991)). With local receptive fields, neurons can learn to extract elementary visual features such as oriented edges, end-points, corners (or similar features in other signals such as speech spectrograms). These features are then combined by the subsequent layers in order to detect higher-order features. As stated earlier, distortions or shifts of the input can cause the position of salient features to vary. In addition, elementary feature detectors that are useful on one part of the image are likely to be useful across the entire image. This knowledge can be applied by forcing a set of units, whose receptive fields are located at different places on the image, to have identical weight vectors ((Fukushima and Miyake 1982)), ((Rumelhart, Hinton and Williams 1986)),
((LeCun et al. 1989)). Units in a layer are organized in planes within which all the units share the same set of weights. The set of outputs of the units in such a plane is called a *feature map*. Units in a feature map are all constrained to perform the same operation on different parts of the image. A complete convolutional layer is composed of several feature maps (with different weight vectors), so that multiple features can be extracted at each location. A concrete example of this is the first layer of LeNet-5 shown in Figure 1. Units in the first hidden layer of LeNet-5 are organized in 6 planes, each of which is a feature map. A unit in a feature map has 25 inputs connected to a 5 by 5 area in the input, called the *receptive field* of the unit. Each unit has 25 inputs, and therefore 25 trainable coefficients plus a trainable bias. The receptive fields of contiguous units in a feature map are centered on correspondingly contiguous units in the previous layer. Therefore receptive fields of neighboring units overlap. For example, in the first hidden layer of LeNet-5, the receptive fields of horizontally contiguous units overlap by 4 columns and 5 rows. As stated earlier, all the units in a feature map share the same set of 25 weights and the same bias so they detect the same feature at all possible locations on the input. The other feature maps in the layer use different sets of weights and biases, thereby extracting different types of local features. In the case of LeNet-5, at each input location six different types of features are extracted by six units in identical locations in the six feature maps. A sequential implementation of a feature map would scan the input image with a single unit that has a local receptive field, and store the states of this unit at corresponding locations in the feature map. This operation is equivalent to a convolution, followed by an additive bias and squashing function, hence the name *convolutional network*. The kernel of the convolution is the set of connection weights used by the units in the feature map. An interesting property of convolutional layers is that if the input image is shifted, the feature map output will be shifted by the same amount, but will be left unchanged otherwise. This property is at the basis of the robustness of convolutional networks to shifts and distortions of the input.

Once a feature has been detected, its exact location becomes less important. Only its approximate position relative to other features is relevant. Using handwritten digits as an example, once we know that the input image contains the endpoint of a roughly horizontal segment in the upper left area, a corner in the upper right area, and the endpoint of a roughly vertical segment in the lower portion of the image, we can tell the input image is a 7. Not only is the precise position of each of those features irrelevant for identifying the pattern, it is potentially harmful because the positions are likely to vary for different instances of the shape. A simple way to reduce the precision with which the position of distinctive features are encoded in a feature map is to reduce the spatial resolution of the feature map. This can be achieved with a so-called *sub-sampling layers* which performs a local averaging and a sub-sampling, reducing the resolution of the feature map, and reducing the sensitivity of the output to shifts and distortions. The second hidden layer of LeNet-5 is a sub-sampling layer. This layer comprises six feature maps, one for each feature map in the previous layer. The receptive field of each unit is a 2 by 2 area in the previous layer's corresponding feature map. Each unit computes the *average* of its four inputs, multiplies it by a trainable coefficient, adds a trainable bias, and passes the result though a sigmoid function. Contiguous units have non-overlapping contiguous receptive fields. Consequently, a sub-sampling layer feature map has half the number of rows and columns as the feature maps in the previous layer. The trainable coefficient and bias control the effect of the sigmoid non-linearity. If the coefficient is small, then the unit operates in a quasi-linear mode, and the sub-sampling layer merely blurs the input. If the coefficient is large, sub-sampling units can be seen as performing a "noisy OR" or a "noisy AND" function depending on the value of the bias. Successive layers of convolutions and sub-sampling are typically alternated, resulting in a "bi-pyramid": at each layer, the number of feature maps is increased as the spatial resolution is decreased. Each unit in the third hidden layer in figure 1 may have input connections from several feature maps in the previous layer. The convolution/sub-sampling combination, inspired by Hubel and Wiesel's notions of "simple" and "complex" cells, was implemented in Fukushima's Neocognitron ((Fukushima and Miyake 1982)), though no globally supervised learning procedure such as back-propagation was available then. A large degree of invariance to geometric transformations of the input can be achieved with this progressive reduction of spatial resolution compensated by a progressive increase of the richness of the representation (the number of feature maps).

Since all the weights are learned with back-propagation, convolutional networks can be seen as synthesizing their own feature extractors, and tuning them to the task at hand. The weight sharing technique has the interesting side effect of reducing the number of free parameters, thereby reducing the "capacity" of the machine and reducing the gap between test error and training error. The network in figure 1 contains 345,308 connections, but only 60,000 trainable free parameters because of the weight sharing.

Fixed-size Convolutional Networks have been applied to many applications, among others: handwriting recognition ((LeCun et al. 1989); (Martin 1993)), as well as machine-printed character recognition ((Wang and Jean 1993)), on-line handwriting recognition ((Bengio et al. 1995)), and face recognition
((Lawrence et al. 1997)). Fixed-size convolutional networks that share weights along a single temporal dimension are known as Time-Delay Neural Networks (TDNNs) and applied widely in speech processing and time-series prediction. Variable-size convolutional networks, which have applications in object detection and location are described in section 3.

2.2 LeNet-5

This section describes in more detail the architecture of LeNet-5, the Convolutional Neural Network used in the experiments. LeNet-5 comprises 7 layers, not counting the output, all of which contain trainable parameters (weights). The input is a 32x32 pixel image. Input shapes should be significantly smaller than that (e.g. on the order of 20x20 pixels). The reason is that it is desirable that potential distinctive features such as end-points or corner can appear *in the center* of the receptive field of the highest-level feature detectors. In LeNet-5 the set of centers of the receptive fields of the last convolutional layer (C3, see below) form a 20x20 area in the center of the 32x32 input. The values of the input pixels are normalized so that the background level (white) corresponds to a value of -0.1 and the foreground (black) corresponds to 1.175. This makes the mean input roughly 0, and the variance roughly 1 which accelerates learning. In the following, convolutional layers are labeled Cx, sub-sampling layers are labeled Sx, and fully-connected layers are labeled Fx, where x is the layer index.

Layer C1 is a convolutional layer with 6 feature maps. Each unit in each feature map is connected to a 5x5 neighborhood in the input. The size of the feature maps is 28x28 which prevents connection from the input from falling off the boundary. C1 contains 156 trainable parameters, and 122,304 connections.

Layer S2 is a sub-sampling layer with 6 feature maps of size 14x14. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C1. The four inputs to a unit in S2 are added, then multiplied by a trainable coefficient, and added to a trainable bias. The result is passed through a sigmoidal function. The 2x2 receptive fields are non-overlapping, therefore feature maps in S2 have half the number of rows and column as feature maps in C1. Layer S2 has 12 trainable parameters and 5,880 connections.

Layer C3 is a convolutional layer with 16 feature maps. Each unit in each feature map is connected to several 5x5 neighborhoods at identical locations in a subset of S2's feature maps. Why not connect every S2 feature map to every C3 feature map? The reason is twofold. First, a non-complete connection scheme keeps the number of connections within reasonable bounds. More importantly, it forces a break of symmetry in the network. Different feature maps are forced to extract different (hopefully complementary) features because they get different sets of inputs. The rationale behind the connection scheme is the following. The first six C3 feature maps take inputs from every contiguous subsets of three feature maps in S2. The next six take input from every contiguous subset of four. The next three take input from some discontinuous subsets of four. Finally the last one takes input from all S2 feature maps. The full connection table is given in ((LeCun et al. 1998)), Layer C3 has 1,516 trainable parameters and 156,000 connections.

Layer S4 is a sub-sampling layer with 16 feature maps of size 5x5. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C3, in a similar way as C1 and S4. Layer S4 has 32 trainable parameters and 2,000 connections.

Layer C5 is a convolutional layer with 120 feature maps. Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. Here, because the size of S4 is also 5x5, the size of C5's feature maps is 1x1: this amounts to a full connection between S4 and C5. C5 is labeled as a convolutional layer, instead of a fully-connected layer, because if LeNet-5 input were made bigger with everything else kept constant, the feature map dimension would be larger than 1x1. This process of dynamically increasing the size of a convolutional network is described in the section Section 3. Layer C5 has 48,120 trainable connections.

Layer F6, contains 84 units (the reason for this number comes from the design of the output layer, explained later) and is fully connected to C5. It has 10,164 trainable parameters.

As in classical neural networks, units in layers up to F6 compute a dot product between their input vector and their weight vector, to which a bias is added. This weighted sum is then passed through a scaled hyperbolic tangent function to produce the state of the unit.

Finally, the output layer is composed of Euclidean Radial Basis Function units (RBF), one for each class, with 84 inputs each. Each output RBF unit computes the Euclidean distance between its input vector and its parameter vector. The output of a particular RBF can be interpreted as a penalty term measuring the fit between the input pattern and a model of the class associated with the RBF. Given an input pattern, the loss function should be designed so as to get the configuration of F6 as close as possible to the parameter vector of the RBF that corresponds to the pattern's desired class. The parameter vectors of these units were chosen by hand and kept fixed (at least initially). The components of those parameters vectors were set to -1 or +1 to predetermined values. The parameter vectors of the RBFs play the role of target vectors for layer F6.

The simplest output loss function that can be used with the above network is:

$$E(W) = \frac{1}{P}\sum_{p=1}^{P} y_{D^p}(Z^p, W) \qquad (2)$$

where y_{D^p} is the output of the D_p-th RBF unit, i.e. the one that corresponds to the correct class of input pattern Z^p. The actual loss function used in our experiments has additional term to make it more discriminative. More details are available in ((LeCun et al. 1998)). Computing the gradient of the loss function with respect to all the weights in all the layers of the convolutional network is done with back-propagation. The standard algorithm must be slightly modified to take account of the weight sharing. An easy way to implement it is to first compute the partial derivatives of the loss function with respect to each *connection*, as if the network were a conventional multi-layer network without weight sharing. Then the partial derivatives of all the connections that share a same parameter are added to form the derivative with respect to that parameter.

2.3 An Example: Recognizing Handwritten Digits

Recognizing individual digits is an excellent benchmark for comparing shape recognition methods. This comparative study concentrates on adaptive methods that operate directly on size-normalized images. Handwritten digit recognition may seem a little simplistic when one's interest is Computer Vision, but the simplicity is only apparent, and the problems to solve are essentially the same as with any 2D shape recognition, only there is abundant training data available, and the intra-class shape variability is considerably larger than with any rigid object recognition problem.

The database used to train and test the systems described in this paper was constructed from the NIST's Special Database 3 and 1 containing binary images of handwritten digits. From these, we built a database called MNIST which contains 60,000 training samples (half from SD1, half from SD3), and 10,000 test images (half from SD1 and half from SD3). The original black and white (bilevel) images were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as result of anti-aliased resampling. Three versions of the database were used. In the first version, the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. In some instances, this 28x28 field was extended to 32x32 with background pixels. This version of the database will be referred to as the *regular* database. In the second version of the database, (referred to as the *deslanted* version) the character images were deslanted using the moments of inertia of the black pixels and cropped down to 20x20 pixels images. In the third version of the database, used in some early experiments, the images were reduced to 16x16 pixels. The regular database is available at `http://www.research.att.com/yann`.

2.4 Results and Comparison with other Classifiers

Several versions of LeNet-5 were trained on the regular database, with typically 20 iterations through the entire training data. The test error rate stabilizes after around 10 passes through the training set at 0.95%. The error rate on the training set reaches 0.35% after 19 passes. The influence of the training set size was measured by training the network with 15,000, 30,000, and 60,000 examples. The results made it clear that additional training data would be beneficial. In another set of experiments, we artificially generated more training examples by randomly distorting the original training images. The increased training set was composed of the 60,000 original patterns plus 540,000 instances of distorted patterns with randomly picked distortion parameters. The distortions were combinations of the following planar affine transformations: horizontal and vertical translations, scaling, squeezing (simultaneous horizontal compression and vertical elongation, or the reverse), and horizontal shearing. Figure 2 shows examples of distorted patterns used for training. When distorted data was used for training, the test error rate dropped to 0.8% (from 0.95% without deformation). Some of the misclassified examples are genuinely ambiguous, but several are perfectly identifiable by humans, although they are written in an under-represented style. This shows that further improvements are to be expected with more training data.

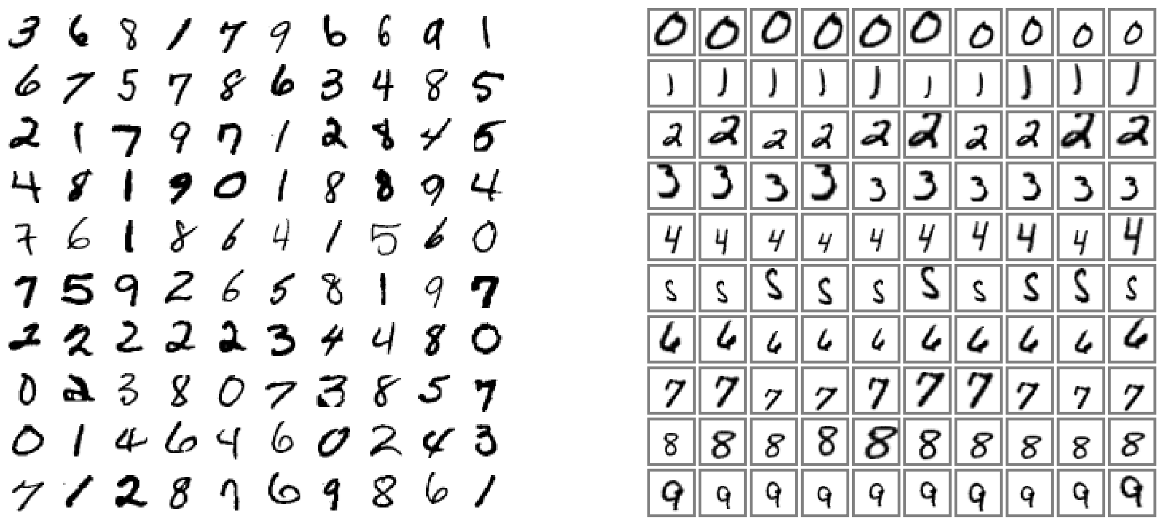

Figure 2: Examples from the test set (left), and examples of distortions of ten training patterns (right).

For the sake of comparison, a variety of other trainable classifiers was trained and tested on the same database. The error rates on the test set for the various methods are shown in figure 3. The experiments included the following methods: *linear classification* with 10 two-way classifiers trained to classify one class from the other nine; *pairwise linear classifier* with 45 two-way classifiers trained to classify one class versus one other followed by a voting mechanism; *K-Nearest Neighbor classifiers* with a simple Euclidean distance on pixel images; *40-dimension principal component analysis followed by degree 2 polynomial classifier*; *radial basis function network* with 1000 Gaussian RBF trained with K-means per class, and followed by a linear classifier; *Tangent Distance classifier*, a nearest-neighbor classifier where the distance is made invariant to small geometric distortions by projecting the pattern onto linear approximations of the manifolds generated by distorting the prototypes; *Support Vector Machines* of various types (regular SVM, reduced-set SVM, virtual SVM) using polynomial kernels; *fully connected neural nets* with one or two hidden layers and various numbers of hidden units; *LeNet-1*, a small convolutional neural net with only 2600 free parameters and 100,000 connections; *LeNet-4*, a convolutional neural net with 17,000 free parameters and 260,000 connection similar to but slightly different from LeNet-5; *Boosted LeNet4*, a classifier obtained by voting three instances of LeNet-4 trained on different subsets of the database; and finally *LeNet-5*.

Concerning fully-connected neural networks, it remains somewhat of a mystery that unstructured neural nets with such a large number of free parameters manage to achieve reasonable performance. We conjecture that the dynamics of gradient descent learning in multilayer nets has a "self-regularization" effect. Because the origin of weight space is a saddle point that is attractive in almost every direction, the weights invariably shrink during the first few epochs. Small weights cause the sigmoids to operate in the quasi-linear region, making the network essentially equivalent to a low-capacity, single-layer network. As the learning proceeds, the weights grow, which progressively increases the effective capacity of the network. This seems to be an almost perfect, if fortuitous, implementation of Vapnik's "Structural Risk Minimization" principle ((Vapnik 1995)).

The Support Vector Machine ((Vapnik 1995)) has excellent accuracy, which is most remarkable, because unlike the other high performance classifiers, it does not include *a priori* knowledge about the problem ((Burges and Schölkopf 1997)). In fact, this classifier would do just as well if the image pixels were permuted with a fixed mapping and lost their pictorial structure. However, reaching levels of performance comparable to the Convolutional Neural Networks can only be done at considerable

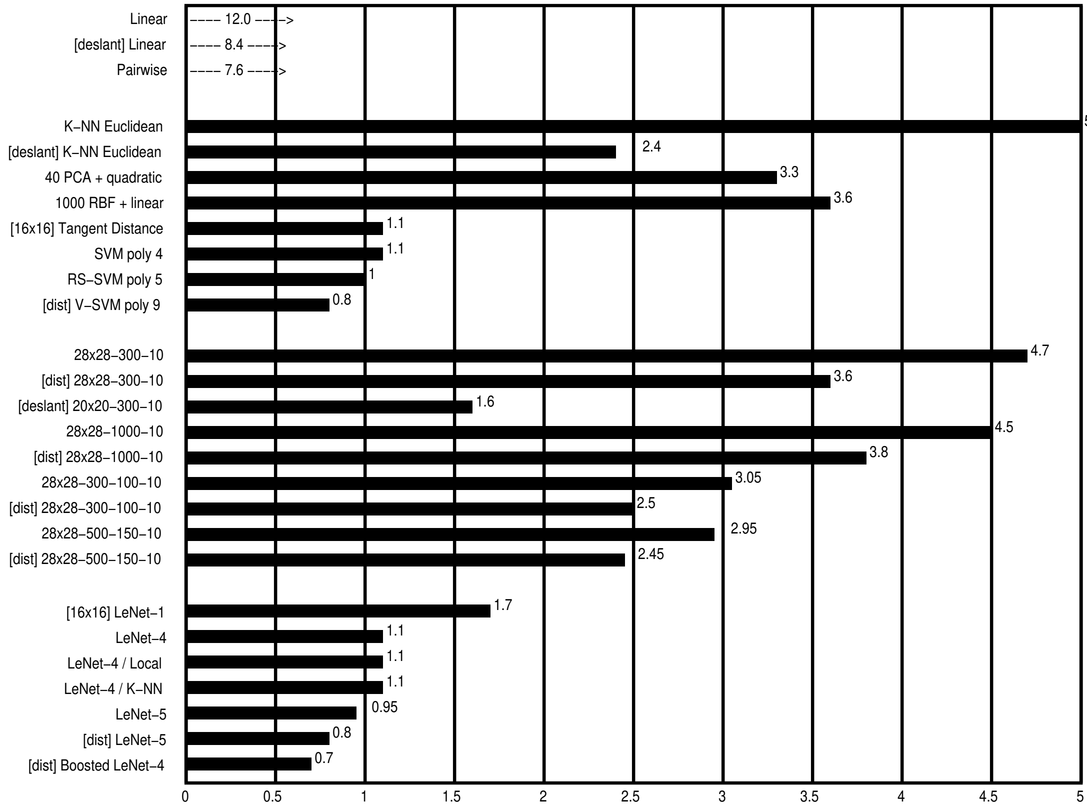

Figure 3: Error rate on the test set (%) for various classification methods. [deslant] indicates that the classifier was trained and tested on the deslanted version of the database. [dist] indicates that the training set was augmented with artificially distorted examples. [16x16] indicates that the system used the 16x16 pixel images. The uncertainty in the quoted error rates is about 0.1%.

expense in memory and computational requirements. The computational requirements of Burges's reduced-set SVM are within a factor of two of LeNet-5, and the error rate is very close. Improvements of those results are expected, as the technique is relatively new.

Boosted LeNet-4 performed best, achieving a score of 0.7%, closely followed by LeNet-5 at 0.8%. Boosted LeNet-4 ((Drucker, Schapire and Simard 1993)) is based on theoretical work by R. Schapire ((Schapire 1990)). Three LeNet-4s are combined: the first one is trained the usual way. the second one is trained on patterns that are filtered by the first net so that the second machine sees a mix of patterns, 50% of which the first net got right, and 50% of which it got wrong. Finally, the third net is trained on new patterns on which the first and the second nets disagree. During testing, the outputs of the three nets are simply added.

When plenty of data is available, many methods can attain respectable accuracy. Compared to other methods, convolutional neural nets offer not only the best accuracy, but also good speed, low memory requirements, and excellent robustness as discussed below.

2.5 Invariance and Noise Resistance

While fully invariant recognition of complex shapes is still an elusive goal, it seems that convolutional networks, because of their architecture, offer a partial answer to the problem of invariance or robustness with respect to distortions, varying position, scale and orientation, as well as intrinsic class variability.

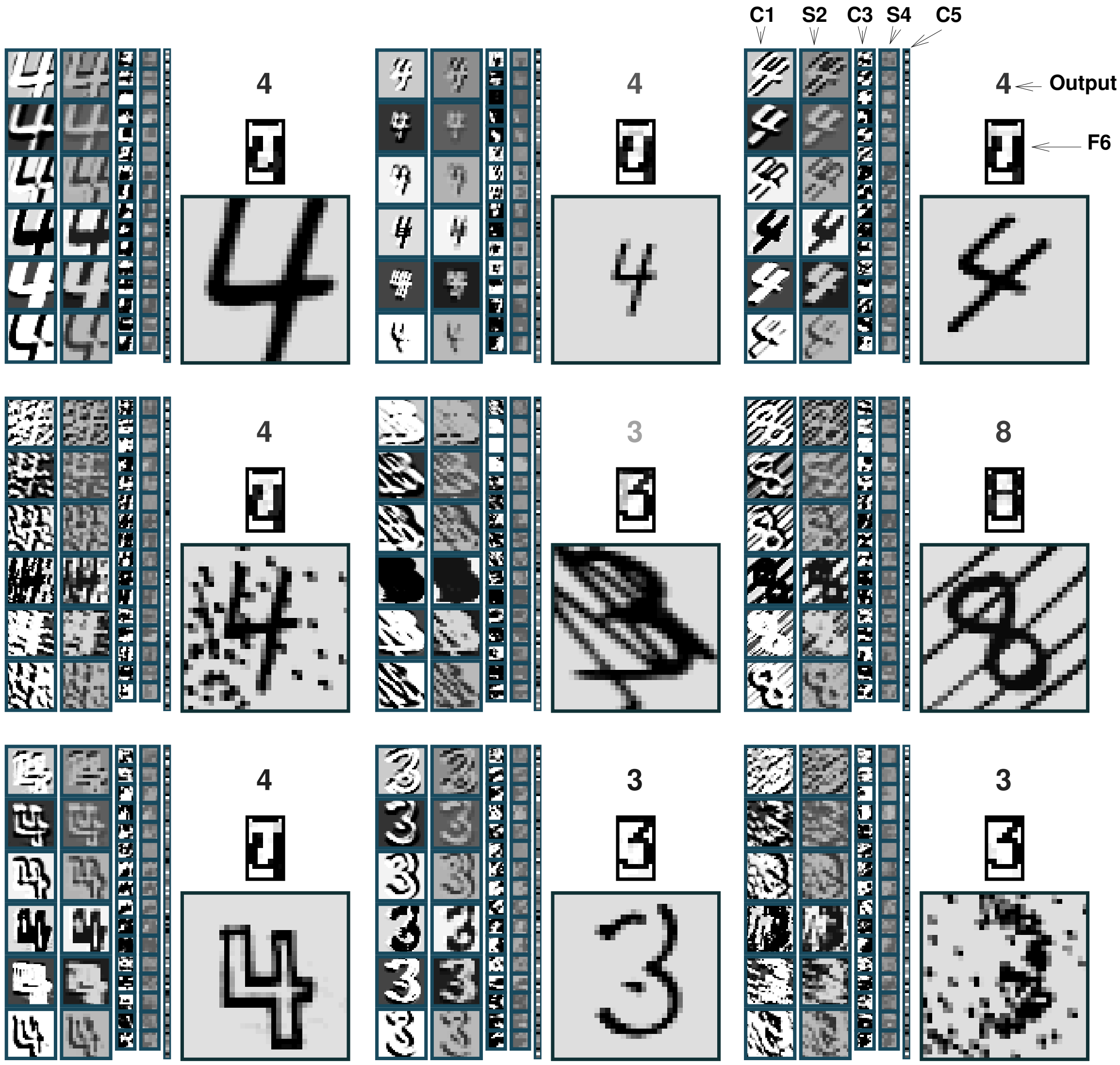

Figure 4: Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).

Figure 4 shows several examples of unusual and distorted characters that are correctly recognized by LeNet-5. For these experiments, the training samples were artificially distorted using random planar affine transformations, and the pixels in the training images were randomly flipped with probability 0.1 to increase the noise resistance. The top row in the figure shows the robustness to size and orientation variations. It is estimated that accurate recognition occurs for scale variations up to about a factor of 2, vertical shift variations of plus or minus about half the height of the character, and rotations up to plus or minus 30 degrees. While the characters are distorted during training, it seems that the robustness of the network subsists for distortions that are significantly larger than the ones used during training. Figure 4 includes examples of characters written in very unusual styles. Needless to say, there are no such examples in the training set. Nevertheless, the network classifies them correctly, which seems to suggest that the features that have been learned have some degree of generality. Lastly, figure 4 includes examples that demonstrates LeNet-5's robustness to extremely high levels of structured noise. Handling these images with traditional segmentation and feature extraction techniques would pose insurmountable problems. Even though the only noise used during training was random pixel flipping, it seems that the network can eliminate the adverse effects of non-sensical but structured marks from images such as the 3 and the 8 in the second row. This demonstrates a somewhat puzzling ability of such networks to perform (if implicitly) a kind of elementary *feature binding* solely through feed-forward linear combinations and sigmoid functions.

Animated examples of LeNet-5 in action are available on the Internet at `http://www.research.att.com/~yann`.

3 Multiple Object Recognition with Space Displacement Neural Networks

A major conceptual problem in vision and pattern recognition is how to recognize individual objects when those objects cannot be easily segmented out of their surrounding. In general, this poses the problem of feature binding: how to identify and bind together features that belong to a single object, while suppressing features that belong to the background or to other objects. The common wisdom is that, except in the simplest case, one cannot identify and bind together the features of an object unless one knows what object to look for.

In handwriting recognition, the problem is to separate a character from its neighbors, given that the neighbors can touch it or overlap with it. The most common solution is called "heuristic over-segmentation". It consists in generating a large number of potential cuts between characters using heuristic image analysis techniques. Candidate characters are formed by combining contiguous segments in multiple ways. The candidate characters are then sent to the recognizer for classification and scoring. A simple graph-search technique then finds the consistent sequence of character candidates with the best overall score.

There is a simple alternative to explicitly segmenting images of character strings using heuristics. The idea is to sweep a recognizer across all possible locations on an image of the entire word or string whose height has been normalized. With this technique, no segmentation heuristics is required. However, there are problems with this approach. First, the method is in general quite expensive. The recognizer must be applied at every possible location on the input, or at least at a large enough subset of locations so that misalignments of characters in the field of view of the recognizers are small enough to have no effect on the error rate. Second, when the recognizer is centered on a character to be recognized, the neighbors of the center character will be present in the field of view of the recognizer, possibly touching the center character. Therefore the recognizer must be able to correctly recognize the character in the center of its input field, even if neighboring characters are very close to, or touching the central character. Third, a word or character string cannot be perfectly size normalized. Individual characters within a string may have widely varying sizes and baseline positions. Therefore the recognizer must be very robust to shifts and size variations.

These three problems are elegantly circumvented if a convolutional network is replicated over the input field. First of all, as shown in the previous section, convolutional neural networks are very robust to shifts and scale variations of the input image, as well as to noise and extraneous marks in the input. These properties take care of the latter two problems mentioned in the previous paragraph. Second, convolutional networks provide a drastic saving in computational requirement when replicated over large input fields. A replicated convolutional network, also called a *Space Displacement Neural Network* or SDNN ((Matan et al. 1992)), is shown in Figure 5. While scanning a recognizer can be prohibitively expensive in general, convolutional networks can be scanned or replicated very efficiently over large, variable-size input fields. Consider one instance of a convolutional net and its *alter ego* at a nearby location. Because of the convolutional nature of the network, units in the two instances that look at identical locations on the input have identical outputs, therefore their states do not need

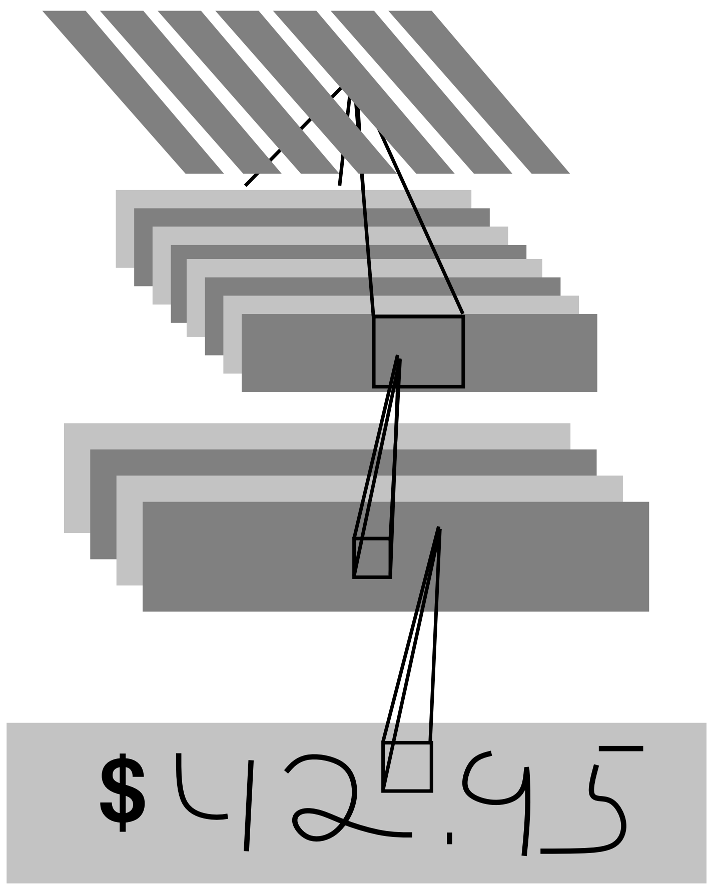

Figure 5: A Space Displacement Neural Network is a convolutional network that has been replicated over a wide input field.

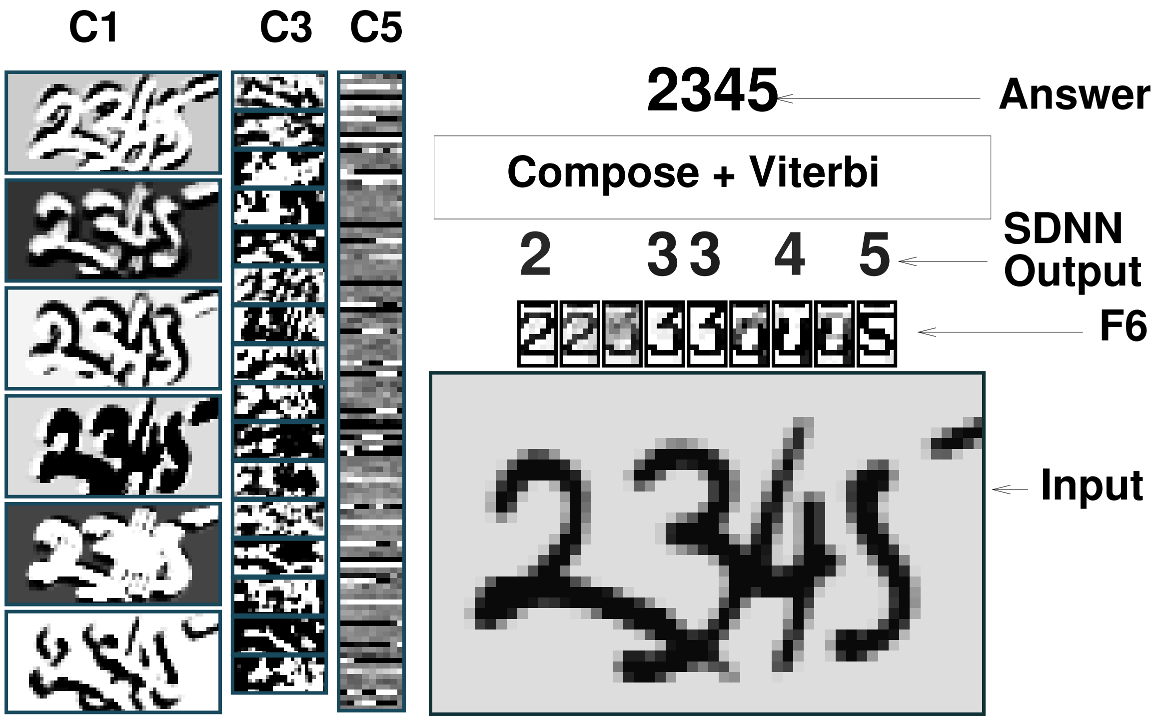

Figure 6: An example of multiple character recognition with SDNN. With SDNN, no explicit segmentation is performed.

to be computed twice. Only a thin "slice" of new states that are not shared by the two network instances needs to be recomputed. When all the slices are put together, the result is simply a larger convolutional network whose structure is identical to the original network, except that the feature maps are larger in the horizontal dimension. In other words, replicating a convolutional network can be done simply by increasing the size of the fields over which the convolutions are performed, and by replicating the output layer accordingly. The output layer effectively becomes a convolutional layer. An output whose receptive field is centered on an elementary object will produce the class of this object, while an in-between output may indicate no character or contain rubbish. The outputs can be interpreted as evidences for the presence of objects at all possible positions in the input field.

The SDNN architecture seems particularly attractive for recognizing cursive handwriting where no obvious segmentation heuristics exist. Although the idea of SDNN is quite old ((Keeler, Rumelhart and Leow 1991); (Matan et al. 1992)), and very attractive by its simplicity, it has not generated wide interest until recently because of the enormous demands it puts on the recognizer.

3.1 Interpreting the Output of an SDNN

The output of a horizontally replicated SDNN is a sequence of vectors which encode the likelihoods, penalties, or scores of finding character of a particular class label at the corresponding location in the input. A post-processor is required to pull out the best possible label sequence from this vector sequence. An example of SDNN output is shown in Figure 6. Very often, individual characters are spotted by several neighboring instances of the recognizer, a consequence of the robustness of the recognizer to horizontal translations. Also quite often, characters are erroneously detected by recognizer instances that see only a piece of a character. For example a recognizer instance that only sees the right third of a "4" might output the label 1. How can we eliminate those extraneous

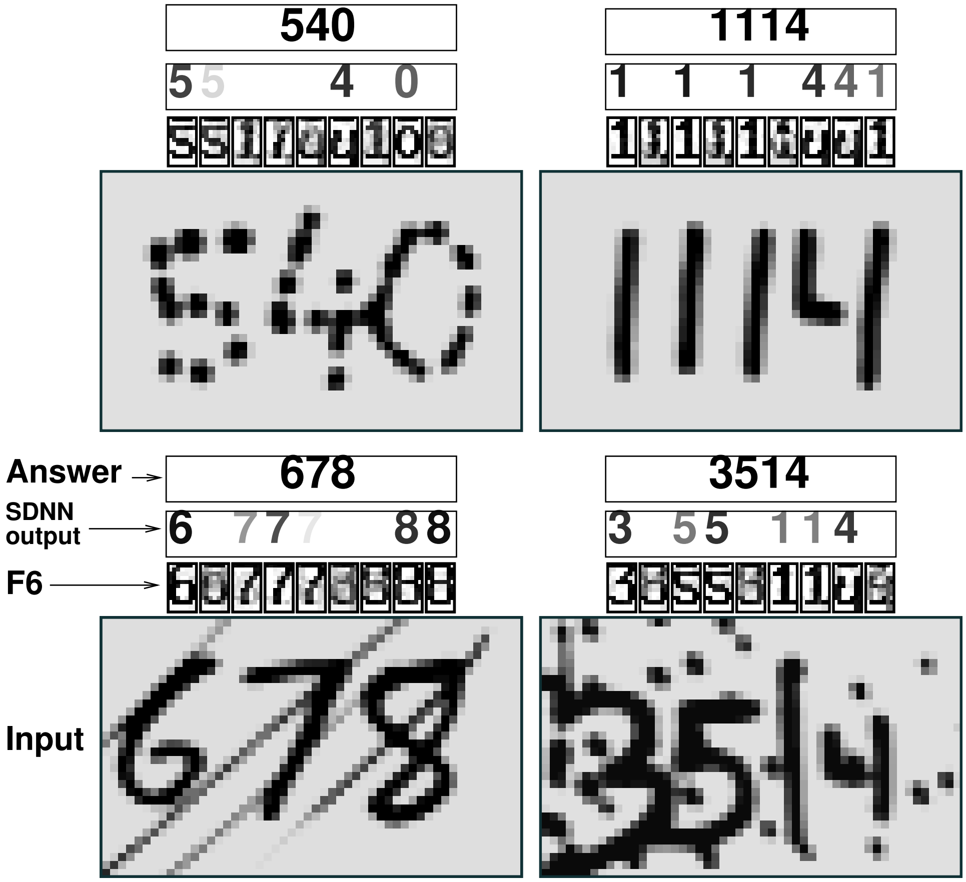

Figure 7: An SDNN applied to a noisy image of digit string. The digits shown in the SDNN output represent the winning class labels, with a lighter grey level for high-penalty answers.

characters from the output sequence and pull-out the best interpretation? This can be done with a simple weighted finite state machine. The sequence of vectors produced by the SDNN is first turned into a linear graph constructed as follows. Each vector in the output sequence is transformed into a bundle of arcs with a common source node and target node. Each arc contains one of the possible character labels, together with its corresponding penalty. Each bundle contains an additional arc bearing the "none of the above"label with a penalty. These bundles are concatenated in the order of the vector sequence (the target node of a bundle becomes the source node of the next bundle). Each path in this graph is a possible interpretation of the input. A grammar is then constructed as a weighted finite-state machine that contains a model for each character. The grammar ensures for example that neighboring characters must be separated by a "none of the above" label (white space), and that successive occurrences of the same label are probably produced by a single input character. The grammar and the linear graph are then *composed* (a graph operation similar to a tensor product). The composed graph contains all the paths of the linear graph that happen to be grammatically correct. A Viterbi algorithm can then be used to find the path with the smallest overall penalty.

3.2 Experiments with SDNN

In a series of experiments, LeNet-5 was trained with the goal of being replicated into an SDNN so as to recognize multiple characters without segmentations. The data was generated from the previously described MNIST set as follows. Training images were composed of a central character, flanked by two side characters picked at random in the training set. The separation between the bounding boxes of the characters were chosen at random between -1 and 4 pixels. In other instances, no central character was present, in which case the desired output of the network was the blank space class. In addition, training images were degraded by randomly flipping the pixels with probability 0.1.

Figures 6 and 7 show a few examples of successful recognitions of multiple characters by the LeNet-5 SDNN. Standard techniques based on Heuristic Over-Segmentation (see section 4.1) would likely fail on most of those examples. The robustness of the network to scale and vertical position variations allows it to recognize characters in such strings. More importantly, it seems that the network is able to individually recognize the characters even when there is a significant overlap with the neighbors. It is also able to correctly group disconnected pieces of ink that form characters, as exemplified in the upper half of the figure. In the top left example, the 4 and the 0 are more connected to each other than they are connected with themselves, yet the system correctly identifies the 4 and the 0 as separate objects. The top right example is interesting for several reasons. First the system correctly identifies the three individual "1". Second, the left half and right half of the disconnected 4 are correctly grouped, even though no simple proximity criterion could decide to associate the left half of the 4 to the vertical bar on its left or on its right. The right half of the 4 does cause the appearance of an erroneous "1" on the SDNN output, but this "1" is removed by the grammar which prevents different non-blank characters from appearing on contiguous outputs. The bottom left example demonstrates that extraneous marks that do not belong to identifiable characters are suppressed even though they may connect genuine characters to each other. The lower right example shows the combined robustness to character overlaps, vertical shifts, size variations, and noise.

Several authors have argued that invariance and feature binding for multiple object recognition requires specific mechanisms involving feedback, explicit switching devices (3-way multiplicative connections) ((Lades et al. 1993)), object-centered representations, graph matching mechanisms, or generative models that attempt to simultaneously extract the pose and the category of the objects. It is somewhat disconcerting to observe that the above SDNN appears to "solve" the feature binding problem, albeit partially and in a restricted context, even though it possesses no built in machinery to do it explicitly. If nothing else, these experiments show that purely feed-forward "numerical" multi-layer systems with a fixed architecture can emulate functions that appear combinatorial, and are qualitatively much more complex than anticipated by most (including the authors).

Several short animations of the LeNet-5 SDNN, including some with characters that move on top of each other, can be
viewed at `http://www.research.att.com/~yann`.

3.3 Face Detection and Spotting with SDNN

An interesting application of SDNNs is object detection and spotting. The invariance properties of Convolutional Networks, combined with the efficiency with which they can be replicated over large fields suggest that they can be used for "brute force" object spotting and detection in large images. The main idea is to train a single Convolutional Network to distinguish images of the object of interest from images present in the background. Once trained, the network is replicated so as to cover the entire image to be analyzed, thereby forming a two-dimensional Space Displacement Neural Network. The output of the SDNN is a two-dimensional plane in which the most activate units indicate the presence of the object of interest in the corresponding receptive field. Since the size of the objects to be detected within the image are unknown, the image can be presented to the network at multiple resolutions, and the results at multiple resolutions combined. The idea has been applied to face location, ((Vaillant, Monrocq and LeCun 1994)), address block location on envelopes ((Wolf and Platt 1994)), and hand tracking in video ((Nowlan and Platt 1995)).

To illustrate the method, we will consider the case of face detection in images as described in ((Vaillant, Monrocq and LeCun 1994)). First, images containing faces at various scales are collected. Those images are filtered through a zero-mean Laplacian filter so as to remove variations in global illumination and large-scale illumination gradients. Then, training samples of faces and non-faces are manually extracted from these images. The face sub-images are then size normalized so that the height of the entire face is approximately 20 pixels while keeping fairly large variations (within a factor of two). The scale of background sub-images are picked at random. A single convolutional network is trained on those samples to classify face sub-images from non-face sub-images. When a scene image is to be analyzed, it is first filtered through the Laplacian filter, and sub-sampled by ratios that are successive powers of the square root of 2. The network is replicated over each of the images at each resolution. A simple voting technique is used to combine the results from multiple resolutions.

More recently, some authors have used Neural Networks, or other classifiers such as Support Vector Machines for face detection with great success ((Rowley, et al. 1996); (Osuna et al. 1997); (Papageorgiou et al. 1998); (Sung and Poggio 1994)). Their systems are somewhat similar to the one described above, including the idea of presenting the image to the network at multiple scales. But since those systems do not use Convolutional Networks, they cannot take advantage of the speedup described here, and have to rely on other techniques, such as pre-filtering and real-time tracking, to keep the computational requirement within reasonable limits. In addition, because those classifiers are much less invariant to scale variations than Convolutional Networks, it is necessary to use a large number multiscale images with finely-spaced scales.

4 Graph Transformer Networks

Despite the apparent ability of the systems described in the previous sections to solve combinatorial problems with non-combinatorial means, there are situations where the need for *compositionality* and combinatorial searches is inescapable. A good example is language modeling and more generally, models that involve finite-state grammars, weighted finite-state machines, or other graph-based knowledge representations such as finite-state transducers. Another good example is the recognition of non-rigid objects composed of multiple movable parts. These objects are often best described by graphs whose nodes and arcs represent the parts and their relationships ((Ioffe and Forsyth)). The main point of this section is to show that gradient-based learning techniques can be extended to situations where those models are used.

It is easy to show that the modular gradient-based learning model presented in section 1 can be applied to networks of modules whose state variables X_n are graphs with numerical information attached to the arcs (scalars, vectors, etc), rather than fixed-size vectors. There are two main conditions for this. First, the modules must produce the values on the output graphs from the values on the input graphs through differentiable functions. Second, the overall loss function should be continuous and differentiable *almost everywhere* with respect to the parameters. Networks of graph-manipulating modules are called *Graph Transformer Networks* ((Bottou, LeCun and Bengio 1997); (LeCun et al. 1998)).

4.1 Word Recognition with a Graph Transformer Network

Though the Space Displacement Neural Net method presented in the previous section is very promising for word recognition applications, the more traditional method (and so far still the most developed) is called heuristic over-segmentation. With this methods, the word is segmented into candidate characters using heuristic image analysis techniques. Unfortunately, it is almost impossible to devise techniques that will infallibly segment naturally written words into well formed characters. This section and the next describe in detail a simple example of GTN for reading words. The method can rely on gradient-based learning to avoids the expensive and unreliable task of manually segmenting and hand-truthing a database so as to train the recognizer on individual characters.

4.1.1 Segmentation.

Given a word, a number of candidate cuts are generated with heuristic methods. The cut generation heuristic is designed so as to generate more cuts than necessary, in the hope that the "correct" set of cuts will be included. Once the cuts have been generated, alternative segmentations are best represented by a graph, called the *segmentation graph*. The segmentation graph is a *directed acyclic graph* with a start node and an end node. Each internal node is associated with a candidate cut produced by the segmentation algorithm. Each arc between a source node and a destination node is associated with an image that contains all the ink between the cut associated with the source node and the cut associated with the destination node. An arc is created between two nodes if the segmenter decided that the piece(s) of ink between the corresponding cuts could form a candidate character. Typically, each individual piece of ink would be associated with an arc. Pairs of successive pieces of ink would also be included, unless they are separated by a wide gap, which is a clear indication that they belong to different characters. Each complete path through the graph contains each piece of ink once and only once. Each path corresponds to a different way of associating pieces of ink together so as to form characters.

4.1.2 Recognition Transformer and Viterbi Transformer.

A simple GTN to recognize character strings is shown in Figure 8. Only the right branch of the top half is used for recognition. The left branch is used for the training procedure described in the next sub-section. The GTN is composed of two main graph transformers called the *recognition transformer* T_{rec}, and the *Viterbi transformer* T_{vit}. The goal of the recognition transformer is to generate a graph, called the *interpretation graph* or *recognition graph* G_{int}, that contains all the possible interpretations for all the possible segmentations of the input. Each path in G_{int} represents one possible interpretation of one particular segmentation of the input. The role of the Viterbi transformer is to extract the best interpretation from the interpretation graph.

The recognition transformer T_{rec} takes the segmentation graph G_{seg} as input, and applies the recognizer for single characters to the images associated with each of the arcs in the segmentation graph. The interpretation graph G_{int} has almost the same structure as the segmentation graph, except that each arc is replaced by a set of arcs from and to the same node. In this set of arcs, there is one arc for each possible class for the image associated with the corresponding arc in G_{seg}. To each arc is attached a class label, and the penalty that the image belongs to this class as produced by the recognizer. If the segmenter has computed penalties for the candidate segments, these penalties are combined with the penalties computed by the character recognizer, to obtain the penalties on the arcs of the interpretation graph. Although combining penalties of different nature seems highly heuristic, the GTN training procedure will tune the penalties and take advantage of this combination anyway. Each path in the interpretation graph corresponds to a possible interpretation of the input word. The penalty of a particular interpretation for a particular segmentation is given by the sum of the arc penalties along the corresponding path in the interpretation graph. Computing the penalty of an interpretation independently of the segmentation requires to combine the penalties of all the paths with that interpretation. This can be done using the "forward" algorithm widely used in Hidden Markov Models.

The Viterbi transformer produces a graph G_{vit} with a single path. This path is the path of least cumulated penalty in the interpretation graph. The result of the recognition can be produced by

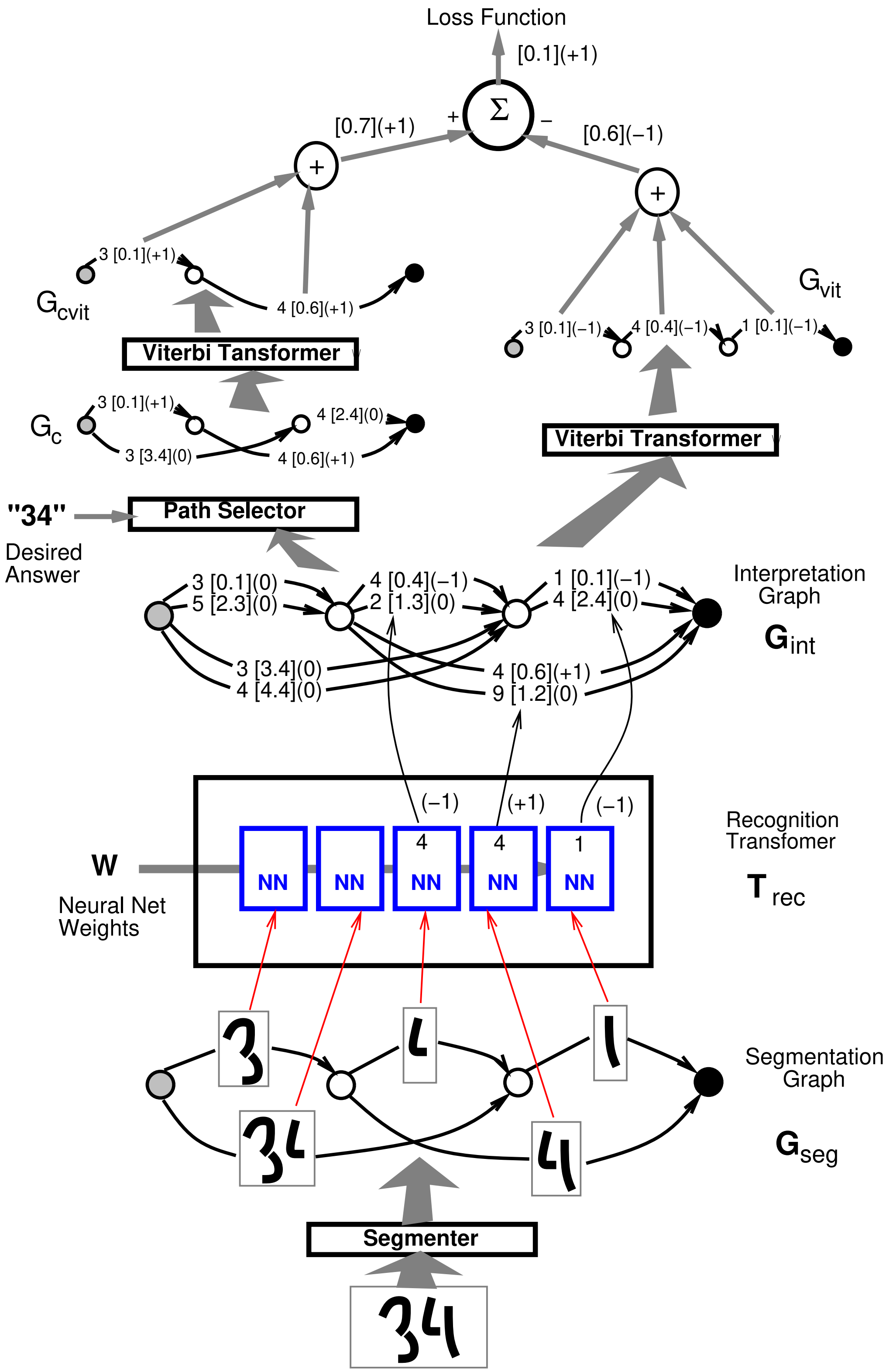

Figure 8: A GTN Architecture for word recognition based on Heuristic Over-Segmentation. During recognition, only the right-hand path of the top part is used. For training with Viterbi training, only the left-hand path is used. For discriminative Viterbi training, both paths are used. Quantities in square brackets are penalties computed during the forward propagation. Quantities in parentheses are partial derivatives computed during the backward propagation.

reading off the labels of the arcs along the graph G_{vit} extracted by the Viterbi transformer. The Viterbi transformer owes its name to the famous *Viterbi algorithm* to find the shortest path in a graph.

4.2 Gradient-Based Training of a GTN

The previous section describes the process of recognizing a string using Heuristic Over-Segmentation, assuming that the recognizer is trained so as to assign low penalties to the correct class label of correctly segmented characters, high penalties to erroneous categories of correctly segmented characters, and high penalties to all categories for badly formed characters. This section explains how to train the system at the string level to do the above without requiring manual labeling of character segments.

In many applications, there is enough a priori knowledge about what is expected from each of the modules in order to train them separately. For example, with Heuristic Over-Segmentation one could individually label single-character images and train a character recognizer on them, but it might be difficult to obtain an appropriate set of non-character images to train the model to reject wrongly segmented candidates. Although separate training is simple, it requires additional supervision information that is often lacking or incomplete (the correct segmentation and the labels of incorrect candidate segments). The following section describes two of the many gradient-based methods for training GTN-based handwriting recognizers at the string level: Viterbi training and discriminative Viterbi training. Unlike similar approaches in the context of speech recognition, we make no recourse to a probabilistic interpretation, but show that, within the Gradient-Based Learning approach, discriminative training is a simple instance of the pervasive principle of error correcting learning.

4.2.1 Viterbi Training.

During recognition, we select the path in the Interpretation Graph that has the lowest penalty with the Viterbi algorithm. Ideally, we would like this path of lowest penalty to be associated with the correct label sequence as often as possible. An obvious loss function to minimize is therefore the average over the training set of the penalty of the path *associated with the correct label sequence* that has the lowest penalty. The goal of training will be to find the set of recognizer parameters (the weights, if the recognizer is a neural network) that minimize the average penalty of this "correct" lowest penalty path. The gradient of this loss function can be computed by back-propagation through the GTN architecture shown in figure 8, using only the left-hand path of the top part, and ignoring the right half. This training architecture contains a graph transformer called a *path selector*, inserted between the Interpretation Graph and the Viterbi Transformer. This transformer takes the interpretation graph and the desired label sequence as input. It extracts from the interpretation graph those paths that contain the correct (desired) label sequence. Its output graph G_{c} is called the *constrained interpretation graph*, and contains all the paths that correspond to the correct label sequence. The constrained interpretation graph is then sent to the Viterbi transformer which produces a graph G_{cvit} with a single path. This path is the "correct" path with the lowest penalty. Finally, a path scorer transformer takes G_{cvit}, and simply computes its cumulated penalty C_{cvit} by adding up the penalties along the path. The output of this GTN is the loss function for the current pattern:

$$E_{\mathrm{vit}} = C_{\mathrm{cvit}} \tag{3}$$

The only label information that is required by the above system is the sequence of desired character labels. No knowledge of the correct segmentation is required on the part of the supervisor, since the system chooses among the segmentations in the interpretation graph the one that yields the lowest penalty.

The process of back-propagating gradients through the Viterbi training GTN is now described. As explained in section 1, the gradients must be propagated backwards through all modules of the GTN, in order to compute gradients in preceding modules and thereafter tune their parameters. Back-propagating gradients through the path scorer is quite straightforward. The partial derivatives of the loss function with respect to the individual penalties on the constrained Viterbi path G_{cvit} are equal to 1, since the loss function is simply the sum of those penalties. Back-propagating through the Viterbi Transformer is equally simple. The partial derivatives of E_{vit} with respect to the penalties on the arcs of the constrained graph G_{c} are 1 for those arcs that appear in the constrained Viterbi path G_{cvit}, and 0 for those that do not. Why is it legitimate to back-propagate through an essentially discrete function such as the Viterbi Transformer? The answer is that the Viterbi Transformer is nothing more than a collection of `min` functions and adders put together. It can be shown easily that gradients can be back-propagated through `min` functions without adverse effects. Back-propagation through the path selector transformer is similar to back-propagation through the Viterbi transformer. Arcs in G_{int} that appear in G_{c} have the same gradient as the corresponding arc in G_{c}, i.e. 1 or 0, depending on whether the arc appear in G_{cvit}. The other arcs, i.e. those that do not have an *alter ego* in G_{c} because they do not contain the right label have a gradient of 0. During the forward propagation through the recognition transformer, one instance of the recognizer for single character was created for each arc in the segmentation graph. The state of recognizer instances was stored. Since each arc penalty in G_{int} is produced by an individual output of a recognizer instance, we now have a gradient (1 or 0) for each output of each instance of the recognizer. Recognizer outputs that have a non zero gradient are part of the correct answer, and will therefore have their value pushed down. The gradients present on the recognizer outputs can be back-propagated through each recognizer instance. For each recognizer instance, we obtain a vector of partial derivatives of the loss function with respect to the recognizer instance parameters. All the recognizer instances share the same parameter vector, since they are merely clones of each other, therefore the full gradient of the loss function with respect to the recognizer's parameter vector is simply the sum of the gradient vectors produced by each recognizer instance. Viterbi training, though formulated differently, is often use in HMM-based speech recognition systems ((Rabiner 1989)).

While it seems simple and satisfying, this training architecture has a flaw that can potentially be fatal. If the recognizer is a simple neural network with sigmoid output units, the minimum of the loss function is attained, not when the recognizer always gives the right answer, but when it ignores the input, and sets its output to a constant vector with small values for all the components. This is known as *the collapse problem*. The collapse only occurs if the recognizer outputs can simultaneously take their minimum value. Another problem with Viterbi training is that the penalty of the answer cannot be used reliably as a measure of confidence because it does not take low-penalty (or high-scoring) competing (possibly incorrect) answers into account. A simple way to address this problem and to avoid the collapse is to train the whole system with a discriminative loss function as described in the next section.

4.2.2 Discriminative Viterbi Training.

The idea of discriminative Viterbi training is to not only minimize the cumulated penalty of the lowest penalty path with the correct interpretation, but also to somehow increase the penalty of competing and possibly incorrect paths that have a dangerously low penalty. This type of criterion is called *discriminative*, because it plays the good answers against the bad ones. Discriminative training procedures can be seen as attempting to build appropriate separating surfaces between classes rather than to model individual classes independently of each other.

One example of discriminative criterion is the difference between the penalty of the Viterbi path in the constrained graph, and the penalty of the Viterbi path in the (unconstrained) interpretation graph, i.e. the difference between the penalty of the best correct path, and the penalty of the best path (correct or incorrect). The corresponding GTN training architecture is shown in figure 8. The left side of the diagram is identical to the GTN used for non-discriminative Viterbi training. This loss function reduces the risk of collapse because it forces the recognizer to *increases* the penalty of wrongly recognized objects. Discriminative training can also be seen as another example of *error correction procedure*, which tends to minimize the difference between the desired output computed in the left half of the GTN in figure 8 and the actual output computed in the right half of figure 8.

Let the discriminative Viterbi loss function be denoted E_{dvit}, and let us call C_{cvit} the penalty of the Viterbi path in the constrained graph, and C_{vit} the penalty of the Viterbi path in the unconstrained interpretation graph:

$$E_{\mathrm{dvit}} = C_{\mathrm{cvit}} - C_{\mathrm{vit}} \tag{4}$$

E_{dvit} is always positive since the constrained graph is a subset of the paths in the interpretation graph, and the Viterbi algorithm selects the path with the lowest total penalty. In the ideal case, the two paths C_{cvit} and C_{vit} coincide, and E_{dvit} is zero.

Back-propagating gradients through the discriminative Viterbi GTN adds some "negative" training to the previously described non-discriminative training. Figure 8 shows how the gradients are back-propagated. The left half is identical to the non-discriminative Viterbi training GTN, therefore the back-propagation is identical. The gradients back-propagated through the right half of the GTN are multiplied by -1, since C_{vit} contributes to the loss with a negative sign. Otherwise the process is similar to the left half. The gradients on arcs of G_{int} get positive contributions from the left half and negative contributions from the right half. The two contributions must be added, since the penalties on G_{int} arcs are sent to the two halves through a "Y" connection in the forward pass. Arcs in G_{int} that appear neither in G_{vit} nor in G_{cvit} have a gradient of zero. They do not contribute to the cost. Arcs that appear in both G_{vit} and G_{cvit} also have zero gradient. The -1 contribution from the right half cancels the the +1 contribution from the left half. In other words, when an arc is rightfully part of the answer, there is no gradient. If an arc appears in G_{cvit} but not in G_{vit}, the gradient is +1. The arc should have had a lower penalty to make it to G_{vit}. If an arc is in G_{vit} but not in G_{cvit}, the gradient is -1. The arc had a low penalty, but should have had a higher penalty since it is not part of the desired answer. Variations of this technique have been used for the speech recognition.

4.2.3 GTN and Vision

GTN-based systems such as the one described in the previous section have been applied with success to handwriting recognition, and are the basis of several commercial document recognition systems used by the banking industry to read checks automatically.

While reading checks has great practical value, the more interesting scientific question is whether GTN can applied to more general computer vision problem. The gradient-based learning methods described in the previous section can in fact be generalized to any architecture that manipulates graphs. The method is not restricted to systems that manipulate left-to-right acyclic graphs, but can be applied to systems that represent object with trees, cliques, or any other graph structure. The only constraint is that the modules that implement the graph search or graph matching procedures must be continuous almost everywhere with respect to the tunable parameters of the modules that produce the values attached to the graphs, so that gradients can be back-propagated through them. Most classical graph matching and search algorithms have that property.

5 Conclusion

The methods described in this paper confirms what the history of Pattern Recognition has already shown repeatedly: finding ways to increase the role of learning and statistical estimation almost invariably improves the performance of recognition systems. For 2D shape recognition, Convolutional Neural Networks have been shown to eliminate the need for hand-crafted feature extractors. Replicated Convolutional Networks have been shown to handle fairly complex instances of the feature binding problem with a completely feed-forward, trained architecture instead of the more traditional combinatorial hypothesis testing methods.

Gradient-based learning architectures face the same segmentation alternative as other object detection or recognition approaches: exhaustive or selective, through hand-crafted heuristics. For both alternatives, they offer effective solutions to problems that have plagued most object recognition systems. Exhaustive exploration of every segmentation possibility results in a huge amount of computations, often redundant: Replicated Convolutional Networks make most computations be shared between many segmentation hypothesis. In situations where segmentation has to be selective, trainable Graph Transformer Networks have been shown to reduce the need for hand-crafted heuristics, manual labeling, and manual parameter tuning in document recognition systems.

Globally-trained Graph Transformer Networks have been applied successfully to on-line handwriting recognition and check recognition ((LeCun et al. 1998)). The check recognition system based on this concept is used commercially in several banks across the US and reads millions of checks per day. The concepts and results in this paper help establish the usefulness and relevance of gradient-based minimization methods as a general organizing principle for learning in large systems. It is clear that Graph Transformer Networks can be applied to many situations where the domain knowledge or the state information can be represented by graphs. This is the case in many visual tasks where graphs can represent alternative interpretations of a scene, multiple instances of an object, or relationship between objects.

References

Bengio, Y., LeCun, Y., Nohl, C., and Burges, C. (1995). LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition. *Neural Computation*, 7(5).

Bottou, L. and Gallinari, P. (1991). A Framework for the Cooperation of Learning Algorithms. In Touretzky, D. and Lippmann, R., editors, *Advances in Neural Information Processing Systems*, volume 3, Denver. Morgan Kaufmann.

Bottou, L., LeCun, Y., and Bengio, Y. (1997). Global Training of Document Processing Systems using Graph Transformer Networks. In *Proc. of Computer Vision and Pattern Recognition*, Puerto-Rico. IEEE.

Burges, C. J. C. and Schoelkopf, B. (1997). Improving the accuracy and speed of support vector machines. In M. Mozer, M. J. and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*. The MIT Press, Cambridge.

Drucker, H., Schapire, R., and Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 42–49, San Mateo, CA. Morgan Kaufmann.

Fukushima, K. (1975). Cognitron: A Self-Organizing Multilayered Neural Network. *Biological Cybernetics*, 20:121–136.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15:455–469.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive Fields, Binocular Interaction, and Functional Architecture in the Cat's Visual Cortex. *Journal of Physiology (London)*, 160:106–154.

Ioffe, S., and Forsyth, D. (1998). Learning to find Pictures of People *Advances in Neural Information Processing Systems 11.* The MIT Press, Cambridge.

Keeler, J., Rumelhart, D., and Leow, W. K. (1991). Integrated segmentation and recognition of hand-printed numerals. In Lippmann, R. P., Moody, J. M., and Touretzky, D. S., editors, *Neural Information Processing Systems*, volume 3, pages 557–563. Morgan Kaufmann Publishers, San Mateo, CA.

Lades, M., Vorbrüggen, J. C., Buhmann, J., and von der Malsburg, C. (1993). Distortion Invariant Object Recognition in the Dynamic Link Architecture. *IEEE Trans. Comp.*, 42(3):300–311.

Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997). Face Recognition: A Convolutional Neural Network Approach. *IEEE Transactions on Neural Networks*, 8(1):98–113.

LeCun, Y. (1986). Learning Processes in an Asymmetric Threshold Network. In Bienenstock, E., Fogelman-Soulié, F., and Weisbuch, G., editors, *Disordered systems and biological organization*, pages 233–240, Les Houches, France. Springer-Verlag.

LeCun, Y. (1988). A theoretical framework for Back-Propagation. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, CMU, Pittsburgh, Pa. Morgan Kaufmann.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, (86)11:2278–2324.

Martin, G. L. (1993). Centered-object integrated segmentation and recognition of overlapping hand-printed characters. *Neural Computation*, 5:419–429.

Matan, O., Burges, C. J. C., LeCun, Y., and Denker, J. S. (1992). Multi-Digit Recognition Using a Space Displacement Neural Network. In Moody, J. M., Hanson, S. J., and Lippman, R. P., editors, *Neural Information Processing Systems*, volume 4. Morgan Kaufmann Publishers, San Mateo, CA.

Moghaddam, B., and Pentland, A. (1995). Probabilistic Visual Learning for Object Recognition. *MIT Media Lab*, Technical Report 326.

Mozer, M. C. (1991). *The perception of multiple objects: A connectionist approach.* MIT Press-Bradford Books, Cambridge, MA.

Nowlan, S. and Platt, J. (1995). A Convolutional Neural Network Hand Tracker. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7*, pages 901–908, San Mateo, CA. Morgan Kaufmann.

Osuna, E., Freund, R., and Girosi, F. (1997). Training Support Vector Machines: an Application to Face Detection. In *Proceedings of CVPR'96*, pages 130–136. IEEE Computer Society Press.

Papageorgiou, C., Oren, M., and Poggio, T. (1998) LeRec: A General Framework for Object Detection. *Proceedings of the International Conference of Computer Vision 1998*, Bombay.

Rabiner, L. R. (1989). A Tutorial On Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286.

Rowley, H. A., Baluja, S., and Kanade, T. (1996). Neural Network-Based Face Detection. In *Proceedings of CVPR'96*, pages 203–208. IEEE Computer Society Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I, pages 318–362. Bradford Books, Cambridge, MA.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Sung, K.K. and Poggio, T. (1994). Example-based Learning for View-based Human Face Detection. *MIT Artificial Intelligence Lab*, A.I. Memo 1521

Vaillant, R., Monrocq, C., and LeCun, Y. (1994). Original approach for the localisation of objects in images. *IEE Proc on Vision, Image, and Signal Processing*, 141(4):245–250.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory.* Springer, New-York.

Wang, J. and Jean, J. (1993). Multi-resolution neural networks for omnifont character recognition. In *Proceedings of International Conference on Neural Networks*, volume III, pages 1588–1593.

Wolf, R. and Platt, J. (1994). Postal address block location using a convolutional locator network. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 745–752.

Yuille, A., Hallinan P., and Cohen, D. (1992). Feature Extraction from Faces using Deformable Templates. *Intl. Journal of Computer Vision*, 8(2):99-111.

Zhu, S.C., and Mumford, D. (1997). Feature Learning Generic Prior Models for Visual Computation. In *Proceedings of CVPR'97*, pages 463–469. IEEE Computer Society Press.