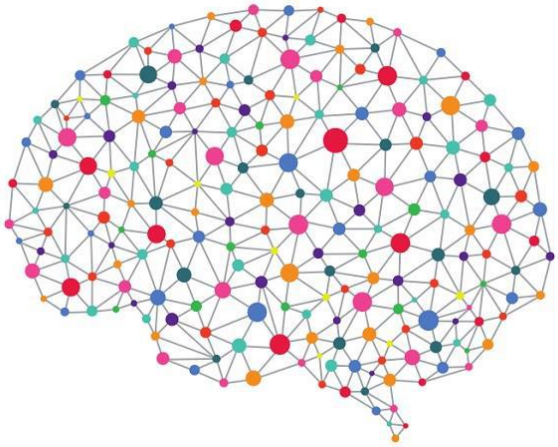
10 Important
AI Research Papers



These papers provide a breadth of information about Artificial intelligence (AI − the simulation of human intelligence processes by machines, especially computer systems) that is generally useful and interesting from a computer science perspective.

Contents

A Computational Approach to Edge Detection

JOHN CANNY, MEMBER, IEEE

Abstract—**This paper describes a computational approach to edge detection. The success of the approach depends on the definition of a *comprehensive* set of goals for the computation of edge points. These goals must be precise enough to delimit the desired behavior of the detector while making minimal assumptions about the form of the solution. We define detection and localization criteria for a class of edges, and present mathematical forms for these criteria as functionals on the operator impulse response. A third criterion is then added to ensure that the detector has only one response to a single edge. We use the criteria in numerical optimization to derive detectors for several common image features, including step edges. On specializing the analysis to step edges, we find that there is a natural uncertainty principle between detection and localization performance, which are the two main goals. With this principle we derive a single operator shape which is optimal at any scale. The optimal detector has a simple approximate implementation in which edges are marked at maxima in gradient magnitude of a Gaussian-smoothed image. We extend this simple detector using operators of several widths to cope with different signal-to-noise ratios in the image. We present a general method, called feature synthesis, for the fine-to-coarse integration of information from operators at different scales. Finally we show that step edge detector performance improves considerably as the operator point spread function is extended along the edge. This detection scheme uses several elongated operators at each point, and the directional operator outputs are integrated with the gradient maximum detector.**

Index Terms—**Edge detection, feature extraction, image processing, machine vision, multiscale image analysis.**

I. Introduction

EDGE detectors of some kind, particularly step edge detectors, have been an essential part of many computer vision systems. The edge detection process serves to simplify the analysis of images by drastically reducing the amount of data to be processed, while at the same time preserving useful structural information about object boundaries. There is certainly a great deal of diversity in the applications of edge detection, but it is felt that many applications share a common set of requirements. These requirements yield an abstract edge detection problem, the solution of which can be applied in any of the original problem domains.

We should mention some specific applications here. The Binford–Horn line finder [14] used the output of an edge detector as input to a program which could isolate simple geometric solids. More recently the model-based vision system ACRONYM [3] used an edge detector as the front end to a sophisticated recognition program. Shape from motion [29], [13] can be used to infer the structure of three-dimensional objects from the motion of edge contours or edge points in the image plane. Several modern theories of stereopsis assume that images are preprocessed by an edge detector before matching is done [19], [20]. Beattie [1] describes an edge-based labeling scheme for low-level image understanding. Finally, some novel methods have been suggested for the extraction of three-dimensional information from image contours, namely shape from contour [27] and shape from texture [31].

In all of these examples there are common criteria relevant to edge detector performance. The first and most obvious is low error rate. It is important that edges that occur in the image should not be missed and that there be no spurious responses. In all the above cases, system performance will be hampered by edge detector errors. The second criterion is that the edge points be well localized. That is, the distance between the points marked by the detector and the "center" of the true edge should be minimized. This is particularly true of stereo and shape from motion, where small disparities are measured between left and right images or between images produced at slightly different times.

In this paper we will develop a mathematical form for these two criteria which can be used to design detectors for arbitrary edges. We will also discover that the first two criteria are not "tight" enough, and that it is necessary to add a third criterion to circumvent the possibility of multiple responses to a single edge. Using numerical optimization, we derive optimal operators for ridge and roof edges. We will then specialize the criteria for step edges and give a parametric closed form for the solution. In the process we will discover that there is an uncertainty principle relating detection and localization of noisy step edges, and that there is a direct tradeoff between the two. One consequence of this relationship is that there is a single unique "shape" of impulse response for an optimal step edge detector, and that the tradeoff between detection and localization can be varied by changing the spatial width of the detector. Several examples of the detector performance on real images will be given.

II. One-Dimensional Formulation

To facilitate the analysis we first consider one-dimensional edge profiles. That is, we will assume that two-

Manuscript received December 10, 1984; revised November 27, 1985. Recommended for acceptance by S. L. Tanimoto. This work was supported in part by the System Development Foundation, in part by the Office of Naval Research under Contract N00014-81-K-0494, and in part by the Advanced Research Projects Agency under Office of Naval Research Contracts N00014-80-C-0505 and N00014-82-K-0334.

The author is with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.

IEEE Log Number 8610412.

0162-8828/86/1100-0679$01.00 © 1986 IEEE

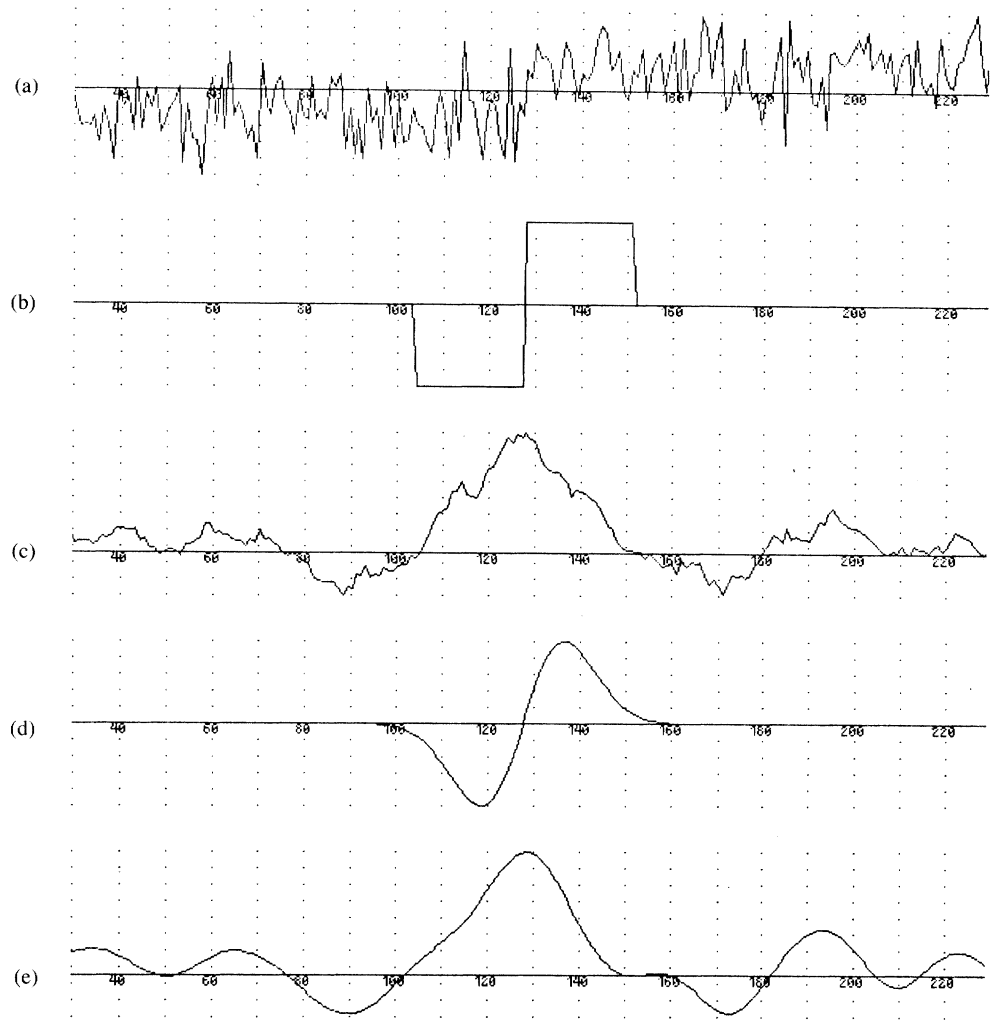


Fig. 1. (a) A noisy step edge. (b) Difference of boxes operator. (c) Difference of boxes operator applied to the edge. (d) First derivative of Gaussian operator. (e) First derivative of Gaussian applied to the edge.

dimensional edges locally have a constant cross-section in some direction. This would be true for example, of smooth edge contours or of ridges, but not true of corners. We will assume that the image consists of the edge and additive white Gaussian noise.

The detection problem is formulated as follows: We begin with an edge of known cross-section bathed in white Gaussian noise as in Fig. 1(a), which shows a step edge. We convolve this with a filter whose impulse response could be illustrated by either Fig. 1(b) or (d). The outputs of the convolutions are shown, respectively, in Fig. 1(c) and (e). We will mark the center of an edge at a local maximum in the output of the convolution. The design problem then becomes one of finding the filter which gives the best performance with respect to the criteria given below. For example, the filter in Fig. 1(d) performs much better than Fig. 1(b) on this example, because the response of the latter exhibits several local maxima in the region of the edge.

In summary, the three performance criteria are as follows:

1) Good detection. There should be a low probability of failing to mark real edge points, and low probability of falsely marking nonedge points. Since both these probabilities are monotonically decreasing functions of the output signal-to-noise ratio, this criterion corresponds to maximizing signal-to-noise ratio.

2) Good localization. The points marked as edge points by the operator should be as close as possible to the center of the true edge.

3) Only one response to a single edge. This is implicitly captured in the first criterion since when there are two responses to the same edge, one of them must be considered false. However, the mathematical form of the first criterion did not capture the multiple response requirement and it had to be made explicit.

A. Detection and Localization Criteria

A crucial step in our method is to capture the intuitive criteria given above in a mathematical form which is readily solvable. We deal first with signal-to-noise ratio and localization. Let the impulse response of the filter be $f(x)$, and denote the edge itself by $G(x)$. We will assume that the edge is centered at $x = 0$. Then the response of the

filter to this edge at its center H_G is given by a convolution integral:

$$H_G = \int_{-W}^{+W} G(-x)\, f(x)\, dx \tag{1}$$

assuming the filter has a finite impulse response bounded by $[-W, W]$. The root-mean-squared response to the noise $n(x)$ only, will be

$$H_n = n_0 \left[\int_{-W}^{+W} f^2(x)\, dx \right]^{1/2} \tag{2}$$

where n_0^2 is the mean-squared noise amplitude per unit length. We define our first criterion, the output signal-to-noise ratio, as the quotient of these two responses.

$$\text{SNR} = \frac{\left| \int_{-W}^{+W} G(-x)\, f(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f^2(x)\, dx}} \tag{3}$$

For the localization criterion, we want some measure which increases as localization improves, and we will use the reciprocal of the root-mean-squared distance of the marked edge from the center of the true edge. Since we have decided to mark edges at local maxima in the response of the operator $f(x)$, the first derivative of the response will be zero at these points. Note also that since edges are centered at $x = 0$, *in the absence of noise* there should be a local maximum in the response at $x = 0$.

Let $H_n(x)$ be the response of the filter to noise only, and $H_G(x)$ be its response to the edge, and suppose there is a local maximum in the total response at the point $x = x_0$. Then we have

$$H_n'(x_0) + H_G'(x_0) = 0. \tag{4}$$

The Taylor expansion of $H_G'(x_0)$ about the origin gives

$$H_G'(x_0) = H_G'(0) + H_G''(0)x_0 + O(x_0^2). \tag{5}$$

By assumption $H_G'(0) = 0$, i.e., the response of the filter in the absence of noise has a local maximum at the origin, so the first term in the expansion can be ignored. The displacement x_0 of the actual maximum is assumed to be small so we will ignore quadratic and higher terms. In fact by a simple argument we can show that if the edge $G(x)$ is either symmetric or antisymmetric, all even terms in x_0 vanish. Suppose $G(x)$ is antisymmetric, and express $f(x)$ as a sum of a symmetric component and an antisymmetric component. The convolution of the symmetric component with $G(x)$ contributes nothing to the numerator of the SNR, but it does contribute to the noise component in the denominator. Therefore, if $f(x)$ has any symmetric component, its SNR will be worse than a purely antisymmetric filter. A dual argument holds for symmetric edges, so that if the edge $G(x)$ is symmetric or antisymmetric, the filter $f(x)$ will follow suit. The net result of this is that the response $H_G(x)$ is always symmetric, and that its derivatives of odd orders [which appear in the coefficients of even order in (5)] are zero at the origin. Equations (4) and (5) give

$$H_G''(0)x_0 \approx -H_n'(x_0). \tag{6}$$

Now $H_n'(x_0)$ is a Gaussian random quantity whose variance is the mean-squared value of $H_n'(x_0)$, and is given by

$$E[H_n'(x_0)^2] = n_0^2 \int_{-W}^{+W} f'^2(x)\, dx \tag{7}$$

where $E[y]$ is the expectation value of y. Combining this result with (6) and substituting for $H_G''(0)$ gives

$$E[x_0^2] \approx \frac{n_0^2 \int_{-W}^{+W} f'^2(x)\, dx}{\left[\int_{-W}^{+W} G'(-x)\, f'(x)\, dx \right]^2} = \delta x_0^2 \tag{8}$$

where δx_0 is an approximation to the standard deviation of x_0. The localization is defined as the reciprocal of δx_0.

$$\text{Localization} = \frac{\left| \int_{-W}^{+W} G'(-x)\, f'(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f'^2(x)\, dx}}. \tag{9}$$

Equations (3) and (9) are mathematical forms for the first two criteria, and the design problem reduces to the maximization of both of these simultaneously. In order to do this, we maximize the product of (3) and (9). We could conceivably have combined (3) and (9) using any function that is monotonic in two arguments, but the use of the product simplifies the analysis for step edges, as should become clear in Section III. For the present we will make use of the product of the criteria for arbitrary edges, i.e., we seek to maximize

$$\frac{\left| \int_{-W}^{+W} G(-x)\, f(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f^2(x)\, dx}} \cdot \frac{\left| \int_{-W}^{+W} G'(-x)\, f'(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f'^2(x)\, dx}}. \tag{10}$$

There may be some additional constraints on the solution, such as the multiple response constraint (12) described next.

B. Eliminating Multiple Responses

In our specification of the edge detection problem, we decided that edges would be marked at local maxima in the response of a linear filter applied to the image. The detection criterion given in the last section measures the effectiveness of the filter in discriminating between signal and noise at the center of an edge. It does not take into account the behavior of the filter *nearby* the edge center. The first two criteria can be trivially maximized as fol-

lows. From the Schwarz inequality for integrals we can show that SNR (3) is bounded above by

$$n_0^{-1}\sqrt{\int_{-W}^{+W} G^2(x)\,dx}$$

and localization (9) by

$$n_0^{-1}\sqrt{\int_{-W}^{+W} G'^2(x)\,dx}.$$

Both bounds are attained, and the product of SNR and localization is maximized when $f(x) = G(-x)$ in $[-W, W]$.

Thus, according to the first two criteria, the optimal detector for step edges is a truncated step, or difference of boxes operator. The difference of boxes was used by Rosenfeld and Thurston [25], and in conjunction with lateral inhibition by Herskovits and Binford [11]. However it has a very high bandwidth and tends to exhibit many maxima in its response to noisy step edges, which is a serious problem when the imaging system adds noise or when the image itself contains textured regions. These extra edges should be considered erroneous according to the first of our criteria. However, the analytic form of this criterion was derived from the response at a single point (the center of the edge) and did not consider the interaction of the responses at several nearby points. If we examine the output of a difference of boxes edge detector we find that the response to a noisy step is a roughly triangular peak with numerous sharp maxima in the vicinity of the edge (see Fig. 1).

These maxima are so close together that it is not possible to select one as the response to the step while identifying the others as noise. We need to add to our criteria the requirement that the function f will not have "too many" responses to a single step edge in the vicinity of the step. We need to limit the number of peaks in the response so that there will be a low probability of declaring more than one edge. Ideally, we would like to make the distance between peaks in the noise response approximate the width of the response of the operator to a single step. This width will be some fraction of the operator width W.

In order to express this as a functional constraint on f, we need to obtain an expression for the distance between adjacent noise peaks. We first note that the mean distance between adjacent maxima in the output is twice the distance between adjacent zero-crossings in the derivative of the operator output. Then we make use of a result due to Rice [24] that the average distance between zero-crossings of the response of a function g to Gaussian noise is

$$x_{\text{ave}} = \pi\left(\frac{-R(0)}{R''(0)}\right)^{1/2} \tag{11}$$

where $R(\tau)$ is the autocorrelation function of g. In our case we are looking for the mean zero-crossing spacing for the function f'. Now since

$$R(0) = \int_{-\infty}^{+\infty} g^2(x)\,dx \quad \text{and} \quad R''(0) = -\int_{-\infty}^{+\infty} g'^2(x)\,dx$$

the mean distance between zero-crossings of f' will be

$$x_{zc}(f) = \pi\left(\frac{\int_{-\infty}^{+\infty} f'^2(x)\,dx}{\int_{-\infty}^{+\infty} f''^2(x)\,dx}\right)^{1/2} \tag{12}$$

The distance between adjacent maxima in the noise response of f, denoted $x_{\max}$, will be twice x_{zc}. We set this distance to be some fraction k of the operator width.

$$x_{\max}(f) = 2x_{zc}(f) = kW. \tag{13}$$

This is a natural form for the constraint because the response of the filter will be concentrated in a region of width $2W$, and the expected number of noise maxima in this region is N_n where

$$N_n = \frac{2W}{x_{\max}} = \frac{2}{k}. \tag{14}$$

Fixing k fixes the number of noise maxima that could lead to a false response.

We remark here that the intermaximum spacing (12) scales with the operator width. That is, we first define an operator f_w which is the result of stretching f by a factor of w, $f_w(x) = f(x/w)$. Then after substituting into (12) we find that the intermaximum spacing for f_w is $x_{zc}(f_w) = wx_{zc}(f)$. Therefore, if a function f satisfies the multiple response constraint (13) for fixed k, then the function f_w will also satisfy it, assuming W scales with w. For any fixed k, the multiple response criterion is invariant with respect to spatial scaling of f.

III. Finding Optimal Detectors by Numerical Optimization

In general it will be difficult (or impossible) to find a closed form for the function f which maximizes (10) subject to the multiple response constraint. Even when G has a particularly simple form (e.g., it is a step edge), the form of f may be complicated. However, if we are given a candidate function f, evaluation of (10) and (12) is straightforward. In particular, if the function f is represented by a discrete time sequence, evaluation of (10) requires only the computation of four inner products between sequences. This suggests that numerical optimization can be done directly on the sampled operator impulse response.

The output will not be an analytic form for the operator, but an implementation of a detector for the edge of interest will require discrete point-spread functions anyway. It is also possible to include additional constraints by using a *penalty method* [15]. In this scheme, the constrained optimization is reduced to one, or possibly several, unconstrained optimizations. For each constraint we define a penalty function which has a nonzero value when one

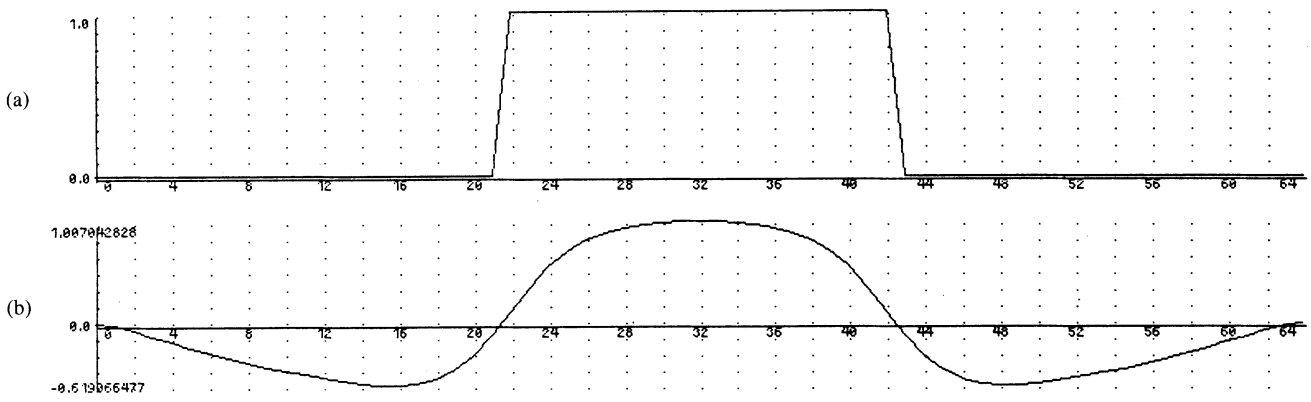


Fig. 2. A ridge profile and the optimal operator for it.

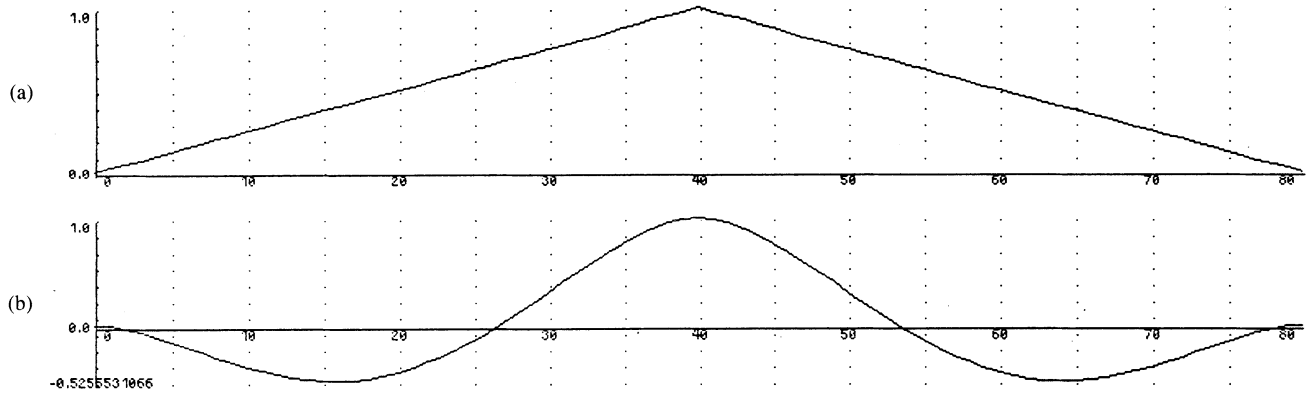


Fig. 3. A roof profile and an optimal operator for roofs.

of the constraints is violated. We then find the f which maximizes

$$\text{SNR}(f) * \text{Localization}\ (f) - \sum \mu_i P_i(f) \qquad (15)$$

where P_i is a function which has a positive value only when a constraint is violated. The larger the value of μ_i the more nearly the constraints will be satisfied, but at the same time the greater the likelihood that the problem will be ill-conditioned. A sequence of values of μ_i may need to be used, with the final form of f from each optimization used as the starting form for the next. The μ_i are increased at each iteration so that the value of $P_i(f)$ will be reduced, until the constraints are "almost" satisfied.

An example of the method applied to the problem of detecting "ridge" profiles is shown in Fig. 2. For a ridge, the function G is defined to be a flat plateau of width w, with step transitions to zero at the ends. The auxiliary constraints are

- The multiple response constraint. This constraint is taken directly from (12), and does not depend on the form of the edge.
- The operator should have zero dc component. That is it should have zero output to constant input.

Since the width of the operator is dependent on the width of the ridge, there is a suggestion that several widths of operators should be used. This has not been done in the present implementation however. A wide ridge can be considered to be two closely spaced edges, and the implementation already includes detectors for these. The only reason for using a ridge detector is that there are ridges in images that are too small to be dealt with effectively by the narrowest edge operator. These occur frequently because there are many edges (e.g., scratches and cracks or printed matter) which lie at or beyond the resolution of the camera and result in contours only one or two pixels wide.

A similar procedure was used to find an optimal operator for roof edges. These edges typically occur at the concave junctions of two planar faces in polyhedral objects. The results are shown in Fig. 3. Again there are two subsidiary constraints, one for multiple responses and one for zero response to constant input.

A roof edge detector has not been incorporated into the implementation of the edge detector because it was found that ideal roof edges were relatively rare. In any case the ridge detector is an approximation to the ideal roof detector, and is adequate to cope with roofs. The situation may be different in the case of an edge detector designed explicitly to deal with images of polyhedra, like the Binford–Horn line-finder [14].

The method just described has been used to find optimal operators for both ridge and roof profiles and in addition it successfully finds the optimal step edge operator derived in Section IV. It should be possible to use it to find operators for arbitrary one-dimensional edges, and it should be possible to apply the method in two dimensions to find optimal detectors for various types of corner.

IV. A Detector for Step Edges

We now specialize the results of the last section to the case where the input $G(x)$ is step edge. Specifically we set $G(x) = Au_{-1}(x)$ where $u_n(x)$ is the nth derivative of a delta function, and A is the amplitude of the step. That is,

$$u_{-1}(x) = \begin{cases} 0, & \text{for } x < 0; \\ 1, & \text{for } x \geq 0; \end{cases} \tag{16}$$

and substituting for $G(x)$ in (3) and (9) gives

$$\text{SNR} = \frac{A\left|\int_{-W}^{0} f(x)\,dx\right|}{n_0\sqrt{\int_{-W}^{+W} f^2(x)\,dx}} \tag{17}$$

$$\text{Localization} = \frac{A|f'(0)|}{n_0\sqrt{\int_{-W}^{+W} f'^2(x)\,dx}}. \tag{18}$$

Both of these criteria improve directly with the ratio A/n_0 which might be termed the signal-to-noise ratio of the image. We now remove this dependence on the image and define two performance measures Σ and Λ which depend on the filter only:

$$\text{SNR} = \frac{A}{n_0}\Sigma(f) \qquad \Sigma(f) = \frac{\left|\int_{-W}^{0} f(x)\,dx\right|}{\sqrt{\int_{-W}^{+W} f^2(x)\,dx}} \tag{19}$$

$$\text{Localization} = \frac{A}{n_0}\Lambda(f') \qquad \Lambda(f') = \frac{|f'(0)|}{\sqrt{\int_{-W}^{+W} f'^2(x)\,dx}}. \tag{20}$$

Suppose now that we form a spatially scaled filter f_w from f, where $f_w(x) = f(x/w)$. Recall from the end of Section II that the multiple response criterion is unaffected by spatial scaling. When we substitute f_w into (19) and (20) we obtain for the performance of the scaled filter:

$$\Sigma(f_w) = \sqrt{w}\Sigma(f) \quad \text{and} \quad \Lambda(f'_w) = \frac{1}{\sqrt{w}}\Lambda(f'). \tag{21}$$

The first of these equations is quite intuitive, and implies that a filter with a broad impulse response will have better signal-to-noise ratio than a narrow filter when applied to a step edge. The second is less obvious, and it implies that a narrow filter will give better localization than a broad one. What is surprising is that the changes are inversely related, that is, both criteria either increase or decrease by $\sqrt{w}$. There is an uncertainty principle relating the detection and localization performance of the step edge detector. Through spatial scaling of f we can trade off detection performance against localization, but we cannot improve both simultaneously. This suggests that a natural choice for the composite criterion would be the product of (19) and (20), since this product would be invariant under changes in scale.

$$\Sigma(f)\,\Lambda(f') = \frac{\left|\int_{-W}^{0} f(x)\,dx\right|}{\sqrt{\int_{-W}^{+W} f^2(x)\,dx}} \frac{|f'(0)|}{\sqrt{\int_{-W}^{+W} f'^2(x)\,dx}}. \tag{22}$$

The solutions to the maximization of this expression will be a *class of functions* all related by spatial scaling. In fact this result is independent of the method of combination of the criteria. To see this we assume that there is a function f which gives the best localization Λ for a particular Σ. That is, we find f such that

$$\Sigma(f) = c_1 \quad \text{and} \quad \Lambda(f') \text{ is maximized.} \tag{23}$$

Now suppose we seek a second function f_w which gives the best possible localization while its signal-to-noise ratio is fixed to a different value, i.e.,

$$\Sigma(f_w) = c_2 \quad \text{while} \quad \Lambda(f'_w) \text{ is maximized.} \tag{24}$$

If we now define $f_1(x)$ in terms of $f_w(x)$ as $f_1(x) = f_w(xw)$ where

$$w = c_2^2/c_1^2$$

then the constraint on f_w in (24) translates to a constraint on f_1 which is identical to (23), and (24) can be rewritten as

$$\Sigma(f_1) = c_1 \quad \text{and} \quad \frac{1}{\sqrt{w}}\Lambda(f'_1) \text{ is maximized} \tag{25}$$

which has the solution $f_1 = f$. So if we find a single such function f, we can obtain maximal localization for any fixed signal-to-noise ratio by scaling f. The design problem for step edge detection has a *single unique* (up to spatial scaling) solution regardless of the absolute values of signal to noise ratio or localization.

The optimal filter is implicitly defined by (22), but we must transform the problem slightly before we can apply the calculus of variations. Specifically, we transform the maximization of (22) into a constrained minimization that involves only integral functionals. All but one of the integrals in (22) are set to undetermined constant values. We then find the extreme value of the remaining integral (since it will correspond to an extreme in the total expression) as a function of the undetermined constants. The values of the constants are then chosen so as to maximize the original expression, which is now a function only of these constants. Given the constants, we can uniquely specify the function $f(x)$ which gives a maximum of the composite criterion.

A second modification involves the limits of the integrals. The two integrals in the denominator of (22) have

limits at $+W$ and $-W$, while the integral in the numerator has one limit at 0 and the other at $-W$. Since the function f should be antisymmetric, we can use the latter limits for all integrals. The denominator integrals will have half the value over this subrange that they would have over the full range. Also, this enables the value of $f'(0)$ to be set as a boundary condition, rather than expressed as an integral of f''. If the integral to be minimized shares the same limits as the constraint integrals, it is possible to exploit the *isoperimetric constraint* condition (see [6, p. 216]). When this condition is fulfilled, the constrained optimization can be reduced to an unconstrained optimization using Lagrange multipliers for the constraint functionals. The problem of finding the maximum of (22) reduces to the minimization of the integral in the denominator of the SNR term, subject to the constraint that the other integrals remain constant. By the principle of reciprocity, we could have chosen to extremize any of the integrals while keeping the others constant, and the solution should be the same.

We seek some function f chosen from a space of *admissible* functions that minimizes the integral

$$\int_{-W}^{0} f^2(x)\,dx \tag{26}$$

subject to

$$\int_{-W}^{0} f(x)\,dx = c_1 \qquad \int_{-W}^{0} f'^2(x)\,dx = c_2$$

$$\int_{-W}^{0} f''^2(x)\,dx = c_3 \qquad f'(0) = c_4. \tag{27}$$

The space of admissible functions in this case will be the space of all continuous functions that satisfy certain boundary conditions, namely that $f(0) = 0$ and $f(-W) = 0$. These boundary conditions are necessary to ensure that the integrals evaluated over finite limits accurately represent the infinite convolution integrals. That is, if the nth derivative of f appears in some integral, the function must be continuous in its $(n-1)$st derivative over the range $(-\infty, +\infty)$. This implies that the values of f and its first $(n-1)$ derivatives must be zero at the limits of integration, since they are zero outside this range.

The functional to be minimized is of the form $\int_a^b F(x, f, f', f'')$ and we have a series of constraints that can be written in the form $\int_a^b G_i(x, f, f', f'') = c_i$. Since the constraints are isoperimetric, i.e., they share the same limits of integration as the integral being minimized, we can form a composite functional using Lagrange multipliers [6]. The functional is a linear combination of the functionals that appear in the expression to be minimized and in the constraints. Finding a solution to the unconstrained maximization of $\Psi(x, f, f', f'')$ is equivalent to finding the solution to the constrained problem. The composite functional is

$$\Psi(x, f, f', f'') = F(x, f, f', f'') + \lambda_1 G_1(x, f, f', f'') + \lambda_2 G_2(x, f, f', f'') + \cdots$$

Substituting,

$$\Psi(x, f, f', f'') = f^2 + \lambda_1 f'^2 + \lambda_2 f''^2 + \lambda_3 f. \tag{28}$$

It may be seen from the form of this equation that the choice of which integral is extremized and which are constraints is arbitrary, the solution will be the same. This is an example of the *reciprocity* that was mentioned earlier. The choice of an integral from the denominator is simply convenient since the standard form of variational problem is a minimization problem. The Euler equation that corresponds to the functional Ψ is

$$\Psi_f - \frac{d}{dx}\Psi_{f'} + \frac{d^2}{dx^2}\Psi_{f''} = 0 \tag{29}$$

where Ψ_f denotes the partial derivative of Ψ with respect to f, etc. We substitute for Ψ from (28) in the Euler equation giving:

$$2f(x) - 2\lambda_1 f''(x) + 2\lambda_2 f''''(x) + \lambda_3 = 0. \tag{30}$$

The solution of this differential equation is the sum of a constant and a set of four exponentials of the form $e^{\gamma x}$ where γ derives from the solution of the corresponding homogeneous differential equation. Now γ must satisfy

$$2 - 2\lambda_1\gamma^2 + 2\lambda_2\gamma^4 = 0$$

so

$$\gamma^2 = \frac{\lambda_1}{2\lambda_2} \pm \frac{\sqrt{\lambda_1^2 - 4\lambda_2}}{2\lambda_2}. \tag{31}$$

This equation may have roots that are purely imaginary, purely real, or complex depending on the values of λ_1 and λ_2. From the composite functional Ψ we can infer that λ_2 is positive (since the integral of f''^2 is to be minimized) but it is not clear what the sign or magnitude of λ_1 should be. The Euler equation supplies a necessary condition for the existence of a minimum, but it is not a sufficient condition. By formulating such a condition we can resolve the ambiguity in the value of λ_1. To do this we must consider the second variation of the functional. Let

$$J[f] = \int_{x0}^{x1} \Psi(x, f, f', f'')\,dx.$$

Then by Taylor's theorem (see also [6, p. 214]),

$$J[f + \epsilon g] = J[f] + \epsilon J_1[f, g] + \tfrac{1}{2}\epsilon^2 J_2[f + \rho g, g]$$

where ρ is some number between 0 and ϵ, and g is chosen from the space of admissible functions, and where

$$J_1[f, g] = \int_{x0}^{x1} \Psi_f g + \Psi_{f'} g' + \Psi_{f''} g''\,dx$$

$$J_2[f, g] = \int_{x0}^{x1} \Psi_{ff} g^2 + \Psi_{f'f'} g'^2 + \Psi_{f''f''} g''^2 + 2\Psi_{ff'} g g' + 2\Psi_{f'f''} g' g'' + 2\Psi_{ff''} g''\,dx. \tag{32}$$

Note that J_1 is nothing more than the integral of g times the Euler equation for f (transformed using integration by parts) and will be zero if f satisfies the Euler equation. We can now define the second variation $\delta^2 J$ as

$$\delta^2 J = \frac{\epsilon^2}{2} J_2[f, g].$$

The necessary condition for a minimum is $\delta^2 J \geq 0$. We compute the second partial derivatives of Ψ from (28) and we get

$$J_1[f + g] = \int_{x_0}^{x_1} 2g^2 + 2\lambda_1 g'^2 + 2\lambda_2 g''^2 \, dx \geq 0. \tag{33}$$

Using the fact that g is an admissible function and therefore vanishes at the integration limits, we transform the above using integration by parts to

$$2 \int_{x_0}^{x_1} g^2 - \lambda_1 g g'' + \lambda_2 g''^2 \, dx \geq 0$$

which can be written as

$$2 \int_{x_0}^{x_1} \left(g^2 - \frac{\lambda_1}{2} g'' \right)^2 + \left(\lambda_2 - \frac{\lambda_1^2}{4} \right) g''^2 \, dx \geq 0.$$

The integral is guaranteed to be positive if the expression being integrated is positive for all x, so if

$$4\lambda_2 > \lambda_1^2$$

then the integral will be positive for all x and for arbitrary g, and the extremum will certainly be a minimum. If we refer back to (31) we find that this condition is precisely that which gives complex roots for γ, so we have both guaranteed the existence of a minimum and resolved a possible ambiguity in the form of the solution. We can now proceed with the derivation and assume four complex roots of the form $\gamma = \pm\alpha \pm i\omega$ with α, ω real. Now $\gamma^2 = \alpha^2 - \omega^2 \pm 2i\alpha\omega$ and equating real and imaginary parts with (31) we obtain

$$\alpha^2 - \omega^2 = \frac{\lambda_1}{2\lambda_2} \quad \text{and} \quad 4\alpha^2\omega^2 = \frac{4\lambda_2 - \lambda_1^2}{4\lambda_2^2}. \tag{34}$$

The general solution in the range $[-W, 0]$ may now be written

$$f(x) = a_1 e^{\alpha x} \sin \omega x + a_2 e^{\alpha x} \cos \omega x + a_3 e^{-\alpha x} \cdot \sin \omega x + a_4 e^{-\alpha x} \cos \omega x + c. \tag{35}$$

This function is subject to the boundary conditions

$$f(0) = 0 \quad f(-W) = 0 \quad f'(0) = s \quad f'(-W) = 0$$

where s is an unknown constant equal to the slope of the function f at the origin. Since $f(x)$ is asymmetric, we can extend the above definition to the range $[-W, W]$ using $f(-x) = -f(x)$. The four boundary conditions enable us to solve for the quantities a_1 through a_4 in terms of the unknown constants α, ω, c, and s. The boundary conditions may be rewritten

$$\begin{aligned}
a_2 + a_4 + c &= 0 \\
a_1 e^{\alpha} \sin \omega + a_2 e^{\alpha} \cos \omega + a_3 e^{-\alpha} \sin \omega & \\
+ a_4 e^{-\alpha} \cos \omega + c &= 0 \\
a_1 \omega + a_2 \alpha + a_3 \omega - a_4 \alpha &= s \\
a_1 e^{\alpha}(\alpha \sin \omega + \omega \cos \omega) + a_2 e^{\alpha}(\alpha \cos \omega & \\
- \omega \sin \omega) + a_3 e^{-\alpha}(-\alpha \sin \omega + \omega \cos \omega) & \\
+ a_4 e^{-\alpha}(-\alpha \cos \omega - \omega \sin \omega) &= 0.
\end{aligned} \tag{36}$$

These equations are linear in the four unknowns a_1, a_2, a_3, a_4 and when solved they yield

$$\begin{aligned}
a_1 &= c(\alpha(\beta - \alpha) \sin 2\omega - \alpha\omega \cos 2\omega + (-2\omega^2 \sinh \alpha \\
&\quad + 2\alpha^2 e^{-\alpha}) \sin \omega + 2\alpha\omega \sinh \alpha \cos \omega \\
&\quad + \omega e^{-2\alpha}(\alpha + \beta) - \beta\omega)/4(\omega^2 \sinh^2 \alpha - \alpha^2 \sin^2 \omega) \\
a_2 &= c(\alpha(\beta - \alpha) \cos 2\omega + \alpha\omega \sin 2\omega - 2\alpha\omega \cosh \alpha \\
&\quad \cdot \sin \omega - 2\omega^2 \sinh \alpha \cos \omega + 2\omega^2 e^{-\alpha} \sinh \alpha \\
&\quad + \alpha(\alpha - \beta))/4(\omega^2 \sinh^2 \alpha - \alpha^2 \sin^2 \omega) \\
a_3 &= c(-\alpha(\beta + \alpha) \sin 2\omega + \alpha\omega \cos 2\omega + (2\omega^2 \sinh \alpha \\
&\quad + 2\alpha^2 e^{\alpha}) \sin \omega + 2\alpha\omega \sinh \alpha \cos \omega \\
&\quad + \omega e^{2\alpha}(\beta - \alpha) - \beta\omega)/4(\omega^2 \sinh^2 \alpha - \alpha^2 \sin^2 \omega) \\
a_4 &= c(-\alpha(\beta + \alpha) \cos 2\omega - \alpha\omega \sin 2\omega + 2\alpha\omega \cosh \alpha \\
&\quad \cdot \sin \omega + 2\omega^2 \sinh \alpha \cos \omega - 2\omega^2 e^{\alpha} \sinh \alpha \\
&\quad + \alpha(\alpha - \beta))/4(\omega^2 \sinh^2 \alpha - \alpha^2 \sin^2 \omega)
\end{aligned} \tag{37}$$

where β is the slope s at the origin divided by the constant c. On inspection of these expressions we can see that a_3 can be obtained from a_1 by replacing α by $-\alpha$, and similarly for a_4 from a_2.

The function f is now parametrized in terms of the constants α, ω, β, and c. We have still to find the values of these parameters which maximize the quotient of integrals that forms our composite criterion. To do this we first express each of the integrals in terms of the constants. Since these integrals are very long and uninteresting, they are not given here but may be found in [4]. We have reduced the problem of optimizing over an infinite-dimensional space of functions to a nonlinear optimization in three variables α, ω, and β (not surprisingly, the combined criterion does not depend on c). Unfortunately the resulting criterion, which must still satisfy the multiple response constraint, is probably too complex to be solved analytically, and numerical methods must be used to provide the final solution.

The shape of f will depend on the multiple response constraint, i.e., it will depend on how far apart we force the adjacent responses. Fig. 5 shows the operators that result from particular choices of this distance. Recall that there was no single best function for arbitrary ω, but a class of functions which were obtained by scaling a pro-

totype function by ω. We will want to force the responses further apart as the signal-to-noise ratio in the image is lowered, but it is not clear what the value of signal-to-noise ratio will be for a single operator. In the context in which this operator is used, several operator widths are available, and a decision procedure is applied to select the smallest operator that has an output signal-to-noise ratio above a fixed threshold. With this arrangement the operators will spend much of the time operating close to their output Σ thresholds. We try to choose a spacing for which the probability of a multiple response error is comparable to the probability of an error due to thresholding.

A rough estimate for the probability of a spurious maximum in the neighborhood of the true maximum can be formed as follows. If we look at the response of f to an ideal step we find that its second derivative has magnitude $|Af'(0)|$ at $x = 0$. There will be only one maximum near the center of the edge if $|Af'(0)|$ is greater than the second derivative of the response to noise only. This latter quantity, denoted s_n, is a Gaussian random variable with standard deviation

$$n_0\sigma_s = n_0\left(\int_{-W}^{+W} f''^2(x)\,dx\right)^{1/2}.$$

The probability p_m that the noise slope s_n exceeds $Af'(0)$ is given in terms of the normal distribution function Φ

$$p_m = 1 - \Phi\left(\frac{A|f'(0)|}{n_0\sigma_s}\right). \tag{38}$$

We can choose a value for this probability as an acceptable error rate and this will determine the ratio of $f'(0)$ to σ_s. We can relate the probability of a multiple response p_m to the probability of falsely marking an edge p_f which is

$$p_f = 1 - \Phi\left(\frac{A}{n_0}\Sigma\right) \tag{39}$$

by setting $p_m = p_f$. This is a natural choice since it makes a detection error or a multiple response error equally likely. Then from (38) and (39) we have

$$\frac{|f'(0)|}{\sigma_s} = \Sigma. \tag{40}$$

In practice it was impossible to find filters which satisfied this constraint, so instead we search for a filter satisfying

$$\frac{|f'(0)|}{\sigma_s} = r\Sigma \tag{41}$$

where r is as close as possible to 1. The performance indexes and parameter values for several filters are given in Fig. 4. The a_i coefficients for all these filters can be found from (37), by fixing c to, say, $c = 1$. Unfortunately, the largest value of r that could be obtained using the constrained numerical optimization was about 0.576 for filter number 6 in the table. In our implementation, we have approximated this filter using the first derivative of a Gaussian as described in the next section.

Filter Parameters						
n	x_{max}	$\Sigma\Lambda$	r	α	ω	β
1	0.15	4.21	0.215	24.59550	0.12250	63.97566
2	0.3	2.87	0.313	12.47120	0.38284	31.26860
3	0.5	2.13	0.417	7.85869	2.62856	18.28800
4	0.8	1.57	0.515	5.06500	2.56770	11.06100
5	1.0	1.33	0.561	3.45580	0.07161	4.80684
6	1.2	1.12	0.576	2.05220	1.56939	2.91540
7	1.4	0.75	0.484	0.00297	3.50350	7.47700

Fig. 4. Filter parameters and performance measures for the filters illustrated in Fig. 5.

The first derivative of Gaussian operator, or even filter 6 itself, should not be taken as the final word in edge detection filters, even with respect to the criteria we have used. If we are willing to tolerate a slight reduction in multiple response performance r, we can obtain significant improvements in the other two criteria. For example, filters 4 and 5 both have significantly better $\Sigma\Lambda$ product than filter 6, and only slightly lower r. From Fig. 5 we can see that these filters have steeper slope at the origin, suggesting that the performance gain is mostly in localization, although this has not been verified experimentally. A thorough empirical comparison of these other operators remains to be done, and the theory in this case is unclear on how best to make the tradeoff.

V. An Efficient Approximation

The operator derived in the last section as filter number 6, and illustrated in Fig. 6, can be approximated by the first derivative of a Gaussian $G'(x)$, where

$$G(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right).$$

The reason for doing this is that there are very efficient ways to compute the two-dimensional extension of the filter if it can be represented as some derivative of a Gaussian. This is described in detail elsewhere [4], but for the present we will compare the theoretical performance of a first derivative of a Gaussian filter to the optimal operator. The impulse response of the first derivative filter is

$$f(x) = -\frac{x}{\chi^2}\exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{42}$$

and the terms in the performance criteria have the values

$$|f'(0)| = \frac{1}{\sigma_s}$$

$$\int_{-\infty}^{0} f(x)\,dx = 1 \qquad \int_{-\infty}^{+\infty} f^2(x)\,dx = \frac{\sqrt{\pi}}{2\sigma}$$

$$\int_{-\infty}^{+\infty} f'^2(x)\,dx = \frac{3\sqrt{\pi}}{4\sigma^3} \qquad \int_{-\infty}^{+\infty} f''^2(x)\,dx = \frac{15\sqrt{\pi}}{8\sigma^5}. \tag{43}$$

The overall performance index for this operator is

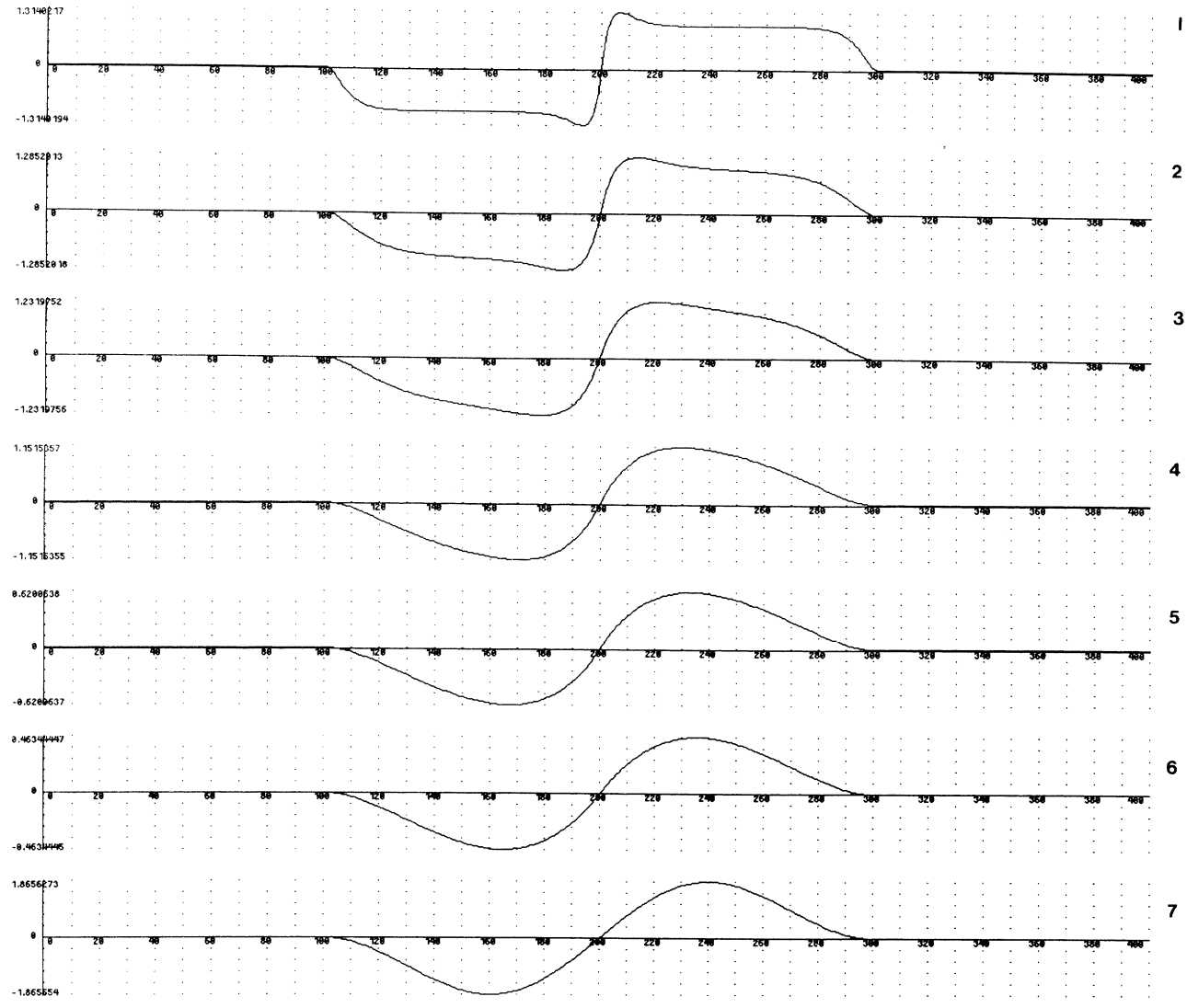


Fig. 5. Optimal step edge operators for various values of x_{max}. From top to bottom, they are $x_{max} = 0.15, 0.3, 0.5, 0.8, 1.0, 1.2, 1.4$.

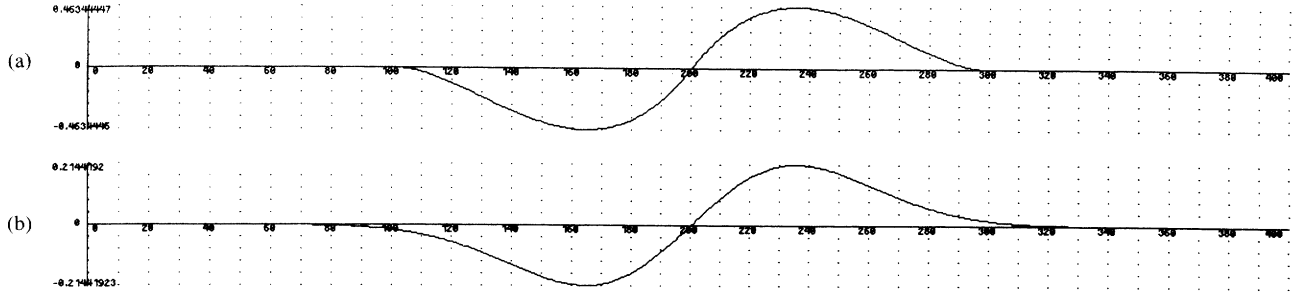


Fig. 6. (a) The optimal step edge operator. (b) The first derivative of a Gaussian.

$$\Sigma\Lambda = \sqrt{\frac{8}{3\pi}} \approx 0.92 \tag{44}$$

while the r value is, from (41),

$$r = \sqrt{\frac{4}{15}} \approx 0.51.$$

The performance of the first derivative of Gaussian operator above is worse than the optimal operator by about 20 percent and its multiple response measure r, is worse by about 10 percent. It would probably be difficult to detect a difference of this magnitude by looking at the performance of the two operators on real images, and because the first derivative of Gaussian operator can be computed with much less effort in two dimensions, it has

been used exclusively in experiments. The impulse responses of the two operators can be compared in Fig. 6.

A close approximation of the first derivative of Gaussian operator was suggested by Macleod [16] for step edge detection. Macleod's operator is a difference of two displaced two-dimensional Gaussians. It was evaluated in Fram and Deutsch [7] and compared very favorably with several other schemes considered in that paper. There are also strong links with the Laplacian of Gaussian operator suggested by Marr and Hildreth [18]. In fact, a one-dimensional Marr–Hildreth edge detector is almost identical with the operator we have derived because maxima in the output of a first derivative operator will correspond to zero-crossings in the Laplacian operator as used by Marr and Hildreth. In two dimensions however, the directional properties of our detector enhance its detection and localization performance compared to the Laplacian. Another important difference is that the amplitude of the response at a maximum provides a good estimate of edge strength, because the SNR criterion is the ratio of this response to the noise response. The Marr–Hildreth operator does not use any form of thresholding, but an adaptive thresholding scheme can be used to advantage with our first derivative operator. In the next section we describe such a scheme, which includes noise estimation and a novel method for thresholding edge points along contours.

We have derived our optimal operator to deal with known image features in Gaussian noise. Edge detection between textured regions is another important problem. This is straightforward if the texture can be modelled as the response of some filter $t(x)$ to Gaussian noise. We can then treat the texture as a noise signal, and the response of the filter $f(x)$ to the texture is the same as the response of the filter $(f * t)\ (x)$ to Gaussian noise. Making this replacement in each integral in the performance criteria that computes a noise response gives us the texture edge design problem. The generalization to other types of texture is not as easy, and for good discrimination between known texture types, a better approach would involve a Markov image model as in [5].

VI. Noise Estimation and Thresholding

To estimate noise from an operator output, we need to be able to separate its response to noise from the response due to step edges. Since the performance of the system will be critically dependent on the accuracy of this estimate, it should also be formulated as an optimization. Wiener filtering is a method for optimally estimating one component of a two-component signal, and can be used to advantage in this application. It requires knowledge of the autocorrelation functions of the two components and of the combined signal. Once the noise component has been optimally separated, we form a global histogram of noise amplitude, and estimate the noise strength from some fixed percentile of the noise signal.

Let $g_1(x)$ be the signal we are trying to detect (in this case the noise output), and $g_2(x)$ be some disturbance (paradoxically this will be the edge response of our filter), then denote the autocorrelation function of g_1 as $R_{11}(\tau)$ and that of g_2 as $R_{22}(\tau)$, and their cross-correlation as $R_{12}(\tau)$, where the correlation of two real functions is defined as follows:

$$R_{ij}(\tau) = \int_{-\infty}^{+\infty} g_i(x)\, g_j(x + \tau)\, dx.$$

We assume in this case that the signal and disturbance are uncorrelated, so $R_{12}(\tau) = 0$. The optimal filter is $K(x)$, which is implicitly defined as follows [30]:

$$R_{11}(\tau) = \int_{-\infty}^{+\infty} (R_{11}(\tau - x) + R_{22}(\tau - x))\, K(x)\, dx.$$

Since the autocorrelation of the output of a filter in response to white noise is equal to the autocorrelation of its impulse response, we have

$$R_{11}(x) = k_3\left(\frac{x^2}{2\sigma^2} - 1\right) \exp\left(-\frac{x^2}{4\sigma^2}\right)$$

If g_2 is the response of the operator derived in (42) to a step edge then we will have $g_2(x) = k \exp(-x/2\sigma^2)$ and

$$R_{22}(x) = k_2 \exp\left(-\frac{x^2}{4\sigma^2}\right).$$

In the case where the amplitude of the edge is large compared to the noise, $R_{22} + R_{11}$ is approximately a Gaussian and R_{11} is the second derivative of a Gaussian of the same σ. Then the optimal form of K is the second derivative of an impulse function.

The filter K above is convolved with the output of the edge detection operator and the result is squared. The next step is the estimation of the mean-squared noise from the local values. Here there are several possibilities. The simplest is to average the squared values over some neighborhood, either using a moving average filter or by taking an average over the entire image. Unfortunately, experience has shown that the filter K is very sensitive to step edges, and that as a consequence the noise estimate from any form of averaging is heavily colored by the density and strength of edges.

In order to gain better separation between signal and noise we can make use of the fact that the amplitude distribution of the filter response tends to be different for edges and noise. By our model, the noise response should have a Gaussian distribution, while the step edge response will be composed of large values occurring very infrequently. If we take a histogram of the filter values, we should find that the positions of the low percentiles (say less than 80 percent) will be determined mainly the noise energy, and that they are therefore useful estimators for noise. A global histogram estimate is actually used in the current implementation of the algorithm.

Even with noise estimation, the edge detector will be susceptible to streaking if it uses only a single threshold. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the

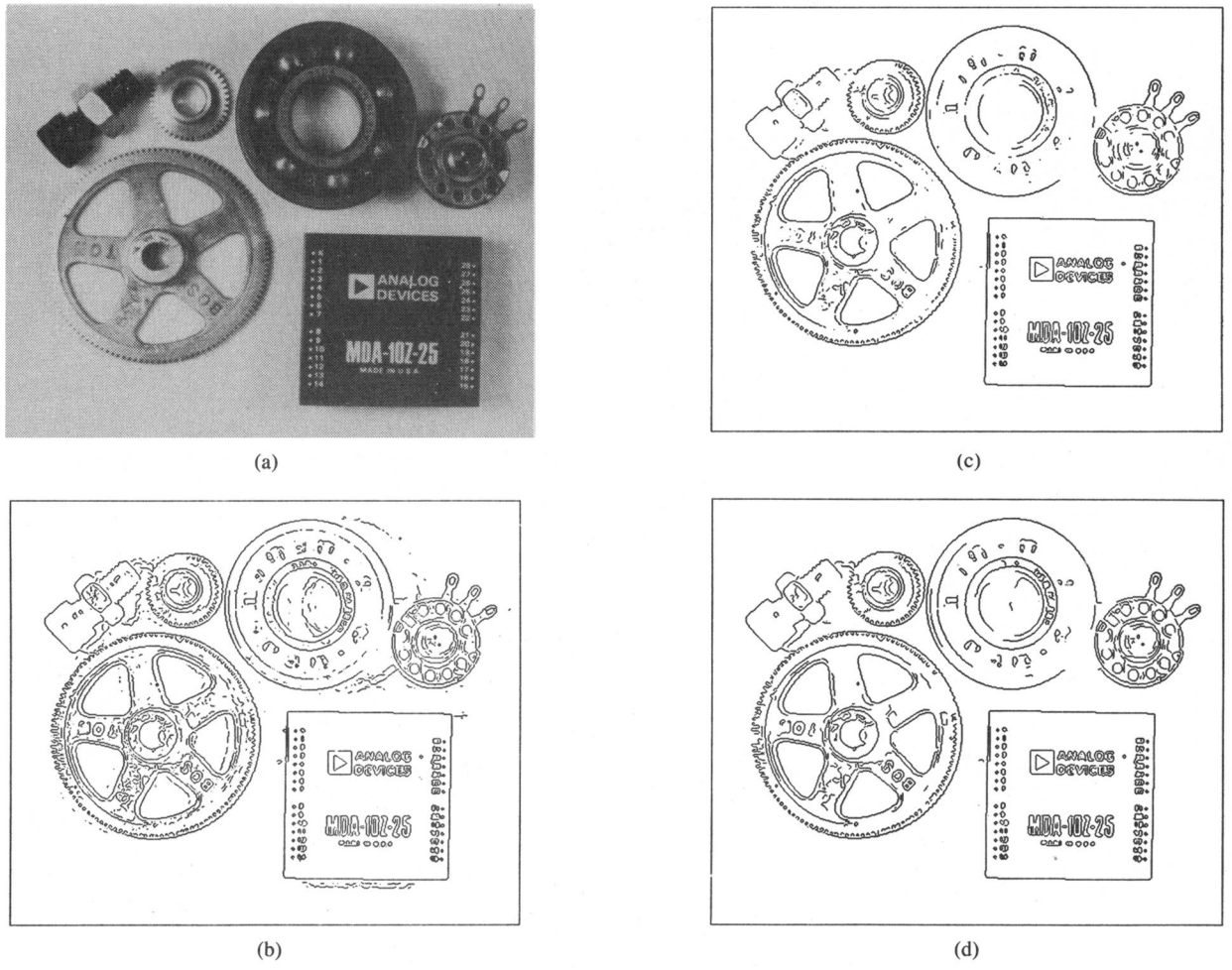


Fig. 7. (a) Parts image, 576 by 454 pixels. (b) Image thesholded at T_1. (c) Image thresholded at 2 T_1. (d) Image thresholded with hysteresis using both the thresholds in (a) and (b).

threshold along the length of the contour. Suppose we have a single threshold set at T_1, and that there is an edge in the image such that the response of the operator has mean value T_1. There will be some fluctuation of the output amplitude due to noise, even if the noise is very slight. We expect the contour to be above threshold only about half the time. This leads to a broken edge contour. While this is a pathological case, streaking is a very common problem with edge detectors that employ thresholding. It is very difficult to set a threshold so that there is small probability of marking noise edges while retaining high sensitivity. An example of the effect of streaking is given in Fig. 7.

One possible solution to this problem, used by Pentland [22] with Marr–Hildreth zero-crossings, is to average the edge strength of a contour over part of its length. If the average is above the threshold, the entire segment is marked. If the average is below threshold, no part of the contour appears in the output. The contour is segmented by breaking it at maxima in curvature. This segmentation is necessary in the case of zero-crossings since the zero-crossings always form closed contours, which obviously do not always correspond to contours in the image.

In the current algorithm, no attempt is made to presegment contours. Instead the thresholding is done with hysteresis. If any part of a contour is above a high threshold, those points are immediately output, as is the entire connected segment of contour which contains the points and which lies above a low threshold. The probability of streaking is greatly reduced because for a contour to be broken it must now fluctuate above the high threshold and below the low threshold. Also the probability of isolated false edge points is reduced because the strength of such points must be above a higher threshold. The ratio of the high to low threshold in the implementation is in the range two or three to one.

VII. Two or More Dimensions

In one dimension we can characterize the position of a step edge in space with one position coordinate. In two dimensions an edge also has an orientation. In this section we will use the term "edge direction" to mean the direction of the tangent to the contour that the edge defines in two dimensions. Suppose we wish to detect edges of a particular orientation. We create a two-dimensional mask for this orientation by convolving a linear edge detection

function aligned normal to the edge direction with a projection function parallel to the edge direction. A substantial savings in computational effort is possible if the projection function is a Gaussian with the same σ as the (first derivative of the) Gaussian used as the detection function. It is possible to create such masks by convolving the image with a symmetric two-dimensional Gaussian and then differentiating normal to the edge direction. In fact we do not have to do this in every direction because the slope of a smooth surface in any direction can be determined exactly from its slope in two directions. This form of directional operator, while simple and inexpensive to compute, forms the heart of the more elaborate detector which will be described in the next few sections.

Suppose we wish to convolve the image with an operator G_n which is the first derivative of a two-dimensional Gaussian G in some direction $\boldsymbol{n}$, i.e.,

$$G = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

and

$$G_n = \frac{\partial G}{\partial \boldsymbol{n}} = \boldsymbol{n} \cdot \nabla G. \tag{45}$$

Ideally, $\boldsymbol{n}$ should be oriented normal to the direction of an edge to be detected, and although this direction is not known *a priori*, we can form a good estimate of it from the smoothed gradient direction

$$\boldsymbol{n} = \frac{\nabla(G * I)}{|\nabla(G * I)|} \tag{46}$$

where $*$ denotes convolution. This turns out to be a very good estimator for edge normal direction for steps, since a smoothed step has strong gradient normal to the edge. It is exact for straight line edges in the absence of noise, and the Gaussian smoothing keeps it relatively insensitive to noise.

An edge point is defined to be a local maximum (in the direction $\boldsymbol{n}$) of the operator G_n applied to the image I. At a local maximum, we have

$$\frac{\partial}{\partial \boldsymbol{n}} G_n * I = 0$$

and substituting for G_n from (45) and associating Gaussian convolution, the above becomes

$$\frac{\partial^2}{\partial \boldsymbol{n}^2} G * I = 0. \tag{47}$$

At such an edge point, the edge strength will be the magnitude of

$$|G_n * I| = |\nabla(G * I)|. \tag{48}$$

Because of the associativity of convolution, we can first convolve with a symmetric Gaussian G and then compute directional second derivative zeros to locate edges (47), and use the magnitude of (48) to estimate edge strength. This is equivalent to detecting and locating the edge using the directional operator G_n, but we need not know the direction $\boldsymbol{n}$ before convolution.

The form of nonlinear second derivative operator in (47) has also been used by Torre and Poggio [28] and by Haralick [10]. It also appears in Prewitt [23] in the context of edge enhancement. A rather different two-dimensional extension is proposed by Spacek [26] who uses one-dimensional filters aligned normal to the edge direction but without extending them along the edge direction. Spacek starts with a one-dimensional formulation which maximizes the product of the three performance criteria defined in Section II, and leads to a step edge operator which differs slightly from the one we derived in Section IV. Gennert [8] addresses the two-dimensional edge detector problem directly, and applies a set of directional first derivative operators at each point in the image. The operators have limited extent along the edge direction and produce good results at sharp changes in edge orientation and corners.

The operator (47) actually locates either maxima or minima by locating the zero-crossings in the second derivative in the edge direction. In principle it could be used to implement an edge detector in an arbitrary number of dimensions, by first convolving the image with a symmetric n-dimensional Gaussian. The convolution with an n-dimensional Gaussian is highly efficient because the Gaussian is separable into n one-dimensional filters.

But there are other more pressing reasons for using a smooth projection function such as a Gaussian. When we apply a linear operator to a two-dimensional image, we form at every point in the output a weighted sum of some of the input values. For the edge detector described here, this sum will be a difference between local averages on different sides of the edge. This output, before nonmaximum suppression, represents a kind of moving average of the image. Ideally we would like to use an infinite projection function, but real edges are of limited extent. It is therefore necessary to window the projection function [9]. If the window function is abruptly truncated, e.g., if it is rectangular, the filtered image will not be smooth because of the very high bandwidth of this window. This effect is related to the Gibbs phenomenon in Fourier theory which occurs when a signal is transformed over a finite window. When nonmaximum suppression is applied to this rough signal we find that edge contours tend to "wander" or that in severe cases they are not even continuous.

The solution is to use a smooth window function. In statistics, the Hamming and Hanning windows are typically used for moving averages. The Gaussian is a reasonable approximation to both of these, and it certainly has very low bandwidth for a given spatial width. (The Gaussian is the unique function with minimal product of bandwidth and frequency.) The effect of the window function becomes very marked for large operator sizes and it is probably the biggest single reason why operators with large support were not practical until the work of Marr and Hildreth on the Laplacian of Gaussian.

It is worthwhile here to compare the performance of

this kind of directional second derivative operator with the Laplacian. First we note that the two-dimensional Laplacian can be decomposed into components of second derivative in two arbitrary orthogonal directions. If we choose to take one of the derivatives in the direction of principal gradient, we find that the operator output will contain one contribution that is essentially the same as the operator described above, and also a contribution that is aligned along the edge direction. This second component contributes nothing to localization or detection (the surface is roughly constant in this direction), but increases the output noise.

In later sections we will describe an edge detector which incorporates operators of varying orientation and aspect ratio, but these are a superset of the operators used in the simple detector described above. In typical images, most of the edges are marked by the operators of the smallest width, and most of these by nonelongated operators. The simple detector performs well enough in these cases, and as detector complexity increases, performance gains tend to diminish. However, as we shall see in the following sections, there are cases when larger or more directional operators should be used, and that they do improve performance when they are applicable. The key to making such a complicated detector produce a coherent output is to design effective decision procedures for choosing between operator outputs at each point in the image.

VIII. The Need for Multiple Widths

Having determined the optimal shape for the operator, we now face the problem of choosing the width of the operator so as to give the best detection/localization tradeoff in a particular application. In general the signal-to-noise ratio will be different for each edge within an image, and so it will be necessary to incorporate several widths of operator in the scheme. The decision as to which operator to use must be made dynamically by the algorithm and this requires a local estimate of the noise energy in the region surrounding the candidate edge. Once the noise energy is known, the signal-to-noise ratios of each of the operators will be known. If we then use a model of the probability distribution of the noise, we can effectively calculate the probability of a candidate edge being a false edge (for a given edge, this probability will be different for different operator widths).

If we assume that the *a priori* penalty associated with a falsely detected edge is independent of the edge strength, it is appropriate to threshold the detector outputs on probability of error rather than on magnitude of response. Once the probability threshold is set, the minimum acceptable signal-to-noise ratio is determined. However, there may be several operators with signal-to-noise ratios above the threshold, and in this case the smallest operator should be chosen, since it gives the best localization. We can afford to be conservative in the setting of the threshold since edges missed by the smallest operators may be picked up by the larger ones. Effectively the global tradeoff between error rate and localization remains, since choosing a high signal-to-noise ratio threshold leads to a lower error rate, but will tend to give poorer localization since fewer edges will be recorded from the smaller operators.

In summary then, the first heuristic for choosing between operator outputs is that *small operator widths should be used whenever they have sufficient* Σ. This is similar to the selection criterion proposed by Marr and Hildreth [18] for choosing between different Laplacian of Gaussian channels. In their case the argument was based on the observation that the smaller channels have higher resolution, i.e., there is less possibility of interference from neighboring edges. That argument is also very relevant in the present context, as to date there has been no consideration of the possibility of more than one edge in a given operator support. Interestingly, Rosenfeld and Thurston [25] proposed exactly the opposite criterion in the choice of operator for edge detection in texture. The argument given was that the larger operators give better averaging and therefore (presumably) better signal-to-noise ratios.

Taking the fine-to-coarse heuristic as a starting point, we need to form a local decision procedure that will enable us to decide whether to mark one or more edges when several operators in a neighborhood are responding. If the operator with the smallest width responds to an edge and if it has a signal-to-noise ratio above the threshold, we should immediately mark an edge at that point. We now face the problem that there will almost certainly be edges marked by the larger operators, but that these edges will probably not be exactly coincident with the first edge. A possible answer to this would be to suppress the outputs of all nearby operators. This has the undesirable effect of preventing the large channels for responding to "fuzzy" edges that are superimposed on the sharp edge.

Instead we use a "feature synthesis" approach. We begin by marking all the edges from the smallest operators. From these edges, we synthesize the large operator outputs than would have been produced if these were the only edges in the image. We then compare the actual operator outputs to the synthetic outputs. We mark additional edges only if the large operator has significantly greater response than what we would predict from the synthetic output. The simplest way to produce the synthetic outputs is to take the edges marked by a small operator in a particular direction, and convolve with a Gaussian normal to the edge direction for this operator. The σ of this Gaussian should be the same as the σ of the large channel detection filter.

This procedure can be applied repeatedly to first mark the edges from the second smallest scale that were not marked by at the first, and then to find the edges from the third scale that were not marked by either of the first two, etc. Thus we build up a cumulative edge map by adding those edges at each scale that were not marked by smaller scales. It turns out that in many cases the majority of edges are picked up by the smallest channel, and the later channels mark mostly shadow and shading edges, or edges between textured regions.

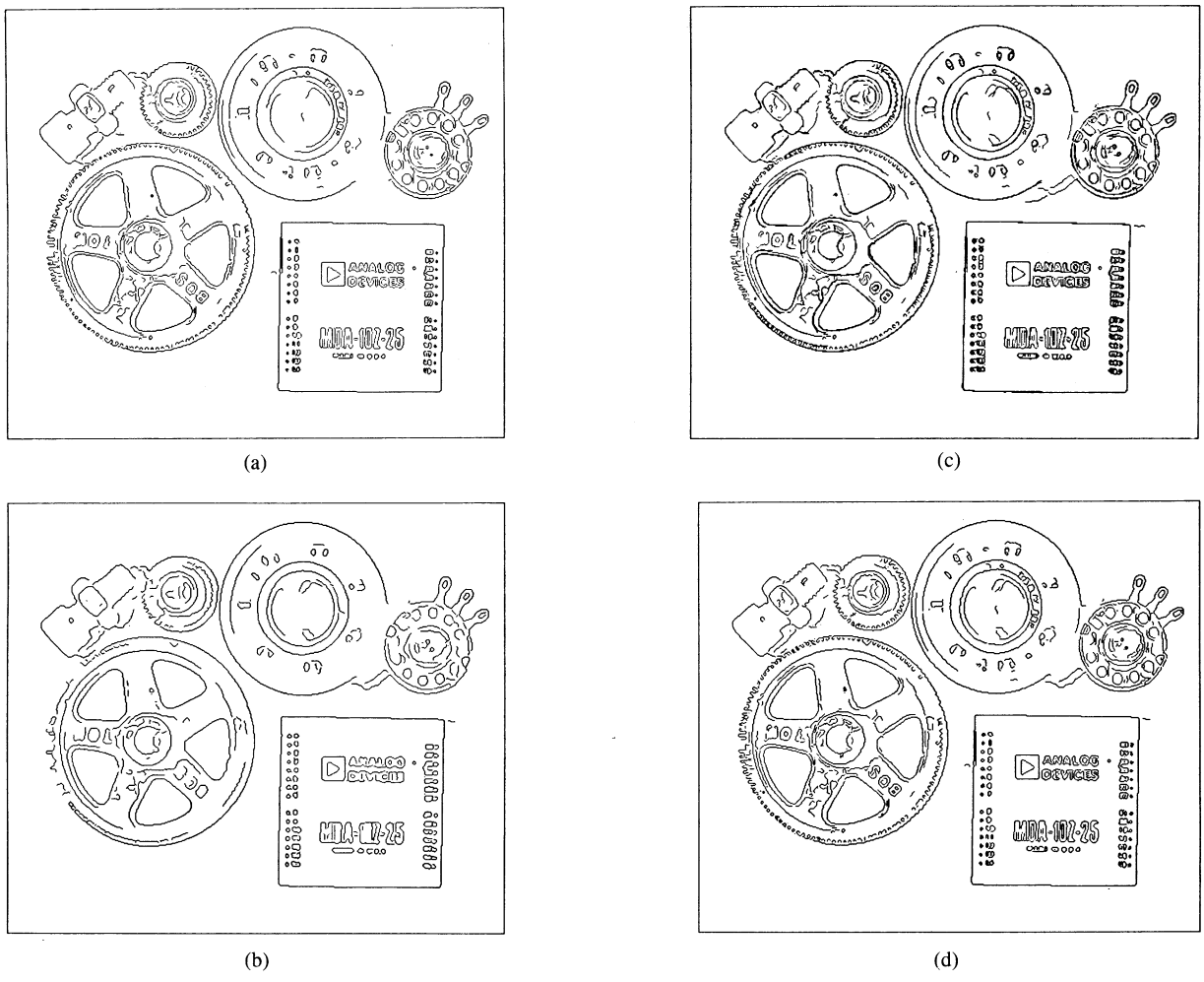


Fig. 8. (a) Edges from parts image at $\sigma = 1.0$. (b) Edges at $\sigma = 2.0$. (c) Superposition of the edges. (d) Edges combined using feature synthesis.

Some examples of feature synthesis applied to some sample images are shown in Figs. 8 and 9. Notice that most of the edges in Fig. 8 are marked by the smaller scale operator, and only a few additional edges, mostly shadows, are picked up by the coarser scale. However when the two sets of edges are superimposed, we notice that in many cases the responses of the two operators to the same edge are not spatially coincident. When feature synthesis is applied we find that redundant responses of the larger operator are eliminated leading to a sharp edge map.

By contrast, in Fig. 9 the edges marked by the two operators are essentially independent, and direct superposition of the edges gives a useful edge map. When we apply feature synthesis to these sets of edges we find that most of the edges at the coarser scale remain. Both Figs. 8 and 9 were produced by the edge detector with exactly the same set of parameters (other than operator size), and they were chosen to represent opposing extremes of image content across scale.

IX. The Need for Directional Operators

So far we have assumed that the projection function is a Gaussian with the same σ as the Gaussian used for the detection function. In fact both the detection and localization of the operator improve as the length of the projection function increases. We now prove this for the operator signal-to-noise ratio. The proof for localization is similar. We will consider a step edge in the x direction which passes through the origin. This edge can be represented by the equation

$$I(x, y) = Au_{-1}(y)$$

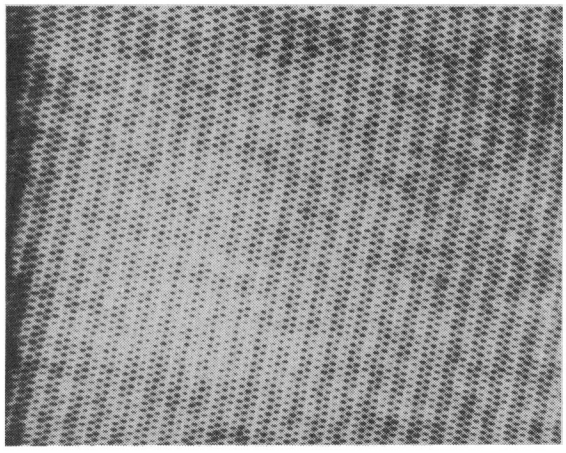
where u_{-1} is the unit step function, and A is the amplitude of the edge as before. Suppose that there is additive Gaussian noise of mean squared value n_{00}^2 per unit area. If we convolve this signal with a filter whose impluse response is $f(x, y)$, then the response to the edge (at the origin) is

$$\int_{-\infty}^{0} \int_{-\infty}^{+\infty} f(x, y)\, dx\, dy.$$

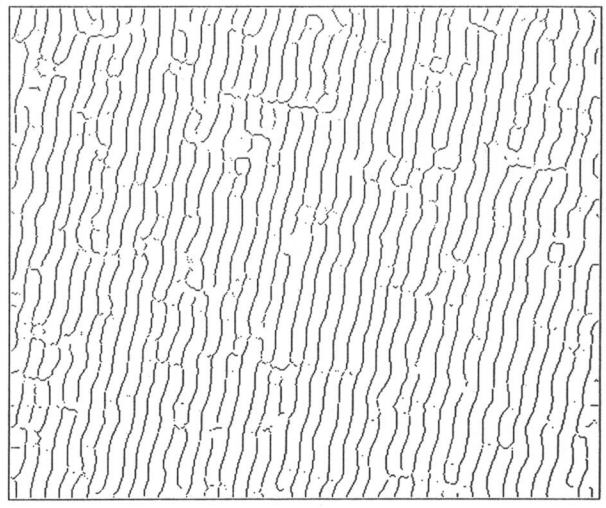
The root mean squared response to the noise only is

$$n_{00} \left(\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f^2(x, y)\, dx\, dy \right)^{1/2}$$

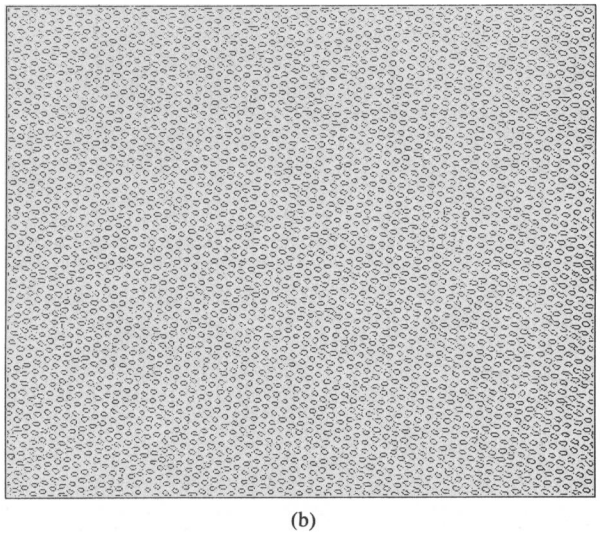
The signal-to-noise ratio is the quotient of these two



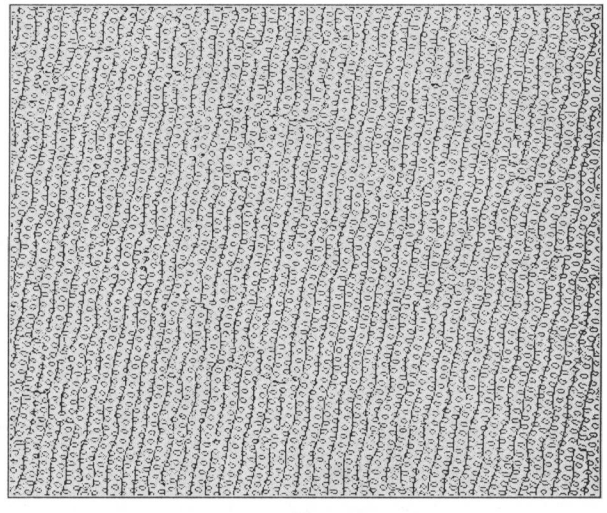
(a)



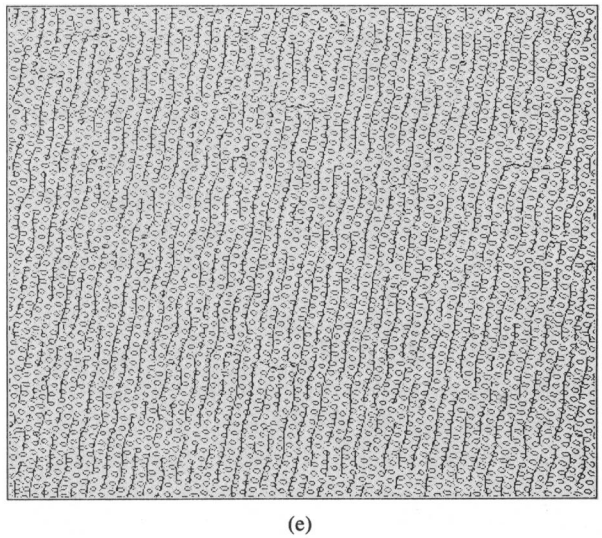
(c)



(b)



(d)



(e)

Fig. 9. (a) Handywipe image 576 by 454 pixels. (b) Edges from handywipe image at $\sigma = 1.0$. (c) $\sigma = 5.0$. (d) Superposition of the edges. (e) Edges combined using feature synthesis.

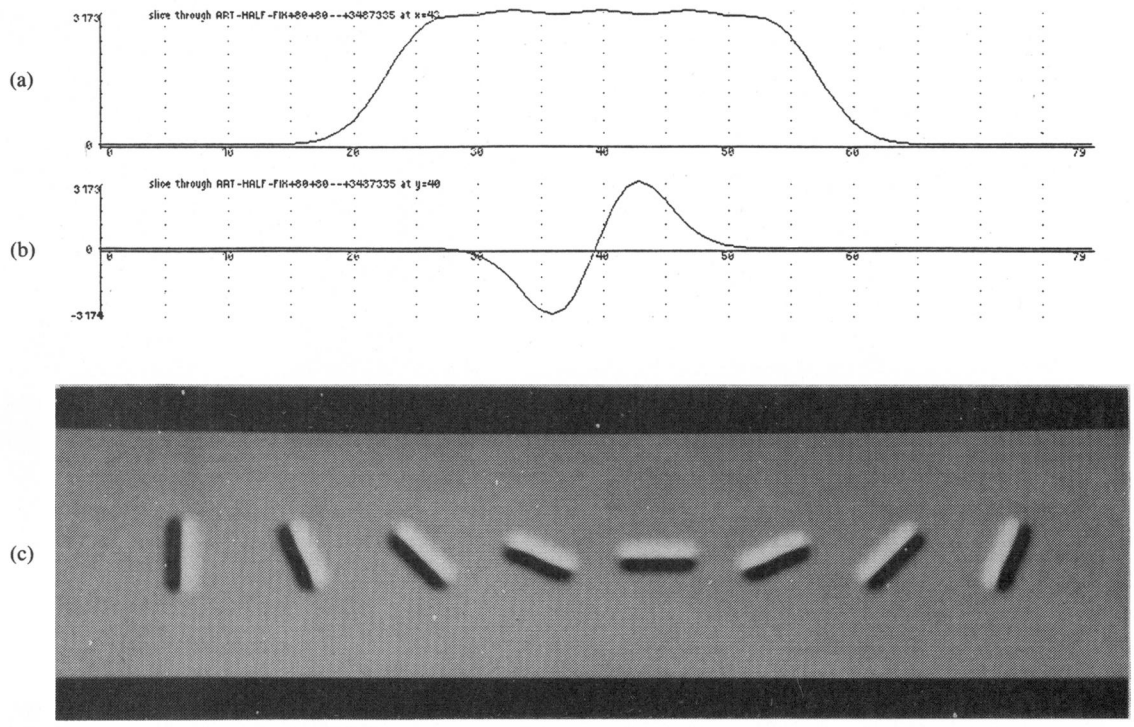


Fig. 10. Directional step edge mask. (a) Cross section parallel to the edge direction. (b) Cross section normal to edge direction. (c) Two-dimensional impulse responses of several masks.

integrals, and will be denoted by Σ. We have already seen what happens if we scale the function normal to the edge (21). We now do the same to the projection function by replacing $f(x, y)$ by $f_l(x, y) = f(x, (y/l))$. The integrals become

$$\int_{-\infty}^{0}\int_{-\infty}^{+\infty} f\left(x, \frac{y}{l}\right) dx\, dy = \int_{-\infty}^{0}\int_{-\infty}^{+\infty} f(x, y_1) l\, dx\, dy_1$$

$$n_{00}\left(\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f^2\left(x, \frac{y}{l}\right) dx\, dy\right)^{1/2} = n_{00}\left(\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f^2(x, y_1) l\, dx\, dy_1\right)^{1/2} \quad (49)$$

And the ratio of the two is now $\sqrt{l}\Sigma$. The localization Λ also improves as $\sqrt{l}$. It is clearly desirable that we use as large a projection function as possible. There are practical limitations on this however, in particular edges in an image are of limited extent, and few are perfectly linear. However, most edges continue for some distance, in fact much further than the 3 or 4 pixel supports of most edge operators. Even curved edges can be approximated by linear segments at a small enough scale. Considering the advantages, it is obviously preferable to use directional operators whenever they are applicable. The only proviso is that the detection scheme must ensure that they are used only when the image fits a linear edge model.

The present algorithm tests for applicability of each directional mask by forming a goodness-of-fit estimate. It does this at the same time as the mask itself is computed. An efficient way of forming long directional masks is to sample the output of nonelongated masks with the same direction. This output is sampled at regular intervals in a line parallel to the edge direction. If the samples are close together (less than 2σ apart), the resulting mask is essentially flat over most of its range in the edge direction and falls smoothly off to zero at its ends. Two cross sections of such a mask are shown in Fig. 10. In this diagram (as in the present implementation) there are five samples over the operator support.

Simultaneously with the computation of the mask, it is possible to establish goodness of fit by a simple squared-error measure. The mask is computed by summing some number of circular mask outputs (say 5) in a line. If the mask lies over a step edge in its preferred direction, these 5 values will be roughly the same. If the edge is curved or not aligned with the mask direction, the values will vary. We use the variance of these values as an estimate of the goodness of fit of the actual edge to an ideal step model. We then suppress the output of a directional mask if its variance is greater than some fraction of the squared output. Where no directional operator has sufficient goodness of fit at a point, the algorithm will use the output of the nonelongated operator described in Section VII. This simple goodness-of-fit measure is sufficient to eliminate the problems that traditionally plague directional operators, such as false responses to highly curved edges and extension of edges beyond corners; see Hildreth [12].

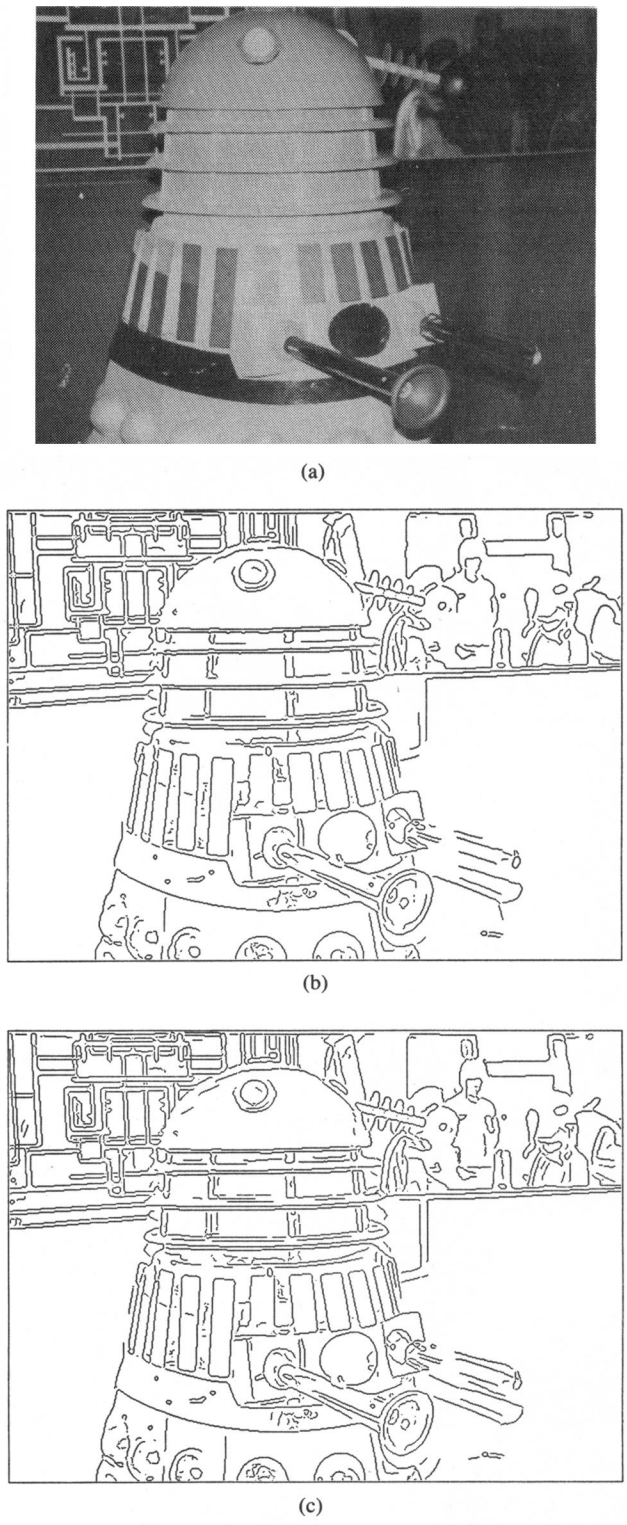
This particular form of projection function, that is a

function with constant value over some range which decays to zero at each end with two roughly half-Gaussians, is very similar to a commonly used extension of the Hanning window. This latter function is flat for some distance and decays to zero at each end with two half-cosine bells [2]. We can therefore expect our function to have good properties as a moving average estimator, which as we saw in Section VII, is an important role fulfilled by the projection function.

All that remains to be done in the design of directional operators is the specification of the number of directions, or equivalently the angle between two adjacent directions. To determine the latter, we need to determine the angular selectivity of a directional operator as a function of the angle θ between the edge direction and the preferred direction of the operator. Assume that we form the operator by taking an odd number $2N + 1$ of samples. Let the number of a sample be n where n is in the range $-N \cdots +N$. Recall that the directional operator is formed by convolving with a symmetric Gaussian, differentiating normal to the preferred edge direction of the operator, and then sampling along the preferred direction. The differentiated surface will be a ridge which makes an angle θ to the preferred edge direction. Its height will vary as cos θ, and the distance of the nth sample from the center of the ridge will be $nd \sin \theta$ where d is the distance between samples. The normalized output will be

$$O_n(\theta) = \frac{\cos \theta}{2N + 1} \left[\sum_{n=-N}^{N} \exp \left(-\frac{(nd \sin \theta)^2}{2\sigma^2} \right) \right]. \quad (50)$$

If there are m operator directions, then the angle between the preferred directions of two adjacent operators will be $180/m$. The worst case angle between the edge and the nearest preferred operator direction is therefore $90/m$. In the current implementation the value of d/σ is about 1.4 and there are 6 operator directions. The worst case for θ is 15 degrees, and for this case the operator output will fall to about 85 percent of its maximum value. Directional operators very much like the ones we have derived were suggested by Marr [17], but were discarded in favor of the Laplacian of Gaussian [18]. In part this was because the computation of several directional operators at each point in the image was thought to require an excessive amount of computation. In fact the sampling scheme described above requires only five multiplications per operator. An example of edge detection using five-point directional operators is given in Fig. 11.



(a) (b) (c)

Fig. 11. (a) Dalek image 576 by 454 pixels. (b) Edges found using circular operator. (c) Directional edges (6 mask orientations).

X. Conclusions

We have described a procedure for the design of edge detectors for arbitrary edge profiles. The design was based on the specification of detection and localization criteria in a mathematical form. It was necessary to augment the original two criteria with a multiple response measure in order to fully capture the intuition of good detection. A mathematical form for the criteria was presented, and numerical optimization was used to find optimal operators for roof and ridge edges. The analysis was then restricted to consideration of optimal operators for step edges. The result was a class of operators related by spatial scaling. There was a direct tradeoff in detection performance versus localization, and this was determined by the spatial

width. The impulse response of the optimal step edge operator was shown to approximate the first derivative of a Gaussian.

A detector was proposed which used adaptive thresholding with hysteresis to eliminate streaking of edge contours. The thresholds were set according to the amount of noise in the image, as determined by a noise estimation scheme. This detector made use of several operator widths to cope with varying image signal-to-noise ratios, and operator outputs were combined using a method called feature synthesis, where the responses of the smaller operators were used to predict the large operator responses. If the actual large operator outputs differ significantly from the predicted values, new edge points are marked. It is therefore possible to describe edges that occur at different scales, even if they are spatially coincident.

In two dimensions it was shown that marking edge points at maxima of gradient magnitude in the gradient direction is equivalent to finding zero-crossings of a certain nonlinear differential operator. It was shown that when edge contours are locally straight, highly directional operators will give better results than operators with a circular support. A method was proposed for the efficient generation of highly directional masks at several orientations, and their integration into a single description.

Among the possible extensions of the work, the most interesting unsolved problem is the integration of different edge detector outputs into a single description. A scheme which combined the edge and ridge detector outputs using feature synthesis was implemented, but the results were inconclusive. The problem is much more complicated here than for edge operators at different scales because there is no clear reason to prefer one edge type over another. Each edge set must be synthesized from the other, without a bias caused by overestimation in one direction.

The criteria we have presented can be used with slight modification for the design of other kinds of operator. For example, we may wish to design detectors for nonlinear two-dimensional features (such as corners). In this case the detection criterion would be a two-dimensional integral similar to (3), while a plausible localization criterion would need to take into account the variation of the edge position in both the x and y directions, and would not directly generalize from (9). There is a natural generalization to the detection of higher-dimensional edges, such as occur at material boundaries in tomographic scans. As was pointed out in Section VII, (47) can be used to find edges in images of arbitrary dimension, and the algorithm remains efficient in higher dimensions because n-dimensional Gaussian convolution can be broken down into n linear convolutions.

Acknowledgment

The author would like to thank Dr. J. M. Brady for his influence on the course of this work and for comments on early drafts of this paper. Thanks to the referees for their suggestions which have greatly improved the presentation of the paper. In particular thanks to the referee who suggested the simple derivation based on the Schwarz inequality that appears on p. 682.

References

[1] R. J. Beattie, "Edge detection for semantically based early visual processing," Ph.D. dissertation, Univ. Edinburgh, 1984.

[2] C. Bingham, M. D. Godfrey, and J. W. Tukey, "Modern techniques of power spectrum estimation," *IEEE Trans. Audio Electroacoust.*, vol. AU-15, no. 2, pp. 56–66, 1967.

[3] R. A. Brooks, "Symbolic reasoning among 3-D models and 2-D images," Dep. Comput. Sci., Stanford Univ., Stanford, CA, Rep. AIM-343, 1981.

[4] J. F. Canny, "Finding edges and lines in images," M.I.T. Artificial Intell. Lab., Cambridge, MA, Rep. AI-TR-720, 1983.

[5] F. S. Cohen, D. B. Cooper, J. F. Silverman, and E. B. Hinkle, "Simple parallel hierarchical and relaxation algorithms for segmenting textured images based on noncasual Markovian random field models," in *Proc. 7th Int. Conf. Pattern Recognition and Image Processing*, Canada, 1984.

[6] R. Courant and D. Hilbert, *Methods of Mathematical Physics*, vol. 1. New York: Wiley-Interscience, 1953.

[7] J. R. Fram and E. S. Deutsch, "On the quantitative evaluation of edge detection schemes and their comparison with human performance," *IEEE Trans. Comput.*, vol. C-24, no. 6, pp. 616–628, 1975.

[8] M. Gennert, "Detecting half-edges and vertices in images," in *IEEE Conf. Comput. Vision and Pattern Recognition*, Miami Beach, FL, June 24–26, 1986.

[9] R. W. Hamming, *Digital Filters*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[10] R. M. Haralick, "Zero-crossings of second directional derivative edge operator," in *SPIE Proc. Robot Vision*, Arlington, VA, 1982.

[11] A. Herskovits and T. O. Binford, "On boundary detection," M.I.T. Artificial Intell. Lab., Cambridge, MA, AI Memo 183, 1970.

[12] E. C. Hildreth, "Implementation of a theory of edge detection," M.I.T. Artificial Intell. Lab., Cambridge, MA, Rep. AI-TR-579, 1980.

[13] ——, *The Measurement of Visual Motion*. Cambridge, MA: M.I.T. Press, 1983.

[14] B. K. P. Horn, "The Binford-Horn line-finder," M.I.T. Artificial Intell. Lab., Cambridge, MA, AI Memo 285, 1971.

[15] D. G. Luenberger, *Introduction to Linear and Non-Linear Programming*. Reading, MA: Addison-Wesley, 1973.

[16] I. D. G. Macleod, "On finding structure in pictures," in *Picture Language Machines*, S. Kaneff, Ed. New York: Academic, 1970, p. 231.

[17] D. C. Marr, "Early processing of visual information," *Phil. Trans. Roy. Soc. London*, vol. B 275, pp. 483–524, 1976.

[18] D. C. Marr and E. Hildreth, "Theory of edge detection," *Proc. Roy. Soc. London.*, vol. B 207, pp. 187–217, 1980.

[19] D. C. Marr and T. Poggio, "A theory of human stereo vision," *Proc. Roy. Soc. London.*, vol. B 204, pp. 301–328, 1979.

[20] J. E. W. Mayhew and J. P. Frisby, "Psychophysical and computational studies toward a theory of human stereopsis," *Artificial Intell. (Special Issue on Computer Vision)*, vol. 17, 1981.

[21] T. Poggio, H. Voorhees, and A. Yuille, "A regularized solution to edge detection," M.I.T. Artificial Intell. Lab., Cambridge, MA, Rep. AIM-833, 1985.

[22] A. P. Pentland, "Visual inference of shape: Computation from local features," Ph.D. dissertation, Dep. Psychol., Massachusetts Inst. Technol., Cambridge, MA, 1982.

[23] J. M. S. Prewitt, "Object enhancement and extraction," in *Picture Processing and Psychopictorics*, B. Lipkin and A. Rosenfeld, Eds. New York: Academic, 1970, pp. 75–149.

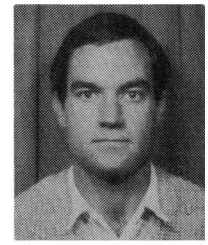
[24] S. O. Rice, "Mathematical analysis of random Noise," *Bell Syst. Tech. J.*, vol. 24, pp. 46–156, 1945.

[25] A. Rosenfeld and M. Thurston, "Edge and curve detection for visual scene analysis," *IEEE Trans. Comput.*, vol. C-20, no. 5, pp. 562–569, 1971.

[26] L. Spacek, "The computation of visual motion," Ph.D. dissertation, Univ. Essex at Colchester, 1984.

[27] K. A. Stevens, "Surface perception from local analysis of texture and contour," M.I.T. Artificial Intell. Lab., Cambridge, MA, Rep. AI-TR-512, 1980.

[28] V. Torre and T. Poggio, "On edge detection," M.I.T. Artificial Intell. Lab., Cambridge, MA, Rep. AIM-768, 1984.
[29] S. Ullman, *The Interpretation of Visual Motion.* Cambridge, MA: M.I.T. Press, 1979.
[30] N. Wiener, *Extrapolation, Interpolation and Smoothing of Stationary Time Series*. Cambridge, MA: M.I.T. Press, 1949.
[31] A. P. Witkin, "Shape from contour," M.I.T. Artificial Intell. Lab., Cambridge, MA, Rep. AI-TR-589, 1980.



John Canny (S'81–M'82) was born in Adelaide, Australia, in 1958. He received the B.Sc. degree in computer science and the B.E. degree from Adelaide University in 1980 and 1981, respectively, and the S.M. degree from the Massachusetts Institute of Technology, Cambridge, in 1983.

He is with the Artificial Intelligence Laboratory, M.I.T. His research interests include low-level vision, model-based vision, motion planning for robots, and computer algebra.

Mr. Canny is a student member of the Association for Computing Machinery.

AI Magazine Volume 27 Number 4 (2006) (© AAAI)

A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence

August 31, 1955

John McCarthy, Marvin L. Minsky,
Nathaniel Rochester,
and Claude E. Shannon

■ The 1956 Dartmouth summer research project on artificial intelligence was initiated by this August 31, 1955 proposal, authored by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. The original typescript consisted of 17 pages plus a title page. Copies of the typescript are housed in the archives at Dartmouth College and Stanford University. The first 5 papers state the proposal, and the remaining pages give qualifications and interests of the four who proposed the study. In the interest of brevity, this article reproduces only the proposal itself, along with the short autobiographical statements of the proposers.

We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

The following are some aspects of the artificial intelligence problem:

1. Automatic Computers

If a machine can do a job, then an automatic calculator can be programmed to simulate the machine. The speeds and memory capacities of present computers may be insufficient to simulate many of the higher functions of the human brain, but the major obstacle is not lack of machine capacity, but our inability to write programs taking full advantage of what we have.

2. How Can a Computer be Programmed to Use a Language

It may be speculated that a large part of human thought consists of manipulating words according to rules of reasoning and rules of conjecture. From this point of view, forming a generalization consists of admitting a new

A Proposal for the

DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE

June 17 - Aug. 16

We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

The following are some aspects of the artificial intelligence problem:

1) Automatic Computers

If a machine can do a job, then an automatic calculator can be programmed to simulate the machine. The speeds and memory capacities of present computers may be insufficient to simulate many of the higher functions of the human brain, but the major obstacle is not lack of machine capacity, but our inability to write programs taking full advantage of what we have.

2) How Can a Computer be Programmed to Use a Language

It may be speculated that a large part of human thought consists of manipulating words according to rules of reasoning

Photo courtesy Dartmouth College.

Page 1 of the Original Proposal.

word and some rules whereby sentences containing it imply and are implied by others. This idea has never been very precisely formulated nor have examples been worked out.

3. Neuron Nets

How can a set of (hypothetical) neurons be arranged so as to form concepts. Considerable theoretical and experimental work has been done on this problem by Uttley, Rashevsky and his group, Farley and Clark, Pitts and McCulloch, Minsky, Rochester and Holland, and others. Partial results have been obtained but the problem needs more theoretical work.

4. Theory of the Size of a Calculation

If we are given a well-defined problem (one for which it is possible to test mechanically whether or not a proposed answer is a valid answer) one way of solving it is to try all possible answers in order. This method is inefficient, and to exclude it one must have some criterion for efficiency of calculation. Some consideration will show that to get a measure of the efficiency of a calculation it is necessary to have on hand a method of measuring the complexity of calculating devices which in turn can be done if one has a theory of the complexity of functions. Some partial results on this problem have been obtained by Shannon, and also by McCarthy.

5. Self-lmprovement

Probably a truly intelligent machine will carry out activities which may best be described as self-improvement. Some schemes for doing this have been proposed and are worth further study. It seems likely that this question can be studied abstractly as well.

6. Abstractions

A number of types of "abstraction" can be distinctly defined and several others less distinctly. A direct attempt to classify these and to describe machine methods of forming abstractions from sensory and other data would seem worthwhile.

7. Randomness and Creativity

A fairly attractive and yet clearly incomplete conjecture is that the difference between creative thinking and unimaginative competent thinking lies in the injection of a some randomness. The randomness must be guided by intuition to be efficient. In other words, the educated guess or the hunch include controlled randomness in otherwise orderly thinking.

The Proposers

Claude E. Shannon

Claude E. Shannon, Mathematician, Bell Telephone Laboratories. Shannon developed the statistical theory of information, the application of propositional calculus to switching circuits, and has results on the efficient synthesis of switching circuits, the design of machines that learn, cryptography, and the theory of Turing machines. He and J. McCarthy are coediting an *Annals of Mathematics* study on "The Theory of Automata".

Marvin L. Minsky

Marvin L. Minsky, Harvard Junior Fellow in Mathematics and Neurology. Minsky has built a machine for simulating learning by nerve nets and has written a Princeton Ph.D thesis in mathematics entitled, "Neural Nets and the Brain Model Problem" which includes results in learning theory and the theory of random neural nets.

Nathaniel Rochester

Nathaniel Rochester, Manager of Information Research, IBM Corporation, Poughkeepsie, New York. Rochester was concerned with the development of radar for seven years and computing machinery for seven years. He and another engineer were jointly responsible for the design of the IBM Type 701 which is a large scale automatic computer in wide use today. He worked out some of the automatic programming techniques which are in wide use today and has been concerned with problems of how to get machines to do tasks which previously could be done only by people. He has also worked on simulation of nerve nets with particular emphasis on using computers to test theories in neurophysiology.

John McCarthy

John McCarthy, Assistant Professor of Mathematics, Dartmouth College. McCarthy has worked on a number of questions connected with the mathematical nature of the thought process including the theory of Turing machines, the speed of computers, the relation of a brain model to its environment, and the use of languages by machines. Some results of this work are included in the forthcoming "Annals Study" edited by Shannon and McCarthy. McCarthy's other work has been in the field of differential equations.

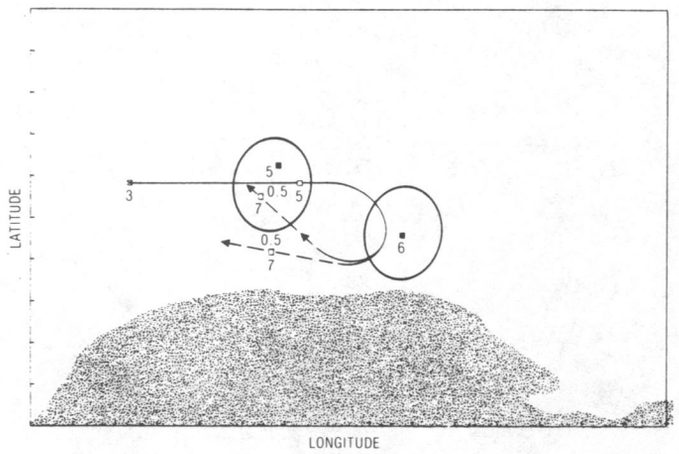


Fig. 6. ID plot of ship 10001 after the second round of operator-imposed assignment constraints.

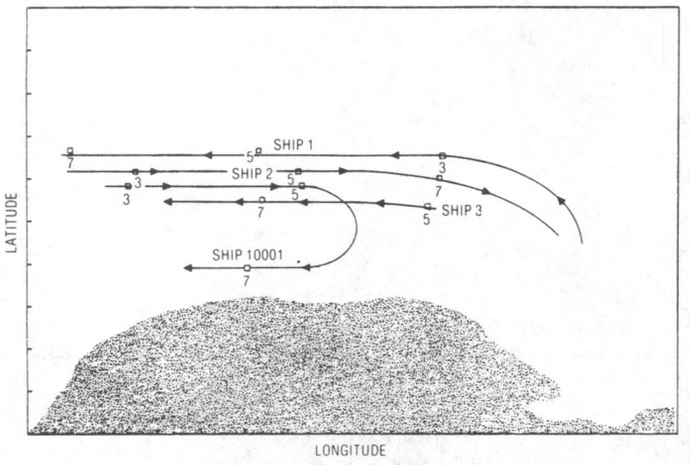


Fig. 7. Actual ship movements.

of the two last sighted locations. The true trajectories are shown in Fig. 7 where it can be seen that ship 10001 did, in fact, turn toward the coast.

IV. Concluding Remarks

The procedure of ship identification from DF sightings has been oversimplified in this discussion. Often DF sightings are not completely identified but, instead, contain only ship class information. The interactive technique still applies, but additional identification and display flexibility must be provided.

Any additional information contained in the sightings can be used to discriminate among radar and DF sightings. Factors such as measured heading and visual ID will permit further automatic reduction of the P and Q matrices.

It is also possible to automate some of the more routine manual functions. However, experience has shown that better results are obtained by having a human operator resolve ambiguous situations arising from sparse data.

References

[1] R. W. Sittler, "An optimal data association problem in surveillance theory," *IEEE Trans. Mil. Elect.*, vol. MIL-8, pp. 125–139, 1964.
[2] M. S. White, "Finding events in a sea of bubbles," *IEEE Trans. Comput.*, vol. C-20 (9), pp. 988–1006, 1971.
[3] A. G. Jaffer and Y. Bar-Shalom, "On optimal tracking in multiple-target environments," *Proc. of the 3rd Sym. on Nonlinear Estimation Theory and its Applications*, San Diego, CA, Sept. 1972.
[4] P. Smith and G. Buechler, "A branching algorithm for discrimination and tracking multiple objects," *IEEE Trans. Automat. Contr.*, vol. AC-20, pp. 101–104, 1975.
[5] D. L. Alspach, "A Gaussian sum approach to the multitarget-tracking problem," *Automatica*, vol. 11, pp. 285–296, 1975.
[6] C. L. Morefield, *Application of 0-1 Integer Programming to a Track Assembly Problem*, TR-0075(5085-10)-1, Aerospace Corp. El Segundo, CA, Apr. 1975.
[7] D. B. Reid, *A Multiple Hypothesis Filter for Tracking Multiple Targets in a Cluttered Environment*, LMSC-D560254, Lockheed Palo Alto Research Laboratories, Palo Alto, CA, Sept. 1977.
[8] P. L. Smith, "Reduction of sea surveillance data using binary matrices," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6 (8), pp. 531–538, Aug. 1976.

A Threshold Selection Method from Gray-Level Histograms

NOBUYUKI OTSU

***Abstract*—A nonparametric and unsupervised method of automatic threshold selection for picture segmentation is presented. An optimal threshold is selected by the discriminant criterion, namely, so as to maximize the separability of the resultant classes in gray levels. The procedure is very simple, utilizing only the zeroth- and the first-order cumulative moments of the gray-level histogram. It is straightforward to extend the method to multithreshold problems. Several experimental results are also presented to support the validity of the method.**

I. Introduction

It is important in picture processing to select an adequate threshold of gray level for extracting objects from their background. A variety of techniques have been proposed in this regard. In an ideal case, the histogram has a deep and sharp valley between two peaks representing objects and background, respectively, so that the threshold can be chosen at the bottom of this valley [1]. However, for most real pictures, it is often difficult to detect the valley bottom precisely, especially in such cases as when the valley is flat and broad, imbued with noise, or when the two peaks are extremely unequal in height, often producing no traceable valley. There have been some techniques proposed in order to overcome these difficulties. They are, for example, the valley sharpening technique [2], which restricts the histogram to the pixels with large absolute values of derivative (Laplacian or gradient), and the difference histogram method [3], which selects the threshold at the gray level with the maximal amount of difference. These utilize information concerning neighboring pixels (or edges) in the original picture to modify the histogram so as to make it useful for thresholding. Another class of methods deals directly with the gray-level histogram by parametric techniques. For example, the histogram is approximated in the least square sense by a sum of Gaussian distributions, and statistical decision procedures are applied [4]. However, such a method requires considerably tedious and sometimes unstable calculations. Moreover, in many cases, the Gaussian distributions turn out to be a meager approximation of the real modes.

In any event, no "goodness" of threshold has been evaluated in

Manuscript received October 13, 1977; revised April 17, 1978 and August 31, 1978.
The author is with the Mathematical Engineering Section, Information Science Division, Electrotechnical Laboratory, Chiyoda-ku, Tokyo 100, Japan.

most of the methods so far proposed. This would imply that it could be the right way of deriving an optimal thresholding method to establish an appropriate criterion for evaluating the "goodness" of threshold from a more general standpoint.

In this correspondence, our discussion will be confined to the elementary case of threshold selection where only the gray-level histogram suffices without other *a priori* knowledge. It is not only important as a standard technique in picture processing, but also essential for unsupervised decision problems in pattern recognition. A new method is proposed from the viewpoint of discriminant analysis; it directly approaches the feasibility of evaluating the "goodness" of threshold and automatically selecting an optimal threshold.

II. Formulation

Let the pixels of a given picture be represented in L gray levels $[1, 2, \cdots, L]$. The number of pixels at level i is denoted by n_i and the total number of pixels by $N = n_1 + n_2 + \cdots + n_L$. In order to simplify the discussion, the gray-level histogram is normalized and regarded as a probability distribution:

$$p_i = n_i/N, \qquad p_i \geq 0, \sum_{i=1}^{L} p_i = 1. \tag{1}$$

Now suppose that we dichotomize the pixels into two classes C_0 and C_1 (background and objects, or vice versa) by a threshold at level k; C_0 denotes pixels with levels $[1, \cdots, k]$, and C_1 denotes pixels with levels $[k+1, \cdots, L]$. Then the probabilities of class occurrence and the class mean levels, respectively, are given by

$$\omega_0 = \Pr(C_0) = \sum_{i=1}^{k} p_i = \omega(k) \tag{2}$$

$$\omega_1 = \Pr(C_1) = \sum_{i=k+1}^{L} p_i = 1 - \omega(k) \tag{3}$$

and

$$\mu_0 = \sum_{i=1}^{k} i \Pr(i|C_0) = \sum_{i=1}^{k} ip_i/\omega_0 = \mu(k)/\omega(k) \tag{4}$$

$$\mu_1 = \sum_{i=k+1}^{L} i \Pr(i|C_1) = \sum_{i=k+1}^{L} ip_i/\omega_1 = \frac{\mu_T - \mu(k)}{1 - \omega(k)}, \tag{5}$$

where

$$\omega(k) = \sum_{i=1}^{k} p_i \tag{6}$$

and

$$\mu(k) = \sum_{i=1}^{k} ip_i \tag{7}$$

are the zeroth- and the first-order cumulative moments of the histogram up to the kth level, respectively, and

$$\mu_T = \mu(L) = \sum_{i=1}^{L} ip_i \tag{8}$$

is the total mean level of the original picture. We can easily verify the following relation for any choice of k:

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T, \qquad \omega_0 + \omega_1 = 1. \tag{9}$$

The class variances are given by

$$\sigma_0^2 = \sum_{i=1}^{k} (i - \mu_0)^2 \Pr(i|C_0) = \sum_{i=1}^{k} (i - \mu_0)^2 p_i/\omega_0 \tag{10}$$

$$\sigma_1^2 = \sum_{i=k+1}^{L} (i - \mu_1)^2 \Pr(i|C_1) = \sum_{i=k+1}^{L} (i - \mu_1)^2 p_i/\omega_1. \tag{11}$$

These require second-order cumulative moments (statistics).

In order to evaluate the "goodness" of the threshold (at level k), we shall introduce the following discriminant criterion measures (or measures of class separability) used in the discriminant analysis [5]:

$$\lambda = \sigma_B^2/\sigma_W^2, \qquad \kappa = \sigma_T^2/\sigma_W^2, \qquad \eta = \sigma_B^2/\sigma_T^2, \tag{12}$$

where

$$\sigma_W^2 = \omega_0\sigma_0^2 + \omega_1\sigma_1^2 \tag{13}$$

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2 \tag{14}$$

(due to (9)) and

$$\sigma_T^2 = \sum_{i=1}^{L} (i - \mu_T)^2 p_i \tag{15}$$

are the within-class variance, the between-class variance, and the total variance of levels, respectively. Then our problem is reduced to an optimization problem to search for a threshold k that maximizes one of the object functions (the criterion measures) in (12).

This standpoint is motivated by a conjecture that well-thresholded classes would be separated in gray levels, and conversely, a threshold giving the best separation of classes in gray levels would be the best threshold.

The discriminant criteria maximizing λ, κ, and η, respectively, for k are, however, equivalent to one another; e.g., $\kappa = \lambda + 1$ and $\eta = \lambda/(\lambda + 1)$ in terms of λ, because the following basic relation always holds:

$$\sigma_W^2 + \sigma_B^2 = \sigma_T^2. \tag{16}$$

It is noticed that σ_W^2 and σ_B^2 are functions of threshold level k, but σ_T^2 is independent of k. It is also noted that σ_W^2 is based on the second-order statistics (class variances), while σ_B^2 is based on the first-order statistics (class means). Therefore, η is the simplest measure with respect to k. Thus we adopt η as the criterion measure to evaluate the "goodness" (or separability) of the threshold at level k.

The optimal threshold k^* that maximizes η, or equivalently maximizes σ_B^2, is selected in the following sequential search by using the simple cumulative quantities (6) and (7), or explicitly using (2)–(5):

$$\eta(k) = \sigma_B^2(k)/\sigma_T^2 \tag{17}$$

$$\sigma_B^2(k) = \frac{[\mu_T\omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]} \tag{18}$$

and the optimal threshold k^* is

$$\sigma_B^2(k^*) = \max_{1 \leq k < L} \sigma_B^2(k). \tag{19}$$

From the problem, the range of k over which the maximum is sought can be restricted to

$$S^* = \{k;\ \omega_0\omega_1 = \omega(k)[1 - \omega(k)] > 0, \qquad \text{or } 0 < \omega(k) < 1\}.$$

We shall call it the effective range of the gray-level histogram. From the definition in (14), the criterion measure σ_B^2 (or η) takes a minimum value of zero for such k as $k \in S - S^* = \{k;\ \omega(k) = 0 \text{ or } 1\}$ (i.e., making all pixels either C_1 or C_0, which is, of course, not our concern) and takes a positive and bounded value for $k \in S^*$. It is, therefore, obvious that the maximum always exists.

III. Discussion and Remarks

A. Analysis of further important aspects

The method proposed in the foregoing affords further means to analyze important aspects other than selecting optimal thresholds.

For the selected threshold k^*, the class probabilities (2) and (3), respectively, indicate the portions of the areas occupied by the classes in the picture so thresholded. The class means (4) and (5) serve as estimates of the mean levels of the classes in the original gray-level picture.

The maximum value $\eta(k^*)$, denoted simply by η^*, can be used as a measure to evaluate the separability of classes (or ease of thresholding) for the original picture or the bimodality of the histogram. This is a significant measure, for it is invariant under affine transformations of the gray-level scale (i.e., for any shift and dilatation, $g_i' = ag_i + b$). It is uniquely determined within the range

$$0 \leq \eta^* \leq 1.$$

The lower bound (zero) is attainable by, and only by, pictures having a single constant gray level, and the upper bound (unity) is attainable by, and only by, two-valued pictures.

B. Extension to Multithresholding

The extension of the method to multihresholding problems is straightforward by virtue of the discriminant criterion. For example, in the case of three-thresholding, we assume two thresholds: $1 \leq k_1 < k_2 < L$, for separating three classes, C_0 for $[1, \cdots, k_1]$, C_1 for $[k_1 + 1, \cdots, k_2]$, and C_2 for $[k_2 + 1, \cdots, L]$. The criterion measure σ_B^2 (also η) is then a function of two variables k_1 and k_2, and an optimal set of thresholds k_1^* and k_2^* is selected by maximizing σ_B^2:

$$\sigma_B^2(k_1^*, k_2^*) = \max_{1 \leq k_1 < k_2 < L} \sigma_B^2(k_1, k_2).$$

It should be noticed that the selected thresholds generally become less credible as the number of classes to be separated increases. This is because the criterion measure (σ_B^2), defined in one-dimensional (gray-level) scale, may gradually lose its meaning as the number of classes increases. The expression of σ_B^2 and the maximization procedure also become more and more complicated. However, they are very simple for $M = 2$ and 3, which cover almost all practical applications, so that a special method to reduce the search processes is hardly needed. It should be remarked that the parameters required in the present method for M-thresholding are $M - 1$ discrete thresholds themselves, while the parametric method, where the gray-level histogram is approximated by the sum of Gaussian distributions, requires $3M - 1$ continuous parameters.

C. Experimental Results

Several examples of experimental results are shown in Figs. 1–3. Throughout these figures, (a) (as also (e)) is an original gray-level picture; (b) (and (f)) is the result of thresholding; (c) (and (g)) is a set of the gray-level histogram (marked at the selected threshold) and the criterion measure $\eta(k)$ related thereto; and (d) (and (h)) is the result obtained by the analysis. The original gray-level pictures are all 64 × 64 in size, and the numbers of gray levels are 16 in Fig. 1, 64 in Fig. 2, 32 in Fig. 3(a), and 256 in Fig. 3(e). (They all had equal outputs in 16 gray levels by superposition of symbols by reason of representation, so that they may be slightly lacking in precise detail in the gray levels.)

Fig. 1 shows the results of the application to an identical character "A" typewritten in different ways, one with a new ribbon (a)

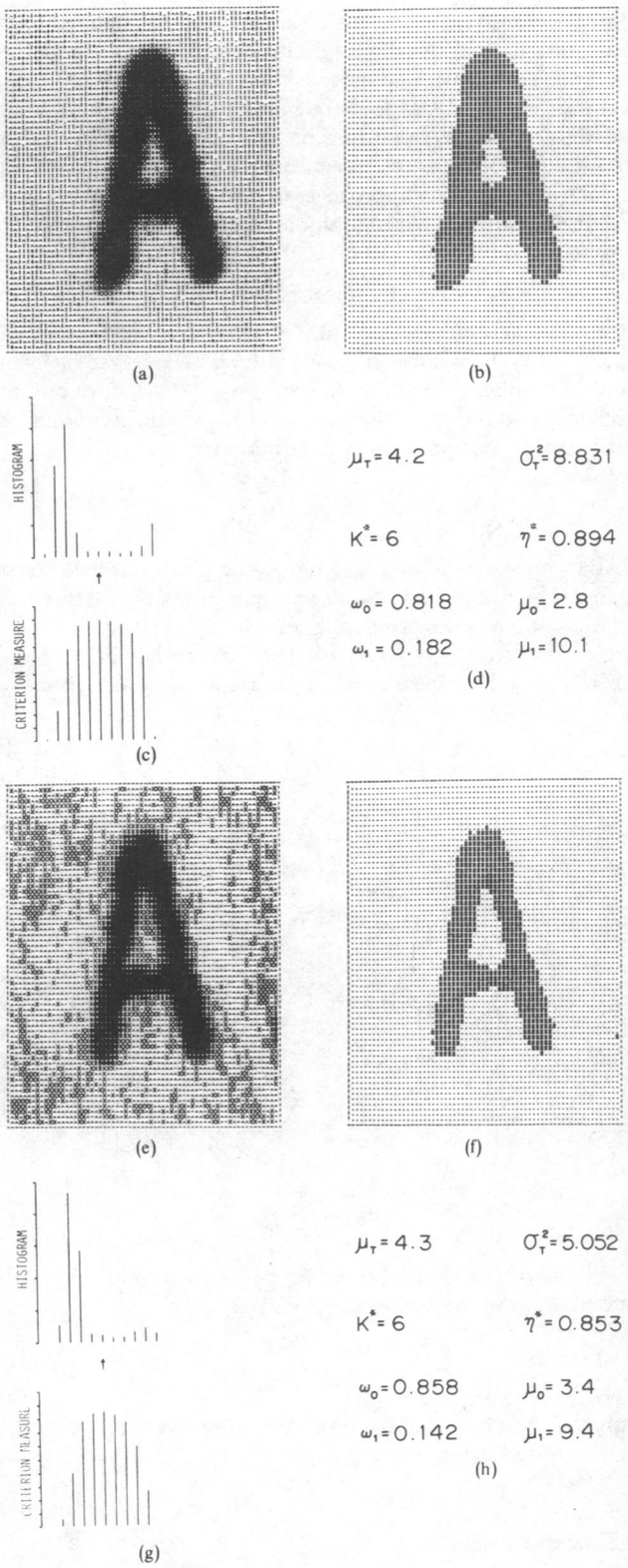


Fig. 1. Application to characters.

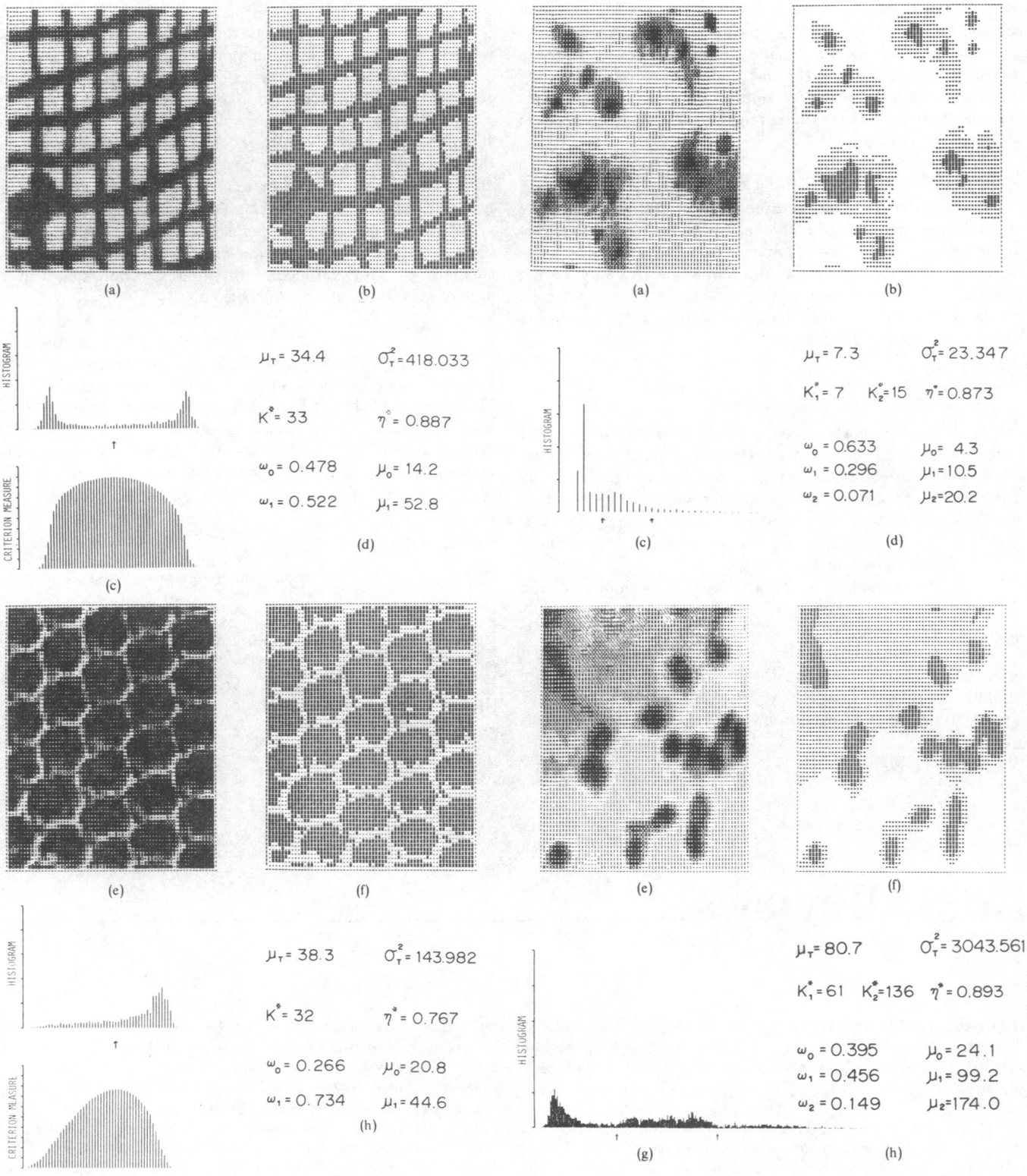


Fig. 2. Application to textures.

Fig. 3. Application to cells. Criterion measures $\eta(k_1, k_2)$ are omitted in (c) and (g) by reason of illustration.

and another with an old one (e), respectively. In Fig. 2, the results are shown for textures, where the histograms typically show the difficult cases of a broad and flat valley (c) and a unimodal peak (g). In order to appropriately illustrate the case of three-thresholding, the method has also been applied to cell images with successful results, shown in Fig. 3, where C_0 stands for the background, C_1 for the cytoplasm, and C_2 for the nucleus. They are indicated in (b) and (f) by ( ), (=), and (*), respectively.

A number of experimental results so far obtained for various examples indicate that the present method derived theoretically is of satisfactory practical use.

D. Unimodality of the object function

The object function $\sigma_B^2(k)$, or equivalently, the criterion measure $\eta(k)$, is always smooth and unimodal, as can be seen in the experimental results in Figs. 1–2. It may attest to an advantage of the suggested criterion and may also imply the stability of the method. The rigorous proof of the unimodality has not yet been obtained. However, it can be dispensed with from our standpoint concerning only the maximum.

IV. Conclusion

A method to select a threshold automatically from a gray level histogram has been derived from the viewpoint of discriminant analysis. This directly deals with the problem of evaluating the goodness of thresholds. An optimal threshold (or set of thresholds) is selected by the discriminant criterion; namely, by maximizing the discriminant measure η (or the measure of separability of the resultant classes in gray levels).

The proposed method is characterized by its nonparametric and unsupervised nature of threshold selection and has the following desirable advantages.

1) The procedure is very simple; only the zeroth and the first order cumulative moments of the gray-level histogram are utilized.

2) A straightforward extension to multithresholding problems is feasible by virtue of the criterion on which the method is based.

3) An optimal threshold (or set of thresholds) is selected automatically and stably, not based on the differentiation (i.e., a local property such as valley), but on the integration (i.e., a global property) of the histogram.

4) Further important aspects can also be analyzed (e.g., estimation of class mean levels, evaluation of class separability, etc.).

5) The method is quite general: it covers a wide scope of unsupervised decision procedure.

The range of its applications is not restricted only to the thresholding of the gray-level picture, such as specifically described in the foregoing, but it may also cover other cases of unsupervised classification in which a histogram of some characteristic (or feature) discriminative for classifying the objects is available.

Taking into account these points, the method suggested in this correspondence may be recommended as the most simple and standard one for automatic threshold selection that can be applied to various practical problems.

Acknowledgment

The author wishes to thank Dr. H. Nishino, Head of the Information Science Division, for his hospitality and encouragement. Thanks are also due to Dr. S. Mori, Chief of the Picture Processing Section, for the data of characters and textures and valuable discussions, and to Dr. Y. Noguchi for cell data. The author is also very grateful to Professor S. Amari of the University of Tokyo for his cordial and helpful suggestions for revising the presentation of the manuscript.

References

[1] J. M. S. Prewitt and M. L. Mendelsohn, "The analysis of cell images," *Ann. N. Y. Acad. Sci.*, vol. 128, pp. 1035–1053, 1966.
[2] J. S. Weszka, R. N. Nagel, and A. Rosenfeld, "A threshold selection technique," *IEEE Trans. Comput.*, vol. C-23, pp. 1322–1326, 1974.
[3] S. Watanabe and CYBEST Group, "An automated apparatus for cancer prescreening: CYBEST," *Comp. Graph. Image Process.*, vol. 3, pp. 350–358, 1974.
[4] C. K. Chow and T. Kaneko, "Automatic boundary detection of the left ventricle from cineangiograms," Comput. Biomed. Res., vol. 5, pp. 388–410, 1972.
[5] K. Fukunage, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1972, pp. 260–267.

Book Reviews

Orthogonal Transforms for Digital Signal Processing—N. Ahmed and K. R. Rao (New York: Springer-Verlag, 1975, 263 pp.). *Reviewed by Lokenath Debnath, Departments of Mathematics and Physics, East Carolina University, Greenville, NC 27834.*

With the advent of high-speed digital computers and the rapid advances in digital technology, orthogonal transforms have received considerable attention in recent years, especially in the area of digital signal processing. This book presents the theory and applications of discrete orthogonal transforms. With some elementary knowledge of Fourier series transforms, differential equations, and matrix algebra as prerequisites, this book is written as a graduate level text for electrical and computer engineering students.

The first two chapters are essentially tutorial and cover signal representation using orthogonal functions, Fourier methods of representating signals, relation between the Fourier series and the Fourier transform, and some aspects of cross correlation, autocorrelation, and convolution. These chapters provide a systematic transition from the Fourier representation of analog signals to that of digital signals.

The third chapter is concerned with the Fourier representation of discrete and digital signals through the discrete Fourier transform (DFT). Some important properties of the DFT including the convolution and correlation theorems are discussed in some detail. The concept of amplitude, power, and phase spectra is introduced. It is shown that the DFT is directly related to the Fourier transform series representation of data sequences $\{X(m)\}$. The two-dimensional DFT and its possible extension to higher dimensions are investigated, and the chapter closes with some discussion on time-varying power and phase spectra.

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [40] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test *set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.*

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1]*, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.*

1. Introduction

Deep convolutional neural networks [22, 21] have led to a series of breakthroughs for image classification [21, 49, 39]. Deep networks naturally integrate low/mid/high-level features [49] and classifiers in an end-to-end multi-layer fashion, and the "levels" of features can be enriched by the number of stacked layers (depth). Recent evidence [40, 43] reveals that network depth is of crucial importance, and the leading results [40, 43, 12, 16] on the challenging ImageNet dataset [35] all exploit "very deep" [40] models, with a depth of sixteen [40] to thirty [16]. Many other non-trivial visual recognition tasks [7, 11, 6, 32, 27] have also greatly benefited from very deep models.

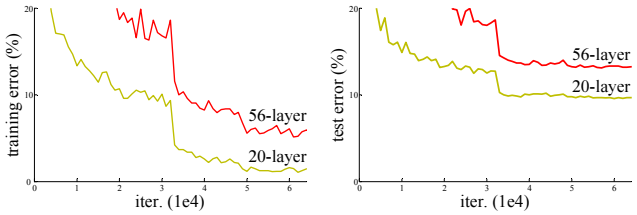


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [14, 1, 8], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 8, 36, 12] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more layers to a suitably deep model leads to *higher training error*, as reported in [10, 41] and thoroughly verified by our experiments. Fig. 1 shows a typical example.

The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution *by construction* to the deeper model: the added layers are *identity* mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart. But experiments show that our current solvers on hand are unable to find solutions that

[1] http://image-net.org/challenges/LSVRC/2015/ and http://mscoco.org/dataset/#detections-challenge2015.

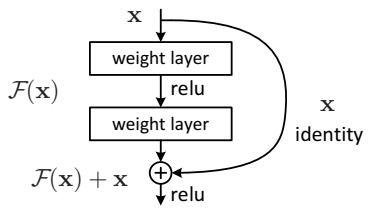


Figure 2. Residual learning: a building block.

are comparably good or better than the constructed solution (or unable to do so in feasible time).

In this paper, we address the degradation problem by introducing a *deep residual learning* framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. Formally, denoting the desired underlying mapping as $\mathcal{H}(\mathbf{x})$, we let the stacked nonlinear layers fit another mapping of $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original mapping is recast into $\mathcal{F}(\mathbf{x})+\mathbf{x}$. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

The formulation of $\mathcal{F}(\mathbf{x})+\mathbf{x}$ can be realized by feedforward neural networks with "shortcut connections" (Fig. 2). Shortcut connections [2, 33, 48] are those skipping one or more layers. In our case, the shortcut connections simply perform *identity* mapping, and their outputs are added to the outputs of the stacked layers (Fig. 2). Identity shortcut connections add neither extra parameter nor computational complexity. The entire network can still be trained end-to-end by SGD with backpropagation, and can be easily implemented using common libraries (*e.g.*, Caffe [19]) without modifying the solvers.

We present comprehensive experiments on ImageNet [35] to show the degradation problem and evaluate our method. We show that: 1) Our extremely deep residual nets are easy to optimize, but the counterpart "plain" nets (that simply stack layers) exhibit higher training error when the depth increases; 2) Our deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks.

Similar phenomena are also shown on the CIFAR-10 set [20], suggesting that the optimization difficulties and the effects of our method are not just akin to a particular dataset. We present successfully trained models on this dataset with over 100 layers, and explore models with over 1000 layers.

On the ImageNet classification dataset [35], we obtain excellent results by extremely deep residual nets. Our 152-layer residual net is the deepest network ever presented on ImageNet, while still having lower complexity than VGG nets [40]. Our ensemble has **3.57%** top-5 error on the ImageNet *test* set, and *won the 1st place in the ILSVRC 2015 classification competition*. The extremely deep representations also have excellent generalization performance on other recognition tasks, and lead us to further *win the 1st places on: ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation* in ILSVRC & COCO 2015 competitions. This strong evidence shows that the residual learning principle is generic, and we expect that it is applicable in other vision and non-vision problems.

2. Related Work

Residual Representations. In image recognition, VLAD [18] is a representation that encodes by the residual vectors with respect to a dictionary, and Fisher Vector [30] can be formulated as a probabilistic version [18] of VLAD. Both of them are powerful shallow representations for image retrieval and classification [4, 47]. For vector quantization, encoding residual vectors [17] is shown to be more effective than encoding original vectors.

In low-level vision and computer graphics, for solving Partial Differential Equations (PDEs), the widely used Multigrid method [3] reformulates the system as subproblems at multiple scales, where each subproblem is responsible for the residual solution between a coarser and a finer scale. An alternative to Multigrid is hierarchical basis preconditioning [44, 45], which relies on variables that represent residual vectors between two scales. It has been shown [3, 44, 45] that these solvers converge much faster than standard solvers that are unaware of the residual nature of the solutions. These methods suggest that a good reformulation or preconditioning can simplify the optimization.

Shortcut Connections. Practices and theories that lead to shortcut connections [2, 33, 48] have been studied for a long time. An early practice of training multi-layer perceptrons (MLPs) is to add a linear layer connected from the network input to the output [33, 48]. In [43, 24], a few intermediate layers are directly connected to auxiliary classifiers for addressing vanishing/exploding gradients. The papers of [38, 37, 31, 46] propose methods for centering layer responses, gradients, and propagated errors, implemented by shortcut connections. In [43], an "inception" layer is composed of a shortcut branch and a few deeper branches.

Concurrent with our work, "highway networks" [41, 42] present shortcut connections with gating functions [15]. These gates are data-dependent and have parameters, in contrast to our identity shortcuts that are parameter-free. When a gated shortcut is "closed" (approaching zero), the layers in highway networks represent *non-residual* functions. On the contrary, our formulation always learns residual functions; our identity shortcuts are never closed, and all information is always passed through, with additional residual functions to be learned. In addition, high-

way networks have not demonstrated accuracy gains with extremely increased depth (*e.g.*, over 100 layers).

3. Deep Residual Learning

3.1. Residual Learning

Let us consider $\mathcal{H}(\mathbf{x})$ as an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with $\mathbf{x}$ denoting the inputs to the first of these layers. If one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions[2], then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, *i.e.*, $\mathcal{H}(\mathbf{x}) - \mathbf{x}$ (assuming that the input and output are of the same dimensions). So rather than expect stacked layers to approximate $\mathcal{H}(\mathbf{x})$, we explicitly let these layers approximate a residual function $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original function thus becomes $\mathcal{F}(\mathbf{x})+\mathbf{x}$. Although both forms should be able to asymptotically approximate the desired functions (as hypothesized), the ease of learning might be different.

This reformulation is motivated by the counterintuitive phenomena about the degradation problem (Fig. 1, left). As we discussed in the introduction, if the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart. The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers. With the residual learning reformulation, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings.

In real cases, it is unlikely that identity mappings are optimal, but our reformulation may help to precondition the problem. If the optimal function is closer to an identity mapping than to a zero mapping, it should be easier for the solver to find the perturbations with reference to an identity mapping, than to learn the function as a new one. We show by experiments (Fig. 7) that the learned residual functions in general have small responses, suggesting that identity mappings provide reasonable preconditioning.

3.2. Identity Mapping by Shortcuts

We adopt residual learning to every few stacked layers. A building block is shown in Fig. 2. Formally, in this paper we consider a building block defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \tag{1}$$

Here $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers considered. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned. For the example in Fig. 2 that has two layers, $\mathcal{F} = W_2\sigma(W_1\mathbf{x})$ in which σ denotes ReLU [29] and the biases are omitted for simplifying notations. The operation $\mathcal{F} + \mathbf{x}$ is performed by a shortcut connection and element-wise addition. We adopt the second nonlinearity after the addition (*i.e.*, $\sigma(\mathbf{y})$, see Fig. 2).

The shortcut connections in Eqn.(1) introduce neither extra parameter nor computation complexity. This is not only attractive in practice but also important in our comparisons between plain and residual networks. We can fairly compare plain/residual networks that simultaneously have the same number of parameters, depth, width, and computational cost (except for the negligible element-wise addition).

The dimensions of $\mathbf{x}$ and $\mathcal{F}$ must be equal in Eqn.(1). If this is not the case (*e.g.*, when changing the input/output channels), we can perform a linear projection W_s by the shortcut connections to match the dimensions:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \tag{2}$$

We can also use a square matrix W_s in Eqn.(1). But we will show by experiments that the identity mapping is sufficient for addressing the degradation problem and is economical, and thus W_s is only used when matching dimensions.

The form of the residual function $\mathcal{F}$ is flexible. Experiments in this paper involve a function $\mathcal{F}$ that has two or three layers (Fig. 5), while more layers are possible. But if $\mathcal{F}$ has only a single layer, Eqn.(1) is similar to a linear layer: $\mathbf{y} = W_1\mathbf{x} + \mathbf{x}$, for which we have not observed advantages.

We also note that although the above notations are about fully-connected layers for simplicity, they are applicable to convolutional layers. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ can represent multiple convolutional layers. The element-wise addition is performed on two feature maps, channel by channel.

3.3. Network Architectures

We have tested various plain/residual nets, and have observed consistent phenomena. To provide instances for discussion, we describe two models for ImageNet as follows.

Plain Network. Our plain baselines (Fig. 3, middle) are mainly inspired by the philosophy of VGG nets [40] (Fig. 3, left). The convolutional layers mostly have 3×3 filters and follow two simple design rules: (i) for the same output feature map size, the layers have the same number of filters; and (ii) if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer. We perform downsampling directly by convolutional layers that have a stride of 2. The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax. The total number of weighted layers is 34 in Fig. 3 (middle).

It is worth noticing that our model has *fewer* filters and *lower* complexity than VGG nets [40] (Fig. 3, left). Our 34-layer baseline has 3.6 billion FLOPs (multiply-adds), which is only 18% of VGG-19 (19.6 billion FLOPs).

[2]This hypothesis, however, is still an open question. See [28].

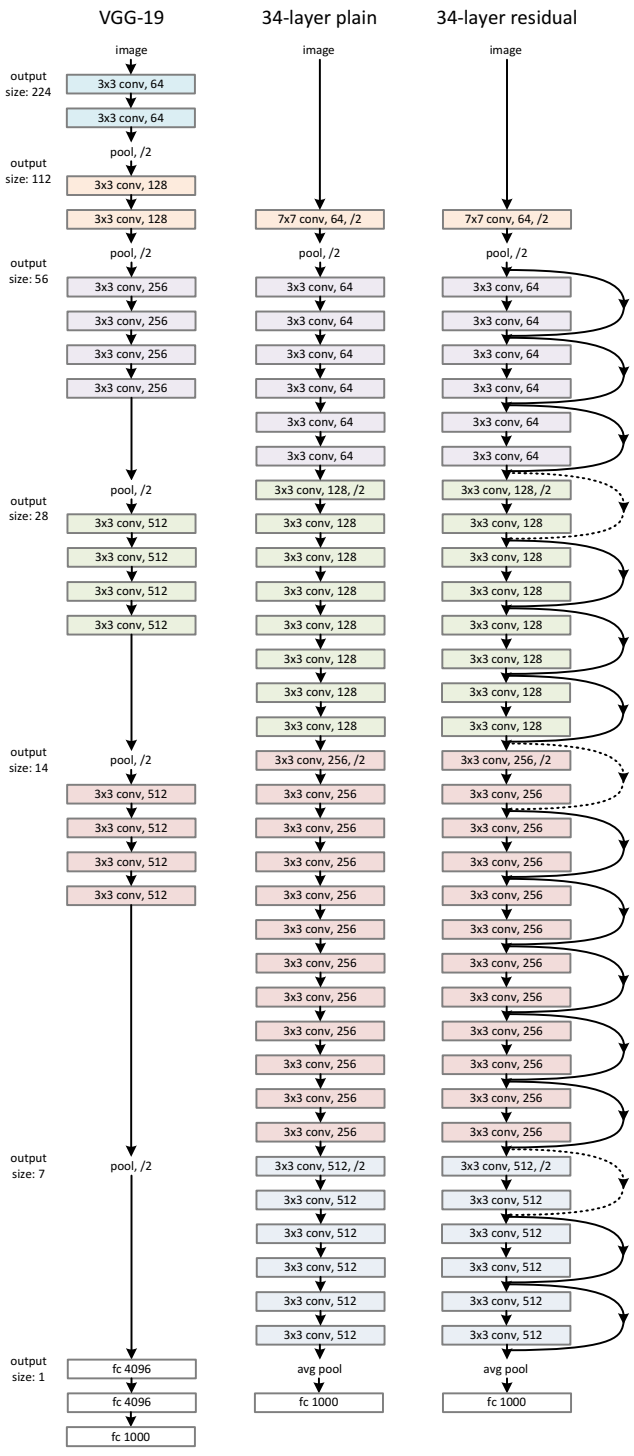


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [40] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Residual Network. Based on the above plain network, we insert shortcut connections (Fig. 3, right) which turn the network into its counterpart residual version. The identity shortcuts (Eqn.(1)) can be directly used when the input and output are of the same dimensions (solid line shortcuts in Fig. 3). When the dimensions increase (dotted line shortcuts in Fig. 3), we consider two options: (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter; (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1×1 convolutions). For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

3.4. Implementation

Our implementation for ImageNet follows the practice in [21, 40]. The image is resized with its shorter side randomly sampled in $[256, 480]$ for scale augmentation [40]. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted [21]. The standard color augmentation in [21] is used. We adopt batch normalization (BN) [16] right after each convolution and before activation, following [16]. We initialize the weights as in [12] and train all plain/residual nets from scratch. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, and the models are trained for up to 60×10^4 iterations. We use a weight decay of 0.0001 and a momentum of 0.9. We do not use dropout [13], following the practice in [16].

In testing, for comparison studies we adopt the standard 10-crop testing [21]. For best results, we adopt the fully-convolutional form as in [40, 12], and average the scores at multiple scales (images are resized such that the shorter side is in $\{224, 256, 384, 480, 640\}$).

4. Experiments

4.1. ImageNet Classification

We evaluate our method on the ImageNet 2012 classification dataset [35] that consists of 1000 classes. The models are trained on the 1.28 million training images, and evaluated on the 50k validation images. We also obtain a final result on the 100k test images, reported by the test server. We evaluate both top-1 and top-5 error rates.

Plain Networks. We first evaluate 18-layer and 34-layer plain nets. The 34-layer plain net is in Fig. 3 (middle). The 18-layer plain net is of a similar form. See Table 1 for detailed architectures.

The results in Table 2 show that the deeper 34-layer plain net has higher validation error than the shallower 18-layer plain net. To reveal the reasons, in Fig. 4 (left) we compare their training/validation errors during the training procedure. We have observed the degradation problem - the

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

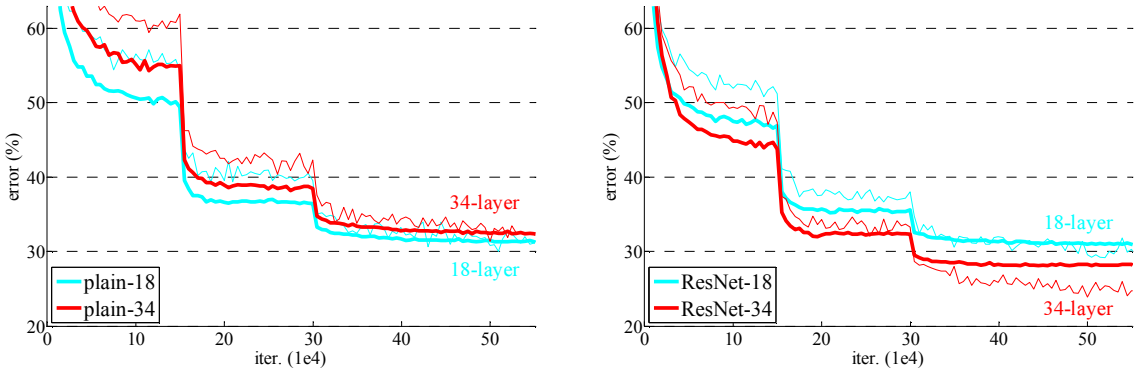


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

34-layer plain net has higher *training* error throughout the whole training procedure, even though the solution space of the 18-layer plain network is a subspace of that of the 34-layer one.

We argue that this optimization difficulty is *unlikely* to be caused by vanishing gradients. These plain networks are trained with BN [16], which ensures forward propagated signals to have non-zero variances. We also verify that the backward propagated gradients exhibit healthy norms with BN. So neither forward nor backward signals vanish. In fact, the 34-layer plain net is still able to achieve competitive accuracy (Table 3), suggesting that the solver works to some extent. We conjecture that the deep plain nets may have exponentially low convergence rates, which impact the reducing of the training error[3]. The reason for such optimization difficulties will be studied in the future.

Residual Networks. Next we evaluate 18-layer and 34-layer residual nets (*ResNets*). The baseline architectures are the same as the above plain nets, expect that a shortcut connection is added to each pair of 3×3 filters as in Fig. 3 (right). In the first comparison (Table 2 and Fig. 4 right), we use identity mapping for all shortcuts and zero-padding for increasing dimensions (option A). So they have *no extra parameter* compared to the plain counterparts.

We have three major observations from Table 2 and Fig. 4. First, the situation is reversed with residual learning – the 34-layer ResNet is better than the 18-layer ResNet (by 2.8%). More importantly, the 34-layer ResNet exhibits considerably lower training error and is generalizable to the validation data. This indicates that the degradation problem is well addressed in this setting and we manage to obtain accuracy gains from increased depth.

Second, compared to its plain counterpart, the 34-layer

[3]We have experimented with more training iterations (3×) and still observed the degradation problem, suggesting that this problem cannot be feasibly addressed by simply using more iterations.

model	top-1 err.	top-5 err.
VGG-16 [40]	28.07	9.33
GoogLeNet [43]	-	9.15
PReLU-net [12]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43†
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

method	top-5 err. (**test**)
VGG [40] (ILSVRC'14)	7.32
GoogLeNet [43] (ILSVRC'14)	6.66
VGG [40] (v5)	6.8
PReLU-net [12]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

ResNet reduces the top-1 error by 3.5% (Table 2), resulting from the successfully reduced training error (Fig. 4 right *vs*. left). This comparison verifies the effectiveness of residual learning on extremely deep systems.

Last, we also note that the 18-layer plain/residual nets are comparably accurate (Table 2), but the 18-layer ResNet converges faster (Fig. 4 right *vs*. left). When the net is "not overly deep" (18 layers here), the current SGD solver is still able to find good solutions to the plain net. In this case, the ResNet eases the optimization by providing faster convergence at the early stage.

Identity *vs*. Projection Shortcuts. We have shown that parameter-free, identity shortcuts help with training. Next we investigate projection shortcuts (Eqn.(2)). In Table 3 we compare three options: (A) zero-padding shortcuts are used for increasing dimensions, and all shortcuts are parameter-free (the same as Table 2 and Fig. 4 right); (B) projection shortcuts are used for increasing dimensions, and other shortcuts are identity; and (C) all shortcuts are projections.

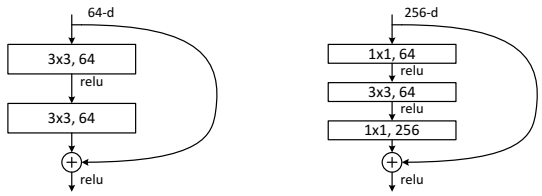


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Table 3 shows that all three options are considerably better than the plain counterpart. B is slightly better than A. We argue that this is because the zero-padded dimensions in A indeed have no residual learning. C is marginally better than B, and we attribute this to the extra parameters introduced by many (thirteen) projection shortcuts. But the small differences among A/B/C indicate that projection shortcuts are not essential for addressing the degradation problem. So we do not use option C in the rest of this paper, to reduce memory/time complexity and model sizes. Identity shortcuts are particularly important for not increasing the complexity of the bottleneck architectures that are introduced below.

Deeper Bottleneck Architectures. Next we describe our deeper nets for ImageNet. Because of concerns on the training time that we can afford, we modify the building block as a *bottleneck* design[4]. For each residual function $\mathcal{F}$, we use a stack of 3 layers instead of 2 (Fig. 5). The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions. Fig. 5 shows an example, where both designs have similar time complexity.

The parameter-free identity shortcuts are particularly important for the bottleneck architectures. If the identity shortcut in Fig. 5 (right) is replaced with projection, one can show that the time complexity and model size are doubled, as the shortcut is connected to the two high-dimensional ends. So identity shortcuts lead to more efficient models for the bottleneck designs.

50-layer ResNet: We replace each 2-layer block in the

[4]Deeper *non*-bottleneck ResNets (*e.g*., Fig. 5 left) also gain accuracy from increased depth (as shown on CIFAR-10), but are not as economical as the bottleneck ResNets. So the usage of bottleneck designs is mainly due to practical considerations. We further note that the degradation problem of plain nets is also witnessed for the bottleneck designs.

34-layer net with this 3-layer bottleneck block, resulting in a 50-layer ResNet (Table 1). We use option B for increasing dimensions. This model has 3.8 billion FLOPs.

101-layer and 152-layer ResNets: We construct 101-layer and 152-layer ResNets by using more 3-layer blocks (Table 1). Remarkably, although the depth is significantly increased, the 152-layer ResNet (11.3 billion FLOPs) still has *lower complexity* than VGG-16/19 nets (15.3/19.6 billion FLOPs).

The 50/101/152-layer ResNets are more accurate than the 34-layer ones by considerable margins (Table 3 and 4). We do not observe the degradation problem and thus enjoy significant accuracy gains from considerably increased depth. The benefits of depth are witnessed for all evaluation metrics (Table 3 and 4).

Comparisons with State-of-the-art Methods. In Table 4 we compare with the previous best single-model results. Our baseline 34-layer ResNets have achieved very competitive accuracy. Our 152-layer ResNet has a single-model top-5 validation error of 4.49%. This single-model result outperforms all previous ensemble results (Table 5). We combine six models of different depth to form an ensemble (only with two 152-layer ones at the time of submitting). This leads to **3.57%** top-5 error on the test set (Table 5). *This entry won the 1st place in ILSVRC 2015.*

4.2. CIFAR-10 and Analysis

We conducted more studies on the CIFAR-10 dataset [20], which consists of 50k training images and 10k testing images in 10 classes. We present experiments trained on the training set and evaluated on the test set. Our focus is on the behaviors of extremely deep networks, but not on pushing the state-of-the-art results, so we intentionally use simple architectures as follows.

The plain/residual architectures follow the form in Fig. 3 (middle/right). The network inputs are 32×32 images, with the per-pixel mean subtracted. The first layer is 3×3 convolutions. Then we use a stack of $6n$ layers with 3×3 convolutions on the feature maps of sizes $\{32, 16, 8\}$ respectively, with $2n$ layers for each feature map size. The numbers of filters are $\{16, 32, 64\}$ respectively. The subsampling is performed by convolutions with a stride of 2. The network ends with a global average pooling, a 10-way fully-connected layer, and softmax. There are totally $6n$+2 stacked weighted layers. The following table summarizes the architecture:

output map size	32×32	16×16	8×8
# layers	1+2n	2n	2n
# filters	16	32	64

When shortcut connections are used, they are connected to the pairs of 3×3 layers (totally $3n$ shortcuts). On this dataset we use identity shortcuts in all cases (*i.e.*, option A), so our residual models have exactly the same depth, width, and number of parameters as the plain counterparts.

method			error (%)
Maxout [9]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [34]	19	2.5M	8.39
Highway [41, 42]	19	2.3M	7.54 (7.72±0.16)
Highway [41, 42]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Table 6. Classification error on the **CIFAR-10** test set. All methods are with data augmentation. For ResNet-110, we run it 5 times and show "best (mean±std)" as in [42].

We use a weight decay of 0.0001 and momentum of 0.9, and adopt the weight initialization in [12] and BN [16] but with no dropout. These models are trained with a mini-batch size of 128 on two GPUs. We start with a learning rate of 0.1, divide it by 10 at 32k and 48k iterations, and terminate training at 64k iterations, which is determined on a 45k/5k train/val split. We follow the simple data augmentation in [24] for training: 4 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip. For testing, we only evaluate the single view of the original 32×32 image.

We compare $n = \{3, 5, 7, 9\}$, leading to 20, 32, 44, and 56-layer networks. Fig. 6 (left) shows the behaviors of the plain nets. The deep plain nets suffer from increased depth, and exhibit higher training error when going deeper. This phenomenon is similar to that on ImageNet (Fig. 4, left) and on MNIST (see [41]), suggesting that such an optimization difficulty is a fundamental problem.

Fig. 6 (middle) shows the behaviors of ResNets. Also similar to the ImageNet cases (Fig. 4, right), our ResNets manage to overcome the optimization difficulty and demonstrate accuracy gains when the depth increases.

We further explore $n = 18$ that leads to a 110-layer ResNet. In this case, we find that the initial learning rate of 0.1 is slightly too large to start converging[5]. So we use 0.01 to warm up the training until the training error is below 80% (about 400 iterations), and then go back to 0.1 and continue training. The rest of the learning schedule is as done previously. This 110-layer network converges well (Fig. 6, middle). It has *fewer* parameters than other deep and thin

[5]With an initial learning rate of 0.1, it starts converging (<90% error) after several epochs, but still reaches similar accuracy.

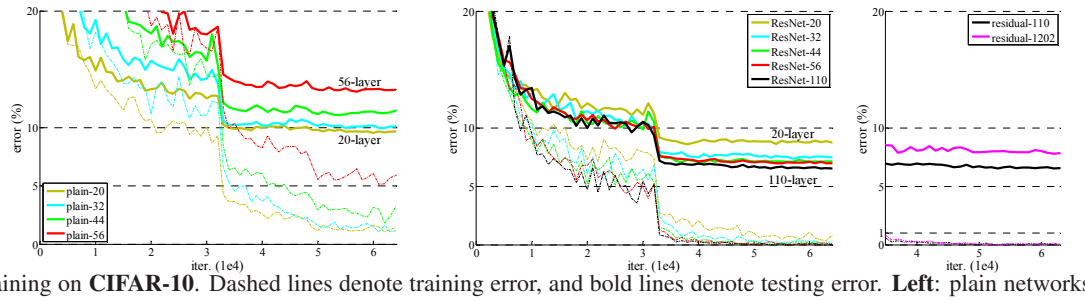


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

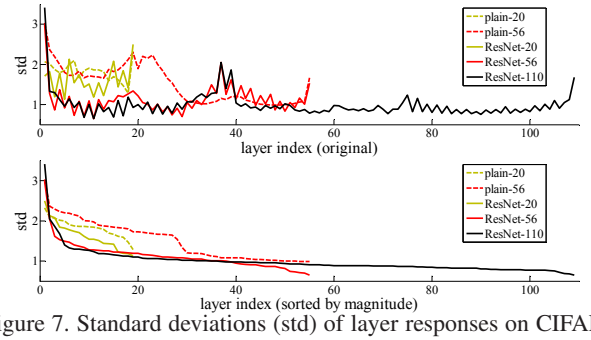


Figure 7. Standard deviations (std) of layer responses on CIFAR-10. The responses are the outputs of each 3×3 layer, after BN and before nonlinearity. **Top**: the layers are shown in their original order. **Bottom**: the responses are ranked in descending order.

networks such as FitNet [34] and Highway [41] (Table 6), yet is among the state-of-the-art results (6.43%, Table 6).

Analysis of Layer Responses. Fig. 7 shows the standard deviations (std) of the layer responses. The responses are the outputs of each 3×3 layer, after BN and before other nonlinearity (ReLU/addition). For ResNets, this analysis reveals the response strength of the residual functions. Fig. 7 shows that ResNets have generally smaller responses than their plain counterparts. These results support our basic motivation (Sec.3.1) that the residual functions might be generally closer to zero than the non-residual functions. We also notice that the deeper ResNet has smaller magnitudes of responses, as evidenced by the comparisons among ResNet-20, 56, and 110 in Fig. 7. When there are more layers, an individual layer of ResNets tends to modify the signal less.

Exploring Over 1000 layers. We explore an aggressively deep model of over 1000 layers. We set $n = 200$ that leads to a 1202-layer network, which is trained as described above. Our method shows *no optimization difficulty*, and this 10^3-layer network is able to achieve *training error* <0.1% (Fig. 6, right). Its test error is still fairly good (7.93%, Table 6).

But there are still open problems on such aggressively deep models. The testing result of this 1202-layer network is worse than that of our 110-layer network, although both have similar training error. We argue that this is because of overfitting. The 1202-layer network may be unnecessarily large (19.4M) for this small dataset. Strong regularization such as maxout [9] or dropout [13] is applied to obtain the best results ([9, 25, 24, 34]) on this dataset. In this paper, we use no maxout/dropout and just simply impose regularization via deep and thin architectures by design, without distracting from the focus on the difficulties of optimization. But combining with stronger regularization may improve results, which we will study in the future.

training data	07+12	07++12
test data	VOC 07 test	VOC 12 test
VGG-16	73.2	70.4
ResNet-101	**76.4**	**73.8**

Table 7. Object detection mAP (%) on the PASCAL VOC 2007/2012 test sets using **baseline** Faster R-CNN. See also appendix for better results.

metric	mAP@.5	mAP@[.5, .95]
VGG-16	41.5	21.2
ResNet-101	**48.4**	**27.2**

Table 8. Object detection mAP (%) on the COCO validation set using **baseline** Faster R-CNN. See also appendix for better results.

4.3. Object Detection on PASCAL and MS COCO

Our method has good generalization performance on other recognition tasks. Table 7 and 8 show the object detection baseline results on PASCAL VOC 2007 and 2012 [5] and COCO [26]. We adopt *Faster R-CNN* [32] as the detection method. Here we are interested in the improvements of replacing VGG-16 [40] with ResNet-101. The detection implementation (see appendix) of using both models is the same, so the gains can only be attributed to better networks. Most remarkably, on the challenging COCO dataset we obtain a 6.0% increase in COCO's standard metric (mAP@[.5, .95]), which is a 28% relative improvement. This gain is solely due to the learned representations.

Based on deep residual nets, we won the 1st places in several tracks in ILSVRC & COCO 2015 competitions: ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. The details are in the appendix.

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[2] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] W. L. Briggs, S. F. McCormick, et al. *A Multigrid Tutorial*. Siam, 2000.

[4] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, 2011.

[5] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *IJCV*, pages 303–338, 2010.

[6] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[9] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv:1302.4389*, 2013.

[10] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[13] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[14] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma thesis, TU Munich*, 1991.

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[17] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33, 2011.

[18] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *TPAMI*, 2012.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[23] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

[24] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv:1409.5185*, 2014.

[25] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*. 2014.

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[28] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NIPS*, 2014.

[29] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[30] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.

[31] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, 2012.

[32] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[33] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge university press, 1996.

[34] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *arXiv:1409.0575*, 2014.

[36] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

[37] N. N. Schraudolph. Accelerated gradient descent by factor-centering decomposition. Technical report, 1998.

[38] N. N. Schraudolph. Centering neural network gradient factors. In *Neural Networks: Tricks of the Trade*, pages 207–226. Springer, 1998.

[39] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[40] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[41] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

[42] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. *1507.06228*, 2015.

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[44] R. Szeliski. Fast surface interpolation using hierarchical basis functions. *TPAMI*, 1990.

[45] R. Szeliski. Locally adapted hierarchical basis preconditioning. In *SIGGRAPH*, 2006.

[46] T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun. Pushing stochastic gradient towards second-order methods–backpropagation learning with transformations in nonlinearities. In *Neural Information Processing*, 2013.

[47] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[48] W. Venables and B. Ripley. Modern applied statistics with s-plus. 1999.

[49] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. In *ECCV*, 2014.

Distinctive Image Features from Scale-Invariant Keypoints

David G. Lowe
Computer Science Department
University of British Columbia
Vancouver, B.C., Canada
lowe@cs.ubc.ca

January 5, 2004

Abstract

This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through least-squares solution for consistent pose parameters. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.

Accepted for publication in the *International Journal of Computer Vision,* 2004.

1 Introduction

Image matching is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking. This paper describes image features that have many properties that make them suitable for matching differing images of an object or scene. The features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images with efficient algorithms. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition.

The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test. Following are the major stages of computation used to generate the set of image features:

1. **Scale-space extrema detection:** The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.

2. **Keypoint localization:** At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.

3. **Orientation assignment:** One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

4. **Keypoint descriptor:** The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

This approach has been named the Scale Invariant Feature Transform (SIFT), as it transforms image data into scale-invariant coordinates relative to local features.

An important aspect of this approach is that it generates large numbers of features that densely cover the image over the full range of scales and locations. A typical image of size 500x500 pixels will give rise to about 2000 stable features (although this number depends on both image content and choices for various parameters). The quantity of features is particularly important for object recognition, where the ability to detect small objects in cluttered backgrounds requires that at least 3 features be correctly matched from each object for reliable identification.

For image matching and recognition, SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors. This paper will discuss fast nearest-neighbor algorithms that can perform this computation rapidly against large databases.

The keypoint descriptors are highly distinctive, which allows a single feature to find its correct match with good probability in a large database of features. However, in a cluttered

image, many features from the background will not have any correct match in the database, giving rise to many false matches in addition to the correct ones. The correct matches can be filtered from the full set of matches by identifying subsets of keypoints that agree on the object and its location, scale, and orientation in the new image. The probability that several features will agree on these parameters by chance is much lower than the probability that any individual feature match will be in error. The determination of these consistent clusters can be performed rapidly by using an efficient hash table implementation of the generalized Hough transform.

Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed verification. First, a least-squared estimate is made for an affine approximation to the object pose. Any other image features consistent with this pose are identified, and outliers are discarded. Finally, a detailed computation is made of the probability that a particular set of features indicates the presence of an object, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

2 Related research

The development of image matching by using a set of local interest points can be traced back to the work of Moravec (1981) on stereo matching using a corner detector. The Moravec detector was improved by Harris and Stephens (1988) to make it more repeatable under small image variations and near edges. Harris also showed its value for efficient motion tracking and 3D structure from motion recovery (Harris, 1992), and the Harris corner detector has since been widely used for many other image matching tasks. While these feature detectors are usually called corner detectors, they are not selecting just corners, but rather any image location that has large gradients in all directions at a predetermined scale.

The initial applications were to stereo and short-range motion tracking, but the approach was later extended to more difficult problems. Zhang *et al.* (1995) showed that it was possible to match Harris corners over a large image range by using a correlation window around each corner to select likely matches. Outliers were then removed by solving for a fundamental matrix describing the geometric constraints between the two views of rigid scene and removing matches that did not agree with the majority solution. At the same time, a similar approach was developed by Torr (1995) for long-range motion matching, in which geometric constraints were used to remove outliers for rigid objects moving within an image.

The ground-breaking work of Schmid and Mohr (1997) showed that invariant local feature matching could be extended to general image recognition problems in which a feature was matched against a large database of images. They also used Harris corners to select interest points, but rather than matching with a correlation window, they used a rotationally invariant descriptor of the local image region. This allowed features to be matched under arbitrary orientation change between the two images. Furthermore, they demonstrated that multiple feature matches could accomplish general recognition under occlusion and clutter by identifying consistent clusters of matched features.

The Harris corner detector is very sensitive to changes in image scale, so it does not provide a good basis for matching images of different sizes. Earlier work by the author (Lowe, 1999) extended the local feature approach to achieve scale invariance. This work also described a new local descriptor that provided more distinctive features while being less

sensitive to local image distortions such as 3D viewpoint change. This current paper provides a more in-depth development and analysis of this earlier work, while also presenting a number of improvements in stability and feature invariance.

There is a considerable body of previous research on identifying representations that are stable under scale change. Some of the first work in this area was by Crowley and Parker (1984), who developed a representation that identified peaks and ridges in scale space and linked these into a tree structure. The tree structure could then be matched between images with arbitrary scale change. More recent work on graph-based matching by Shokoufandeh, Marsic and Dickinson (1999) provides more distinctive feature descriptors using wavelet coefficients. The problem of identifying an appropriate and consistent scale for feature detection has been studied in depth by Lindeberg (1993, 1994). He describes this as a problem of scale selection, and we make use of his results below.

Recently, there has been an impressive body of work on extending local features to be invariant to full affine transformations (Baumberg, 2000; Tuytelaars and Van Gool, 2000; Mikolajczyk and Schmid, 2002; Schaffalitzky and Zisserman, 2002; Brown and Lowe, 2002). This allows for invariant matching to features on a planar surface under changes in orthographic 3D projection, in most cases by resampling the image in a local affine frame. However, none of these approaches are yet fully affine invariant, as they start with initial feature scales and locations selected in a non-affine-invariant manner due to the prohibitive cost of exploring the full affine space. The affine frames are are also more sensitive to noise than those of the scale-invariant features, so in practice the affine features have lower repeatability than the scale-invariant features unless the affine distortion is greater than about a 40 degree tilt of a planar surface (Mikolajczyk, 2002). Wider affine invariance may not be important for many applications, as training views are best taken at least every 30 degrees rotation in viewpoint (meaning that recognition is within 15 degrees of the closest training view) in order to capture non-planar changes and occlusion effects for 3D objects.

While the method to be presented in this paper is not fully affine invariant, a different approach is used in which the local descriptor allows relative feature positions to shift significantly with only small changes in the descriptor. This approach not only allows the descriptors to be reliably matched across a considerable range of affine distortion, but it also makes the features more robust against changes in 3D viewpoint for non-planar surfaces. Other advantages include much more efficient feature extraction and the ability to identify larger numbers of features. On the other hand, affine invariance is a valuable property for matching planar surfaces under very large view changes, and further research should be performed on the best ways to combine this with non-planar 3D viewpoint invariance in an efficient and stable manner.

Many other feature types have been proposed for use in recognition, some of which could be used in addition to the features described in this paper to provide further matches under differing circumstances. One class of features are those that make use of image contours or region boundaries, which should make them less likely to be disrupted by cluttered backgrounds near object boundaries. Matas *et al.*, (2002) have shown that their maximally-stable extremal regions can produce large numbers of matching features with good stability. Mikolajczyk *et al.*, (2003) have developed a new descriptor that uses local edges while ignoring unrelated nearby edges, providing the ability to find stable features even near the boundaries of narrow shapes superimposed on background clutter. Nelson and Selinger (1998) have shown good results with local features based on groupings of image contours. Similarly,

Pope and Lowe (2000) used features based on the hierarchical grouping of image contours, which are particularly useful for objects lacking detailed texture.

The history of research on visual recognition contains work on a diverse set of other image properties that can be used as feature measurements. Carneiro and Jepson (2002) describe phase-based local features that represent the phase rather than the magnitude of local spatial frequencies, which is likely to provide improved invariance to illumination. Schiele and Crowley (2000) have proposed the use of multidimensional histograms summarizing the distribution of measurements within image regions. This type of feature may be particularly useful for recognition of textured objects with deformable shapes. Basri and Jacobs (1997) have demonstrated the value of extracting local region boundaries for recognition. Other useful properties to incorporate include color, motion, figure-ground discrimination, region shape descriptors, and stereo depth cues. The local feature approach can easily incorporate novel feature types because extra features contribute to robustness when they provide correct matches, but otherwise do little harm other than their cost of computation. Therefore, future systems are likely to combine many feature types.

3 Detection of scale-space extrema

As described in the introduction, we will detect keypoints using a cascade filtering approach that uses efficient algorithms to identify candidate locations that are then examined in further detail. The first stage of keypoint detection is to identify locations and scales that can be repeatably assigned under differing views of the same object. Detecting locations that are invariant to scale change of the image can be accomplished by searching for stable features across all possible scales, using a continuous function of scale known as scale space (Witkin, 1983).

It has been shown by Koenderink (1984) and Lindeberg (1994) that under a variety of reasonable assumptions the only possible scale-space kernel is the Gaussian function. Therefore, the scale space of an image is defined as a function, $L(x, y, \sigma)$, that is produced from the convolution of a variable-scale Gaussian, $G(x, y, \sigma)$, with an input image, $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

where $*$ is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}.$$

To efficiently detect stable keypoint locations in scale space, we have proposed (Lowe, 1999) using scale-space extrema in the difference-of-Gaussian function convolved with the image, $D(x, y, \sigma)$, which can be computed from the difference of two nearby scales separated by a constant multiplicative factor k:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned} \quad (1)$$

There are a number of reasons for choosing this function. First, it is a particularly efficient function to compute, as the smoothed images, L, need to be computed in any case for scale space feature description, and D can therefore be computed by simple image subtraction.

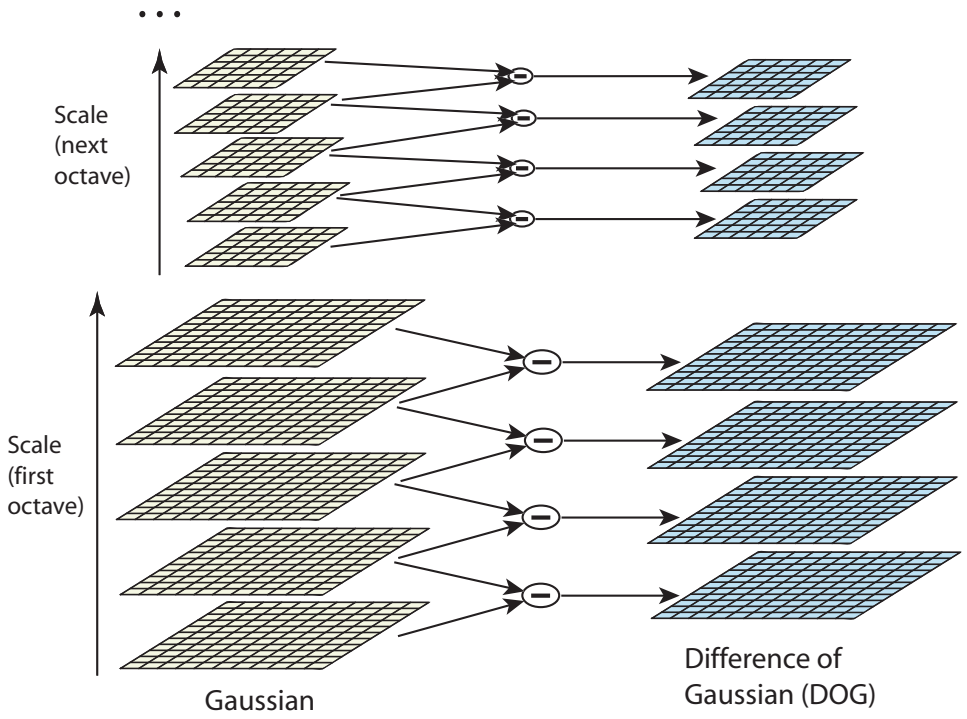


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

In addition, the difference-of-Gaussian function provides a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$, as studied by Lindeberg (1994). Lindeberg showed that the normalization of the Laplacian with the factor σ^2 is required for true scale invariance. In detailed experimental comparisons, Mikolajczyk (2002) found that the maxima and minima of $\sigma^2\nabla^2 G$ produce the most stable image features compared to a range of other possible image functions, such as the gradient, Hessian, or Harris corner function.

The relationship between D and $\sigma^2\nabla^2 G$ can be understood from the heat diffusion equation (parameterized in terms of σ rather than the more usual $t = \sigma^2$):

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G.$$

From this, we see that $\nabla^2 G$ can be computed from the finite difference approximation to $\partial G/\partial\sigma$, using the difference of nearby scales at $k\sigma$ and σ:

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

and therefore,

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G.$$

This shows that when the difference-of-Gaussian function has scales differing by a constant factor it already incorporates the σ^2 scale normalization required for the scale-invariant

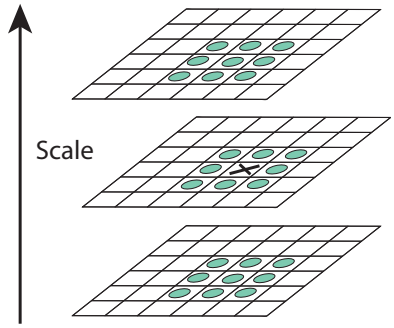


Figure 2: Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

Laplacian. The factor $(k-1)$ in the equation is a constant over all scales and therefore does not influence extrema location. The approximation error will go to zero as k goes to 1, but in practice we have found that the approximation has almost no impact on the stability of extrema detection or localization for even significant differences in scale, such as $k = \sqrt{2}$.

An efficient approach to construction of $D(x, y, \sigma)$ is shown in Figure 1. The initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space, shown stacked in the left column. We choose to divide each octave of scale space (i.e., doubling of σ) into an integer number, s, of intervals, so $k = 2^{1/s}$. We must produce $s + 3$ images in the stack of blurred images for each octave, so that final extrema detection covers a complete octave. Adjacent image scales are subtracted to produce the difference-of-Gaussian images shown on the right. Once a complete octave has been processed, we resample the Gaussian image that has twice the initial value of σ (it will be 2 images from the top of the stack) by taking every second pixel in each row and column. The accuracy of sampling relative to σ is no different than for the start of the previous octave, while computation is greatly reduced.

3.1 Local extrema detection

In order to detect the local maxima and minima of $D(x, y, \sigma)$, each sample point is compared to its eight neighbors in the current image and nine neighbors in the scale above and below (see Figure 2). It is selected only if it is larger than all of these neighbors or smaller than all of them. The cost of this check is reasonably low due to the fact that most sample points will be eliminated following the first few checks.

An important issue is to determine the frequency of sampling in the image and scale domains that is needed to reliably detect the extrema. Unfortunately, it turns out that there is no minimum spacing of samples that will detect all extrema, as the extrema can be arbitrarily close together. This can be seen by considering a white circle on a black background, which will have a single scale space maximum where the circular positive central region of the difference-of-Gaussian function matches the size and location of the circle. For a very elongated ellipse, there will be two maxima near each end of the ellipse. As the locations of maxima are a continuous function of the image, for some ellipse with intermediate elongation there will be a transition from a single maximum to two, with the maxima arbitrarily close to

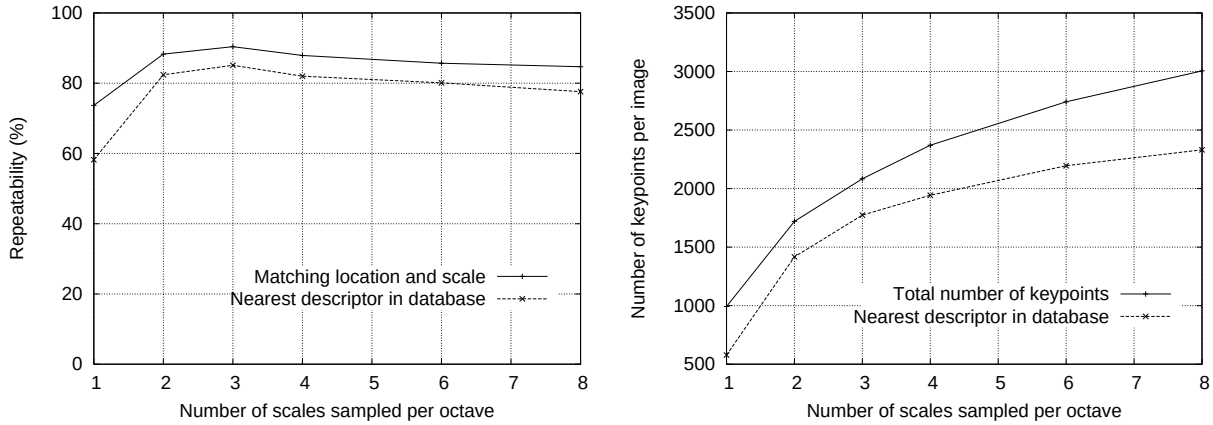


Figure 3: The top line of the first graph shows the percent of keypoints that are repeatably detected at the same location and scale in a transformed image as a function of the number of scales sampled per octave. The lower line shows the percent of keypoints that have their descriptors correctly matched to a large database. The second graph shows the total number of keypoints detected in a typical image as a function of the number of scale samples.

each other near the transition.

Therefore, we must settle for a solution that trades off efficiency with completeness. In fact, as might be expected and is confirmed by our experiments, extrema that are close together are quite unstable to small perturbations of the image. We can determine the best choices experimentally by studying a range of sampling frequencies and using those that provide the most reliable results under a realistic simulation of the matching task.

3.2 Frequency of sampling in scale

The experimental determination of sampling frequency that maximizes extrema stability is shown in Figures 3 and 4. These figures (and most other simulations in this paper) are based on a matching task using a collection of 32 real images drawn from a diverse range, including outdoor scenes, human faces, aerial photographs, and industrial images (the image domain was found to have almost no influence on any of the results). Each image was then subject to a range of transformations, including rotation, scaling, affine stretch, change in brightness and contrast, and addition of image noise. Because the changes were synthetic, it was possible to precisely predict where each feature in an original image should appear in the transformed image, allowing for measurement of correct repeatability and positional accuracy for each feature.

Figure 3 shows these simulation results used to examine the effect of varying the number of scales per octave at which the image function is sampled prior to extrema detection. In this case, each image was resampled following rotation by a random angle and scaling by a random amount between 0.2 of 0.9 times the original size. Keypoints from the reduced resolution image were matched against those from the original image so that the scales for all keypoints would be be present in the matched image. In addition, 1% image noise was added, meaning that each pixel had a random number added from the uniform interval [-0.01,0.01] where pixel values are in the range [0,1] (equivalent to providing slightly less than 6 bits of accuracy for image pixels).

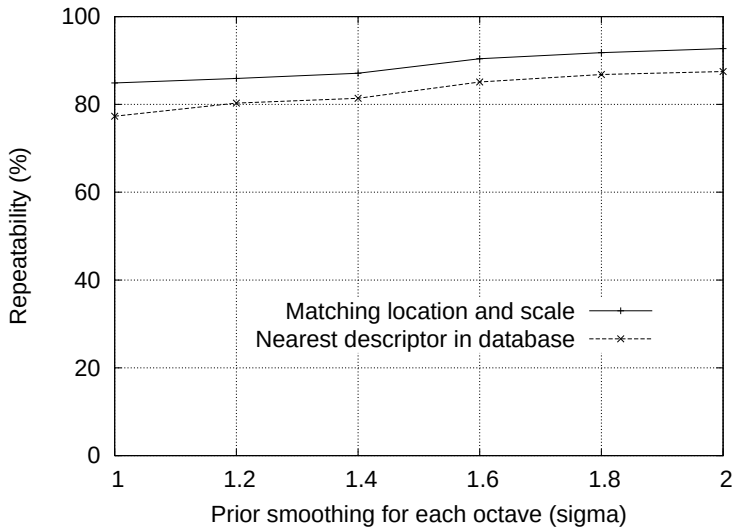


Figure 4: The top line in the graph shows the percent of keypoint locations that are repeatably detected in a transformed image as a function of the prior image smoothing for the first level of each octave. The lower line shows the percent of descriptors correctly matched against a large database.

The top line in the first graph of Figure 3 shows the percent of keypoints that are detected at a matching location and scale in the transformed image. For all examples in this paper, we define a matching scale as being within a factor of $\sqrt{2}$ of the correct scale, and a matching location as being within σ pixels, where σ is the scale of the keypoint (defined from equation (1) as the standard deviation of the smallest Gaussian used in the difference-of-Gaussian function). The lower line on this graph shows the number of keypoints that are correctly matched to a database of 40,000 keypoints using the nearest-neighbor matching procedure to be described in Section 6 (this shows that once the keypoint is repeatably located, it is likely to be useful for recognition and matching tasks). As this graph shows, the highest repeatability is obtained when sampling 3 scales per octave, and this is the number of scale samples used for all other experiments throughout this paper.

It might seem surprising that the repeatability does not continue to improve as more scales are sampled. The reason is that this results in many more local extrema being detected, but these extrema are on average less stable and therefore are less likely to be detected in the transformed image. This is shown by the second graph in Figure 3, which shows the average number of keypoints detected and correctly matched in each image. The number of keypoints rises with increased sampling of scales and the total number of correct matches also rises. Since the success of object recognition often depends more on the quantity of correctly matched keypoints, as opposed to their percentage correct matching, for many applications it will be optimal to use a larger number of scale samples. However, the cost of computation also rises with this number, so for the experiments in this paper we have chosen to use just 3 scale samples per octave.

To summarize, these experiments show that the scale-space difference-of-Gaussian function has a large number of extrema and that it would be very expensive to detect them all. Fortunately, we can detect the most stable and useful subset even with a coarse sampling of scales.

3.3 Frequency of sampling in the spatial domain

Just as we determined the frequency of sampling per octave of scale space, so we must determine the frequency of sampling in the image domain relative to the scale of smoothing. Given that extrema can be arbitrarily close together, there will be a similar trade-off between sampling frequency and rate of detection. Figure 4 shows an experimental determination of the amount of prior smoothing, σ, that is applied to each image level before building the scale space representation for an octave. Again, the top line is the repeatability of keypoint detection, and the results show that the repeatability continues to increase with σ. However, there is a cost to using a large σ in terms of efficiency, so we have chosen to use $\sigma = 1.6$, which provides close to optimal repeatability. This value is used throughout this paper and was used for the results in Figure 3.

Of course, if we pre-smooth the image before extrema detection, we are effectively discarding the highest spatial frequencies. Therefore, to make full use of the input, the image can be expanded to create more sample points than were present in the original. We double the size of the input image using linear interpolation prior to building the first level of the pyramid. While the equivalent operation could effectively have been performed by using sets of subpixel-offset filters on the original image, the image doubling leads to a more efficient implementation. We assume that the original image has a blur of at least $\sigma = 0.5$ (the minimum needed to prevent significant aliasing), and that therefore the doubled image has $\sigma = 1.0$ relative to its new pixel spacing. This means that little additional smoothing is needed prior to creation of the first octave of scale space. The image doubling increases the number of stable keypoints by almost a factor of 4, but no significant further improvements were found with a larger expansion factor.

4 Accurate keypoint localization

Once a keypoint candidate has been found by comparing a pixel to its neighbors, the next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge.

The initial implementation of this approach (Lowe, 1999) simply located keypoints at the location and scale of the central sample point. However, recently Brown has developed a method (Brown and Lowe, 2002) for fitting a 3D quadratic function to the local sample points to determine the interpolated location of the maximum, and his experiments showed that this provides a substantial improvement to matching and stability. His approach uses the Taylor expansion (up to the quadratic terms) of the scale-space function, $D(x, y, \sigma)$, shifted so that the origin is at the sample point:

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^\mathbf{T}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{2}$$

where D and its derivatives are evaluated at the sample point and $\mathbf{x} = (x, y, \sigma)^T$ is the offset from this point. The location of the extremum, $\hat{\mathbf{x}}$, is determined by taking the derivative of this function with respect to $\mathbf{x}$ and setting it to zero, giving

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}}. \tag{3}$$

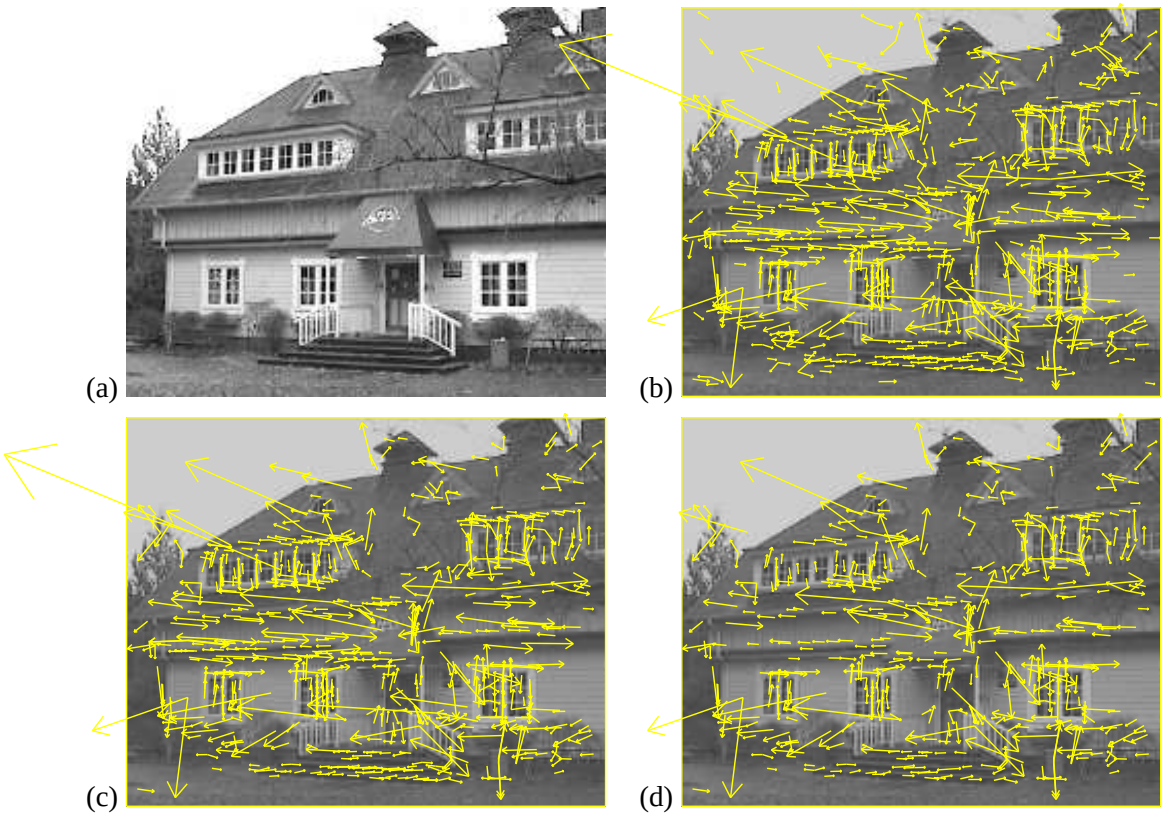


Figure 5: This figure shows the stages of keypoint selection. (a) The 233x189 pixel original image. (b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. (c) After applying a threshold on minimum contrast, 729 keypoints remain. (d) The final 536 keypoints that remain following an additional threshold on ratio of principal curvatures.

As suggested by Brown, the Hessian and derivative of D are approximated by using differences of neighboring sample points. The resulting 3x3 linear system can be solved with minimal cost. If the offset $\hat{\mathbf{x}}$ is larger than 0.5 in any dimension, then it means that the extremum lies closer to a different sample point. In this case, the sample point is changed and the interpolation performed instead about that point. The final offset $\hat{\mathbf{x}}$ is added to the location of its sample point to get the interpolated estimate for the location of the extremum.

The function value at the extremum, $D(\hat{\mathbf{x}})$, is useful for rejecting unstable extrema with low contrast. This can be obtained by substituting equation (3) into (2), giving

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D}{\partial \mathbf{x}}^T \hat{\mathbf{x}}.$$

For the experiments in this paper, all extrema with a value of $|D(\hat{\mathbf{x}})|$ less than 0.03 were discarded (as before, we assume image pixel values in the range [0,1]).

Figure 5 shows the effects of keypoint selection on a natural image. In order to avoid too much clutter, a low-resolution 233 by 189 pixel image is used and keypoints are shown as vectors giving the location, scale, and orientation of each keypoint (orientation assignment is described below). Figure 5 (a) shows the original image, which is shown at reduced contrast behind the subsequent figures. Figure 5 (b) shows the 832 keypoints at all detected maxima

and minima of the difference-of-Gaussian function, while (c) shows the 729 keypoints that remain following removal of those with a value of $|D(\hat{\mathbf{x}})|$ less than 0.03. Part (d) will be explained in the following section.

4.1 Eliminating edge responses

For stability, it is not sufficient to reject keypoints with low contrast. The difference-of-Gaussian function will have a strong response along edges, even if the location along the edge is poorly determined and therefore unstable to small amounts of noise.

A poorly defined peak in the difference-of-Gaussian function will have a large principal curvature across the edge but a small one in the perpendicular direction. The principal curvatures can be computed from a 2x2 Hessian matrix, $\mathbf{H}$, computed at the location and scale of the keypoint:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{4}$$

The derivatives are estimated by taking differences of neighboring sample points.

The eigenvalues of $\mathbf{H}$ are proportional to the principal curvatures of D. Borrowing from the approach used by Harris and Stephens (1988), we can avoid explicitly computing the eigenvalues, as we are only concerned with their ratio. Let α be the eigenvalue with the largest magnitude and β be the smaller one. Then, we can compute the sum of the eigenvalues from the trace of $\mathbf{H}$ and their product from the determinant:

$$\begin{aligned} \mathrm{Tr}(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta, \\ \mathrm{Det}(\mathbf{H}) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta. \end{aligned}$$

In the unlikely event that the determinant is negative, the curvatures have different signs so the point is discarded as not being an extremum. Let r be the ratio between the largest magnitude eigenvalue and the smaller one, so that $\alpha = r\beta$. Then,

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

which depends only on the ratio of the eigenvalues rather than their individual values. The quantity $(r+1)^2/r$ is at a minimum when the two eigenvalues are equal and it increases with r. Therefore, to check that the ratio of principal curvatures is below some threshold, r, we only need to check

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}.$$

This is very efficient to compute, with less than 20 floating point operations required to test each keypoint. The experiments in this paper use a value of $r = 10$, which eliminates keypoints that have a ratio between the principal curvatures greater than 10. The transition from Figure 5 (c) to (d) shows the effects of this operation.

5 Orientation assignment

By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation. This approach contrasts with the orientation invariant descriptors of Schmid and Mohr (1997), in which each image property is based on a rotationally invariant measure. The disadvantage of that approach is that it limits the descriptors that can be used and discards image information by not requiring all measures to be based on a consistent rotation.

Following experimentation with a number of approaches to assigning a local orientation, the following approach was found to give the most stable results. The scale of the keypoint is used to select the Gaussian smoothed image, L, with the closest scale, so that all computations are performed in a scale-invariant manner. For each image sample, $L(x, y)$, at this scale, the gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, is precomputed using pixel differences:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint.

Peaks in the orientation histogram correspond to dominant directions of local gradients. The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation. Therefore, for locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale but different orientations. Only about 15% of points are assigned multiple orientations, but these contribute significantly to the stability of matching. Finally, a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy.

Figure 6 shows the experimental stability of location, scale, and orientation assignment under differing amounts of image noise. As before the images are rotated and scaled by random amounts. The top line shows the stability of keypoint location and scale assignment. The second line shows the stability of matching when the orientation assignment is also required to be within 15 degrees. As shown by the gap between the top two lines, the orientation assignment remains accurate 95% of the time even after addition of $\pm 10\%$ pixel noise (equivalent to a camera providing less than 3 bits of precision). The measured variance of orientation for the correct matches is about 2.5 degrees, rising to 3.9 degrees for 10% noise. The bottom line in Figure 6 shows the final accuracy of correctly matching a keypoint descriptor to a database of 40,000 keypoints (to be discussed below). As this graph shows, the SIFT features are resistant to even large amounts of pixel noise, and the major cause of error is the initial location and scale detection.

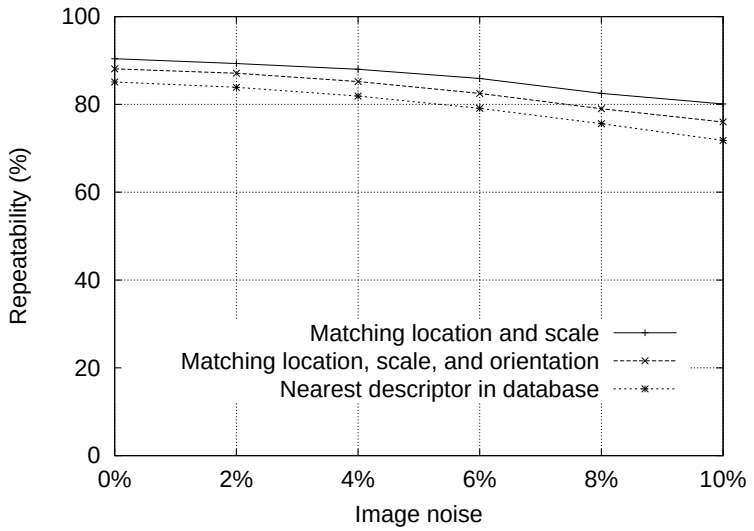


Figure 6: The top line in the graph shows the percent of keypoint locations and scales that are repeatably detected as a function of pixel noise. The second line shows the repeatability after also requiring agreement in orientation. The bottom line shows the final percent of descriptors correctly matched to a large database.

6 The local image descriptor

The previous operations have assigned an image location, scale, and orientation to each keypoint. These parameters impose a repeatable local 2D coordinate system in which to describe the local image region, and therefore provide invariance to these parameters. The next step is to compute a descriptor for the local image region that is highly distinctive yet is as invariant as possible to remaining variations, such as change in illumination or 3D viewpoint.

One obvious approach would be to sample the local image intensities around the keypoint at the appropriate scale, and to match these using a normalized correlation measure. However, simple correlation of image patches is highly sensitive to changes that cause misregistration of samples, such as affine or 3D viewpoint change or non-rigid deformations. A better approach has been demonstrated by Edelman, Intrator, and Poggio (1997). Their proposed representation was based upon a model of biological vision, in particular of complex neurons in primary visual cortex. These complex neurons respond to a gradient at a particular orientation and spatial frequency, but the location of the gradient on the retina is allowed to shift over a small receptive field rather than being precisely localized. Edelman *et al.* hypothesized that the function of these complex neurons was to allow for matching and recognition of 3D objects from a range of viewpoints. They have performed detailed experiments using 3D computer models of object and animal shapes which show that matching gradients while allowing for shifts in their position results in much better classification under 3D rotation. For example, recognition accuracy for 3D objects rotated in depth by 20 degrees increased from 35% for correlation of gradients to 94% using the complex cell model. Our implementation described below was inspired by this idea, but allows for positional shift using a different computational mechanism.

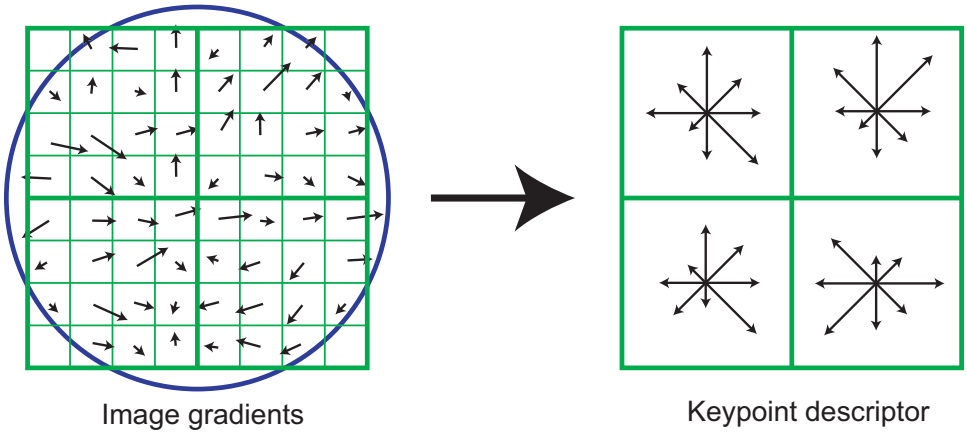


Figure 7: A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2x2 descriptor array computed from an 8x8 set of samples, whereas the experiments in this paper use 4x4 descriptors computed from a 16x16 sample array.

6.1 Descriptor representation

Figure 7 illustrates the computation of the keypoint descriptor. First the image gradient magnitudes and orientations are sampled around the keypoint location, using the scale of the keypoint to select the level of Gaussian blur for the image. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. For efficiency, the gradients are precomputed for all levels of the pyramid as described in Section 5. These are illustrated with small arrows at each sample location on the left side of Figure 7.

A Gaussian weighting function with σ equal to one half the width of the descriptor window is used to assign a weight to the magnitude of each sample point. This is illustrated with a circular window on the left side of Figure 7, although, of course, the weight falls off smoothly. The purpose of this Gaussian window is to avoid sudden changes in the descriptor with small changes in the position of the window, and to give less emphasis to gradients that are far from the center of the descriptor, as these are most affected by misregistration errors.

The keypoint descriptor is shown on the right side of Figure 7. It allows for significant shift in gradient positions by creating orientation histograms over 4x4 sample regions. The figure shows eight directions for each orientation histogram, with the length of each arrow corresponding to the magnitude of that histogram entry. A gradient sample on the left can shift up to 4 sample positions while still contributing to the same histogram on the right, thereby achieving the objective of allowing for larger local positional shifts.

It is important to avoid all boundary affects in which the descriptor abruptly changes as a sample shifts smoothly from being within one histogram to another or from one orientation to another. Therefore, trilinear interpolation is used to distribute the value of each gradient sample into adjacent histogram bins. In other words, each entry into a bin is multiplied by a weight of $1 - d$ for each dimension, where d is the distance of the sample from the central value of the bin as measured in units of the histogram bin spacing.

The descriptor is formed from a vector containing the values of all the orientation histogram entries, corresponding to the lengths of the arrows on the right side of Figure 7. The figure shows a 2x2 array of orientation histograms, whereas our experiments below show that the best results are achieved with a 4x4 array of histograms with 8 orientation bins in each. Therefore, the experiments in this paper use a 4x4x8 = 128 element feature vector for each keypoint.

Finally, the feature vector is modified to reduce the effects of illumination change. First, the vector is normalized to unit length. A change in image contrast in which each pixel value is multiplied by a constant will multiply gradients by the same constant, so this contrast change will be canceled by vector normalization. A brightness change in which a constant is added to each image pixel will not affect the gradient values, as they are computed from pixel differences. Therefore, the descriptor is invariant to affine changes in illumination. However, non-linear illumination changes can also occur due to camera saturation or due to illumination changes that affect 3D surfaces with differing orientations by different amounts. These effects can cause a large change in relative magnitudes for some gradients, but are less likely to affect the gradient orientations. Therefore, we reduce the influence of large gradient magnitudes by thresholding the values in the unit feature vector to each be no larger than 0.2, and then renormalizing to unit length. This means that matching the magnitudes for large gradients is no longer as important, and that the distribution of orientations has greater emphasis. The value of 0.2 was determined experimentally using images containing differing illuminations for the same 3D objects.

6.2 Descriptor testing

There are two parameters that can be used to vary the complexity of the descriptor: the number of orientations, r, in the histograms, and the width, n, of the $n \times n$ array of orientation histograms. The size of the resulting descriptor vector is rn^2. As the complexity of the descriptor grows, it will be able to discriminate better in a large database, but it will also be more sensitive to shape distortions and occlusion.

Figure 8 shows experimental results in which the number of orientations and size of the descriptor were varied. The graph was generated for a viewpoint transformation in which a planar surface is tilted by 50 degrees away from the viewer and 4% image noise is added. This is near the limits of reliable matching, as it is in these more difficult cases that descriptor performance is most important. The results show the percent of keypoints that find a correct match to the single closest neighbor among a database of 40,000 keypoints. The graph shows that a single orientation histogram ($n = 1$) is very poor at discriminating, but the results continue to improve up to a 4x4 array of histograms with 8 orientations. After that, adding more orientations or a larger descriptor can actually hurt matching by making the descriptor more sensitive to distortion. These results were broadly similar for other degrees of viewpoint change and noise, although in some simpler cases discrimination continued to improve (from already high levels) with 5x5 and higher descriptor sizes. Throughout this paper we use a 4x4 descriptor with 8 orientations, resulting in feature vectors with 128 dimensions. While the dimensionality of the descriptor may seem high, we have found that it consistently performs better than lower-dimensional descriptors on a range of matching tasks and that the computational cost of matching remains low when using the approximate nearest-neighbor methods described below.

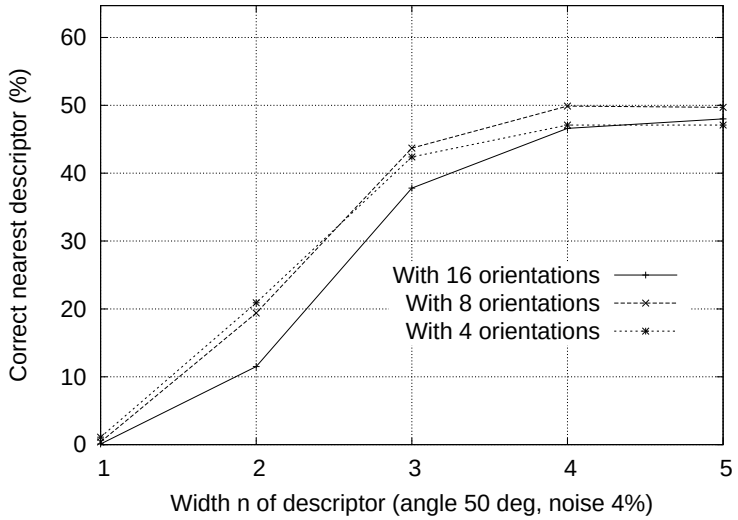


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and addition of 4% noise.

6.3 Sensitivity to affine change

The sensitivity of the descriptor to affine change is examined in Figure 9. The graph shows the reliability of keypoint location and scale selection, orientation assignment, and nearest-neighbor matching to a database as a function of rotation in depth of a plane away from a viewer. It can be seen that each stage of computation has reduced repeatability with increasing affine distortion, but that the final matching accuracy remains above 50% out to a 50 degree change in viewpoint.

To achieve reliable matching over a wider viewpoint angle, one of the affine-invariant detectors could be used to select and resample image regions, as discussed in Section 2. As mentioned there, none of these approaches is truly affine-invariant, as they all start from initial feature locations determined in a non-affine-invariant manner. In what appears to be the most affine-invariant method, Mikolajczyk (2002) has proposed and run detailed experiments with the Harris-affine detector. He found that its keypoint repeatability is below that given here out to about a 50 degree viewpoint angle, but that it then retains close to 40% repeatability out to an angle of 70 degrees, which provides better performance for extreme affine changes. The disadvantages are a much higher computational cost, a reduction in the number of keypoints, and poorer stability for small affine changes due to errors in assigning a consistent affine frame under noise. In practice, the allowable range of rotation for 3D objects is considerably less than for planar surfaces, so affine invariance is usually not the limiting factor in the ability to match across viewpoint change. If a wide range of affine invariance is desired, such as for a surface that is known to be planar, then a simple solution is to adopt the approach of Pritchard and Heidrich (2003) in which additional SIFT features are generated from 4 affine-transformed versions of the training image corresponding to 60 degree viewpoint changes. This allows for the use of standard SIFT features with no additional cost when processing the image to be recognized, but results in an increase in the size of the feature database by a factor of 3.

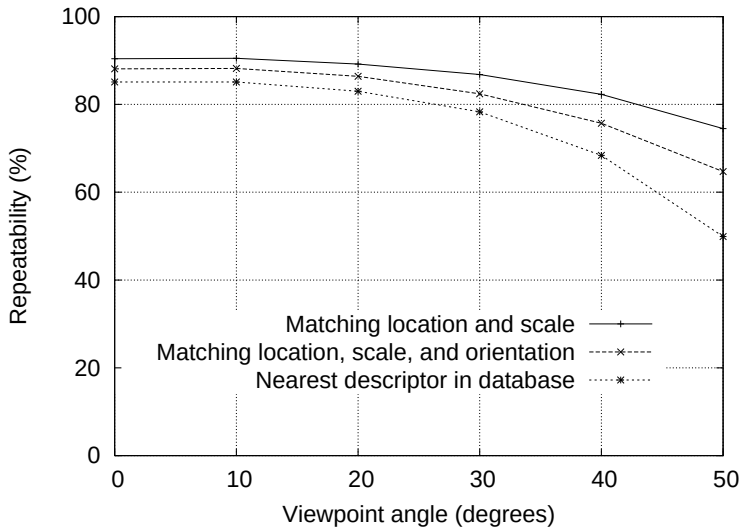


Figure 9: This graph shows the stability of detection for keypoint location, orientation, and final matching to a database as a function of affine distortion. The degree of affine distortion is expressed in terms of the equivalent viewpoint rotation in depth for a planar surface.

6.4 Matching to large databases

An important remaining issue for measuring the distinctiveness of features is how the reliability of matching varies as a function of the number of features in the database being matched. Most of the examples in this paper are generated using a database of 32 images with about 40,000 keypoints. Figure 10 shows how the matching reliability varies as a function of database size. This figure was generated using a larger database of 112 images, with a viewpoint depth rotation of 30 degrees and 2% image noise in addition to the usual random image rotation and scale change.

The dashed line shows the portion of image features for which the nearest neighbor in the database was the correct match, as a function of database size shown on a logarithmic scale. The leftmost point is matching against features from only a single image while the rightmost point is selecting matches from a database of all features from the 112 images. It can be seen that matching reliability does decrease as a function of the number of distractors, yet all indications are that many correct matches will continue to be found out to very large database sizes.

The solid line is the percentage of keypoints that were identified at the correct matching location and orientation in the transformed image, so it is only these points that have any chance of having matching descriptors in the database. The reason this line is flat is that the test was run over the full database for each value, while only varying the portion of the database used for distractors. It is of interest that the gap between the two lines is small, indicating that matching failures are due more to issues with initial feature localization and orientation assignment than to problems with feature distinctiveness, even out to large database sizes.

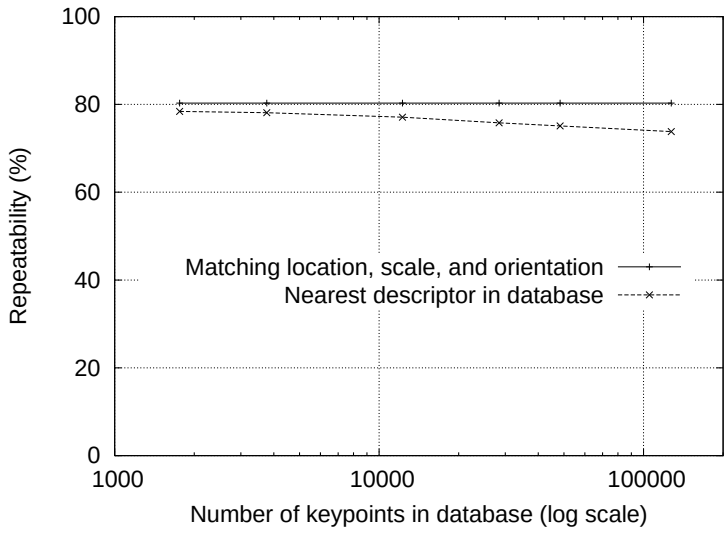


Figure 10: The dashed line shows the percent of keypoints correctly matched to a database as a function of database size (using a logarithmic scale). The solid line shows the percent of keypoints assigned the correct location, scale, and orientation. Images had random scale and rotation changes, an affine transform of 30 degrees, and image noise of 2% added prior to matching.

7 Application to object recognition

The major topic of this paper is the derivation of distinctive invariant keypoints, as described above. To demonstrate their application, we will now give a brief description of their use for object recognition in the presence of clutter and occlusion. More details on applications of these features to recognition are available in other papers (Lowe, 1999; Lowe, 2001; Se, Lowe and Little, 2002).

Object recognition is performed by first matching each keypoint independently to the database of keypoints extracted from training images. Many of these initial matches will be incorrect due to ambiguous features or features that arise from background clutter. Therefore, clusters of at least 3 features are first identified that agree on an object and its pose, as these clusters have a much higher probability of being correct than individual feature matches. Then, each cluster is checked by performing a detailed geometric fit to the model, and the result is used to accept or reject the interpretation.

7.1 Keypoint matching

The best candidate match for each keypoint is found by identifying its nearest neighbor in the database of keypoints from training images. The nearest neighbor is defined as the keypoint with minimum Euclidean distance for the invariant descriptor vector as was described in Section 6.

However, many features from an image will not have any correct match in the training database because they arise from background clutter or were not detected in the training images. Therefore, it would be useful to have a way to discard features that do not have any good match to the database. A global threshold on distance to the closest feature does not perform well, as some descriptors are much more discriminative than others. A more effective measure is obtained by comparing the distance of the closest neighbor to that of the

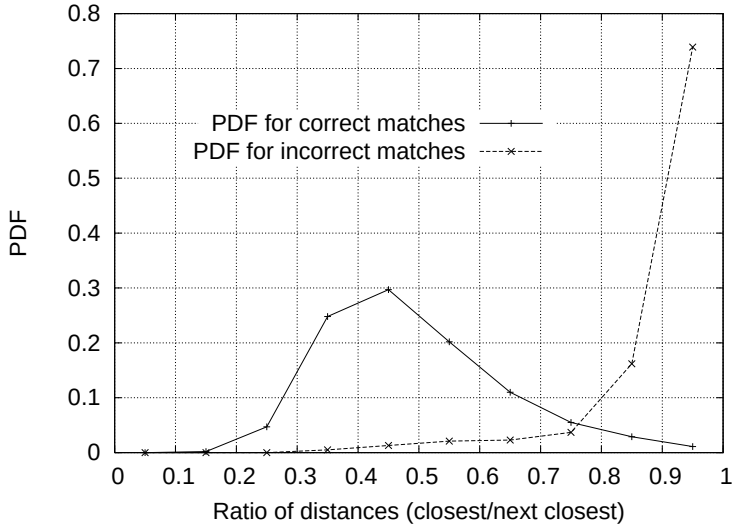


Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

second-closest neighbor. If there are multiple training images of the same object, then we define the second-closest neighbor as being the closest neighbor that is known to come from a different object than the first, such as by only using images known to contain different objects. This measure performs well because correct matches need to have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching. For false matches, there will likely be a number of other false matches within similar distances due to the high dimensionality of the feature space. We can think of the second-closest match as providing an estimate of the density of false matches within this portion of the feature space and at the same time identifying specific instances of feature ambiguity.

Figure 11 shows the value of this measure for real image data. The probability density functions for correct and incorrect matches are shown in terms of the ratio of closest to second-closest neighbors of each keypoint. Matches for which the nearest neighbor was a correct match have a PDF that is centered at a much lower ratio than that for incorrect matches. For our object recognition implementation, we reject all matches in which the distance ratio is greater than 0.8, which eliminates 90% of the false matches while discarding less than 5% of the correct matches. This figure was generated by matching images following random scale and orientation change, a depth rotation of 30 degrees, and addition of 2% image noise, against a database of 40,000 keypoints.

7.2 Efficient nearest neighbor indexing

No algorithms are known that can identify the exact nearest neighbors of points in high dimensional spaces that are any more efficient than exhaustive search. Our keypoint descriptor has a 128-dimensional feature vector, and the best algorithms, such as the k-d tree (Friedman *et al.*, 1977) provide no speedup over exhaustive search for more than about 10 dimensional spaces. Therefore, we have used an approximate algorithm, called the Best-Bin-First (BBF) algorithm (Beis and Lowe, 1997). This is approximate in the sense that it returns the closest

neighbor with high probability.

The BBF algorithm uses a modified search ordering for the k-d tree algorithm so that bins in feature space are searched in the order of their closest distance from the query location. This priority search order was first examined by Arya and Mount (1993), and they provide further study of its computational properties in (Arya *et al.*, 1998). This search order requires the use of a heap-based priority queue for efficient determination of the search order. An approximate answer can be returned with low cost by cutting off further search after a specific number of the nearest bins have been explored. In our implementation, we cut off search after checking the first 200 nearest-neighbor candidates. For a database of 100,000 keypoints, this provides a speedup over exact nearest neighbor search by about 2 orders of magnitude yet results in less than a 5% loss in the number of correct matches. One reason the BBF algorithm works particularly well for this problem is that we only consider matches in which the nearest neighbor is less than 0.8 times the distance to the second-nearest neighbor (as described in the previous section), and therefore there is no need to exactly solve the most difficult cases in which many neighbors are at very similar distances.

7.3 Clustering with the Hough transform

To maximize the performance of object recognition for small or highly occluded objects, we wish to identify objects with the fewest possible number of feature matches. We have found that reliable recognition is possible with as few as 3 features. A typical image contains 2,000 or more features which may come from many different objects as well as background clutter. While the distance ratio test described in Section 7.1 will allow us to discard many of the false matches arising from background clutter, this does not remove matches from other valid objects, and we often still need to identify correct subsets of matches containing less than 1% inliers among 99% outliers. Many well-known robust fitting methods, such as RANSAC or Least Median of Squares, perform poorly when the percent of inliers falls much below 50%. Fortunately, much better performance can be obtained by clustering features in pose space using the Hough transform (Hough, 1962; Ballard, 1981; Grimson 1990).

The Hough transform identifies clusters of features with a consistent interpretation by using each feature to vote for all object poses that are consistent with the feature. When clusters of features are found to vote for the same pose of an object, the probability of the interpretation being correct is much higher than for any single feature. Each of our keypoints specifies 4 parameters: 2D location, scale, and orientation, and each matched keypoint in the database has a record of the keypoint's parameters relative to the training image in which it was found. Therefore, we can create a Hough transform entry predicting the model location, orientation, and scale from the match hypothesis. This prediction has large error bounds, as the similarity transform implied by these 4 parameters is only an approximation to the full 6 degree-of-freedom pose space for a 3D object and also does not account for any non-rigid deformations. Therefore, we use broad bin sizes of 30 degrees for orientation, a factor of 2 for scale, and 0.25 times the maximum projected training image dimension (using the predicted scale) for location. To avoid the problem of boundary effects in bin assignment, each keypoint match votes for the 2 closest bins in each dimension, giving a total of 16 entries for each hypothesis and further broadening the pose range.

In most implementations of the Hough transform, a multi-dimensional array is used to represent the bins. However, many of the potential bins will remain empty, and it is difficult to compute the range of possible bin values due to their mutual dependence (for example,

the dependency of location discretization on the selected scale). These problems can be avoided by using a pseudo-random hash function of the bin values to insert votes into a one-dimensional hash table, in which collisions are easily detected.

7.4 Solution for affine parameters

The Hough transform is used to identify all clusters with at least 3 entries in a bin. Each such cluster is then subject to a geometric verification procedure in which a least-squares solution is performed for the best affine projection parameters relating the training image to the new image.

An affine transformation correctly accounts for 3D rotation of a planar surface under orthographic projection, but the approximation can be poor for 3D rotation of non-planar objects. A more general solution would be to solve for the fundamental matrix (Luong and Faugeras, 1996; Hartley and Zisserman, 2000). However, a fundamental matrix solution requires at least 7 point matches as compared to only 3 for the affine solution and in practice requires even more matches for good stability. We would like to perform recognition with as few as 3 feature matches, so the affine solution provides a better starting point and we can account for errors in the affine approximation by allowing for large residual errors. If we imagine placing a sphere around an object, then rotation of the sphere by 30 degrees will move no point within the sphere by more than 0.25 times the projected diameter of the sphere. For the examples of typical 3D objects used in this paper, an affine solution works well given that we allow residual errors up to 0.25 times the maximum projected dimension of the object. A more general approach is given in (Brown and Lowe, 2002), in which the initial solution is based on a similarity transform, which then progresses to solution for the fundamental matrix in those cases in which a sufficient number of matches are found.

The affine transformation of a model point $[x\ y]^T$ to an image point $[u\ v]^T$ can be written as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

where the model translation is $[t_x\ t_y]^T$ and the affine rotation, scale, and stretch are represented by the m_i parameters.

We wish to solve for the transformation parameters, so the equation above can be rewritten to gather the unknowns into a column vector:

$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \\ & & \cdots & & & \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u \\ v \\ \vdots \end{bmatrix}$$

This equation shows a single match, but any number of further matches can be added, with each match contributing two more rows to the first and last matrix. At least 3 matches are needed to provide a solution.

We can write this linear system as

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

The least-squares solution for the parameters $\mathbf{x}$ can be determined by solving the corresponding normal equations,

$$\mathbf{x} = [\mathbf{A}^\mathbf{T}\mathbf{A}]^{-\mathbf{1}}\mathbf{A}^\mathbf{T}\mathbf{b},$$

which minimizes the sum of the squares of the distances from the projected model locations to the corresponding image locations. This least-squares approach could readily be extended to solving for 3D pose and internal parameters of articulated and flexible objects (Lowe, 1991).

Outliers can now be removed by checking for agreement between each image feature and the model. Given the more accurate least-squares solution, we now require each match to agree within half the error range that was used for the parameters in the Hough transform bins. If fewer than 3 points remain after discarding outliers, then the match is rejected. As outliers are discarded, the least-squares solution is re-solved with the remaining points, and the process iterated. In addition, a top-down matching phase is used to add any further matches that agree with the projected model position. These may have been missed from the Hough transform bin due to the similarity transform approximation or other errors.

The final decision to accept or reject a model hypothesis is based on a detailed probabilistic model given in a previous paper (Lowe, 2001). This method first computes the expected number of false matches to the model pose, given the projected size of the model, the number of features within the region, and the accuracy of the fit. A Bayesian analysis then gives the probability that the object is present based on the actual number of matching features found. We accept a model if the final probability for a correct interpretation is greater than 0.98. For objects that project to small regions of an image, 3 features may be sufficient for reliable recognition. For large objects covering most of a heavily textured image, the expected number of false matches is higher, and as many as 10 feature matches may be necessary.

8 Recognition examples

Figure 12 shows an example of object recognition for a cluttered and occluded image containing 3D objects. The training images of a toy train and a frog are shown on the left.

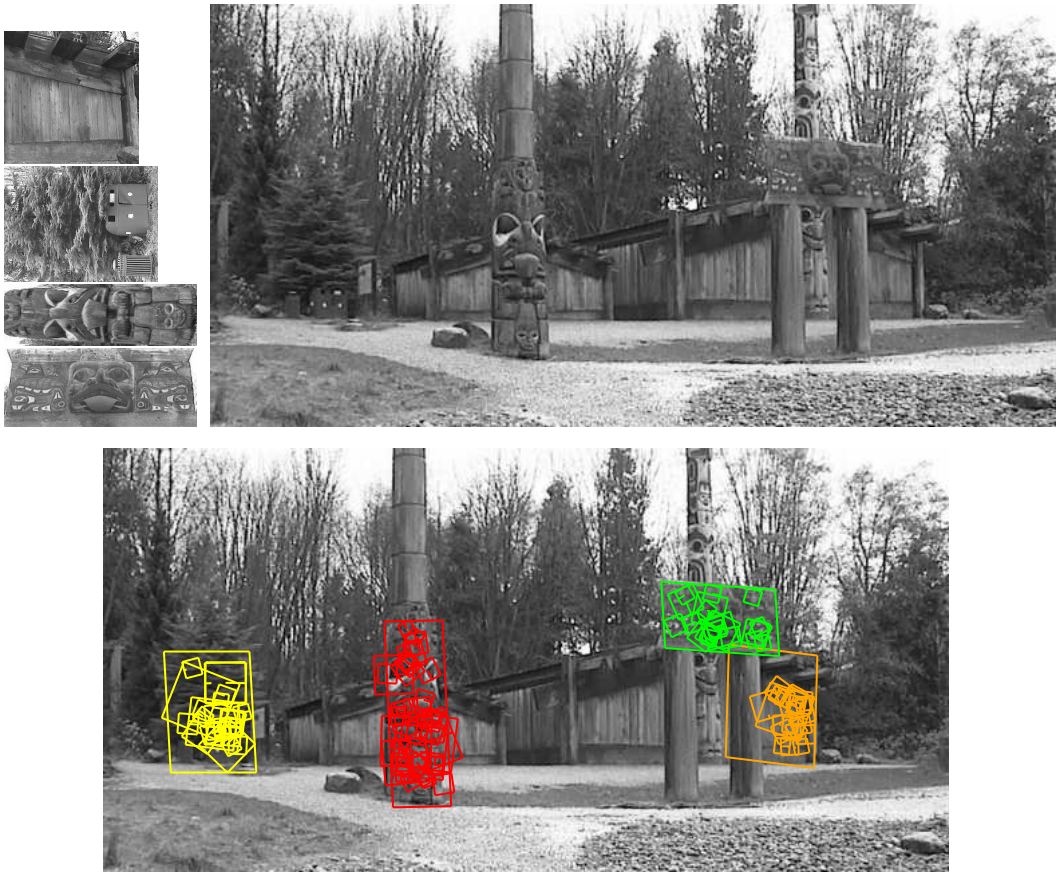


Figure 13: This example shows location recognition within a complex scene. The training images for locations are shown at the upper left and the 640x315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affine transform used for recognition.

The middle image (of size 600x480 pixels) contains instances of these objects hidden behind others and with extensive background clutter so that detection of the objects may not be immediate even for human vision. The image on the right shows the final correct identification superimposed on a reduced contrast version of the image. The keypoints that were used for recognition are shown as squares with an extra line to indicate orientation. The sizes of the squares correspond to the image regions used to construct the descriptor. An outer parallelogram is also drawn around each instance of recognition, with its sides corresponding to the boundaries of the training images projected under the final affine transformation determined during recognition.

Another potential application of the approach is to place recognition, in which a mobile device or vehicle could identify its location by recognizing familiar locations. Figure 13 gives an example of this application, in which training images are taken of a number of locations. As shown on the upper left, these can even be of such seemingly non-distinctive items as a wooden wall or a tree with trash bins. The test image (of size 640 by 315 pixels) on the upper right was taken from a viewpoint rotated about 30 degrees around the scene from the original positions, yet the training image locations are easily recognized.

All steps of the recognition process can be implemented efficiently, so the total time to recognize all objects in Figures 12 or 13 is less than 0.3 seconds on a 2GHz Pentium 4 processor. We have implemented these algorithms on a laptop computer with attached video camera, and have tested them extensively over a wide range of conditions. In general, textured planar surfaces can be identified reliably over a rotation in depth of up to 50 degrees in any direction and under almost any illumination conditions that provide sufficient light and do not produce excessive glare. For 3D objects, the range of rotation in depth for reliable recognition is only about 30 degrees in any direction and illumination change is more disruptive. For these reasons, 3D object recognition is best performed by integrating features from multiple views, such as with local feature view clustering (Lowe, 2001).

These keypoints have also been applied to the problem of robot localization and mapping, which has been presented in detail in other papers (Se, Lowe and Little, 2001). In this application, a trinocular stereo system is used to determine 3D estimates for keypoint locations. Keypoints are used only when they appear in all 3 images with consistent disparities, resulting in very few outliers. As the robot moves, it localizes itself using feature matches to the existing 3D map, and then incrementally adds features to the map while updating their 3D positions using a Kalman filter. This provides a robust and accurate solution to the problem of robot localization in unknown environments. This work has also addressed the problem of place recognition, in which a robot can be switched on and recognize its location anywhere within a large map (Se, Lowe and Little, 2002), which is equivalent to a 3D implementation of object recognition.

9 Conclusions

The SIFT keypoints described in this paper are particularly useful due to their distinctiveness, which enables the correct match for a keypoint to be selected from a large database of other keypoints. This distinctiveness is achieved by assembling a high-dimensional vector representing the image gradients within a local region of the image. The keypoints have been shown to be invariant to image rotation and scale and robust across a substantial range of affine distortion, addition of noise, and change in illumination. Large numbers of keypoints can be extracted from typical images, which leads to robustness in extracting small objects among clutter. The fact that keypoints are detected over a complete range of scales means that small local features are available for matching small and highly occluded objects, while large keypoints perform well for images subject to noise and blur. Their computation is efficient, so that several thousand keypoints can be extracted from a typical image with near real-time performance on standard PC hardware.

This paper has also presented methods for using the keypoints for object recognition. The approach we have described uses approximate nearest-neighbor lookup, a Hough transform for identifying clusters that agree on object pose, least-squares pose determination, and final verification. Other potential applications include view matching for 3D reconstruction, motion tracking and segmentation, robot localization, image panorama assembly, epipolar calibration, and any others that require identification of matching locations between images.

There are many directions for further research in deriving invariant and distinctive image features. Systematic testing is needed on data sets with full 3D viewpoint and illumination changes. The features described in this paper use only a monochrome intensity image, so further distinctiveness could be derived from including illumination-invariant color descriptors

(Funt and Finlayson, 1995; Brown and Lowe, 2002). Similarly, local texture measures appear to play an important role in human vision and could be incorporated into feature descriptors in a more general form than the single spatial frequency used by the current descriptors. An attractive aspect of the invariant local feature approach to matching is that there is no need to select just one feature type, and the best results are likely to be obtained by using many different features, all of which can contribute useful matches and improve overall robustness.

Another direction for future research will be to individually learn features that are suited to recognizing particular objects categories. This will be particularly important for generic object classes that must cover a broad range of possible appearances. The research of Weber, Welling, and Perona (2000) and Fergus, Perona, and Zisserman (2003) has shown the potential of this approach by learning small sets of local features that are suited to recognizing generic classes of objects. In the long term, feature sets are likely to contain both prior and learned features that will be used according to the amount of training data that has been available for various object classes.

Acknowledgments

I would particularly like to thank Matthew Brown, who has suggested numerous improvements to both the content and presentation of this paper and whose own work on feature localization and invariance has contributed to this approach. In addition, I would like to thank many others for their valuable suggestions, including Stephen Se, Jim Little, Krystian Mikolajczyk, Cordelia Schmid, Tony Lindeberg, and Andrew Zisserman. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and through the Institute for Robotics and Intelligent Systems (IRIS) Network of Centres of Excellence.

References

Arya, S., and Mount, D.M. 1993. Approximate nearest neighbor queries in fixed dimensions. In *Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'93),* pp. 271-280.

Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., and Wu, A.Y. 1998. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM,* 45:891-923.

Ballard, D.H. 1981. Generalizing the Hough transform to detect arbitrary patterns. *Pattern Recognition,* 13(2):111-122.

Basri, R., and Jacobs, D.W. 1997. Recognition using region correspondences. *International Journal of Computer Vision,* 25(2):145-166.

Baumberg, A. 2000. Reliable feature matching across widely separated views. In *Conference on Computer Vision and Pattern Recognition,* Hilton Head, South Carolina, pp. 774-781.

Beis, J. and Lowe, D.G. 1997. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Conference on Computer Vision and Pattern Recognition,* Puerto Rico, pp. 1000-1006.

Brown, M. and Lowe, D.G. 2002. Invariant features from interest point groups. In *British Machine Vision Conference,* Cardiff, Wales, pp. 656-665.

Carneiro, G., and Jepson, A.D. 2002. Phase-based local features. In *European Conference on Computer Vision (ECCV),* Copenhagen, Denmark, pp. 282-296.

Crowley, J. L. and Parker, A.C. 1984. A representation for shape based on peaks and ridges in the difference of low-pass transform. *IEEE Trans. on Pattern Analysis and Machine Intelligence,* 6(2):156-170.

Edelman, S., Intrator, N. and Poggio, T. 1997. Complex cells and object recognition. Unpublished manuscript: http://kybele.psych.cornell.edu/~edelman/archive.html

Fergus, R., Perona, P., and Zisserman, A. 2003. Object class recognition by unsupervised scale-invariant learning. In *IEEE Conference on Computer Vision and Pattern Recognition,* Madison, Wisconsin, pp. 264-271.

Friedman, J.H., Bentley, J.L. and Finkel, R.A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software,* 3(3):209-226.

Funt, B.V. and Finlayson, G.D. 1995. Color constant color indexing. *IEEE Trans. on Pattern Analysis and Machine Intelligence,* 17(5):522-529.

Grimson, E. 1990. *Object Recognition by Computer: The Role of Geometric Constraints,* The MIT Press: Cambridge, MA.

Harris, C. 1992. Geometry from visual motion. In *Active Vision,* A. Blake and A. Yuille (Eds.), MIT Press, pp. 263-284.

Harris, C. and Stephens, M. 1988. A combined corner and edge detector. In *Fourth Alvey Vision Conference,* Manchester, UK, pp. 147-151.

Hartley, R. and Zisserman, A. 2000. *Multiple view geometry in computer vision,* Cambridge University Press: Cambridge, UK.

Hough, P.V.C. 1962. *Method and means for recognizing complex patterns.* U.S. Patent 3069654.

Koenderink, J.J. 1984. The structure of images. *Biological Cybernetics,* 50:363-396.

Lindeberg, T. 1993. Detecting salient blob-like image structures and their scales with a scale-space primal sketch: a method for focus-of-attention. *International Journal of Computer Vision,* 11(3): 283-318.

Lindeberg, T. 1994. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics,* 21(2):224-270.

Lowe, D.G. 1991. Fitting parameterized three-dimensional models to images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(5):441-450.

Lowe, D.G. 1999. Object recognition from local scale-invariant features. In *International Conference on Computer Vision,* Corfu, Greece, pp. 1150-1157.

Lowe, D.G. 2001. Local feature view clustering for 3D object recognition. *IEEE Conference on Computer Vision and Pattern Recognition,* Kauai, Hawaii, pp. 682-688.

Luong, Q.T., and Faugeras, O.D. 1996. The fundamental matrix: Theory, algorithms, and stability analysis. *International Journal of Computer Vision,* 17(1):43-76.

Matas, J., Chum, O., Urban, M., and Pajdla, T. 2002. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference,* Cardiff, Wales, pp. 384-393.

Mikolajczyk, K. 2002. *Detection of local features invariant to affine transformations,* Ph.D. thesis, Institut National Polytechnique de Grenoble, France.

Mikolajczyk, K., and Schmid, C. 2002. An affine invariant interest point detector. In *European Conference on Computer Vision (ECCV),* Copenhagen, Denmark, pp. 128-142.

Mikolajczyk, K., Zisserman, A., and Schmid, C. 2003. Shape recognition with edge-based features. In *Proceedings of the British Machine Vision Conference,* Norwich, U.K.

Moravec, H. 1981. Rover visual obstacle avoidance. In *International Joint Conference on Artificial Intelligence,* Vancouver, Canada, pp. 785-790.

Nelson, R.C., and Selinger, A. 1998. Large-scale tests of a keyed, appearance-based 3-D object recognition system. *Vision Research,* 38(15):2469-88.

Pope, A.R., and Lowe, D.G. 2000. Probabilistic models of appearance for 3-D object recognition. *International Journal of Computer Vision,* 40(2):149-167.

Pritchard, D., and Heidrich, W. 2003. Cloth motion capture. *Computer Graphics Forum (Eurographics 2003),* 22(3):263-271.

Schaffalitzky, F., and Zisserman, A. 2002. Multi-view matching for unordered image sets, or 'How do I organize my holiday snaps?"' In *European Conference on Computer Vision,* Copenhagen, Denmark, pp. 414-431.

Schiele, B., and Crowley, J.L. 2000. Recognition without correspondence using multidimensional receptive field histograms. *International Journal of Computer Vision,* 36(1):31-50.

Schmid, C., and Mohr, R. 1997. Local grayvalue invariants for image retrieval. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(5):530-534.

Se, S., Lowe, D.G., and Little, J. 2001. Vision-based mobile robot localization and mapping using scale-invariant features. In *International Conference on Robotics and Automation,* Seoul, Korea, pp. 2051-58.

Se, S., Lowe, D.G., and Little, J. 2002. Global localization using distinctive visual features. In *International Conference on Intelligent Robots and Systems, IROS 2002,* Lausanne, Switzerland, pp. 226-231.

Shokoufandeh, A., Marsic, I., and Dickinson, S.J. 1999. View-based object recognition using saliency maps. *Image and Vision Computing,* 17:445-460.

Torr, P. 1995. *Motion Segmentation and Outlier Detection,* Ph.D. Thesis, Dept. of Engineering Science, University of Oxford, UK.

Tuytelaars, T., and Van Gool, L. 2000. Wide baseline stereo based on local, affinely invariant regions. In *British Machine Vision Conference,* Bristol, UK, pp. 412-422.

Weber, M., Welling, M. and Perona, P. 2000. Unsupervised learning of models for recognition. In *European Conference on Computer Vision,* Dublin, Ireland, pp. 18-32.

Witkin, A.P. 1983. Scale-space filtering. In *International Joint Conference on Artificial Intelligence,* Karlsruhe, Germany, pp. 1019-1022.

Zhang, Z., Deriche, R., Faugeras, O., and Luong, Q.T. 1995. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence,* 78:87-119.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe SIOFFE@GOOGLE.COM
Christian Szegedy SZEGEDY@GOOGLE.COM
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.82% top-5 test error, exceeding the accuracy of human raters.

1. Introduction

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochastic gradient descent (SGD) has proved to be an effective way of training deep networks, and SGD variants such as momentum (Sutskever et al., 2013) and Adagrad (Duchi et al., 2011) have been used to achieve state of the art performance. SGD optimizes the parameters Θ of the network, so as to minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathrm{x}_i, \Theta)$$

where $\mathrm{x}_{1\ldots N}$ is the training data set. With SGD, the training proceeds in steps, at each step considering a *mini-batch* $\mathrm{x}_{1\ldots m}$ of size m. Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch $\frac{1}{m}\sum_{i=1}^{m} \frac{\partial \ell(\mathrm{x}_i,\Theta)}{\partial \Theta}$ is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a mini-batch can be more efficient than m computations for individual examples on modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate and the initial parameter values. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(\mathrm{u}, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ. Learning Θ_2 can be viewed as if the inputs $\mathrm{x} = F_1(\mathrm{u}, \Theta_1)$ are fed into the sub-network

$$\ell = F_2(\mathrm{x}, \Theta_2).$$

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

For example, a gradient descent step

$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m}\sum_{i=1}^{m}\frac{\partial F_2(\mathrm{x}_i, \Theta_2)}{\partial \Theta_2}$$

(for mini-batch size m and learning rate α) is exactly equivalent to that for a stand-alone network F_2 with input x. Therefore, the input distribution properties that aid the network generalization – such as having the same distribution between the training and test data – apply to training the sub-network as well. As such it is advantageous for the distribution of x to remain fixed over time. Then, Θ_2 does not have to readjust to compensate for the change in the distribution of x.

Fixed distribution of inputs to a sub-network would have positive consequences for the layers *outside* the sub-network, as well. Consider a layer with a sigmoid activation function $\mathrm{z} = g(W\mathrm{u} + \mathrm{b})$ where u is the layer input, the weight matrix W and bias vector b are the layer parameters to be learned, and $g(x) = \frac{1}{1+\exp(-x)}$. As $|x|$ increases, $g'(x)$ tends to zero. This means that for all dimensions of $\mathrm{x} = W\mathrm{u} + \mathrm{b}$ except those with small absolute values, the gradient flowing down to u will vanish and the model will train slowly. However, since x is affected by W, b and the parameters of all the layers below, changes to those parameters during training will likely move many dimensions of x into the saturated regime of the nonlinearity and slow down the convergence. This effect is amplified as the network depth increases. In practice, the saturation problem and the resulting vanishing gradients are usually addressed by using Rectified Linear Units (Nair & Hinton, 2010) $ReLU(x) = \max(x, 0)$, careful initialization (Bengio & Glorot, 2010; Saxe et al., 2013), and small learning rates. If, however, we could ensure that the distribution of nonlinearity inputs remains more stable as the network trains, then the optimizer would be less likely to get stuck in the saturated regime, and the training would accelerate.

We refer to the change in the distributions of internal nodes of a deep network, in the course of training, as *Internal Covariate Shift*. Eliminating it offers a promise of faster training. We propose a new mechanism, which we call *Batch Normalization*, that takes a step towards reducing internal covariate shift, and in doing so dramatically accelerates the training of deep neural nets. It accomplishes this via a normalization step that fixes the means and variances of layer inputs. Batch Normalization also has a beneficial effect on the gradient flow through the network, by reducing the dependence of gradients on the scale of the parameters or of their initial values. This allows us to use much higher learning rates without the risk of divergence. Furthermore, batch normalization regularizes the model and reduces the need for Dropout (Srivastava et al., 2014). Finally, Batch Normalization makes it possible to use saturating nonlinearities by preventing the network from getting stuck in the saturated modes.

In Sec. 4.2, we apply Batch Normalization to the best-performing ImageNet classification network, and show that we can match its performance using only 7% of the training steps, and can further exceed its accuracy by a substantial margin. Using an ensemble of such networks trained with Batch Normalization, we achieve the top-5 error rate that improves upon the best known results on ImageNet classification.

2. Towards Reducing Internal Covariate Shift

We define *Internal Covariate Shift* as the change in the distribution of network activations due to the change in network parameters during training. To improve the training, we seek to reduce the internal covariate shift. By fixing the distribution of the layer inputs x as the training progresses, we expect to improve the training speed. It has been long known (LeCun et al., 1998b; Wiesler & Ney, 2011) that the network training converges faster if its inputs are whitened – i.e., linearly transformed to have zero means and unit variances, and decorrelated. As each layer observes the inputs produced by the layers below, it would be advantageous to achieve the same whitening of the inputs of each layer. By whitening the inputs to each layer, we would take a step towards achieving the fixed distributions of inputs that would remove the ill effects of the internal covariate shift.

We could consider whitening activations at every training step or at some interval, either by modifying the network directly or by changing the parameters of the optimization algorithm to depend on the network activation values (Wiesler et al., 2014; Raiko et al., 2012; Povey et al., 2014; Desjardins & Kavukcuoglu). However, if these modifications are interspersed with the optimization steps, then the gradient descent step may attempt to update the parameters in a way that requires the normalization to be updated, which reduces the effect of the gradient step. For example, consider a layer with the input u that adds the learned bias b, and normalizes the result by subtracting the mean of the activation computed over the training data: $\widehat{x} = x - E[x]$ where $x = u + b$, $\mathcal{X} = \{x_{1\ldots N}\}$ is the set of values of x over the training set, and $\mathrm{E}[x] = \frac{1}{N}\sum_{i=1}^{N} x_i$. If a gradient descent step ignores the dependence of $\mathrm{E}[x]$ on b, then it will update $b \leftarrow b + \Delta b$, where $\Delta b \propto -\partial\ell/\partial\widehat{x}$. Then $u + (b + \Delta b) - \mathrm{E}[u + (b + \Delta b)] = u + b - \mathrm{E}[u + b]$. Thus, the combination of the update to b and subsequent change in normalization led to no change in the output of the layer nor, consequently, the loss. As the training continues, b will grow indefinitely while the loss remains fixed. This problem can get worse if the normalization not only centers but also scales the activations. We have observed this

empirically in initial experiments, where the model blows up when the normalization parameters are computed outside the gradient descent step.

The issue with the above approach is that the gradient descent optimization does not take into account the fact that the normalization takes place. To address this issue, we would like to ensure that, for any parameter values, the network *always* produces activations with the desired distribution. Doing so would allow the gradient of the loss with respect to the model parameters to account for the normalization, and for its dependence on the model parameters Θ. Let again x be a layer input, treated as a vector, and $\mathcal{X}$ be the set of these inputs over the training data set. The normalization can then be written as a transformation

$$\widehat{\mathrm{x}} = \mathrm{Norm}(\mathrm{x}, \mathcal{X})$$

which depends not only on the given training example x but on all examples $\mathcal{X}$ – each of which depends on Θ if x is generated by another layer. For backpropagation, we would need to compute the Jacobians $\frac{\partial \mathrm{Norm}(\mathrm{x},\mathcal{X})}{\partial \mathrm{x}}$ and $\frac{\partial \mathrm{Norm}(\mathrm{x},\mathcal{X})}{\partial \mathcal{X}}$; ignoring the latter term would lead to the explosion described above. Within this framework, whitening the layer inputs is expensive, as it requires computing the covariance matrix $\mathrm{Cov}[\mathrm{x}] = \mathrm{E}_{\mathrm{x}\in\mathcal{X}}[\mathrm{x}\mathrm{x}^T] - \mathrm{E}[\mathrm{x}]\mathrm{E}[\mathrm{x}]^T$ and its inverse square root, to produce the whitened activations $\mathrm{Cov}[\mathrm{x}]^{-1/2}(\mathrm{x} - \mathrm{E}[\mathrm{x}])$, as well as the derivatives of these transforms for backpropagation. This motivates us to seek an alternative that performs input normalization in a way that is differentiable and does not require the analysis of the entire training set after every parameter update.

Some of the previous approaches (e.g. (Lyu & Simoncelli, 2008)) use statistics computed over a single training example, or, in the case of image networks, over different feature maps at a given location. However, this changes the representation ability of a network by discarding the absolute scale of activations. We want to a preserve the information in the network, by normalizing the activations in a training example relative to the statistics of the entire training data.

3. Normalization via Mini-Batch Statistics

Since the full whitening of each layer's inputs is costly, we make two necessary simplifications. The first is that instead of whitening the features in layer inputs and outputs jointly, we will normalize each scalar feature independently, by making it have zero mean and unit variance. For a layer with d-dimensional input $\mathrm{x} = (x^{(1)} \ldots x^{(d)})$, we will normalize each dimension

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

where the expectation and variance are computed over the training data set. As shown in (LeCun et al., 1998b), such normalization speeds up convergence, even when the features are not decorrelated.

Note that simply normalizing each input of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. To address this, we make sure that *the transformation inserted in the network can represent the identity transform*. To accomplish this, we introduce, for each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}, \beta^{(k)}$, which scale and shift the normalized value:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$

These parameters are learned along with the original model parameters, and restore the representation power of the network. Indeed, by setting $\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$ and $\beta^{(k)} = \mathrm{E}[x^{(k)}]$, we could recover the original activations, if that were the optimal thing to do.

In the batch setting where each training step is based on the entire training set, we would use the whole set to normalize activations. However, this is impractical when using stochastic optimization. Therefore, we make the second simplification: since we use mini-batches in stochastic gradient training, *each mini-batch produces estimates of the mean and variance* of each activation. This way, the statistics used for normalization can fully participate in the gradient backpropagation. Note that the use of mini-batches is enabled by computation of per-dimension variances rather than joint covariances; in the joint case, regularization would be required since the mini-batch size is likely to be smaller than the number of activations being whitened, resulting in singular covariance matrices.

Consider a mini-batch $\mathcal{B}$ of size m. Since the normalization is applied to each activation independently, let us focus on a particular activation $x^{(k)}$ and omit k for clarity. We have m values of this activation in the mini-batch,

$$\mathcal{B} = \{x_{1\ldots m}\}.$$

Let the normalized values be $\widehat{x}_{1\ldots m}$, and their linear transformations be $y_{1\ldots m}$. We refer to the transform

$$\mathrm{BN}_{\gamma,\beta} : x_{1\ldots m} \rightarrow y_{1\ldots m}$$

as the *Batch Normalizing Transform*. We present the BN Transform in Algorithm 1. In the algorithm, ϵ is a constant added to the mini-batch variance for numerical stability.

The BN transform can be added to a network to manipulate any activation. In the notation $y = \mathrm{BN}_{\gamma,\beta}(x)$, we indicate that the parameters γ and β are to be learned, but it should be noted that the BN transform does not independently process the activation in each training example. Rather, $\mathrm{BN}_{\gamma,\beta}(x)$ depends both on the training example *and the other examples in the mini-batch*. The scaled and

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

shifted values y are passed to other network layers. The normalized activations $\widehat{x}$ are internal to our transformation, but their presence is crucial. The distributions of values of any $\widehat{x}$ has the expected value of 0 and the variance of 1, as long as the elements of each mini-batch are sampled from the same distribution, and if we neglect ϵ. This can be seen by observing that $\sum_{i=1}^{m}\widehat{x}_i = 0$ and $\frac{1}{m}\sum_{i=1}^{m}\widehat{x}_i^2 = 1$, and taking expectations. Each normalized activation $\widehat{x}^{(k)}$ can be viewed as an input to a sub-network composed of the linear transform $y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$, followed by the other processing done by the original network. These sub-network inputs all have fixed means and variances, and although the joint distribution of these normalized $\widehat{x}^{(k)}$ can change over the course of training, we expect that the introduction of normalized inputs accelerates the training of the sub-network and, consequently, the network as a whole.

During training we need to backpropagate the gradient of loss ℓ through this transformation, as well as compute the gradients with respect to the parameters of the BN transform. We use chain rule, as follows:

$$\begin{aligned}
\frac{\partial \ell}{\partial \widehat{x}_i} &= \frac{\partial \ell}{\partial y_i}\cdot\gamma \\
\frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \widehat{x}_i}\cdot(x_i-\mu_{\mathcal{B}})\cdot\frac{-1}{2}(\sigma^2_{\mathcal{B}}+\epsilon)^{-3/2} \\
\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \widehat{x}_i}\cdot\frac{-1}{\sqrt{\sigma^2_{\mathcal{B}}+\epsilon}} \\
\frac{\partial \ell}{\partial x_i} &= \frac{\partial \ell}{\partial \widehat{x}_i}\cdot\frac{1}{\sqrt{\sigma^2_{\mathcal{B}}+\epsilon}} + \frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}}\cdot\frac{2(x_i-\mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}}\cdot\frac{1}{m} \\
\frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}\cdot\widehat{x}_i \\
\frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}
\end{aligned}$$

Thus, BN transform is a differentiable transformation that introduces normalized activations into the network. This ensures that as the model is training, layers can continue learning on input distributions that exhibit less internal covariate shift, thus accelerating the training. Furthermore, the learned affine transform applied to these normalized activations allows the BN transform to represent the identity transformation and preserves the network capacity.

3.1. Training and Inference with Batch-Normalized Networks

To *Batch-Normalize* a network, we specify a subset of activations and insert the BN transform for each of them, according to Alg. 1. Any layer that previously received x as the input, now receives $\text{BN}(x)$. A model employing Batch Normalization can be trained using batch gradient descent, or Stochastic Gradient Descent with a mini-batch size $m > 1$, or with any of its variants such as Adagrad (Duchi et al., 2011). The normalization of activations that depends on the mini-batch allows efficient training, but is neither necessary nor desirable during inference; we want the output to depend only on the input, deterministically. For this, once the network has been trained, we use the normalization

$$\widehat{x} = \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}}$$

using the population, rather than mini-batch, statistics. Neglecting ϵ, these normalized activations have the same mean 0 and variance 1 as during training. We use the unbiased variance estimate $\text{Var}[x] = \frac{m}{m-1}\cdot\text{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$, where the expectation is over training mini-batches of size m and $\sigma^2_{\mathcal{B}}$ are their sample variances. Using moving averages instead, we can track the accuracy of a model as it trains. Since the means and variances are fixed during inference, the normalization is simply a linear transform applied to each activation. It may further be composed with the scaling by γ and shift by β, to yield a single linear transform that replaces $\text{BN}(x)$. Algorithm 2 summarizes the procedure for training batch-normalized networks.

3.2. Batch-Normalized Convolutional Networks

Batch Normalization can be applied to any set of activations in the network. Here, we focus on transforms that consist of an affine transformation followed by an element-wise nonlinearity:

$$\text{z} = g(W\text{u} + \text{b})$$

where W and b are learned parameters of the model, and $g(\cdot)$ is the nonlinearity such as sigmoid or ReLU. This formulation covers both fully-connected and convolutional layers. We add the BN transform immediately before the nonlinearity, by normalizing $\text{x} = W\text{u} + \text{b}$. We could have also normalized the layer inputs u, but since u is likely the output of another nonlinearity, the shape of its distribution is likely to change during training, and constraining its first and second moments would not eliminate the covariate shift. In contrast, $W\text{u} + \text{b}$ is more likely to have a symmetric, non-sparse distribution, that is "more Gaus-

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

sian" (Hyvärinen & Oja, 2000); normalizing it is likely to produce activations with a stable distribution.

Note that, since we normalize $W\mathrm{u} + \mathrm{b}$, the bias b can be ignored since its effect will be canceled by the subsequent mean subtraction (the role of the bias is subsumed by β in Alg. 1). Thus, $\mathrm{z} = g(W\mathrm{u} + \mathrm{b})$ is replaced with

$$\mathrm{z} = g(\text{BN}(W\mathrm{u}))$$

where the BN transform is applied independently to each dimension of $\mathrm{x} = W\mathrm{u}$, with a separate pair of learned parameters $\gamma^{(k)}$, $\beta^{(k)}$ per dimension.

For convolutional layers, we additionally want the normalization to obey the convolutional property – so that different elements of the same feature map, at different locations, are normalized in the same way. To achieve this, we jointly normalize all the activations in a mini-batch, over all locations. In Alg. 1, we let $\mathcal{B}$ be the set of all values in a feature map across both the elements of a mini-batch and spatial locations – so for a mini-batch of size m and feature maps of size $p \times q$, we use the effective mini-batch of size $m' = |\mathcal{B}| = m \cdot p\,q$. We learn a pair of parameters $\gamma^{(k)}$ and $\beta^{(k)}$ per feature map, rather than per activation. Alg. 2 is modified similarly, so that during inference the BN transform applies the same linear transformation to each activation in a given feature map.

3.3. Batch Normalization enables higher learning rates

In traditional deep networks, too high a learning rate may result in the gradients that explode or vanish, as well as getting stuck in poor local minima. Batch Normalization helps address these issues. By normalizing activations throughout the network, it prevents small changes in layer parameters from amplifying as the data propagates through a deep network. For example, this enables the sigmoid nonlinearities to more easily stay in their non-saturated regimes, which is crucial for training deep sigmoid networks but has traditionally been hard to accomplish.

Batch Normalization also makes training more resilient to the parameter scale. Normally, large learning rates may increase the scale of layer parameters, which then amplify the gradient during backpropagation and lead to the model explosion. However, with Batch Normalization, backpropagation through a layer is unaffected by the scale of its parameters. Indeed, for a scalar a,

$$\text{BN}(W\mathrm{u}) = \text{BN}((aW)\mathrm{u})$$

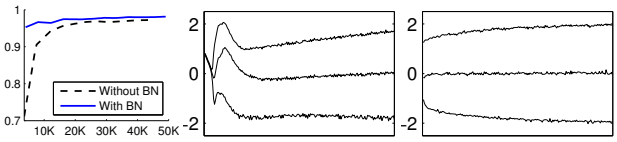
and thus $\frac{\partial \text{BN}((aW)\mathrm{u})}{\partial \mathrm{u}} = \frac{\partial \text{BN}(W\mathrm{u})}{\partial \mathrm{u}}$, so the scale does not affect the layer Jacobian nor, consequently, the gradient propagation. Moreover, $\frac{\partial \text{BN}((aW)\mathrm{u})}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial \text{BN}(W\mathrm{u})}{\partial W}$, so larger weights lead to *smaller* gradients, and Batch Normalization will stabilize the parameter growth.

We further conjecture that Batch Normalization may lead the layer Jacobians to have singular values close to 1, which is known to be beneficial for training (Saxe et al., 2013). Consider two consecutive layers with normalized inputs, and the transformation between these normalized vectors: $\widehat{\mathrm{z}} = F(\widehat{\mathrm{x}})$. If we assume that $\widehat{\mathrm{x}}$ and $\widehat{\mathrm{z}}$ are Gaussian and uncorrelated, and that $F(\widehat{\mathrm{x}}) \approx J\widehat{\mathrm{x}}$ is a linear transformation for the given model parameters, then both $\widehat{\mathrm{x}}$ and $\widehat{\mathrm{z}}$ have unit covariances, and $I = \text{Cov}[\widehat{\mathrm{z}}] = J\text{Cov}[\widehat{\mathrm{x}}]J^T = JJ^T$. Thus, J is orthogonal, which preserves the gradient magnitudes during backpropagation. Although the above assumptions are not true in reality, we expect Batch Normalization to help make gradient propagation better behaved. This remains an area of further study.

4. Experiments

4.1. Activations over time

To verify the effects of internal covariate shift on training, and the ability of Batch Normalization to combat it, we considered the problem of predicting the digit class on the MNIST dataset (LeCun et al., 1998a). We used a very simple network, with a 28x28 binary image as input, and 3 fully-connected hidden layers with 100 activations each. Each hidden layer computes $\mathrm{y} = g(W\mathrm{u} + \mathrm{b})$ with sigmoid nonlinearity, and the weights W initialized to small random Gaussian values. The last hidden layer is followed



(a) (b) Without BN (c) With BN

Figure 1. (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

by a fully-connected layer with 10 activations (one per class) and cross-entropy loss. We trained the network for 50000 steps, with 60 examples per mini-batch. We added Batch Normalization to each hidden layer of the network, as in Sec. 3.1. We were interested in the comparison between the baseline and batch-normalized networks, rather than achieving the state of the art performance on MNIST (which the described architecture does not).

Figure 1(a) shows the fraction of correct predictions by the two networks on held-out test data, as training progresses. The batch-normalized network enjoys the higher test accuracy. To investigate why, we studied inputs to the sigmoid, in the original network N and batch-normalized network $N_{\text{BN}}^{\text{tr}}$ (Alg. 2) over the course of training. In Fig. 1(b,c) we show, for one typical activation from the last hidden layer of each network, how its distribution evolves. The distributions in the original network change significantly over time, both in their mean and the variance, which complicates the training of the subsequent layers. In contrast, the distributions in the batch-normalized network are much more stable as training progresses, which aids the training.

4.2. ImageNet classification

We applied Batch Normalization to a new variant of the Inception network (Szegedy et al., 2014), trained on the ImageNet classification task (Russakovsky et al., 2014). The network has a large number of convolutional and pooling layers, with a softmax layer to predict the image class, out of 1000 possibilities. Convolutional layers use ReLU as the nonlinearity. The main difference to the network described in (Szegedy et al., 2014) is that the 5×5 convolutional layers are replaced by two consecutive layers of 3×3 convolutions with up to 128 filters. The network contains $13.6 \cdot 10^6$ parameters, and, other than the top softmax layer, has no fully-connected layers. We refer to this model as *Inception* in the rest of the text. The training was performed on a large-scale, distributed architecture (Dean et al., 2012), using 5 concurrent steps on each of 10 model replicas, using asynchronous SGD with momentum (Sutskever et al., 2013), with the mini-batch size of 32. All networks are evaluated as training progresses by computing the validation accuracy @1, i.e. the probability of predicting the correct label out of 1000 possibilities, on a held-out set, using a single crop per image.

In our experiments, we evaluated several modifications of Inception with Batch Normalization. In all cases, Batch Normalization was applied to the input of each nonlinearity, in a convolutional way, as described in section 3.2, while keeping the rest of the architecture constant.

4.2.1. Accelerating BN Networks

Simply adding Batch Normalization to a network does not take full advantage of our method. To do so, we applied the following modifications:

Increase learning rate. In a batch-normalized model, we have been able to achieve a training speedup from higher learning rates, with no ill side effects (Sec. 3.3).

Remove Dropout. We have found that removing Dropout from BN-Inception allows the network to achieve higher validation accuracy. We conjecture that Batch Normalization provides similar regularization benefits as Dropout, since the activations observed for a training example are affected by the random selection of examples in the same mini-batch.

Shuffle training examples more thoroughly. We enabled within-shard shuffling of the training data, which prevents the same examples from always appearing in a mini-batch together. This led to about 1% improvement in the validation accuracy, which is consistent with the view of Batch Normalization as a regularizer: the randomization inherent in our method should be most beneficial when it affects an example differently each time it is seen.

Reduce the L_2 weight regularization. While in Inception an L_2 loss on the model parameters controls overfitting, in modified BN-Inception the weight of this loss is reduced by a factor of 5. We find that this improves the accuracy on the held-out validation data.

Accelerate the learning rate decay. In training Inception, learning rate was decayed exponentially. Because our network trains faster than Inception, we lower the learning rate 6 times faster.

Remove Local Response Normalization While Inception and other networks (Srivastava et al., 2014) benefit from it, we found that with Batch Normalization it is not necessary.

Reduce the photometric distortions. Because batch-normalized networks train faster and observe each training example fewer times, we let the trainer focus on more "real" images by distorting them less.

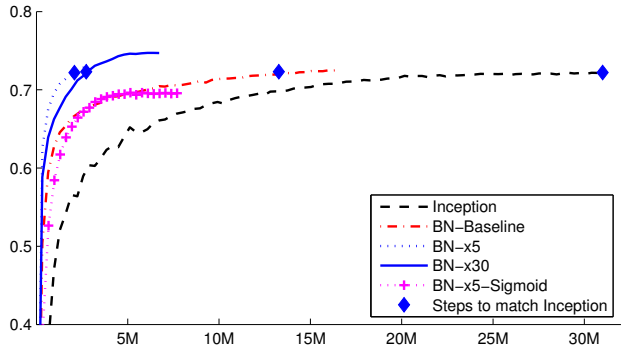


Figure 2. Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Figure 3. For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.

4.2.2. Single-Network Classification

We evaluated the following networks, all trained on the LSVRC2012 training data, and tested on the validation data:

Inception: the network described at the beginning of Section 4.2, trained with the initial learning rate of 0.0015.

BN-Baseline: Same as Inception with Batch Normalization before each nonlinearity.

BN-x5: Inception with Batch Normalization and the modifications in Sec. 4.2.1. The initial learning rate was increased by a factor of 5, to 0.0075. The same learning rate increase with original Inception caused the model parameters to reach machine infinity.

BN-x30: Like *BN-x5*, but with the initial learning rate 0.045 (30 times that of Inception).

BN-x5-Sigmoid: Like *BN-x5*, but with sigmoid nonlinearity $g(t) = \frac{1}{1+\exp(-x)}$ instead of ReLU. We also attempted to train the original Inception with sigmoid, but the model remained at the accuracy equivalent to chance.

In Figure 2, we show the validation accuracy of the networks, as a function of the number of training steps. Inception reached the accuracy of 72.2% after $31 \cdot 10^6$ training steps. The Figure 3 shows, for each network, the number of training steps required to reach the same 72.2% accuracy, as well as the maximum validation accuracy reached by the network and the number of steps to reach it.

By only using Batch Normalization (*BN-Baseline*), we match the accuracy of Inception in less than half the number of training steps. By applying the modifications in Sec. 4.2.1, we significantly increase the training speed of the network. *BN-x5* needs 14 times fewer steps than Inception to reach the 72.2% accuracy. Interestingly, increasing the learning rate further (*BN-x30*) causes the model to train somewhat *slower* initially, but allows it to reach a higher final accuracy. This phenomenon is counterintuitive and should be investigated further. *BN-x30* reaches 74.8% after $6 \cdot 10^6$ steps, i.e. 5 times fewer steps than required by Inception to reach 72.2%.

We also verified that the reduction in internal covariate shift allows deep networks with Batch Normalization to be trained when sigmoid is used as the nonlinearity, despite the well-known difficulty of training such networks. Indeed, *BN-x5-Sigmoid* achieves the accuracy of 69.8%. Without Batch Normalization, Inception with sigmoid never achieves better than $1/1000$ accuracy.

4.2.3. Ensemble Classification

The current reported best results on the ImageNet Large Scale Visual Recognition Competition are reached by the Deep Image ensemble of traditional models (Wu et al., 2015) and the ensemble model of (He et al., 2015). The latter reports the top-5 error of 4.94%, as evaluated by the ILSVRC test server. Here we report a test error of 4.82% on test server. This improves upon the previous best result, and exceeds the estimated accuracy of human raters according to (Russakovsky et al., 2014).

For our ensemble, we used 6 networks. Each was based on *BN-x30*, modified via some of the following: increased initial weights in the convolutional layers; using Dropout (with the Dropout probability of 5% or 10%, vs. 40% for the original Inception); and using non-convolutional Batch Normalization with last hidden layers of the model. Each network achieved its maximum accuracy after about $6 \cdot 10^6$ training steps. The ensemble prediction was based on the arithmetic average of class probabilities predicted by the constituent networks. The details of ensemble and multi-crop inference are similar to (Szegedy et al., 2014).

We demonstrate in Fig. 4 that batch normalization allows us to set new state-of-the-art on the ImageNet classification challenge benchmarks.

Model	Resolution	Crops	Models	Top-1 error	Top-5 error
GoogLeNet ensemble	224	144	7	-	6.67%
Deep Image low-res	256	-	1	-	7.96%
Deep Image high-res	512	-	1	24.88	7.42%
Deep Image ensemble	up to 512	-	-	-	5.98%
MSRA multicrop	up to 480	-	-	-	5.71%
MSRA ensemble	up to 480	-	-	-	4.94%*
BN-Inception single crop	224	1	1	25.2%	7.82%
BN-Inception multicrop	224	144	1	21.99%	5.82%
BN-Inception ensemble	224	144	6	20.1%	**4.82%***

*Figure 4. Batch-Normalized Inception comparison with previous state of the art on the provided validation set comprising 50000 images. *Ensemble results are test server evaluation results on the test set. The BN-Inception ensemble has reached 4.9% top-5 error on the 50000 images of the validation set. All other reported results are on the validation set.*

5. Conclusion

We have presented a novel mechanism for dramatically accelerating the training of deep networks. It is based on the premise that covariate shift, which is known to complicate the training of machine learning systems, also applies to sub-networks and layers, and removing it from internal activations of the network may aid in training. Our proposed method draws its power from normalizing activations, and from incorporating this normalization in the network architecture itself. This ensures that the normalization is appropriately handled by any optimization method that is being used to train the network. To enable stochastic optimization methods commonly used in deep network training, we perform the normalization for each mini-batch, and backpropagate the gradients through the normalization parameters. Batch Normalization adds only two extra parameters per activation, and in doing so preserves the representation ability of the network. We presented an algorithm for constructing, training, and performing inference with batch-normalized networks. The resulting networks can be trained with saturating nonlinearities, are more tolerant to increased training rates, and often do not require Dropout for regularization.

Merely adding Batch Normalization to a state-of-the-art image classification model yields a substantial speedup in training. By further increasing the learning rates, removing Dropout, and applying other modifications afforded by Batch Normalization, we reach the previous state of the art with only a small fraction of training steps – and then beat the state of the art in single-network image classification. Furthermore, by combining multiple models trained with Batch Normalization, we perform better than the best known system on ImageNet, by a significant margin.

Our method bears similarity to the standardization layer of (Gülçehre & Bengio, 2013), though the two address different goals. Batch Normalization seeks a stable distribution of activation values throughout training, and normalizes the inputs of a nonlinearity since that is where matching the moments is more likely to stabilize the distribution. On the contrary, the standardization layer is applied to the *output* of the nonlinearity, which results in sparser activations. We have not observed the nonlinearity *inputs* to be sparse, neither with nor without Batch Normalization. Other notable differences of Batch Normalization include the learned scale and shift that allow the BN transform to represent identity, handling of convolutional layers, and deterministic inference that does not depend on the mini-batch.

In this work, we have not explored the full range of possibilities that Batch Normalization potentially enables. Our future work includes applications of our method to Recurrent Neural Networks (Pascanu et al., 2013), where the internal covariate shift and the vanishing or exploding gradients may be especially severe, and which would allow us to more thoroughly test the hypothesis that normalization improves gradient propagation (Sec. 3.3). More study is needed of the regularization properties of Batch Normalization, which we believe to be responsible for the improvements we have observed when Dropout is removed from BN-Inception. We plan to investigate whether Batch Normalization can help with domain adaptation, in its traditional sense – i.e. whether the normalization performed by the network would allow it to more easily generalize to new data distributions, perhaps with just a recomputation of the population means and variances (Alg. 2). Finally, we believe that further theoretical analysis of the algorithm would allow still more improvements and applications.

Acknowledgments

We thank Vincent Vanhoucke and Jay Yagnik for help and discussions, and the reviewers for insightful comments.

References

Bengio, Yoshua and Glorot, Xavier. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS 2010*, volume 9, pp. 249–256, May 2010.

Dean, Jeffrey, Corrado, Greg S., Monga, Rajat, Chen, Kai, Devin, Matthieu, Le, Quoc V., Mao, Mark Z., Ranzato, Marc'Aurelio, Senior, Andrew, Tucker, Paul, Yang, Ke, and Ng, Andrew Y. Large scale distributed deep networks. In *NIPS*, 2012.

Desjardins, Guillaume and Kavukcuoglu, Koray. Natural neural networks. (unpublished).

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435.

Gülçehre, Çaglar and Bengio, Yoshua. Knowledge matters: Importance of prior information for optimization. *CoRR*, abs/1301.4083, 2013.

He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ArXiv e-prints*, February 2015.

Hyvärinen, A. and Oja, E. Independent component analysis: Algorithms and applications. *Neural Netw.*, 13(4-5): 411–430, May 2000.

Jiang, Jing. A literature survey on domain adaptation of statistical classifiers, 2008.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998a.

LeCun, Y., Bottou, L., Orr, G., and Muller, K. Efficient backprop. In Orr, G. and K., Muller (eds.), *Neural Networks: Tricks of the trade*. Springer, 1998b.

Lyu, S and Simoncelli, E P. Nonlinear image representation using divisive normalization. In *Proc. Computer Vision and Pattern Recognition*, pp. 1–8. IEEE Computer Society, Jun 23-28 2008. doi: 10.1109/CVPR.2008.4587821.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *ICML*, pp. 807–814. Omnipress, 2010.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1310–1318, 2013.

Povey, Daniel, Zhang, Xiaohui, and Khudanpur, Sanjeev. Parallel training of deep neural networks with natural gradient and parameter averaging. *CoRR*, abs/1410.7455, 2014.

Raiko, Tapani, Valpola, Harri, and LeCun, Yann. Deep learning made easier by linear transformations in perceptrons. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 924–932, 2012.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge, 2014.

Saxe, Andrew M., McClelland, James L., and Ganguli, Surya. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013.

Shimodaira, Hidetoshi. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90 (2):227–244, October 2000.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

Sutskever, Ilya, Martens, James, Dahl, George E., and Hinton, Geoffrey E. On the importance of initialization and momentum in deep learning. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pp. 1139–1147. JMLR.org, 2013.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

Wiesler, Simon and Ney, Hermann. A convergence analysis of log-linear training. In Shawe-Taylor, J., Zemel, R.S., Bartlett, P., Pereira, F.C.N., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 24*, pp. 657–665, Granada, Spain, December 2011.

Wiesler, Simon, Richard, Alexander, Schlüter, Ralf, and Ney, Hermann. Mean-normalized stochastic gradient for large-scale deep learning. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 180–184, Florence, Italy, May 2014.

Wu, Ren, Yan, Shengen, Shan, Yi, Dang, Qingqing, and Sun, Gang. Deep image: Scaling up image recognition, 2015.

Large-scale Video Classification with Convolutional Neural Networks

Andrej Karpathy[1,2] George Toderici[1] Sanketh Shetty[1]
karpathy@cs.stanford.edu gtoderici@google.com sanketh@google.com
Thomas Leung[1] Rahul Sukthankar[1] Li Fei-Fei[2]
leungt@google.com sukthankar@google.com feifeili@cs.stanford.edu
[1]Google Research [2]Computer Science Department, Stanford University
http://cs.stanford.edu/people/karpathy/deepvideo

Abstract

Convolutional Neural Networks (CNNs) have been established as a powerful class of models for image recognition problems. Encouraged by these results, we provide an extensive empirical evaluation of CNNs on large-scale video classification using a new dataset of 1 million YouTube videos belonging to 487 classes. We study multiple approaches for extending the connectivity of a CNN in time domain to take advantage of local spatio-temporal information and suggest a multiresolution, foveated architecture as a promising way of speeding up the training. Our best spatio-temporal networks display significant performance improvements compared to strong feature-based baselines (55.3% to 63.9%), but only a surprisingly modest improvement compared to single-frame models (59.3% to 60.9%). We further study the generalization performance of our best model by retraining the top layers on the UCF-101 Action Recognition dataset and observe significant performance improvements compared to the UCF-101 baseline model (63.3% up from 43.9%).

1. Introduction

Images and videos have become ubiquitous on the internet, which has encouraged the development of algorithms that can analyze their semantic content for various applications, including search and summarization. Recently, Convolutional Neural Networks (CNNs) [15] have been demonstrated as an effective class of models for understanding image content, giving state-of-the-art results on image recognition, segmentation, detection and retrieval [11, 3, 2, 20, 9, 18]. The key enabling factors behind these results were techniques for scaling up the networks to tens of millions of parameters and massive labeled datasets that can support the learning process. Under these conditions, CNNs have been shown to learn powerful and interpretable image features [28]. Encouraged by positive results in domain of images, we study the performance of CNNs in large-scale video classification, where the networks have access to not only the appearance information present in single, static images, but also their complex temporal evolution. There are several challenges to extending and applying CNNs in this setting.

From a practical standpoint, there are currently no video classification benchmarks that match the scale and variety of existing image datasets because videos are significantly more difficult to collect, annotate and store. To obtain sufficient amount of data needed to train our CNN architectures, we collected a new Sports-1M dataset, which consists of 1 million YouTube videos belonging to a taxonomy of 487 classes of sports. We make Sports-1M available to the research community to support future work in this area.

From a modeling perspective, we are interested in answering the following questions: what temporal connectivity pattern in a CNN architecture is best at taking advantage of local motion information present in the video? How does the additional motion information influence the predictions of a CNN and how much does it improve performance overall? We examine these questions empirically by evaluating multiple CNN architectures that each take a different approach to combining information across the time domain.

From a computational perspective, CNNs require extensively long periods of training time to effectively optimize the millions of parameters that parametrize the model. This difficulty is further compounded when extending the connectivity of the architecture in time because the network must process not just one image but several frames of video at a time. To mitigate this issue, we show that an effective approach to speeding up the runtime performance of CNNs is to modify the architecture to contain two separate streams of processing: a *context* stream that learns features on low-resolution frames and a high-resolution *fovea* stream that only operates on the middle portion of the frame. We

observe a 2-4x increase in runtime performance of the network due to the reduced dimensionality of the input, while retaining the classification accuracy.

Finally, a natural question that arises is whether features learned on the Sports-1M dataset are generic enough to generalize to a different, smaller dataset. We investigate the transfer learning problem empirically, achieving significantly better performance (65.4%, up from 41.3%) on UCF-101 by re-purposing low-level features learned on the Sports-1M dataset than by training the entire network on UCF-101 alone. Furthermore, since only some classes in UCF-101 are related to sports, we can quantify the relative improvements of the transfer learning in both settings.

Our contributions can be summarized as follows:

- We provide extensive experimental evaluation of multiple approaches for extending CNNs into video classification on a large-scale dataset of 1 million videos with 487 categories (which we release as Sports-1M dataset) and report significant gains in performance over strong feature-based baselines.
- We highlight an architecture that processes input at two spatial resolutions - a low-resolution context stream and a high-resolution fovea stream - as a promising way of improving the runtime performance of CNNs at no cost in accuracy.
- We apply our networks to the UCF-101 dataset and report significant improvement over feature-based state-of-the-art results and baselines established by training networks on UCF-101 alone.

2. Related Work

The standard approach to video classification [26, 16, 21, 17] involves three major stages: First, local visual features that describe a region of the video are extracted either densely [25] or at a sparse set of interest points [12, 8]. Next, the features get combined into a fixed-sized video-level description. One popular approach is to quantize all features using a learned k-means dictionary and accumulate the visual words over the duration of the video into histograms of varying spatio-temporal positions and extents [13]. Lastly, a classifier (such as an SVM) is trained on the resulting "bag of words" representation to distinguish among the visual classes of interest.

Convolutional Neural Networks [15] are a biologically-inspired class of deep learning models that replace all three stages with a single neural network that is trained end to end from raw pixel values to classifier outputs. The spatial structure of images is explicitly taken advantage of for regularization through restricted connectivity between layers (local filters), parameter sharing (convolutions) and special local invariance-building neurons (max pooling). Thus, these architectures effectively shift the required engineering from feature design and accumulation strategies to design of the network connectivity structure and hyperparameter choices. Due to computational constraints, CNNs have until recently been applied to relatively small scale image recognition problems (on datasets such as MNIST, CIFAR-10/100, NORB, and Caltech-101/256), but improvements on GPU hardware have enabled CNNs to scale to networks of millions of parameters, which has in turn led to significant improvements in image classification[11], object detection [20, 9], scene labeling [3], indoor segmentation [4] and house number digit classification [19]. Additionally, features learned by large networks trained on ImageNet [7] have been shown to yield state-of-the-art performance across many standard image recognition datasets when classified with an SVM, even with no fine-tuning [18].

Compared to image data domains, there is relatively little work on applying CNNs to video classification. Since all successful applications of CNNs in image domains share the availability of a large training set, we speculate that this is partly attributable to lack of large-scale video classification benchmarks. In particular, commonly used datasets (KTH, Weizmann, UCF Sports, IXMAS, Hollywood 2, UCF-50) only contain up to few thousand clips and up to few dozen classes. Even the largest available datasets such as CCV (9,317 videos and 20 classes) and the recently introduced UCF-101[22] (13,320 videos and 101 classes) are still dwarfed by available image datasets in the number of instances and their variety [7]. Despite these limitations, some extensions of CNNs into the video domain have been explored. [1] and [10] extend an image CNN to video domains by treating space and time as equivalent dimensions of the input and perform convolutions in both time and space. We consider these extensions as only one of the possible generalizations in this work. Unsupervised learning schemes for training spatio-temporal features have also been developed, based on Convolutional Gated Restricted Boltzmann Machines [23] and Independent Subspace Analysis [14]. In contrast, our models are trained end to end fully supervised.

3. Models

Unlike images which can be cropped and rescaled to a fixed size, videos vary widely in temporal extent and cannot be easily processed with a fixed-sized architecture. In this work we treat every video as a bag of short, fixed-sized clips. Since each clip contains several contiguous frames in time, we can extend the connectivity of the network in time dimension to learn spatio-temporal features. There are multiple options for the precise details of the extended connectivity and we describe three broad connectivity pattern categories (Early Fusion, Late Fusion and Slow Fusion) below. Afterwards, we describe a multiresolution architecture for addressing the computational efficiency.

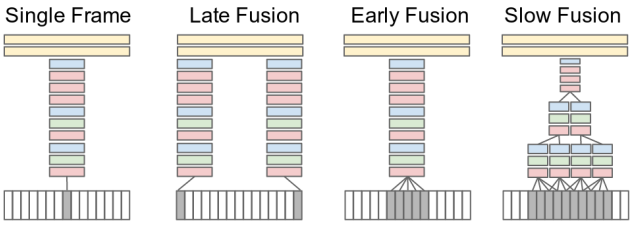


Figure 1: Explored approaches for fusing information over temporal dimension through the network. Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively. In the Slow Fusion model, the depicted columns share parameters.

3.1. Time Information Fusion in CNNs

We investigate several approaches to fusing information across temporal domain (Figure 1): the fusion can be done early in the network by modifying the first layer convolutional filters to extend in time, or it can be done late by placing two separate single-frame networks some distance in time apart and fusing their outputs later in the processing. We first describe a baseline single-frame CNN and then discuss its extensions in time according to different types of fusion.

Single-frame. We use a single-frame baseline architecture to understand the contribution of static appearance to the classification accuracy. This network is similar to the ImageNet challenge winning model [11], but accepts inputs of size $170 \times 170 \times 3$ pixels instead of the original $224 \times 224 \times 3$. Using shorthand notation, the full architecture is $C(96, 11, 3)$-N-P-$C(256, 5, 1)$-N-P-$C(384, 3, 1)$-$C(384, 3, 1)$-$C(256, 3, 1)$-P-$FC(4096)$-$FC(4096)$, where $C(d, f, s)$ indicates a convolutional layer with d filters of spatial size $f \times f$, applied to the input with stride s. $FC(n)$ is a fully connected layer with n nodes. All pooling layers P pool spatially in non-overlapping 2×2 regions and all normalization layers N are defined as described in Krizhevsky et al. [11] and use the same parameters: $k = 2, n = 5, \alpha = 10^{-4}, \beta = 0.5$. The final layer is connected to a softmax classifier with dense connections.

Early Fusion. The Early Fusion extension combines information across an entire time window immediately on the pixel level. This is implemented by modifying the filters on the first convolutional layer in the single-frame model by extending them to be of size $11 \times 11 \times 3 \times T$ pixels, where T is some temporal extent (we use $T = 10$, or approximately a third of a second). The early and direct connectivity to pixel data allows the network to precisely detect local motion direction and speed.

Late Fusion. The Late Fusion model places two separate single-frame networks (as described above, up to last convolutional layer $C(256, 3, 1)$ with shared parameters a distance of 15 frames apart and then merges the two streams in the first fully connected layer. Therefore, neither single-frame tower alone can detect any motion, but the first fully connected layer can compute global motion characteristics by comparing outputs of both towers.

Slow Fusion. The Slow Fusion model is a balanced mix between the two approaches that slowly fuses temporal information throughout the network such that higher layers get access to progressively more global information in both spatial and temporal dimensions. This is implemented by extending the connectivity of all convolutional layers in time and carrying out temporal convolutions in addition to spatial convolutions to compute activations, as seen in [1, 10]. In the model we use, the first convolutional layer is extended to apply every filter of temporal extent $T = 4$ on an input clip of 10 frames through valid convolution with stride 2 and produces 4 responses in time. The second and third layers above iterate this process with filters of temporal extent $T = 2$ and stride 2. Thus, the third convolutional layer has access to information across all 10 input frames.

3.2. Multiresolution CNNs

Since CNNs normally take on orders of weeks to train on large-scale datasets even on the fastest available GPUs, the runtime performance is a critical component to our ability to experiment with different architecture and hyperparameter settings. This motivates approaches for speeding up the models while still retaining their performance. There are multiple fronts to these endeavors, including improvements in hardware, weight quantization schemes, better optimization algorithms and initialization strategies, but in this work we focus on changes in the architecture that enable faster running times without sacrificing performance.

One approach to speeding up the networks is to reduce the number of layers and neurons in each layer, but similar to [28] we found that this consistently lowers the performance. Instead of reducing the size of the network, we conducted further experiments on training with images of lower resolution. However, while this improved the running time of the network, the high-frequency detail in the images proved critical to achieving good accuracy.

Fovea and context streams. The proposed multiresolution architecture aims to strike a compromise by having two separate streams of processing over two spatial resolutions (Figure 2). A 178×178 frame video clip forms an input to the network. The context stream receives the downsampled frames at half the original spatial resolution (89×89 pixels), while the fovea stream receives the center 89×89 region at the original resolution. In this way, the the total input dimensionality is halved. Notably, this design takes advantage of the camera bias present in many online videos, since the object of interest often occupies the center region.

Architecture changes. Both streams are processed by identical network as the full frame models, but starting at

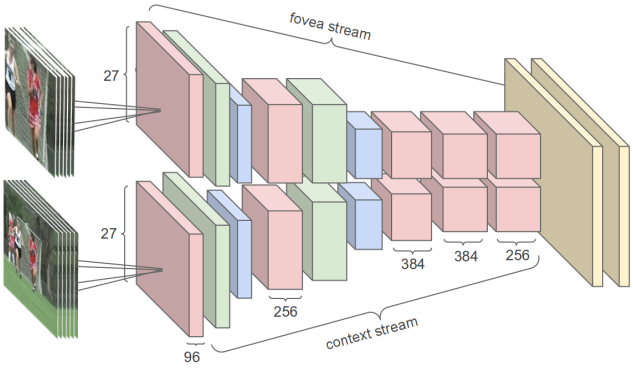


Figure 2: Multiresolution CNN architecture. Input frames are fed into two separate streams of processing: a *context stream* that models low-resolution image and a *fovea stream* that processes high-resolution center crop. Both streams consist of alternating convolution (red), normalization (green) and pooling (blue) layers. Both streams converge to two fully connected layers (yellow).

89×89 clips of video. Since the input is only of half the spatial size as the full-frame models, we take out the last pooling layer to ensure that both streams still terminate in a layer of size $7 \times 7 \times 256$. The activations from both streams are concatenated and fed into the first fully connected layer with dense connections.

3.3. Learning

Optimization. We use Downpour Stochastic Gradient Descent [6] to optimize our models across a computing cluster. The number of replicas for each model varies between 10 and 50 and every model is further split across 4 to 32 partitions. We use mini-batches of 32 examples, momentum of 0.9 and weight decay of 0.0005. All models are initialized with learning rates of $1e^{-3}$ and this value is further reduced by hand whenever the validation error stops improving.

Data augmentation and preprocessing. Following [11], we take advantage of data augmentation to reduce the effects of overfitting. Before presenting an example to a network, we preprocess all images by first cropping to center region, resizing them to 200×200 pixels, randomly sampling a 170×170 region, and finally randomly flipping the images horizontally with 50% probability. These preprocessing steps are applied consistently to all frames that are part of the same clip. As a last step of preprocessing we subtract a constant value of 117 from raw pixel values, which is the approximate value of the mean of all pixels in our images.

4. Results

We first present results on our Sports-1M dataset and qualitatively analyze the learned features and network predictions. We then describe our transfer learning experiments on UCF-101.

4.1. Experiments on Sports-1M

Dataset. The Sports-1M dataset consists of 1 million YouTube videos annotated with 487 classes. The classes are arranged in a manually-curated taxonomy that contains internal nodes such as *Aquatic Sports, Team Sports, Winter Sports, Ball Sports, Combat Sports, Sports with Animals,* and generally becomes fine-grained by the leaf level. For example, our dataset contains 6 different types of bowling, 7 different types of American football and 23 types of billiards.

There are 1000-3000 videos per class and approximately 5% of the videos are annotated with more than one class. The annotations are produced automatically by analyzing the text metadata surrounding the videos. Thus, our data is weakly annotated on two levels: first, the label of a video may be wrong if the tag prediction algorithm fails or if the provided description does not match the video content, and second, even when a video is correctly annotated it may still exhibit significant variation on the frame level. For example, a video tagged as soccer may contain several shots of the scoreboard, interviews, news anchors, the crowd, etc.

We split the dataset by assigning 70% of the videos to the training set, 10% to a validation set and 20% to a test set. As YouTube may contain duplicate videos, it is possible that the same video could appear in both the training and test set. To get an idea about the extent of this problem we processed all videos with a near-duplicate finding algorithm on the frame level and determined that only 1755 videos (out of 1 million) contain a significant fraction of near-duplicate frames. Furthermore, since we only use a random collection of up to 100 half-second clips from every video and our videos are 5 minutes and 36 seconds in length on average, it is unlikely that the same frames occur across data splits.

Training. We trained our models over a period of one month, with models processing approximately 5 clips per second for full-frame networks and up to 20 clips per second for multiresolution networks on a single model replica. The rate of 5 clips per second is roughly 20 times slower than what one could expect from a high-end GPU, but we expect to reach comparable speeds overall given that we use 10-50 model replicas. We further estimate the size of our dataset of sampled frames to be on the order of 50 million examples and that our networks have each seen approximately 500 million examples throughout the training period in total.

Video-level predictions. To produce predictions for an entire video we randomly sample 20 clips and present each clip individually to the network. Every clip is propagated through the network 4 times (with different crops and flips)

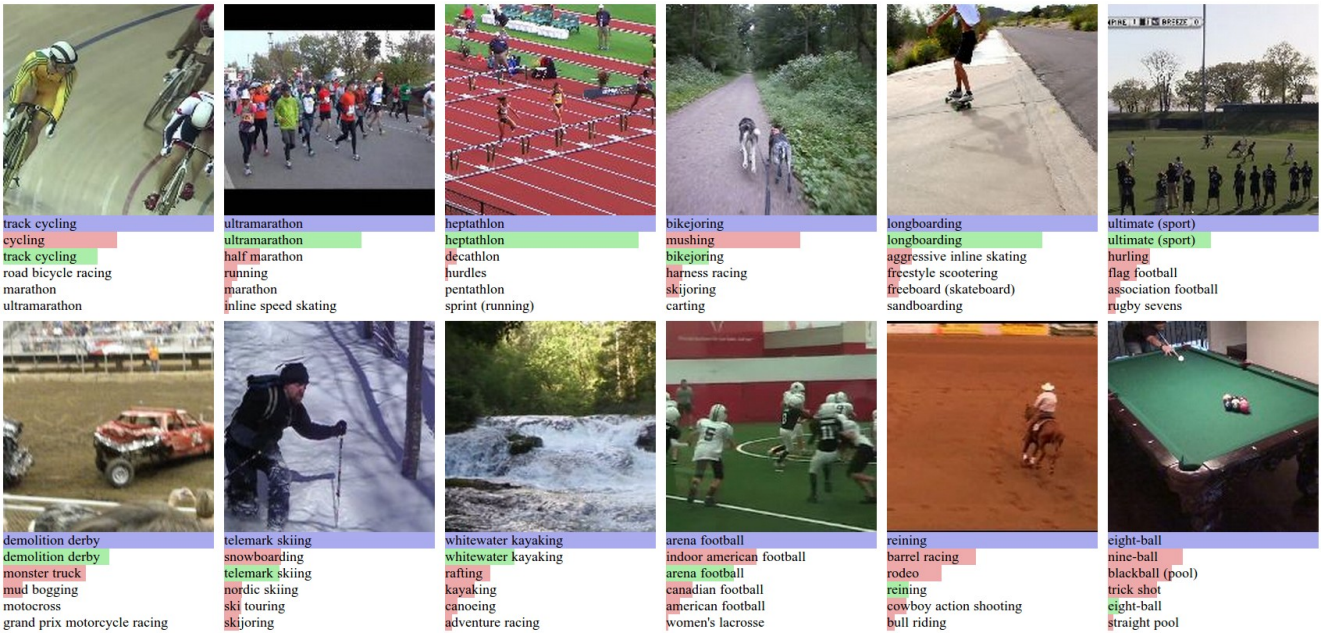


Figure 4: Predictions on Sports-1M test data. Blue (first row) indicates ground truth label and the bars below show model predictions sorted in decreasing confidence. Green and red distinguish correct and incorrect predictions, respectively.

Model	Clip Hit@1	Video Hit@1	Video Hit@5
Feature Histograms + Neural Net	-	55.3	-
Single-Frame	41.1	59.3	77.7
Single-Frame + Multires	**42.4**	**60.0**	**78.5**
Single-Frame Fovea Only	30.0	49.9	72.8
Single-Frame Context Only	38.1	56.0	77.2
Early Fusion	38.9	57.7	76.8
Late Fusion	40.7	59.3	78.7
Slow Fusion	**41.9**	**60.9**	**80.2**
CNN Average (Single+Early+Late+Slow)	41.4	63.9	82.4

Table 1: Results on the 200,000 videos of the Sports-1M test set. Hit@k values indicate the fraction of test samples that contained at least one of the ground truth labels in the top k predictions.

and the network class predictions are averaged to produce a more robust estimate of the class probabilities. To produce video-level predictions we opted for the simplest approach of averaging individual clip predictions over the durations of each video. We expect more elaborate techniques to further improve performance but consider these to be outside of the scope of the paper.

Feature histogram baselines. In addition to comparing CNN architectures among each other, we also report the accuracy of a feature-based approach. Following a standard bag-of-words pipeline we extract several types of features at all frames of our videos, discretize them using k-means vector quantization and accumulate words into histograms with spatial pyramid encoding and soft quantization. Every histogram is normalized to sum to 1 and all histograms are concatenated into a 25,000 dimensional video-level feature vector. Our features are similar to Yang & Toderici [27] and consist of local features (HOG [5], Texton [24], Cuboids [8], etc.) extracted both densely and at sparse interest points, as well as global features (such as Hue-Saturation, Color moments, number of faces detected). As a classifier we use a multilayer neural network with Rectified Linear Units followed by a Softmax classifier. We found that a multilayer network performs consistently and significantly better than linear models on separate validation experiments. Furthermore, we performed extensive cross-validations across many of the network's hyperparameters by training multiple models and choosing the one with best performance on a validation set. The tuned hyper parameters include the learning rate, weight decay, the number of hidden layers (between 1-2), dropout probabilities and the

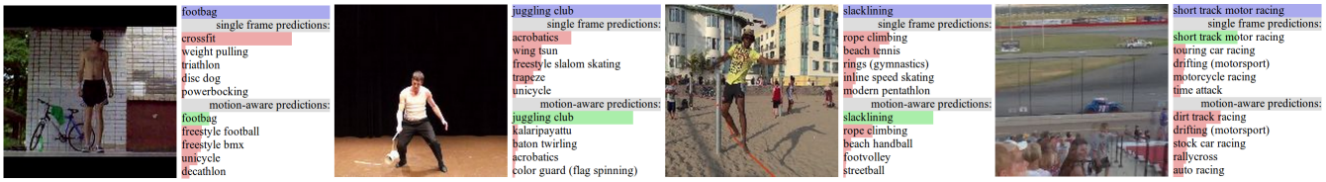


Figure 5: Examples that illustrate qualitative differences between single-frame network and Slow Fusion (motion-aware) network in the same color scheme as Figure 4. A few classes are easier to disambiguate with motion information (left three).

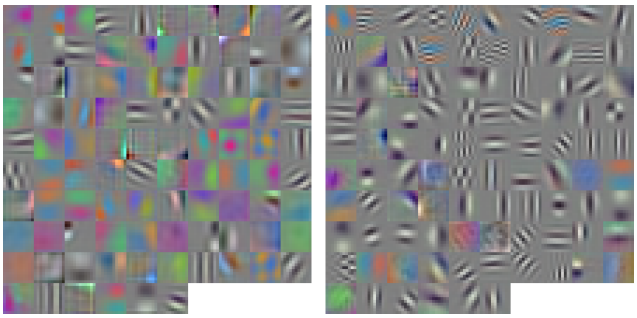


Figure 3: Filters learned on first layer of a multiresolution network. Left: context stream, Right: fovea stream. Notably, the fovea stream learns grayscale, high-frequency features while the context stream models lower frequencies and colors. GIFs of moving video features can be found on our website (linked on first page).

Sports class	Δ AP	Δ AP	Sports class
Juggling Club	0.12	-0.07	Short Track Motor Racing
Pole Climbing	0.10	-0.07	Road Racing
Mountain Unicycling	0.08	-0.07	Jeet Kune Do
Tricking	0.07	-0.06	Paintball
Footbag	0.07	-0.06	Freeride
Skipping Rope	0.06	-0.06	Cricket
Rope Climbing	0.06	-0.06	Wrestling
Slacklining	0.05	-0.06	Modern Pentathlon
Tee Ball	0.05	-0.06	Krav Maga
Sheepdog Trial	0.05	-0.05	Rally Cross

Table 2: Classes for which a (motion-aware) Slow Fusion CNN performs better than the single-frame CNN (left) and vice versa (right), as measured by difference in per-class average precision.

number of nodes in all layers.

Quantitative results. The results for the Sports-1M dataset test set, which consists of 200,000 videos and 4,000,000 clips, are summarized in Table 1. As can be seen from the table, our networks consistently and significantly outperform the feature-based baseline. We emphasize that the feature-based approach computes visual words densely over the duration of the video and produces predictions based on the entire video-level feature vector, while our networks only see 20 randomly sampled clips individually. Moreover, our networks seem to learn well despite significant label noise: the training videos are subject to incorrect annotations and even the correctly-labeled videos often contain a large amount of artifacts such as text, effects, cuts, and logos, none of which we attempted to filter out explicitly.

Compared to the wide gap relative to the feature-based baseline, the variation among different CNN architectures turns out to be surprisingly insignificant. Notably, the single-frame model already displays strong performance. Furthermore, we observe that the foveated architectures are between 2-4$\times$ faster in practice due to reduced input dimensionality. The precise speedups are in part a function of the details of model partitioning and our implementation, but in our experiments we observe a speedup during training of 6 to 21 clips per second (3.5x) for the single-frame model and 5 to 10 clips per second (2x) for the Slow Fusion model.

Contributions of motion. We conduct further experiments to understand the differences between the single-frame network and networks that have access to motion information. We choose the Slow Fusion network as a representative motion-aware network because it performs best. We compute and compare the per-class average precision for all Sports classes and highlight the ones that exhibit largest differences (Table 2). Manually inspecting some of the associated clips (Figure 5), we qualitatively observe that the motion-aware network clearly benefits from motion information in some cases, but these seem to be relatively uncommon. On the other hand, balancing the improvements from access to motion information, we observe that motion-aware networks are more likely to underperform when there is camera motion present. We hypothesize that the CNNs struggle to learn complete invariance across all possible angles and speeds of camera translation and zoom.

Qualitative analysis. Our learned features for the first convolutional layer can be inspected on Figure 3. Interestingly, the context stream learns more color features while the high-resolution fovea stream learns high frequency grayscale filters.

As can be seen on Figure 4, our networks produce interpretable predictions and generally make reasonable mistakes. Further analysis of the confusion matrix (attached in the supplementary material) reveals that most errors are among the fine-grained classes of our dataset. For example, the top 5 most commonly confused pairs of classes are deer hunting vs. hunting, hiking vs. backpacking, powered paragliding vs. paragliding, sledding vs. toboggan, and bujinkan vs. ninjutsu.

Model	3-fold Accuracy
Soomro et al [22]	43.9%
Feature Histograms + Neural Net	59.0%
Train from scratch	41.3%
Fine-tune top layer	64.1%
Fine-tune top 3 layers	**65.4**%
Fine-tune all layers	62.2%

Table 3: Results on UCF-101 for various Transfer Learning approaches using the Slow Fusion network.

Group	mAP from scratch	mAP fine-tune top 3	mAP fine-tune top
Human-Object Interaction	0.26	0.55	0.52
Body-Motion Only	0.32	0.57	0.52
Human-Human Interaction	0.40	0.68	0.65
Playing Musical Instruments	0.42	0.65	0.46
Sports	**0.57**	**0.79**	**0.80**
All groups	0.44	0.68	0.66

Table 4: Mean Average Precision of the Slow Fusion network on UCF-101 classes broken down by category groups.

4.2. Transfer Learning Experiments on UCF-101

The results of our analysis on the Sports-1M dataset indicate that the networks learn powerful motion features. A natural question that arises is whether these features also generalize to other datasets and class categories. We examine this question in detail by performing transfer learning experiments on the UCF-101 [22] Activity Recognition dataset. The dataset consists of 13,320 videos belonging to 101 categories that are separated into 5 broad groups: Human-Object interaction (Applying eye makeup, brushing teeth, hammering, etc.), Body-Motion (Baby crawling, push ups, blowing candles, etc.), Human-Human interaction (Head massage, salsa spin, haircut, etc.), Playing Instruments (flute, guitar, piano, etc.) and Sports. This grouping allows us to separately study the performance improvements on Sports classes relative to classes from unrelated videos that are less numerous in our training data.

Transfer learning. Since we expect that CNNs learn more generic features on the bottom of the network (such as edges, local shapes) and more intricate, dataset-specific features near the top of the network, we consider the following scenarios for our transfer learning experiments:

Fine-tune top layer. We treat the CNN as a fixed feature extractor and train a classifier on the last 4096-dimensional layer, with dropout regularization. We found that as little as 10% chance of keeping each unit active to be effective.

Fine-tune top 3 layers. Instead of only retraining the final classifier layer, we consider also retraining both fully connected layers. We initialize with a fully trained Sports CNN and then begin training the top 3 layers. We introduce dropout before all trained layers, with as little as 10% chance of keeping units active.

Fine-tune all layers. In this scenario we retrain all network parameters, including all convolutional layers on the bottom of the network.

Train from scratch. As a baseline we train the full network from scratch on UCF-101 alone.

Results. To prepare UCF-101 data for classification we sampled 50 clips from every video and followed the same evaluation protocol as for Sports across the 3 suggested folds. We reached out to the authors of [22] to obtain the YouTube video IDs of UCF-101 videos, but unfortunately these were not available and hence we cannot guarantee that the Sports-1M dataset has no overlap with UCF-101. However, these concerns are somewhat mitigated as we only use a few sampled clips from every video.

We use the Slow Fusion network in our UCF-101 experiments as it provides the best performance on Sports-1M. The results of the experiments can be seen on Table 3. Interestingly, retraining the softmax layer alone does not perform best (possibly because the high-level features are too specific to sports) and the other extreme of fine-tuning all layers is also not adequate (likely due to overfitting). Instead, the best performance is obtained by taking a balanced approach and retraining the top few layers of the network. Lastly, training the entire network from scratch consistently leads to massive overfitting and dismal performance.

Performance by group. We further break down our performance by 5 broad groups of classes present in the UCF-101 dataset. We compute the average precision of every class and then compute the mean average precision over classes in each group. As can be seen from Table 4, large fractions of our performance can be attributed to the Sports categories in UCF-101, but the other groups still display impressive performance considering that the only way to observe these types of frames in the training data is due to label noise. Moreover, the gain in performance when retraining only the top to retraining the top 3 layers is almost entirely due to improvements on non-Sports categories: Sports performance only decreases from 0.80 to 0.79, while mAP improves on all other categories.

5. Conclusions

We studied the performance of convolutional neural networks in large-scale video classification. We found that CNN architectures are capable of learning powerful features from weakly-labeled data that far surpass feature-based methods in performance and that these benefits are surprisingly robust to details of the connectivity of the architectures in time. Qualitative examination of network outputs and confusion matrices reveals interpretable errors.

Our results indicate that while the performance is not particularly sensitive to the architectural details of the connectivity in time, a Slow Fusion model consistently performs better than the early and late fusion alternatives. Sur-

prisingly, we find that a single-frame model already displays very strong performance, suggesting that local motion cues may not be critically important, even for a dynamic dataset such as Sports. An alternative theory is that more careful treatment of camera motion may be necessary (for example by extracting features in the local coordinate system of a tracked point, as seen in [25]), but this requires significant changes to a CNN architecture that we leave for future work. We also identified mixed-resolution architectures that consist of a low-resolution context and a high-resolution fovea stream as an effective way of speeding up CNNs without sacrificing accuracy.

Our transfer learning experiments on UCF-101 suggest that the learned features are generic and generalize other video classification tasks. In particular, we achieved the highest transfer learning performance by retraining the top 3 layers of the network.

In future work we hope to incorporate broader categories in the dataset to obtain more powerful and generic features, investigate approaches that explicitly reason about camera motion, and explore recurrent neural networks as a more powerful technique for combining clip-level predictions into global video-level predictions.

Acknowledgments: We thank Saurabh Singh, Abhinav Shrivastava, Jay Yagnik, Alex Krizhevsky, Quoc Le, Jeff Dean and Rajat Monga for helpful discussions.

References

[1] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. In *Human Behavior Understanding*, pages 29–39. Springer, 2011. 2, 3

[2] D. Ciresan, A. Giusti, J. Schmidhuber, et al. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, 2012. 1

[3] L. N. Clement Farabet, Camille Couprie and Y. LeCun. Learning hierarchical features for scene labeling. *PAMI*, 35(8), 2013. 1, 2

[4] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. *Internatinal Conference on Learning Representation*, 2013. 2

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, volume 1, 2005. 5

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012. 4

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2

[8] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, 2005. 2, 5

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2

[10] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *PAMI*, 35(1):221–231, 2013. 2, 3

[11] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 2, 3, 4

[12] I. Laptev. On space-time interest points. *IJCV*, 64(2-3):107–123, 2005. 2

[13] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008. 2

[14] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011. 2

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 2

[16] J. Liu, J. Luo, and M. Shah. Recognizing realistic actions from videos "in the wild". In *CVPR*, 2009. 2

[17] J. C. Niebles, C.-W. Chen, and L. Fei-Fei. Modeling temporal structure of decomposable motion segments for activity classification. In *ECCV*, pages 392–405. Springer, 2010. 2

[18] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *arXiv preprint arXiv:1403.6382*, 2014. 1, 2

[19] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *ICPR*, 2012. 2

[20] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 1, 2

[21] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, 2003. 2

[22] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 2, 7

[23] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *ECCV*. Springer, 2010. 2

[24] M. Varma and A. Zisserman. A statistical approach to texture classification from single images. *IJCV*, 62(1-2):61–81, 2005. 5

[25] H. Wang, A. Klaser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *CVPR*. IEEE, 2011. 2, 8

[26] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, C. Schmid, et al. Evaluation of local spatio-temporal features for action recognition. In *BMVC*, 2009. 2

[27] W. Yang and G. Toderici. Discriminative tag learning on youtube videos with latent sub-tags. In *CVPR*, 2011. 5

[28] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. *arXiv preprint arXiv:1311.2901*, 2013. 1, 3

Artificial Intelligence 48 (1991) 117–124
Elsevier

Book Review

Judea Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference**

Stig Kjær Andersen

Department of Medical Information and Image Analysis, University of Aalborg, Fr. Bajers vej 7, DK-9220 Aalborg, Denmark

Received February 1990
Revised June 1990

Introductory remarks

Judea Pearl's book, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, offers a comprehensive and coherent discussion of the important results of the Renaissance of the probabilistic approach to reasoning under uncertainty in AI. It is a book for which the author should be highly credited. Pearl has dedicated himself to presenting all aspects of a graph representation of the Bayesian probabilistic inference constituted by the belief-network formalism. He has accomplished this goal by providing the reader with a complete background for the formalism, by presenting the central issues, and by extending the scope smoothly to cover decision analysis and structural learning. Finally, Pearl relates this approach to other probabilistic (non-Bayesian) formalisms for handling uncertainty in AI.

Although the book is certainly not easy to read, especially for people who want to understand the full details of the subject, the casual reader can grasp the essence of the probabilistic approach to uncertainty in AI. To get the full benefit of the book, the reader should follow the author's advice on how to tackle the text, which is given in terms of the reader's background.

* (Morgan Kaufmann, San Mateo, CA, 1988); xix + 552 pages.

The Renaissance of probabilistic reasoning in AI

The Bayesian probabilistic view, which is essential for this book, is founded on the fundamental premise that all uncertainties can be represented and measured as probabilities. The primary justification for this premise lies in the formal, axiomatic development of the normative framework for rational behavior of individuals, in the face of uncertainty. Any individual, desiring to behave (at least in principle) in this way, is led to act as though his/her uncertain knowledge is represented using subjective probability.

Beginning in the mid-1970s, the coherent use of probabilities was abandoned for nearly a decade as a vehicle for reasoning under uncertainty in AI. To follow a normative probabilistic approach in uncertainty management, investigators essentially made a choice between oversimplified domain models, where a high degree of independence among variables was required, and computationally intractable domain models, described by the full joint of probabilities. Neither of these alternatives satisfied the requirement of the AI community for a knowledge language, rich enough in its semantics to handle uncertainty. Furthermore, the first alternative was too simplistic for real-world cases, whereas the second was combinatorially explosive in the acquisition of the probabilities as well as in the inference.

Instead, new calculi, such as the MYCIN certainty-factor, became popular. This calculus was fitted to the modularity of rule-based systems; however, the cost for the computational convenience of this calculus, when interpreted in the framework of probabilities, was that the normative validity was limited to simple tree-structured rule bases where no evidence supported more than one hypothesis.

Against this backdrop, in the early 1980s, Pearl laid the basis for circumventing those difficulties by introducing the AI community to a paradigm combining the belief-network knowledge representation and the Bayesian probabilistic updating scheme. This formalism provided the semantics for constructing domain models, which was missing in the earlier use of probabilities in AI. Through the topology of the graphical representation, it was possible precisely to describe dependencies and independencies among domain variables, and to utilize this description through an efficient, normative, and conceptually meaningful updating scheme. The use of probabilities in AI was thus brought out of its Middle Ages.

Pearl's book is a review of an astonishing amount of research that has as its backbone the belief-network paradigm. The kernel is half a dozen of the author's articles, already published in this journal. There is a clear commitment to exploit and defend to the extent possible the point that the Bayesian statistics combined with the semantics of the belief network offer the AI community an operational framework that goes beyond the traditional view of probabilities.

Belief networks: a short introduction to the central concept

A belief network as a domain model is essentially a graph consisting of nodes and arcs. The semantics of the nodes is domain concepts or domain variables, and the semantics of the arcs is the interactions between these nodes, described as conditional probabilities. Unfortunately, the literature does not maintain a consistent naming convention for belief networks, so several synonyms are used (see p. 13 in Pearl's book). More names can be added to that list, such as *probabilistic influence diagram* and *causal probabilistic net.*

To give the flavor of the type of inference we can accomplish with the belief-network methodology, we illustrate a simple network in Fig. 1. This example is taken from the book (p. 56 and Fig. 10.6, p. 503). Here proposition A, "The grass is wet" can be caused either by C_1, "It rained last night", or by C_2, "The sprinkler was on", and we can observe that O_1, "The grass is cold and shiny", or O_2, "My shoes are wet", could be caused by A. The semantics of the network reveals that C_1 and C_2 are conditionally dependent given A, and that O_1 and O_2 are conditionally independent given A. The prior and conditional probabilities, $p(C_1)$, $p(C_2)$, $p(A \mid C_1, C_2)$, $p(O_1 \mid A)$, and $p(O_2 \mid A)$ are assumed to be known, and the variables are either true or false.

One aspect of the probabilistic inference is exemplified in the following scenario: If the belief in A is updated by observation of C_2, our belief in C_1 does not change from the prior value; C_1 and C_2 are independent; however if the belief in A on the other hand is updated by observation of O_1 or O_2, then C_1 and C_2 will become dependent. That is, if we again observe C_2, the updating algorithm will change the belief in C_1 accordingly. The probability that "it rained last night" will decrease and the observation that "the sprinkler was on" is a plausible explanation for "the grass is wet". Although it is an oversimplified model, the scenario illustrates the point that the inference realizes changes between dependency and independency for pairs of variables as a built-in feature.

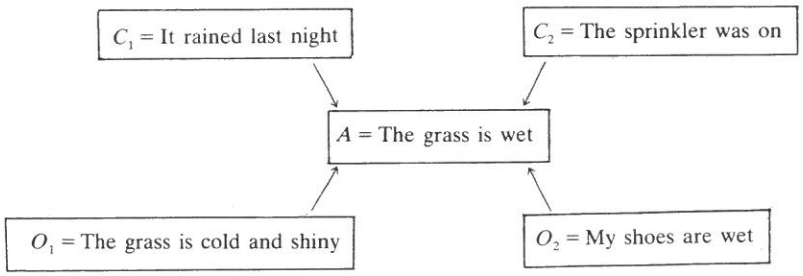


Fig. 1. An example of a simple belief network.

The form

Based on the solid foundation of the Bayesian assumptions, Pearl has developed an inference scheme for propagation of beliefs in a singly connected belief network. The common thread throughout the book is this scheme, where every action is locally scoped and autonomous, and where the formalism inherently distinguishes between *causal* and *evidential* reasoning support. The book builds up the arguments and the apparatus for using the belief network as a knowledge-representation scheme; when the reader is thus prepared, the author carefully presents the propagation scheme. No matter what relation the reader had to the Greek letters pi and lambda before reading the book, Pearl makes sure that these letters, in the mind of the reader, will be associated strongly with the two essential concepts in the formalism: the local propagation of causal and evidential information, respectively.

The readability of the inference scheme has been enhanced greatly over the years since Pearl first presented it. This scheme is the core of the book, and Pearl communicates it clearly to the reader, regardless of the reader's background. With this foundation, Pearl systematically guides the reader through all aspects of using the inference algorithm, focusing on current research within this field.

Pearl's platform is the intentional approach to uncertainty in AI, in which, technically speaking, uncertainty is a built-in feature of the modeling tool used to describe a domain. This perspective is contrary to the extensional approach, where the uncertainty is attached to model components and is combined by a mechanism not inherently belonging to the knowledge-representation paradigm employed.

The text contains clarifying examples that highlight the capability and power of the belief-network approach; a few of them will become quite familiar to the reader by the end of the book. For example, Tweety has survived this book as well. However, the reader must be alert to get a sense of the characteristics of problems that the formalism is less capable of handling. The examples are typical for a book with an underlying theoretical trait; that is, they are made as simple as possible to illustrate a certain problem, but fail to give the reader a feeling of real-world applications. More insight into the pitfalls the reader would face when using the graph language to model complex problems would have increased the value of the discussion.

Pearl explores the calculus of plausible inference in great detail, to illustrate that the tool at hand is superior, and it is difficult to point out any aspect on which he does not touch. A reader who has explored all the corners of this book will have the necessary tools and the background for constructing practical applications, utilizing the best of this paradigm. When it comes to formalizing the problems in the real world, however, the reader will be essentially on his or her own.

The reader could easily be left with the impression that most problems can be represented by a singly connected network. The reality is that single connectivity is often a too coarse model approximation. Pearl claims that his presentation is made with computational feasibility in mind, but although implementation of the basic actions can be done easily from the book, many practical issues, essential to get real systems running efficiently, are not discussed. Issues involving computational complexity are not the subject of this book.

The presentation of alternative views is restricted to those that are claimed to belong to the family of probabilistic approaches. There is a clear bias toward the Bayesian perspective in the treatment of the non-Bayesian probabilistic formalisms for handling uncertainty.

When all the features of the paradigm are presented, it is important that the reader keep in mind the basic philosophy of the author:

> We take for granted that probability calculus is unique in the way it handles context-dependent information and that no competing calculus exists that closely covers so many qualitative aspects of plausible reasoning. So the calculus is worthy of exploitation, emulation, or at the very least, serious exploration. We therefore take probability calculus as an initial model of human reasoning from which more refined models may originate, if needed. By exploring the limits of using probability calculus in machine implementations of plausible inference, we hope to identify conditions under which extensions, refinements, and simplifications are warranted. (p. 20)

One of the intentions of the book is to show that the probabilistic paradigm, built on a concept that is several hundred years old, adds power to the pool of techniques available to automate the human task of solving complex problems. Indeed Pearl succeeds in this task. Readers with a nonmathematical or nonstatistical background will probably find it difficult to penetrate the barrier between the "asterisk" sections and the "nonasterisk" sections—asterisks being the author's indication of a pathway.

The book is a valuable reference for the field of belief-network reasoning, and is a "must" for researchers in this field. What makes the book useful for a broader audience too, is the way the author has wrapped discussions, arguments, and examples of the use of probabilities around this research, achieving his goal of advocating the probabilistic method by constantly stressing the paradigm's compatibility with human reasoning.

The author has intended to make the book readable for a multilevel audience, and the reader should be prepared for a change in the flavor of different sections from purely philosophical to highly technical. This style maps well the dynamics of the author, but may give the reader trouble. As already

mentioned, part of the material is not easily accessible. Each chapter begins, to a lesser or greater extent, with a general discussion, and Pearl has done a good job in bridging discussions in picturesque language to the more stringent, mathematical, and axiomatic formulation of the major part of most chapters. This feature makes certain sections of the book eligible for use in courses, provided that the students' mathematical knowledge is adequate.

The contents

The book has 10 chapters. Each contains a few pages of bibliographical and historical remarks, as well as exercises (for which no answers are provided). At the back of the book is an 18-page bibliography, a five-page author index, and a seven-page subject index.

In the first chapter, Pearl clearly answers the question "Why probabilities?" by reviewing a few essential issues, such as causality (phrased in a pragmatic way as a practical computational device), bidirectional reasoning, and explaining away, all of which are pursued through the rest of the book; probability is something other than the manipulation of numbers.

The two ingredients needed to set the scene for the belief propagation—the machinery of Bayesian inference and the foundation for graphical representations—are the subjects of Chapters 2 and 3. The author tries to pinpoint what can be modeled in the unification of graph-based knowledge representation and Bayesian inference techniques using Markov networks and Bayesian networks: What emerges is a comprehensive picture of the expressiveness—or lack thereof—of modeling dependencies and independencies. The key to understanding the language of graphs, used to express a domain model, is a clear notion of the explicit versus the implicit expressiveness of this tool for formalizing knowledge.

Chapter 4 is devoted to the local belief-updating paradigm in belief networks. This pivotal chapter is self-contained, as though the reader was climbing up the ladder from updating beliefs in simple tree structures to dealing in complex ways with multiple paths (loops) in networks. The calculus is outlined carefully, and the basic algorithms for using the belief network are deduced, to provide a consistent set of beliefs when new evidence has been absorbed. In this chapter, Pearl is casting his tool.

With the tool at hand, Pearl adds different aspects to the use of the local updating paradigm in the following four chapters.

Chapter 5 deals with the handling and explaining of belief commitment, and explains how this fits with the autonomous message-passing scheme. Once more, Pearl uses the opportunity to anchor the basic propagation concepts. In Chapter 6, belief-network representation is extended to influence diagrams: that is, a new type of nodes is added that represents a decision, based on utility

functions. The connection to the field of decision theory is made, and the class of problems the extended paradigm can handle is expanded. This extension is consistently related to local belief propagation.

Chapter 7 adds further items to the list of means to utilize the local updating. First, taxonomic hierarchies are viewed from the belief-network paradigm, then the handling of continuous variables within the same framework is described, and, finally, Pearl focuses on second-order probabilities and argues that this concept already is a built-in feature of the Bayesian approach. Pearl has moved from the central core of the methodology outward, expanding the territory of probabilities, and here he enters an area where the arguments become more controversial.

Up to this point, the knowledge-acquisition process has not been addressed at all; however, the possibility for structural learning from data (joint probabilities) is the subject of Chapter 8. Unfortunately, only simple structures, such as trees, seem to be feasible objects for structural learning with the current state of the art, so structural learning does not solve the problems of knowledge acquisition.

In Chapter 9, Pearl boils down the major difference between the Bayesian-based methodology and the Dempster–Shafer technique, which is also based on probability theory, to the degree that they handle models that are not fully specified. Whereas the Bayesian approach basically requires a complete specification, the Dempster–Shafer technique for handling uncertainty is characterized by computing the probability of extensions to models that are not completely specified. The applicability of the different approaches is certainly dependent on the type of problem to which the technique is applied; the major example, in this chapter—the three-prisoner problem—is well within the territory of Bayesian reasoning.

The final chapter (Chapter 10) establishes a bridge between probability and logic. Pearl extends nonmonotonic logic to causal–evidential logic, which recognizes the bidirectional reasoning scheme (the foundation of the book), although it lacks the soft nature of expression that the probabilities allow for. The story is closed by a dialogue between a probabilist and a logician that gives the reader new perspectives on causality, commitment, and dependencies.

Summary

In conclusion, in the past decade, a large part of the AI community has disregarded the use of probabilities. This book knocks the breath out of the arguments against probability from these days, and proves that the probabilistic approach can be used to build real AI systems. The mission of this book is to collect the research that lays the foundation of a new era of probabilistic inference. Pearl has developed a coherent story from these different projects.

The book is a landmark in the landscape of probabilistic reasoning, and already provides an anchor for new research within this field. This book will definitely remain valid as a reference in the future given its multilevel readability; however, considering the potential applicability, this text is probably only the first of several monographs on this subject.

Readers, who are shopping for methods to be adapted in the practical construction of AI systems, may regard this book as a reference manual for the probabilistic approach. The process of constructing and formalizing domain knowledge in this probabilistic language, however, is not something that readers can simply look up in the index: they must synthesize the material themselves.

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

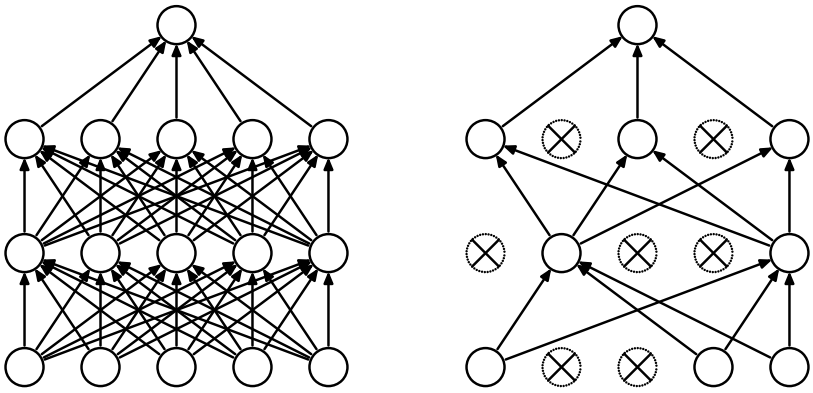
Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Keywords: neural networks, regularization, model combination, deep learning

1. Introduction

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting and many methods have been developed for reducing it. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and soft weight sharing (Nowlan and Hinton, 1992).

With unlimited computation, the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by



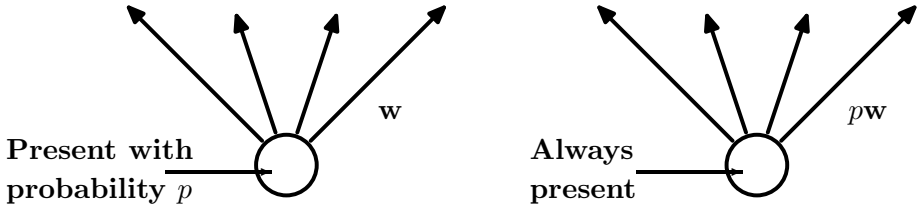
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters.

Model combination nearly always improves the performance of machine learning methods. With large neural networks, however, the obvious idea of averaging the outputs of many separately trained nets is prohibitively expensive. Combining several models is most helpful when the individual models are different from each other and in order to make neural net models different, they should either have different architectures or be trained on different data. Training many different architectures is hard because finding optimal hyperparameters for each architecture is a daunting task and training each large network requires a lot of computation. Moreover, large networks normally require large amounts of training data and there may not be enough data available to train different networks on different subsets of the data. Even if one was able to train many different large networks, using them all at test time is infeasible in applications where it is important to respond quickly.

Dropout is a technique that addresses both these issues. It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The term "dropout" refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure 1. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5.



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout (Figure 1b). A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. These networks all share weights so that the total number of parameters is still $O(n^2)$, or less. For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice. The idea is to use a single neural net at test time without dropout. The weights of this network are scaled-down versions of the trained weights. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time as shown in Figure 2. This ensures that for any hidden unit the *expected* output (under the distribution used to drop units at training time) is the same as the actual output at test time. By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time. We found that training a network with dropout and using this approximate averaging method at test time leads to significantly lower generalization error on a wide variety of classification problems compared to training with other regularization methods.

The idea of dropout is not limited to feed-forward neural nets. It can be more generally applied to graphical models such as Boltzmann Machines. In this paper, we introduce the dropout Restricted Boltzmann Machine model and compare it to standard Restricted Boltzmann Machines (RBM). Our experiments show that dropout RBMs are better than standard RBMs in certain respects.

This paper is structured as follows. Section 2 describes the motivation for this idea. Section 3 describes relevant previous work. Section 4 formally describes the dropout model. Section 5 gives an algorithm for training dropout networks. In Section 6, we present our experimental results where we apply dropout to problems in different domains and compare it with other forms of regularization and model combination. Section 7 analyzes the effect of dropout on different properties of a neural network and describes how dropout interacts with the network's hyperparameters. Section 8 describes the Dropout RBM model. In Section 9 we explore the idea of marginalizing dropout. In Appendix A we present a practical guide

for training dropout nets. This includes a detailed analysis of the practical considerations involved in choosing hyperparameters when training dropout networks.

2. Motivation

A motivation for dropout comes from a theory of the role of sex in evolution (Livnat et al., 2010). Sexual reproduction involves taking half the genes of one parent and half of the other, adding a very small amount of random mutation, and combining them to produce an offspring. The asexual alternative is to create an offspring with a slightly mutated copy of the parent's genes. It seems plausible that asexual reproduction should be a better way to optimize individual fitness because a good set of genes that have come to work well together can be passed on directly to the offspring. On the other hand, sexual reproduction is likely to break up these co-adapted sets of genes, especially if these sets are large and, intuitively, this should decrease the fitness of organisms that have already evolved complicated co-adaptations. However, sexual reproduction is the way most advanced organisms evolved.

One possible explanation for the superiority of sexual reproduction is that, over the long term, the criterion for natural selection may not be individual fitness but rather mix-ability of genes. The ability of a set of genes to be able to work well with another random set of genes makes them more robust. Since a gene cannot rely on a large set of partners to be present at all times, it must learn to do something useful on its own or in collaboration with a *small* number of other genes. According to this theory, the role of sexual reproduction is not just to allow useful new genes to spread throughout the population, but also to facilitate this process by reducing complex co-adaptations that would reduce the chance of a new gene improving the fitness of an individual. Similarly, each hidden unit in a neural network trained with dropout must learn to work with a randomly chosen sample of other units. This should make each hidden unit more robust and drive it towards creating useful features on its own without relying on other hidden units to correct its mistakes. However, the hidden units within a layer will still learn to do different things from each other. One might imagine that the net would become robust against dropout by making many copies of each hidden unit, but this is a poor solution for exactly the same reason as replica codes are a poor way to deal with a noisy channel.

A closely related, but slightly different motivation for dropout comes from thinking about successful conspiracies. Ten conspiracies each involving five people is probably a better way to create havoc than one big conspiracy that requires fifty people to all play their parts correctly. If conditions do not change and there is plenty of time for rehearsal, a big conspiracy can work well, but with non-stationary conditions, the smaller the conspiracy the greater its chance of still working. Complex co-adaptations can be trained to work well on a training set, but on novel test data they are far more likely to fail than multiple simpler co-adaptations that achieve the same thing.

3. Related Work

Dropout can be interpreted as a way of regularizing a neural network by adding noise to its hidden units. The idea of adding noise to the states of units has previously been used in the context of Denoising Autoencoders (DAEs) by Vincent et al. (2008, 2010) where noise

is added to the input units of an autoencoder and the network is trained to reconstruct the noise-free input. Our work extends this idea by showing that dropout can be effectively applied in the hidden layers as well and that it can be interpreted as a form of model averaging. We also show that adding noise is not only useful for unsupervised feature learning but can also be extended to supervised learning problems. In fact, our method can be applied to other neuron-based architectures, for example, Boltzmann Machines. While 5% noise typically works best for DAEs, we found that our weight scaling procedure applied at test time enables us to use much higher noise levels. Dropping out 20% of the input units and 50% of the hidden units was often found to be optimal.

Since dropout can be seen as a *stochastic* regularization technique, it is natural to consider its *deterministic* counterpart which is obtained by marginalizing out the noise. In this paper, we show that, in simple cases, dropout can be analytically marginalized out to obtain deterministic regularization methods. Recently, van der Maaten et al. (2013) also explored deterministic regularizers corresponding to different exponential-family noise distributions, including dropout (which they refer to as "blankout noise"). However, they apply noise to the inputs and only explore models with no hidden layers. Wang and Manning (2013) proposed a method for speeding up dropout by marginalizing dropout noise. Chen et al. (2012) explored marginalization in the context of denoising autoencoders.

In dropout, we minimize the loss function stochastically under a noise distribution. This can be seen as minimizing an expected loss function. Previous work of Globerson and Roweis (2006); Dekel et al. (2010) explored an alternate setting where the loss is minimized when an adversary gets to pick which units to drop. Here, instead of a noise distribution, the maximum number of units that can be dropped is fixed. However, this work also does not explore models with hidden units.

4. Model Description

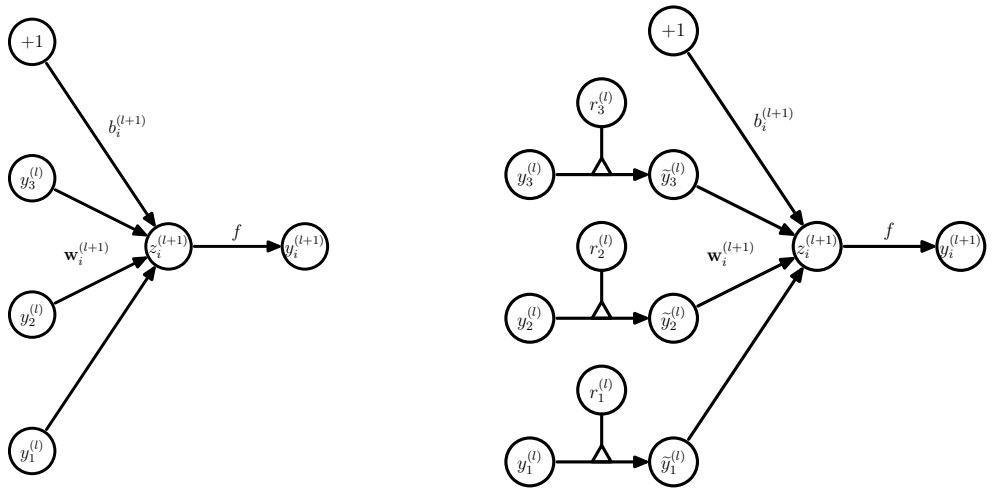
This section describes the dropout neural network model. Consider a neural network with L hidden layers. Let $l \in \{1, \ldots, L\}$ index the hidden layers of the network. Let $\mathbf{z}^{(l)}$ denote the vector of inputs into layer l, $\mathbf{y}^{(l)}$ denote the vector of outputs from layer l ($\mathbf{y}^{(0)} = \mathbf{x}$ is the input). $W^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases at layer l. The feed-forward operation of a standard neural network (Figure 3a) can be described as (for $l \in \{0, \ldots, L-1\}$ and any hidden unit i)

$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}), \end{aligned}$$

where f is any activation function, for example, $f(x) = 1/\left(1 + \exp(-x)\right)$.

With dropout, the feed-forward operation becomes (Figure 3b)

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}). \end{aligned}$$



(a) Standard network

(b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

Here $*$ denotes an element-wise product. For any layer l, $\mathbf{r}^{(l)}$ is a vector of independent Bernoulli random variables each of which has probability p of being 1. This vector is sampled and multiplied element-wise with the outputs of that layer, $\mathbf{y}^{(l)}$, to create the thinned outputs $\widetilde{\mathbf{y}}^{(l)}$. The thinned outputs are then used as input to the next layer. This process is applied at each layer. This amounts to sampling a sub-network from a larger network. For learning, the derivatives of the loss function are backpropagated through the sub-network. At test time, the weights are scaled as $W_{test}^{(l)} = pW^{(l)}$ as shown in Figure 2. The resulting neural network is used without dropout.

5. Learning Dropout Nets

This section describes a procedure for training dropout neural nets.

5.1 Backpropagation

Dropout neural networks can be trained using stochastic gradient descent in a manner similar to standard neural nets. The only difference is that for each training case in a mini-batch, we sample a thinned network by dropping out units. Forward and backpropagation for that training case are done only on this thinned network. The gradients for each parameter are averaged over the training cases in each mini-batch. Any training case which does not use a parameter contributes a gradient of zero for that parameter. Many methods have been used to improve stochastic gradient descent such as momentum, annealed learning rates and L2 weight decay. Those were found to be useful for dropout neural networks as well.

One particular form of regularization was found to be especially useful for dropout—constraining the norm of the incoming weight vector at each hidden unit to be upper bounded by a fixed constant c. In other words, if $\mathbf{w}$ represents the vector of weights incident on any hidden unit, the neural network was optimized under the constraint $||\mathbf{w}||_2 \leq c$. This constraint was imposed during optimization by projecting $\mathbf{w}$ onto the surface of a ball of radius c, whenever $\mathbf{w}$ went out of it. This is also called max-norm regularization since it implies that the maximum value that the norm of any weight can take is c. The constant

c is a tunable hyperparameter, which is determined using a validation set. Max-norm regularization has been previously used in the context of collaborative filtering (Srebro and Shraibman, 2005). It typically improves the performance of stochastic gradient descent training of deep neural nets, even when no dropout is used.

Although dropout alone gives significant improvements, using dropout along with max-norm regularization, large decaying learning rates and high momentum provides a significant boost over just using dropout. A possible justification is that constraining weight vectors to lie inside a ball of fixed radius makes it possible to use a huge learning rate without the possibility of weights blowing up. The noise provided by dropout then allows the optimization process to explore different regions of the weight space that would have otherwise been difficult to reach. As the learning rate decays, the optimization takes shorter steps, thereby doing less exploration and eventually settles into a minimum.

5.2 Unsupervised Pretraining

Neural networks can be pretrained using stacks of RBMs (Hinton and Salakhutdinov, 2006), autoencoders (Vincent et al., 2010) or Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009). Pretraining is an effective way of making use of unlabeled data. Pretraining followed by finetuning with backpropagation has been shown to give significant performance boosts over finetuning from random initializations in certain cases.

Dropout can be applied to finetune nets that have been pretrained using these techniques. The pretraining procedure stays the same. The weights obtained from pretraining should be scaled up by a factor of $1/p$. This makes sure that for each unit, the expected output from it under random dropout will be the same as the output during pretraining. We were initially concerned that the stochastic nature of dropout might wipe out the information in the pretrained weights. This did happen when the learning rates used during finetuning were comparable to the best learning rates for randomly initialized nets. However, when the learning rates were chosen to be smaller, the information in the pretrained weights seemed to be retained and we were able to get improvements in terms of the final generalization error compared to not using dropout when finetuning.

6. Experimental Results

We trained dropout neural networks for classification problems on data sets in different domains. We found that dropout improved generalization performance on *all* data sets compared to neural networks that did not use dropout. Table 1 gives a brief description of the data sets. The data sets are

- MNIST : A standard toy data set of handwritten digits.
- TIMIT : A standard speech benchmark for clean speech recognition.
- CIFAR-10 and CIFAR-100 : Tiny natural images (Krizhevsky, 2009).
- Street View House Numbers data set (SVHN) : Images of house numbers collected by Google Street View (Netzer et al., 2011).
- ImageNet : A large collection of natural images.
- Reuters-RCV1 : A collection of Reuters newswire articles.

- Alternative Splicing data set: RNA features for predicting alternative gene splicing (Xiong et al., 2011).

We chose a diverse set of data sets to demonstrate that dropout is a general technique for improving neural nets and is not specific to any particular application domain. In this section, we present some key results that show the effectiveness of dropout. A more detailed description of all the experiments and data sets is provided in Appendix B.

Data Set	**Domain**	**Dimensionality**	**Training Set**	**Test Set**
MNIST	Vision	784 (28×28 grayscale)	60K	10K
SVHN	Vision	3072 (32×32 color)	600K	26K
CIFAR-10/100	Vision	3072 (32×32 color)	60K	10K
ImageNet (ILSVRC-2012)	Vision	65536 (256×256 color)	1.2M	150K
TIMIT	Speech	2520 (120-dim, 21 frames)	1.1M frames	58K frames
Reuters-RCV1	Text	2000	200K	200K
Alternative Splicing	Genetics	1014	2932	733

Table 1: Overview of the data sets used in this paper.

6.1 Results on Image Data Sets

We used five image data sets to evaluate dropout—MNIST, SVHN, CIFAR-10, CIFAR-100 and ImageNet. These data sets include different image types and training set sizes. Models which achieve state-of-the-art results on *all* of these data sets use dropout.

6.1.1 MNIST

Method	**Unit Type**	**Architecture**	**Error %**
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5×240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**

Table 2: Comparison of different models on MNIST.

The MNIST data set consists of 28×28 pixel handwritten digit images. The task is to classify the images into 10 digit classes. Table 2 compares the performance of dropout with other techniques. The best performing neural networks for the permutation invariant

setting that do not use dropout or unsupervised pretraining achieve an error of about 1.60% (Simard et al., 2003). With dropout the error reduces to 1.35%. Replacing logistic units with rectified linear units (ReLUs) (Jarrett et al., 2009) further reduces the error to 1.25%. Adding max-norm regularization again reduces it to 1.06%. Increasing the size of the network leads to better results. A neural net with 2 layers and 8192 units per layer gets down to 0.95% error. Note that this network has more than 65 million parameters and is being trained on a data set of size 60,000. Training a network of this size to give good generalization error is very hard with standard regularization methods and early stopping. Dropout, on the other hand, prevents overfitting, even in this case. It does not even need early stopping. Goodfellow et al. (2013) showed that results can be further improved to 0.94% by replacing ReLU units with maxout units. All dropout nets use $p = 0.5$ for hidden units and $p = 0.8$ for input units. More experimental details can be found in Appendix B.1.

Dropout nets pretrained with stacks of RBMs and Deep Boltzmann Machines also give improvements as shown in Table 2. DBM—pretrained dropout nets achieve a test error of 0.79% which is the best performance ever reported for the permutation invariant setting. We note that it possible to obtain better results by using 2-D spatial information and augmenting the training set with distorted versions of images from the standard training set. We demonstrate the effectiveness of dropout in that setting on more interesting data sets.

In order to test the robustness of dropout, classification experiments were done with networks of many different architectures keeping all hyperparameters, including p, fixed. Figure 4 shows the test error rates obtained for these different architectures as training progresses. The same architectures trained with and without dropout have drastically different test errors as seen as by the two separate clusters of trajectories. Dropout gives a huge improvement across all architectures, without using hyperparameters that were tuned specifically for each architecture.

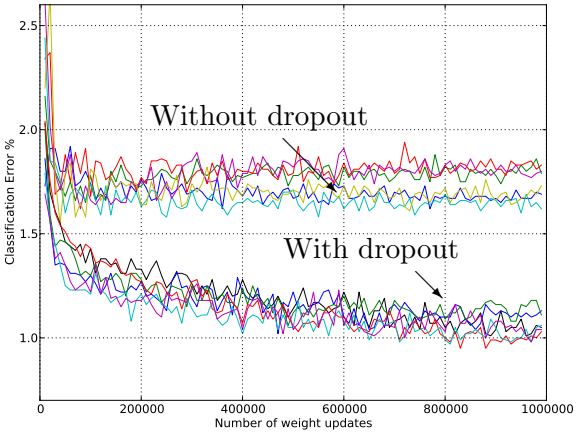


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

6.1.2 Street View House Numbers

The Street View House Numbers (SVHN) Data Set (Netzer et al., 2011) consists of color images of house numbers collected by Google Street View. Figure 5a shows some examples of images from this data set. The part of the data set that we use in our experiments consists of 32×32 color images roughly centered on a digit in a house number. The task is to identify that digit.

For this data set, we applied dropout to convolutional neural networks (LeCun et al., 1989). The best architecture that we found has three convolutional layers followed by 2 fully connected hidden layers. All hidden units were ReLUs. Each convolutional layer was

Method	**Error %**
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

Table 3: Results on the Street View House Numbers data set.

followed by a max-pooling layer. Appendix B.2 describes the architecture in more detail. Dropout was applied to all the layers of the network with the probability of retaining a hidden unit being $p = (0.9, 0.75, 0.75, 0.5, 0.5, 0.5)$ for the different layers of the network (going from input to convolutional layers to fully connected layers). Max-norm regularization was used for weights in both convolutional and fully connected layers. Table 3 compares the results obtained by different methods. We find that convolutional nets outperform other methods. The best performing convolutional nets that do not use dropout achieve an error rate of 3.95%. Adding dropout only to the fully connected layers reduces the error to 3.02%. Adding dropout to the convolutional layers as well further reduces the error to 2.55%. Even more gains can be obtained by using maxout units.

The additional gain in performance obtained by adding dropout in the convolutional layers (3.02% to 2.55%) is worth noting. One may have presumed that since the convolutional layers don't have a lot of parameters, overfitting is not a problem and therefore dropout would not have much effect. However, dropout in the lower layers still helps because it provides noisy inputs for the higher fully connected layers which prevents them from overfitting.

6.1.3 CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 data sets consist of 32×32 color images drawn from 10 and 100 categories respectively. Figure 5b shows some examples of images from this data set. A detailed description of the data sets, input preprocessing, network architectures and other experimental details is given in Appendix B.3. Table 4 shows the error rate obtained by different methods on these data sets. Without any data augmentation, Snoek et al. (2012) used Bayesian hyperparameter optimization to obtained an error rate of 14.98% on CIFAR-10. Using dropout in the fully connected layers reduces that to 14.32% and adding dropout in every layer further reduces the error to 12.61%. Goodfellow et al. (2013) showed that the error is further reduced to 11.68% by replacing ReLU units with maxout units. On CIFAR-100, dropout reduces the error from 43.48% to 37.20% which is a huge improvement. No data augmentation was used for either data set (apart from the input dropout).

(a) Street View House Numbers (SVHN) (b) CIFAR-10

Figure 5: Samples from image data sets. Each row corresponds to a different category.

Method	**CIFAR-10**	**CIFAR-100**
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

6.1.4 ImageNet

ImageNet is a data set of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. A subset of ImageNet with roughly 1000 images in each of 1000 categories is used in this challenge. Since the number of categories is rather large, it is conventional to report two error rates: top-1 and top-5, where the top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model. Figure 6 shows some predictions made by our model on a few test images.

ILSVRC-2010 is the only version of ILSVRC for which the test set labels are available, so most of our experiments were performed on this data set. Table 5 compares the performance of different methods. Convolutional nets with dropout outperform other methods by a large margin. The architecture and implementation details are described in detail in Krizhevsky et al. (2012).

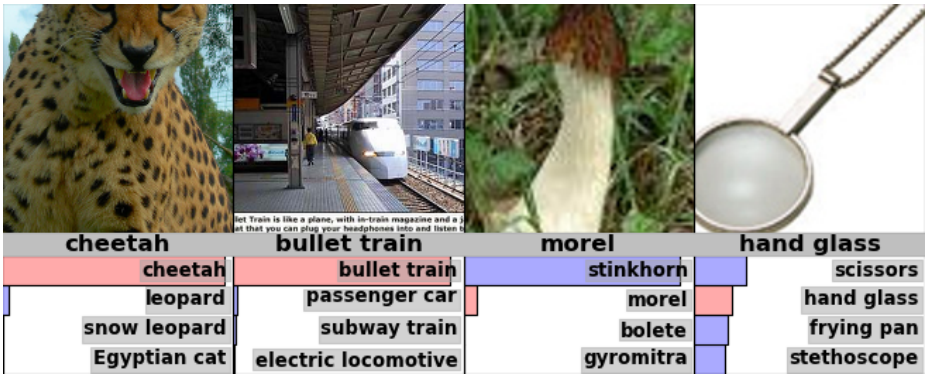


Figure 6: Some ImageNet test cases with the 4 most probable labels as predicted by our model. The length of the horizontal bars is proportional to the probability assigned to the labels by the model. Pink indicates ground truth.

Model	Top-1	Top-5
Sparse Coding (Lin et al., 2010)	47.1	28.2
SIFT + Fisher Vectors (Sanchez and Perronnin, 2011)	45.7	25.7
Conv Net + dropout (Krizhevsky et al., 2012)	37.5	17.0

Table 5: Results on the ILSVRC-2010 test set.

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Our model based on convolutional nets and dropout won the ILSVRC-2012 competition. Since the labels for the test set are not available, we report our results on the test set for the final submission and include the validation set results for different variations of our model. Table 6 shows the results from the competition. While the best methods based on standard vision features achieve a top-5 error rate of about 26%, convolutional nets with dropout achieve a test error of about 16% which is a staggering difference. Figure 6 shows some examples of predictions made by our model. We can see that the model makes very reasonable predictions, even when its best guess is not correct.

6.2 Results on TIMIT

Next, we applied dropout to a speech recognition task. We use the TIMIT data set which consists of recordings from 680 speakers covering 8 major dialects of American English reading ten phonetically-rich sentences in a controlled noise-free environment. Dropout neural networks were trained on windows of 21 log-filter bank frames to predict the label of the central frame. No speaker dependent operations were performed. Appendix B.4 describes the data preprocessing and training details. Table 7 compares dropout neural

nets with other models. A 6-layer net gives a phone error rate of 23.4%. Dropout further improves it to 21.8%. We also trained dropout nets starting from pretrained weights. A 4-layer net pretrained with a stack of RBMs get a phone error rate of 22.7%. With dropout, this reduces to 19.7%. Similarly, for an 8-layer net the error reduces from 20.5% to 19.7%.

Method	**Phone Error Rate%**
NN (6 layers) (Mohamed et al., 2010)	23.4
Dropout NN (6 layers)	21.8
DBN-pretrained NN (4 layers)	22.7
DBN-pretrained NN (6 layers) (Mohamed et al., 2010)	22.4
DBN-pretrained NN (8 layers) (Mohamed et al., 2010)	20.7
mcRBM-DBN-pretrained NN (5 layers) (Dahl et al., 2010)	20.5
DBN-pretrained NN (4 layers) + dropout	**19.7**
DBN-pretrained NN (8 layers) + dropout	**19.7**

Table 7: Phone error rate on the TIMIT core test set.

6.3 Results on a Text Data Set

To test the usefulness of dropout in the text domain, we used dropout networks to train a document classifier. We used a subset of the Reuters-RCV1 data set which is a collection of over 800,000 newswire articles from Reuters. These articles cover a variety of topics. The task is to take a bag of words representation of a document and classify it into 50 disjoint topics. Appendix B.5 describes the setup in more detail. Our best neural net which did not use dropout obtained an error rate of 31.05%. Adding dropout reduced the error to 29.62%. We found that the improvement was much smaller compared to that for the vision and speech data sets.

6.4 Comparison with Bayesian Neural Networks

Dropout can be seen as a way of doing an equally-weighted averaging of exponentially many models with shared weights. On the other hand, Bayesian neural networks (Neal, 1996) are the proper way of doing model averaging over the space of neural network structures and parameters. In dropout, each model is weighted equally, whereas in a Bayesian neural network each model is weighted taking into account the prior and how well the model fits the data, which is the more correct approach. Bayesian neural nets are extremely useful for solving problems in domains where data is scarce such as medical diagnosis, genetics, drug discovery and other computational biology applications. However, Bayesian neural nets are slow to train and difficult to scale to very large network sizes. Besides, it is expensive to get predictions from many large nets at test time. On the other hand, dropout neural nets are much faster to train and use at test time. In this section, we report experiments that compare Bayesian neural nets with dropout neural nets on a small data set where Bayesian neural networks are known to perform well and obtain state-of-the-art results. The aim is to analyze how much does dropout lose compared to Bayesian neural nets.

The data set that we use (Xiong et al., 2011) comes from the domain of genetics. The task is to predict the occurrence of alternative splicing based on RNA features. Alternative splicing is a significant cause of cellular diversity in mammalian tissues. Predicting the

Method	Code Quality (bits)
Neural Network (early stopping) (Xiong et al., 2011)	440
Regression, PCA (Xiong et al., 2011)	463
SVM, PCA (Xiong et al., 2011)	487
Neural Network with dropout	567
Bayesian Neural Network (Xiong et al., 2011)	**623**

Table 8: Results on the Alternative Splicing Data Set.

occurrence of alternate splicing in certain tissues under different conditions is important for understanding many human diseases. Given the RNA features, the task is to predict the probability of three splicing related events that biologists care about. The evaluation metric is Code Quality which is a measure of the negative KL divergence between the target and the predicted probability distributions (higher is better). Appendix B.6 includes a detailed description of the data set and this performance metric.

Table 8 summarizes the performance of different models on this data set. Xiong et al. (2011) used Bayesian neural nets for this task. As expected, we found that Bayesian neural nets perform better than dropout. However, we see that dropout improves significantly upon the performance of standard neural nets and outperforms all other methods. The challenge in this data set is to prevent overfitting since the size of the training set is small. One way to prevent overfitting is to reduce the input dimensionality using PCA. Thereafter, standard techniques such as SVMs or logistic regression can be used. However, with dropout we were able to prevent overfitting without the need to do dimensionality reduction. The dropout nets are very large (1000s of hidden units) compared to a few tens of units in the Bayesian network. This shows that dropout has a strong regularizing effect.

6.5 Comparison with Standard Regularizers

Several regularization methods have been proposed for preventing overfitting in neural networks. These include L2 weight decay (more generally Tikhonov regularization (Tikhonov, 1943)), lasso (Tibshirani, 1996), KL-sparsity and max-norm regularization. Dropout can be seen as another way of regularizing neural networks. In this section we compare dropout with some of these regularization methods using the MNIST data set.

The same network architecture (784-1024-1024-2048-10) with ReLUs was trained using stochastic gradient descent with different regularizations. Table 9 shows the results. The values of different hyperparameters associated with each kind of regularization (decay constants, target sparsity, dropout rate, max-norm upper bound) were obtained using a validation set. We found that dropout combined with max-norm regularization gives the lowest generalization error.

7. Salient Features

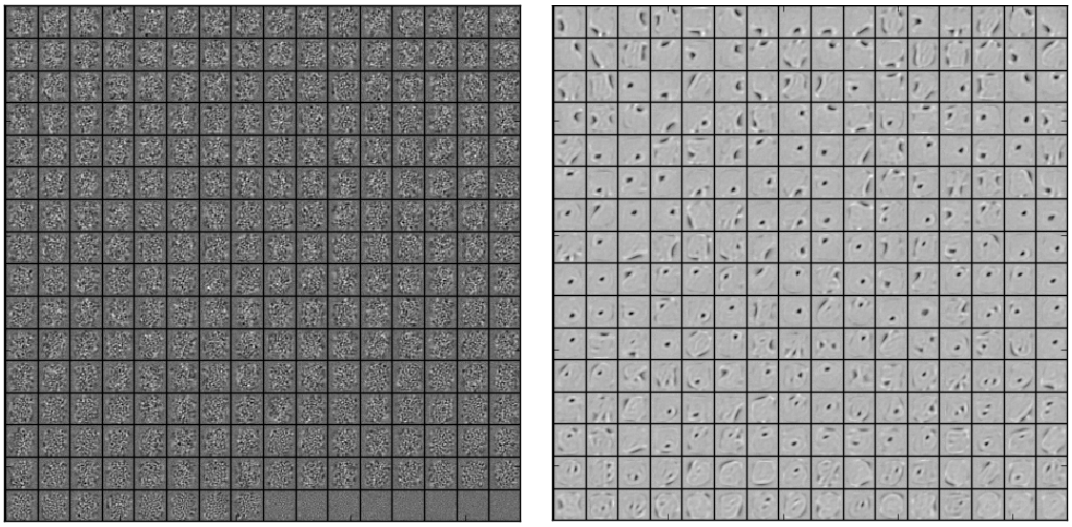
The experiments described in the previous section provide strong evidence that dropout is a useful technique for improving neural networks. In this section, we closely examine how dropout affects a neural network. We analyze the effect of dropout on the quality of features produced. We see how dropout affects the sparsity of hidden unit activations. We

Method	Test Classification error %
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

also see how the advantages obtained from dropout vary with the probability of retaining units, size of the network and the size of the training set. These observations give some insight into why dropout works so well.

7.1 Effect on Features



(a) Without dropout

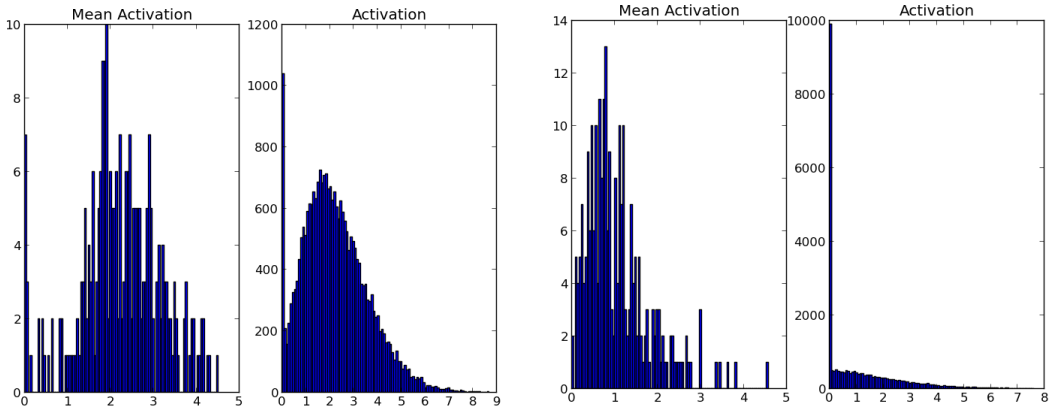
(b) Dropout with $p = 0.5$.

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

In a standard neural network, the derivative received by each parameter tells it how it should change so the final loss function is reduced, *given* what all other units are doing. Therefore, units may change in a way that they fix up the mistakes of the other units. This may lead to complex co-adaptations. This in turn leads to overfitting because these co-adaptations do not generalize to unseen data. We hypothesize that for each hidden unit, dropout prevents co-adaptation by making the presence of other hidden units unreliable. Therefore, a hidden unit cannot rely on other specific units to correct its mistakes. It must perform well in a wide variety of different contexts provided by the other hidden units. To observe this effect directly, we look at the first level features learned by neural networks trained on visual tasks with and without dropout.

Figure 7a shows features learned by an autoencoder on MNIST with a single hidden layer of 256 rectified linear units without dropout. Figure 7b shows the features learned by an identical autoencoder which used dropout in the hidden layer with $p = 0.5$. Both autoencoders had similar test reconstruction errors. However, it is apparent that the features shown in Figure 7a have co-adapted in order to produce good reconstructions. Each hidden unit on its own does not seem to be detecting a meaningful feature. On the other hand, in Figure 7b, the hidden units seem to detect edges, strokes and spots in different parts of the image. This shows that dropout does break up co-adaptations, which is probably the main reason why it leads to lower generalization errors.

7.2 Effect on Sparsity



(a) Without dropout

(b) Dropout with $p = 0.5$.

Figure 8: Effect of dropout on sparsity. ReLUs were used for both models. **Left**: The histogram of mean activations shows that most units have a mean activation of about 2.0. The histogram of activations shows a huge mode away from zero. Clearly, a large fraction of units have high activation. **Right**: The histogram of mean activations shows that most units have a smaller mean mean activation of about 0.7. The histogram of activations shows a sharp peak at zero. Very few units have high activation.

We found that as a side-effect of doing dropout, the activations of the hidden units become sparse, even when no sparsity inducing regularizers are present. Thus, dropout automatically leads to sparse representations. To observe this effect, we take the autoencoders trained in the previous section and look at the sparsity of hidden unit activations on a random mini-batch taken from the test set. Figure 8a and Figure 8b compare the sparsity for the two models. In a good sparse model, there should only be a few highly activated units for any data case. Moreover, the average activation of any unit across data cases should be low. To assess both of these qualities, we plot two histograms for each model. For each model, the histogram on the left shows the distribution of mean activations of hidden units across the minibatch. The histogram on the right shows the distribution of activations of the hidden units.

Comparing the histograms of activations we can see that fewer hidden units have high activations in Figure 8b compared to Figure 8a, as seen by the significant mass away from

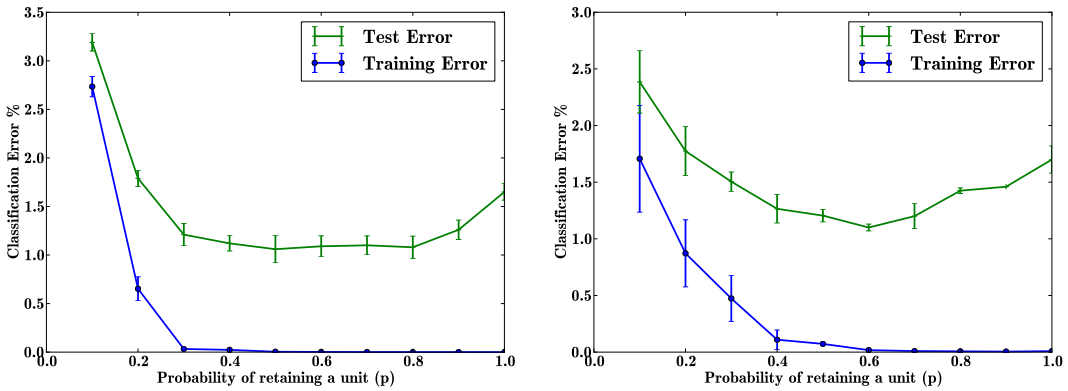
zero for the net that does not use dropout. The mean activations are also smaller for the dropout net. The overall mean activation of hidden units is close to 2.0 for the autoencoder without dropout but drops to around 0.7 when dropout is used.

7.3 Effect of Dropout Rate

Dropout has a tunable hyperparameter p (the probability of retaining a unit in the network). In this section, we explore the effect of varying this hyperparameter. The comparison is done in two situations.

1. The number of hidden units is held constant.
2. The number of hidden units is changed so that the expected number of hidden units that will be retained after dropout is held constant.

In the first case, we train the same network architecture with different amounts of dropout. We use a 784-2048-2048-2048-10 architecture. No input dropout was used. Figure 9a shows the test error obtained as a function of p. If the architecture is held constant, having a small p means very few units will turn on during training. It can be seen that this has led to underfitting since the training error is also high. We see that as p increases, the error goes down. It becomes flat when $0.4 \leq p \leq 0.8$ and then increases as p becomes close to 1.



(a) Keeping n fixed. (b) Keeping pn fixed.

Figure 9: Effect of changing dropout rates on MNIST.

Another interesting setting is the second case in which the quantity pn is held constant where n is the number of hidden units in any particular layer. This means that networks that have small p will have a large number of hidden units. Therefore, after applying dropout, the expected number of units that are present will be the same across different architectures. However, the test networks will be of different sizes. In our experiments, we set $pn = 256$ for the first two hidden layers and $pn = 512$ for the last hidden layer. Figure 9b shows the test error obtained as a function of p. We notice that the magnitude of errors for small values of p has reduced by a lot compared to Figure 9a (for $p = 0.1$ it fell from 2.7% to 1.7%). Values of p that are close to 0.6 seem to perform best for this choice of pn but our usual default value of 0.5 is close to optimal.

7.4 Effect of Data Set Size

One test of a good regularizer is that it should make it possible to get good generalization error from models with a large number of parameters trained on small data sets. This section explores the effect of changing the data set size when dropout is used with feed-forward networks. Huge neural networks trained in the standard way overfit massively on small data sets. To see if dropout can help, we run classification experiments on MNIST and vary the amount of data given to the network.

The results of these experiments are shown in Figure 10. The network was given data sets of size 100, 500, 1K, 5K, 10K and 50K chosen randomly from the MNIST training set. The same network architecture (784-1024-1024-2048-10) was used for all data sets. Dropout with $p = 0.5$ was performed at all the hidden layers and $p = 0.8$ at the input layer. It can be observed that for extremely small data sets (100, 500) dropout does not give any improvements. The model has enough parameters that it can overfit on the training data, even with all the noise coming from dropout. As the size of the data set is increased, the gain from doing dropout increases up to a point and then declines. This suggests that for any given architecture and dropout rate, there is a "sweet spot" corresponding to some amount of data that is large enough to not be memorized in spite of the noise but not so large that overfitting is not a problem anyways.

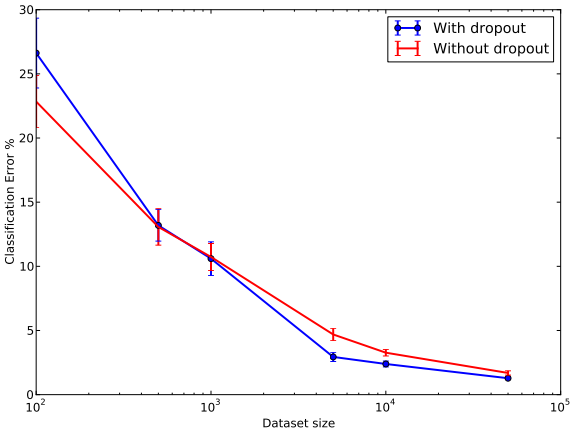


Figure 10: Effect of varying data set size.

7.5 Monte-Carlo Model Averaging vs. Weight Scaling

The efficient test time procedure that we propose is to do an approximate model combination by scaling down the weights of the trained neural network. An expensive but more correct way of averaging the models is to sample k neural nets using dropout for each test case and average their predictions. As $k \to \infty$, this Monte-Carlo model average gets close to the true model average. It is interesting to see empirically how many samples k are needed to match the performance of the approximate averaging method. By computing the error for different values of k we can see how quickly the error rate of the finite-sample average approaches the error rate of the true model average.

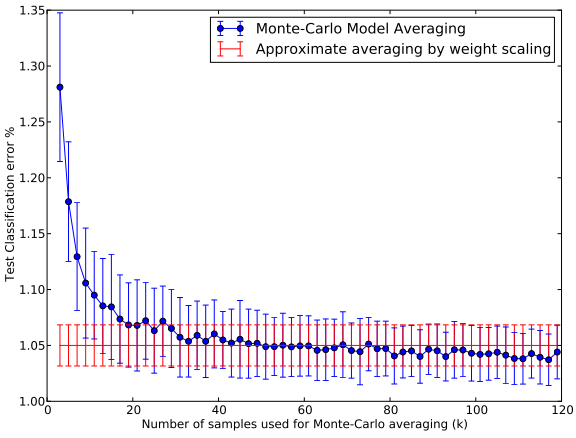


Figure 11: Monte-Carlo model averaging vs. weight scaling.

We again use the MNIST data set and do classification by averaging the predictions of k randomly sampled neural networks. Figure 11 shows the test error rate obtained for different values of k. This is compared with the error obtained using the weight scaling method (shown as a horizontal line). It can be seen that around $k = 50$, the Monte-Carlo method becomes as good as the approximate method. Thereafter, the Monte-Carlo method is slightly better than the approximate method but well within one standard deviation of it. This suggests that the weight scaling method is a fairly good approximation of the true model average.

8. Dropout Restricted Boltzmann Machines

Besides feed-forward neural networks, dropout can also be applied to Restricted Boltzmann Machines (RBM). In this section, we formally describe this model and show some results to illustrate its key properties.

8.1 Model Description

Consider an RBM with visible units $\mathbf{v} \in \{0,1\}^D$ and hidden units $\mathbf{h} \in \{0,1\}^F$. It defines the following probability distribution

$$P(\mathbf{h}, \mathbf{v}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \exp(\mathbf{v}^\top W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}).$$

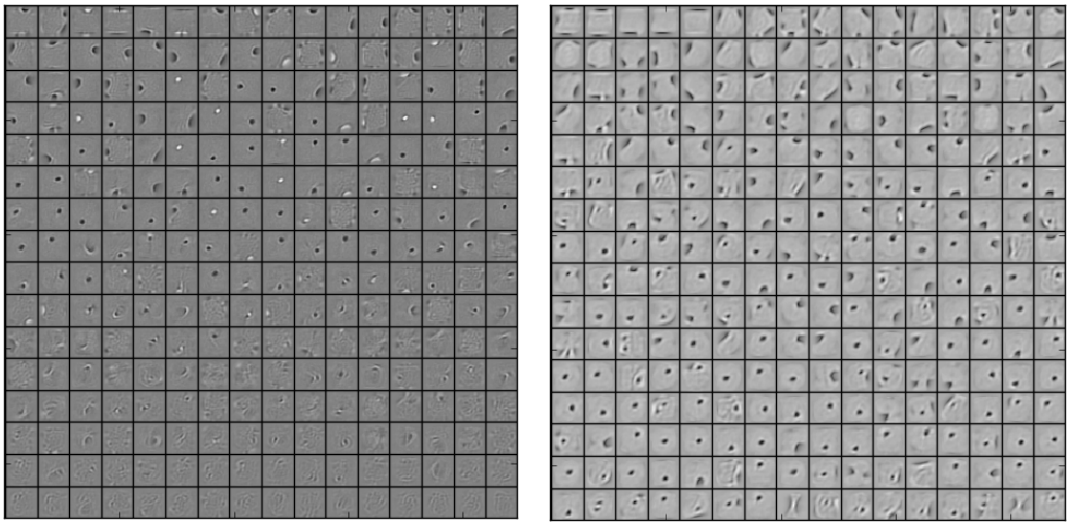
Where $\theta = \{W, \mathbf{a}, \mathbf{b}\}$ represents the model parameters and $\mathcal{Z}$ is the partition function.

Dropout RBMs are RBMs augmented with a vector of binary random variables $\mathbf{r} \in \{0,1\}^F$. Each random variable r_j takes the value 1 with probability p, independent of others. If r_j takes the value 1, the hidden unit h_j is retained, otherwise it is dropped from the model. The joint distribution defined by a Dropout RBM can be expressed as

$$\begin{aligned}
P(\mathbf{r}, \mathbf{h}, \mathbf{v}; p, \theta) &= P(\mathbf{r}; p) P(\mathbf{h}, \mathbf{v} | \mathbf{r}; \theta), \\
P(\mathbf{r}; p) &= \prod_{j=1}^{F} p^{r_j} (1-p)^{1-r_j}, \\
P(\mathbf{h}, \mathbf{v} | \mathbf{r}; \theta) &= \frac{1}{\mathcal{Z}'(\theta, \mathbf{r})} \exp(\mathbf{v}^\top W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}) \prod_{j=1}^{F} g(h_j, r_j), \\
g(h_j, r_j) &= \mathbb{1}(r_j = 1) + \mathbb{1}(r_j = 0)\mathbb{1}(h_j = 0).
\end{aligned}$$

$\mathcal{Z}'(\theta, \mathbf{r})$ is the normalization constant. $g(h_j, r_j)$ imposes the constraint that if $r_j = 0$, h_j must be 0. The distribution over $\mathbf{h}$, conditioned on $\mathbf{v}$ and $\mathbf{r}$ is factorial

$$\begin{aligned}
P(\mathbf{h} | \mathbf{r}, \mathbf{v}) &= \prod_{j=1}^{F} P(h_j | r_j, \mathbf{v}), \\
P(h_j = 1 | r_j, \mathbf{v}) &= \mathbb{1}(r_j = 1) \sigma\left(b_j + \sum_i W_{ij} v_i\right).
\end{aligned}$$



(a) Without dropout (b) Dropout with $p = 0.5$.

Figure 12: Features learned on MNIST by 256 hidden unit RBMs. The features are ordered by L2 norm.

The distribution over $\mathbf{v}$ conditioned on $\mathbf{h}$ is same as that of an RBM

$$P(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{D} P(v_i|\mathbf{h}),$$

$$P(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_j W_{ij} h_j\right).$$

Conditioned on $\mathbf{r}$, the distribution over $\{\mathbf{v}, \mathbf{h}\}$ is same as the distribution that an RBM would impose, except that the units for which $r_j = 0$ are dropped from $\mathbf{h}$. Therefore, the Dropout RBM model can be seen as a mixture of exponentially many RBMs with shared weights each using a different subset of $\mathbf{h}$.

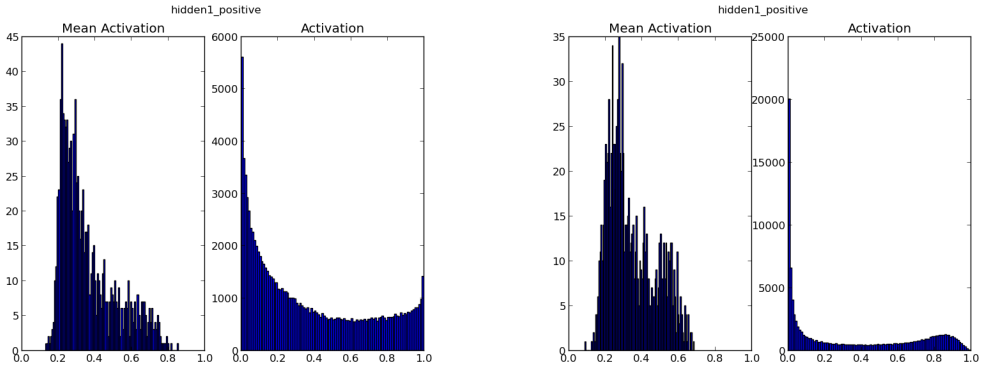
8.2 Learning Dropout RBMs

Learning algorithms developed for RBMs such as Contrastive Divergence (Hinton et al., 2006) can be directly applied for learning Dropout RBMs. The only difference is that $\mathbf{r}$ is first sampled and only the hidden units that are retained are used for training. Similar to dropout neural networks, a different $\mathbf{r}$ is sampled for each training case in every minibatch. In our experiments, we use CD-1 for training dropout RBMs.

8.3 Effect on Features

Dropout in feed-forward networks improved the quality of features by reducing co-adaptations. This section explores whether this effect transfers to Dropout RBMs as well.

Figure 12a shows features learned by a binary RBM with 256 hidden units. Figure 12b shows features learned by a dropout RBM with the same number of hidden units. Features



(a) Without dropout

(b) Dropout with $p = 0.5$.

Figure 13: Effect of dropout on sparsity. **Left**: The activation histogram shows that a large number of units have activations away from zero. **Right**: A large number of units have activations close to zero and very few units have high activation.

learned by the dropout RBM appear qualitatively different in the sense that they seem to capture features that are coarser compared to the sharply defined stroke-like features in the standard RBM. There seem to be very few dead units in the dropout RBM relative to the standard RBM.

8.4 Effect on Sparsity

Next, we investigate the effect of dropout RBM training on sparsity of the hidden unit activations. Figure 13a shows the histograms of hidden unit activations and their means on a test mini-batch after training an RBM. Figure 13b shows the same for dropout RBMs. The histograms clearly indicate that the dropout RBMs learn much sparser representations than standard RBMs even when no additional sparsity inducing regularizer is present.

9. Marginalizing Dropout

Dropout can be seen as a way of adding noise to the states of hidden units in a neural network. In this section, we explore the class of models that arise as a result of marginalizing this noise. These models can be seen as deterministic versions of dropout. In contrast to standard ("Monte-Carlo") dropout, these models do not need random bits and it is possible to get gradients for the marginalized loss functions. In this section, we briefly explore these models.

Deterministic algorithms have been proposed that try to learn models that are robust to feature deletion at test time (Globerson and Roweis, 2006). Marginalization in the context of denoising autoencoders has been explored previously (Chen et al., 2012). The marginalization of dropout noise in the context of linear regression was discussed in Srivastava (2013). Wang and Manning (2013) further explored the idea of marginalizing dropout to speed-up training. van der Maaten et al. (2013) investigated different input noise distributions and

the regularizers obtained by marginalizing this noise. Wager et al. (2013) describes how dropout can be seen as an adaptive regularizer.

9.1 Linear Regression

First we explore a very simple case of applying dropout to the classical problem of linear regression. Let $X \in \mathbb{R}^{N \times D}$ be a data matrix of N data points. $\mathbf{y} \in \mathbb{R}^N$ be a vector of targets. Linear regression tries to find a $\mathbf{w} \in \mathbb{R}^D$ that minimizes

$$||\mathbf{y} - X\mathbf{w}||^2.$$

When the input X is dropped out such that any input dimension is retained with probability p, the input can be expressed as $R * X$ where $R \in \{0, 1\}^{N \times D}$ is a random matrix with $R_{ij} \sim \text{Bernoulli}(p)$ and $*$ denotes an element-wise product. Marginalizing the noise, the objective function becomes

$$\underset{\mathbf{w}}{\text{minimize}} \quad \mathbb{E}_{R \sim \text{Bernoulli(p)}} \left[||\mathbf{y} - (R * X)\mathbf{w}||^2 \right].$$

This reduces to

$$\underset{\mathbf{w}}{\text{minimize}} \quad ||\mathbf{y} - pX\mathbf{w}||^2 + p(1-p)||\Gamma\mathbf{w}||^2,$$

where $\Gamma = (\text{diag}(X^\top X))^{1/2}$. Therefore, dropout with linear regression is equivalent, in expectation, to ridge regression with a particular form for Γ. This form of Γ essentially scales the weight cost for weight w_i by the standard deviation of the ith dimension of the data. If a particular data dimension varies a lot, the regularizer tries to squeeze its weight more.

Another interesting way to look at this objective is to absorb the factor of p into $\mathbf{w}$. This leads to the following form

$$\underset{\mathbf{w}}{\text{minimize}} \quad ||\mathbf{y} - X\widetilde{\mathbf{w}}||^2 + \frac{1-p}{p}||\Gamma\widetilde{\mathbf{w}}||^2,$$

where $\widetilde{\mathbf{w}} = p\mathbf{w}$. This makes the dependence of the regularization constant on p explicit. For p close to 1, all the inputs are retained and the regularization constant is small. As more dropout is done (by decreasing p), the regularization constant grows larger.

9.2 Logistic Regression and Deep Networks

For logistic regression and deep neural nets, it is hard to obtain a closed form marginalized model. However, Wang and Manning (2013) showed that in the context of dropout applied to logistic regression, the corresponding marginalized model can be trained approximately. Under reasonable assumptions, the distributions over the inputs to the logistic unit and over the gradients of the marginalized model are Gaussian. Their means and variances can be computed efficiently. This approximate marginalization outperforms Monte-Carlo dropout in terms of training time and generalization performance.

However, the assumptions involved in this technique become successively weaker as more layers are added. Therefore, the results are not directly applicable to deep networks.

Data Set	Architecture	Bernoulli dropout	Gaussian dropout
MNIST	2 layers, 1024 units each	1.08 ± 0.04	0.95 ± 0.04
CIFAR-10	3 conv + 2 fully connected layers	12.6 ± 0.1	12.5 ± 0.1

Table 10: Comparison of classification error % with Bernoulli and Gaussian dropout. For MNIST, the Bernoulli model uses $p = 0.5$ for the hidden units and $p = 0.8$ for the input units. For CIFAR-10, we use $p = (0.9, 0.75, 0.75, 0.5, 0.5, 0.5)$ going from the input layer to the top. The value of σ for the Gaussian dropout models was set to be $\sqrt{\frac{1-p}{p}}$. Results were averaged over 10 different random seeds.

10. Multiplicative Gaussian Noise

Dropout involves multiplying hidden activations by Bernoulli distributed random variables which take the value 1 with probability p and 0 otherwise. This idea can be generalized by multiplying the activations with random variables drawn from other distributions. We recently discovered that multiplying by a random variable drawn from $\mathcal{N}(1, 1)$ works just as well, or perhaps better than using Bernoulli noise. This new form of dropout amounts to adding a Gaussian distributed random variable with zero mean and standard deviation equal to the activation of the unit. That is, each hidden activation h_i is perturbed to $h_i + h_i r$ where $r \sim \mathcal{N}(0, 1)$, or equivalently $h_i r'$ where $r' \sim \mathcal{N}(1, 1)$. We can generalize this to $r' \sim \mathcal{N}(1, \sigma^2)$ where σ becomes an additional hyperparameter to tune, just like p was in the standard (Bernoulli) dropout. The expected value of the activations remains unchanged, therefore no weight scaling is required at test time.

In this paper, we described dropout as a method where we retain units with probability p at training time and scale down the weights by multiplying them by a factor of p at test time. Another way to achieve the same effect is to scale up the retained activations by multiplying by $1/p$ at training time and not modifying the weights at test time. These methods are equivalent with appropriate scaling of the learning rate and weight initializations at each layer.

Therefore, dropout can be seen as multiplying h_i by a Bernoulli random variable r_b that takes the value $1/p$ with probability p and 0 otherwise. $E[r_b] = 1$ and $Var[r_b] = (1 - p)/p$. For the Gaussian multiplicative noise, if we set $\sigma^2 = (1 - p)/p$, we end up multiplying h_i by a random variable r_g, where $E[r_g] = 1$ and $Var[r_g] = (1 - p)/p$. Therefore, both forms of dropout can be set up so that the random variable being multiplied by has the same mean and variance. However, given these first and second order moments, r_g has the highest entropy and r_b has the lowest. Both these extremes work well, although preliminary experimental results shown in Table 10 suggest that the high entropy case might work slightly better. For each layer, the value of σ in the Gaussian model was set to be $\sqrt{\frac{1-p}{p}}$ using the p from the corresponding layer in the Bernoulli model.

11. Conclusion

Dropout is a technique for improving neural networks by reducing overfitting. Standard backpropagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout breaks up these co-adaptations by

making the presence of any particular hidden unit unreliable. This technique was found to improve the performance of neural nets in a wide variety of application domains including object classification, digit recognition, speech recognition, document classification and analysis of computational biology data. This suggests that dropout is a general technique and is not specific to any domain. Methods that use dropout achieve state-of-the-art results on SVHN, ImageNet, CIFAR-100 and MNIST. Dropout considerably improved the performance of standard neural nets on other data sets as well.

This idea can be extended to Restricted Boltzmann Machines and other graphical models. The central idea of dropout is to take a large model that overfits easily and repeatedly sample and train smaller sub-models from it. RBMs easily fit into this framework. We developed Dropout RBMs and empirically showed that they have certain desirable properties.

One of the drawbacks of dropout is that it increases training time. A dropout network typically takes 2-3 times longer to train than a standard neural network of the same architecture. A major cause of this increase is that the parameter updates are very noisy. Each training case effectively tries to train a different random architecture. Therefore, the gradients that are being computed are not gradients of the final architecture that will be used at test time. Therefore, it is not surprising that training takes a long time. However, it is likely that this stochasticity prevents overfitting. This creates a trade-off between overfitting and training time. With more training time, one can use high dropout and suffer less overfitting. However, one way to obtain some of the benefits of dropout without stochasticity is to marginalize the noise to obtain a regularizer that does the same thing as the dropout procedure, in expectation. We showed that for linear regression this regularizer is a modified form of L2 regularization. For more complicated models, it is not obvious how to obtain an equivalent regularizer. Speeding up dropout is an interesting direction for future work.

Acknowledgments

This research was supported by OGS, NSERC and an Early Researcher Award.

Appendix A. A Practical Guide for Training Dropout Networks

Neural networks are infamous for requiring extensive hyperparameter tuning. Dropout networks are no exception. In this section, we describe heuristics that might be useful for applying dropout.

A.1 Network Size

It is to be expected that dropping units will reduce the capacity of a neural network. If n is the number of hidden units in any layer and p is the probability of retaining a unit, then instead of n hidden units, only pn units will be present after dropout, in expectation. Moreover, this set of pn units will be different each time and the units are not allowed to build co-adaptations freely. Therefore, if an n-sized layer is optimal for a standard neural net on any given task, a good dropout net should have at least n/p units. We found this to be a useful heuristic for setting the number of hidden units in both convolutional and fully connected networks.

A.2 Learning Rate and Momentum

Dropout introduces a significant amount of noise in the gradients compared to standard stochastic gradient descent. Therefore, a lot of gradients tend to cancel each other. In order to make up for this, a dropout net should typically use 10-100 times the learning rate that was optimal for a standard neural net. Another way to reduce the effect the noise is to use a high momentum. While momentum values of 0.9 are common for standard nets, with dropout we found that values around 0.95 to 0.99 work quite a lot better. Using high learning rate and/or momentum significantly speed up learning.

A.3 Max-norm Regularization

Though large momentum and learning rate speed up learning, they sometimes cause the network weights to grow very large. To prevent this, we can use max-norm regularization. This constrains the norm of the vector of incoming weights at each hidden unit to be bound by a constant c. Typical values of c range from 3 to 4.

A.4 Dropout Rate

Dropout introduces an extra hyperparameter—the probability of retaining a unit p. This hyperparameter controls the intensity of dropout. $p = 1$, implies no dropout and low values of p mean more dropout. Typical values of p for hidden units are in the range 0.5 to 0.8. For input layers, the choice depends on the kind of input. For real-valued inputs (image patches or speech frames), a typical value is 0.8. For hidden layers, the choice of p is coupled with the choice of number of hidden units n. Smaller p requires big n which slows down the training and leads to underfitting. Large p may not produce enough dropout to prevent overfitting.

Appendix B. Detailed Description of Experiments and Data Sets

.

This section describes the network architectures and training details for the experimental results reported in this paper. The code for reproducing these results can be obtained from `http://www.cs.toronto.edu/~nitish/dropout`. The implementation is GPU-based. We used the excellent CUDA libraries—cudamat (Mnih, 2009) and cuda-convnet (Krizhevsky et al., 2012) to implement our networks.

B.1 MNIST

The MNIST data set consists of 60,000 training and 10,000 test examples each representing a 28×28 digit image. We held out 10,000 random training images for validation. Hyperparameters were tuned on the validation set such that the best validation error was produced after 1 million weight updates. The validation set was then combined with the training set and training was done for 1 million weight updates. This net was used to evaluate the performance on the test set. This way of using the validation set was chosen because we found that it was easy to set up hyperparameters so that early stopping was not required at all. Therefore, once the hyperparameters were fixed, it made sense to combine the validation and training sets and train for a very long time.

The architectures shown in Figure 4 include all combinations of 2, 3, and 4 layer networks with 1024 and 2048 units in each layer. Thus, there are six architectures in all. For all the architectures (including the ones reported in Table 2), we used $p = 0.5$ in all hidden layers and $p = 0.8$ in the input layer. A final momentum of 0.95 and weight constraints with $c = 2$ was used in all the layers.

To test the limits of dropout's regularization power, we also experimented with 2 and 3 layer nets having 4096 and 8192 units. 2 layer nets gave improvements as shown in Table 2. However, the three layer nets performed slightly worse than 2 layer ones with the same level of dropout. When we increased dropout, performance improved but not enough to outperform the 2 layer nets.

B.2 SVHN

The SVHN data set consists of approximately 600,000 training images and 26,000 test images. The training set consists of two parts—A standard labeled training set and another set of labeled examples that are easy. A validation set was constructed by taking examples from both the parts. Two-thirds of it were taken from the standard set (400 per class) and one-third from the extra set (200 per class), a total of 6000 samples. This same process is used by Sermanet et al. (2012). The inputs were RGB pixels normalized to have zero mean and unit variance. Other preprocessing techniques such as global or local contrast normalization or ZCA whitening did not give any noticeable improvements.

The best architecture that we found uses three convolutional layers each followed by a max-pooling layer. The convolutional layers have 96, 128 and 256 filters respectively. Each convolutional layer has a 5×5 receptive field applied with a stride of 1 pixel. Each max pooling layer pools 3×3 regions at strides of 2 pixels. The convolutional layers are followed by two fully connected hidden layers having 2048 units each. All units use the rectified linear activation function. Dropout was applied to all the layers of the network with the probability of retaining the unit being $p = (0.9, 0.75, 0.75, 0.5, 0.5, 0.5)$ for the different layers of the network (going from input to convolutional layers to fully connected layers). In addition, the max-norm constraint with $c = 4$ was used for all the weights. A momentum of 0.95 was used in all the layers. These hyperparameters were tuned using a validation set. Since the training set was quite large, we did not combine the validation set with the training set for final training. We reported test error of the model that had smallest validation error.

B.3 CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 data sets consists of 50,000 training and 10,000 test images each. They have 10 and 100 image categories respectively. These are 32×32 color images. We used 5,000 of the training images for validation. We followed the procedure similar to MNIST, where we found the best hyperparameters using the validation set and then combined it with the training set. The images were preprocessed by doing global contrast normalization in each color channel followed by ZCA whitening. Global contrast normalization means that for image and each color channel in that image, we compute the mean of the pixel intensities and subtract it from the channel. ZCA whitening means that we mean center the data, rotate it onto its principle components, normalize each component

and then rotate it back. The network architecture and dropout rates are same as that for SVHN, except the learning rates for the input layer which had to be set to smaller values.

B.4 TIMIT

The open source Kaldi toolkit (Povey et al., 2011) was used to preprocess the data into log-filter banks. A monophone system was trained to do a forced alignment and to get labels for speech frames. Dropout neural networks were trained on windows of 21 consecutive frames to predict the label of the central frame. No speaker dependent operations were performed. The inputs were mean centered and normalized to have unit variance.

We used probability of retention $p = 0.8$ in the input layers and 0.5 in the hidden layers. Max-norm constraint with $c = 4$ was used in all the layers. A momentum of 0.95 with a high learning rate of 0.1 was used. The learning rate was decayed as $\epsilon_0(1 + t/T)^{-1}$. For DBN pretraining, we trained RBMs using CD-1. The variance of each input unit for the Gaussian RBM was fixed to 1. For finetuning the DBN with dropout, we found that in order to get the best results it was important to use a smaller learning rate (about 0.01). Adding max-norm constraints did not give any improvements.

B.5 Reuters

The Reuters RCV1 corpus contains more than 800,000 documents categorized into 103 classes. These classes are arranged in a tree hierarchy. We created a subset of this data set consisting of 402,738 articles and a vocabulary of 2000 words comprising of 50 categories in which each document belongs to exactly one class. The data was split into equal sized training and test sets. We tried many network architectures and found that dropout gave improvements in classification accuracy over all of them. However, the improvement was not as significant as that for the image and speech data sets. This might be explained by the fact that this data set is quite big (more than 200,000 training examples) and overfitting is not a very serious problem.

B.6 Alternative Splicing

The alternative splicing data set consists of data for 3665 cassette exons, 1014 RNA features and 4 tissue types derived from 27 mouse tissues. For each input, the target consists of 4 softmax units (one for tissue type). Each softmax unit has 3 states (*inc*, *exc*, *nc*) which are of the biological importance. For each softmax unit, the aim is to predict a distribution over these 3 states that matches the observed distribution from wet lab experiments as closely as possible. The evaluation metric is Code Quality which is defined as

$$\sum_{i=1}^{|\text{data points}|} \sum_{t \in \text{tissue types}} \sum_{s \in \{\text{inc, exc, nc}\}} p_{i,t}^s \log(\frac{q_t^s(r_i)}{\bar{p}^s}),$$

where, $p_{i,t}^s$ is the target probability for state s and tissue type t in input i; $q_t^s(r_i)$ is the predicted probability for state s in tissue type t for input r_i and $\bar{p}^s$ is the average of $p_{i,t}^s$ over i and t.

A two layer dropout network with 1024 units in each layer was trained on this data set. A value of $p = 0.5$ was used for the hidden layer and $p = 0.7$ for the input layer. Max-norm regularization with high decaying learning rates was used. Results were averaged across the same 5 folds used by Xiong et al. (2011).

References

M. Chen, Z. Xu, K. Weinberger, and F. Sha. Marginalized denoising autoencoders for domain adaptation. In *Proceedings of the 29th International Conference on Machine Learning*, pages 767–774. ACM, 2012.

G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. In *Advances in Neural Information Processing Systems 23*, pages 469–477, 2010.

O. Dekel, O. Shamir, and L. Xiao. Learning to classify with missing and corrupted features. *Machine Learning*, 81(2):149–178, 2010.

A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 353–360. ACM, 2006.

I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1319–1327. ACM, 2013.

G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of the International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, Z. Li, M.-H. Tsai, X. Zhou, T. Huang, and T. Zhang. Imagenet classification: fast descriptor coding and large-scale svm training. Large scale visual recognition challenge, 2010.

A. Livnat, C. Papadimitriou, N. Pippenger, and M. W. Feldman. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4):1452–1457, 2010.

V. Mnih. CUDAMat: a CUDA-based matrix class for Python. Technical Report UTML TR 2009-004, Department of Computer Science, University of Toronto, November 2009.

A. Mohamed, G. E. Dahl, and G. E. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 2010.

R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., 1996.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4), 1992.

D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, 2011.

R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455, 2009.

R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008.

J. Sanchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1665–1672, 2011.

P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR 2012)*, 2012.

P. Simard, D. Steinkraus, and J. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, volume 2, pages 958–962, 2003.

J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2960–2968, 2012.

N. Srebro and A. Shraibman. Rank, trace-norm and max-norm. In *Proceedings of the 18th annual conference on Learning Theory*, COLT'05, pages 545–560. Springer-Verlag, 2005.

N. Srivastava. Improving Neural Networks with Dropout. Master's thesis, University of Toronto, January 2013.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B. Methodological*, 58(1):267–288, 1996.

A. N. Tikhonov. On the stability of inverse problems. *Doklady Akademii Nauk SSSR*, 39(5): 195–198, 1943.

L. van der Maaten, M. Chen, S. Tyree, and K. Q. Weinberger. Learning with marginalized corrupted features. In *Proceedings of the 30th International Conference on Machine Learning*, pages 410–418. ACM, 2013.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103. ACM, 2008.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. In *Proceedings of the 27th International Conference on Machine Learning*, pages 3371–3408. ACM, 2010.

S. Wager, S. Wang, and P. Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems 26*, pages 351–359, 2013.

S. Wang and C. D. Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning*, pages 118–126. ACM, 2013.

H. Y. Xiong, Y. Barash, and B. J. Frey. Bayesian prediction of tissue-regulated splicing using RNA sequence and cellular context. *Bioinformatics*, 27(18):2554–2562, 2011.

M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.

Machine Learning 1: 81–106, 1986

Induction of Decision Trees

J.R. QUINLAN (munnari!nswitgould.oz!quinlan@seismo.css.gov)
Centre for Advanced Computing Sciences, New South Wales Institute of Technology, Sydney 2007, Australia

(Received August 1, 1985)

Key words: classification, induction, decision trees, information theory, knowledge acquisition, expert systems

Abstract. The technology for building knowledge-based systems by inductive inference from examples has been demonstrated successfully in several practical applications. This paper summarizes an approach to synthesizing decision trees that has been used in a variety of systems, and it describes one such system, ID3, in detail. Results from recent studies show ways in which the methodology can be modified to deal with information that is noisy and/or incomplete. A reported shortcoming of the basic algorithm is discussed and two means of overcoming it are compared. The paper concludes with illustrations of current research directions.

1. Introduction

Since artificial intelligence first achieved recognition as a discipline in the mid 1950's, machine learning has been a central research area. Two reasons can be given for this prominence. The ability to learn is a hallmark of intelligent behavior, so any attempt to understand intelligence as a phenomenon must include an understanding of learning. More concretely, learning provides a potential methodology for building high-performance systems.

Research on learning is made up of diverse subfields. At one extreme there are adaptive systems that monitor their own performance and attempt to improve it by adjusting internal parameters. This approach, characteristic of a large proportion of the early learning work, produced self-improving programs for playing games (Samuel, 1967), balancing poles (Michie, 1982), solving problems (Quinlan, 1969) and many other domains. A quite different approach sees learning as the acquisition of structured knowledge in the form of concepts (Hunt, 1962; Winston, 1975), discrimination nets (Feigenbaum and Simon, 1963), or production rules (Buchanan, 1978).

The practical importance of machine learning of this latter kind has been underlin-

ed by the advent of knowledge-based expert systems. As their name suggests, these systems are powered by knowledge that is represented explicitly rather than being implicit in algorithms. The knowledge needed to drive the pioneering expert systems was codified through protracted interaction between a domain specialist and a knowledge engineer. While the typical rate of knowledge elucidation by this method is a few rules per man day, an expert system for a complex task may require hundreds or even thousands of such rules. It is obvious that the interview approach to knowledge acquisition cannot keep pace with the burgeoning demand for expert systems; Feigenbaum (1981) terms this the 'bottleneck' problem. This perception has stimulated the investigation of machine learning methods as a means of explicating knowledge (Michie, 1983).

This paper focusses on one microcosm of machine learning and on a family of learning systems that have been used to build knowledge-based systems of a simple kind. Section 2 outlines the features of this family and introduces its members. All these systems address the same task of inducing decision trees from examples. After a more complete specification of this task, one system (ID3) is described in detail in Section 4. Sections 5 and 6 present extensions to ID3 that enable it to cope with noisy and incomplete information. A review of a central facet of the induction algorithm reveals possible improvements that are set out in Section 7. The paper concludes with two novel initiatives that give some idea of the directions in which the family may grow.

2. The TDIDT family of learning systems

Carbonell, Michalski and Mitchell (1983) identify three principal dimensions along which machine learning systems can be classified:

- the underlying learning strategies used;
- the representation of knowledge acquired by the system; and
- the application domain of the system.

This paper is concerned with a family of learning systems that have strong common bonds in these dimensions.

Taking these features in reverse order, the *application domain* of these systems is not limited to any particular area of intellectual activity such as Chemistry or Chess; they can be applied to any such area. While they are thus general-purpose systems, the applications that they address all involve *classification*. The product of learning is a piece of procedural knowledge that can assign a hitherto-unseen object to one of a specified number of disjoint classes. Examples of classification tasks are:

1. the diagnosis of a medical condition from symptoms, in which the classes could be either the various disease states or the possible therapies;
2. determining the game-theoretic value of a chess position, with the classes *won for white*, *lost for white*, and *drawn*; and
3. deciding from atmospheric observations whether a severe thunderstorm is unlikely, possible or probable.

It might appear that classification tasks are only a minuscule subset of procedural tasks, but even activities such as robot planning can be recast as classification problems (Dechter and Michie, 1985).

The members of this family are sharply characterized by their *representation of acquired knowledge* as decision trees. This is a relatively simple knowledge formalism that lacks the expressive power of semantic networks or other first-order representations. As a consequence of this simplicity, the learning methodologies used in the TDIDT family are considerably less complex than those employed in systems that can express the results of their learning in a more powerful language. Nevertheless, it is still possible to generate knowledge in the form of decision trees that is capable of solving difficult problems of practical significance.

The *underlying strategy* is non-incremental learning from examples. The systems are presented with a set of cases relevant to a classification task and develop a decision tree from the top down, guided by frequency information in the examples but not by the particular order in which the examples are given. This contrasts with incremental methods such as that employed in MARVIN (Sammut, 1985), in which a dialog is carried on with an instructor to 'debug' partially correct concepts, and that used by Winston (1975), in which examples are analyzed one at a time, each producing a small change in the developing concept; in both of these systems, the order in which examples are presented is most important. The systems described here search for patterns in the given examples and so must be able to examine and re-examine all of them at many stages during learning. Other well-known programs that share this data-driven approach include BACON (Langley, Bradshaw and Simon, 1983) and INDUCE (Michalski, 1980).

In summary, then, the systems described here develop decision trees for classification tasks. These trees are constructed beginning with the root of the tree and proceeding down to its leaves. The family's palindromic name emphasizes that its members carry out the *T*op-*D*own *I*nduction of *D*ecision *T*rees.

The example objects from which a classification rule is developed are known only through their values of a set of properties or attributes, and the decision trees in turn are expressed in terms of these same attributes. The examples themselves can be assembled in two ways. They might come from an existing database that forms a history of observations, such as patient records in some area of medicine that have accumulated at a diagnosis center. Objects of this kind give a reliable statistical picture but, since they are not organized in any way, they may be redundant or omit

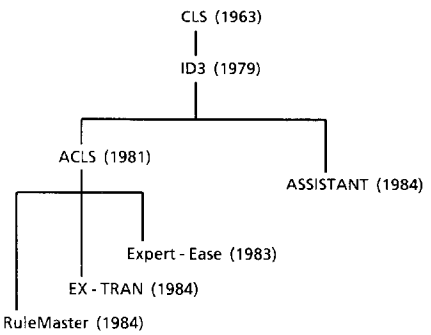


Figure 1. The TDIDT family tree.

uncommon cases that have not been encountered during the period of record-keeping. On the other hand, the objects might be a carefully culled set of tutorial examples prepared by a domain expert, each with some particular relevance to a complete and correct classification rule. The expert might take pains to avoid redundancy and to include examples of rare cases. While the family of systems will deal with collections of either kind in a satisfactory way, it should be mentioned that earlier TDIDT systems were designed with the 'historical record' approach in mind, but all systems described here are now often used with tutorial sets (Michie, 1985).

Figure 1 shows a family tree of the TDIDT systems. The patriarch of this family is Hunt's Concept Learning System framework (Hunt, Marin and Stone, 1966). CLS constructs a decision tree that attempts to minimize the cost of classifying an object. This cost has components of two types: the measurement cost of determining the value of property A exhibited by the object, and the misclassification cost of deciding that the object belongs to class J when its real class is K. CLS uses a lookahead strategy similar to minimax. At each stage, CLS explores the space of possible decision trees to a fixed depth, chooses an action to minimize cost in this limited space, then moves one level down in the tree. Depending on the depth of lookahead chosen, CLS can require a substantial amount of computation, but has been able to unearth subtle patterns in the objects shown to it.

ID3 (Quinlan, 1979, 1983a) is one of a series of programs developed from CLS in response to a challenging induction task posed by Donald Michie, viz. to decide from pattern-based features alone whether a particular chess position in the King-Rook vs King-Knight endgame is lost for the Knight's side in a fixed number of ply. A full description of ID3 appears in Section 4, so it is sufficient to note here that it embeds a tree-building method in an iterative outer shell, and abandons the cost-driven lookahead of CLS with an information-driven evaluation function.

ACLS (Paterson and Niblett, 1983) is a generalization of ID3. CLS and ID3 both require that each property used to describe objects has only values from a specified set. In addition to properties of this type, ACLS permits properties that have

unrestricted integer values. The capacity to deal with attributes of this kind has allowed ACLS to be applied to difficult tasks such as image recognition (Shepherd, 1983).

ASSISTANT (Kononenko, Bratko and Roskar, 1984) also acknowledges ID3 as its direct ancestor. It differs from ID3 in many ways, some of which are discussed in detail in later sections. ASSISTANT further generalizes on the integer-valued attributes of ACLS by permitting attributes with continuous (real) values. Rather than insisting that the classes be disjoint, ASSISTANT allows them to form a hierarchy, so that one class may be a finer division of another. ASSISTANT does not form a decision tree iteratively in the manner of ID3, but does include algorithms for choosing a 'good' training set from the objects available. ASSISTANT has been used in several medical domains with promising results.

The bottom-most three systems in the figure are commercial derivatives of ACLS. While they do not significantly advance the underlying theory, they incorporate many user-friendly innovations and utilities that expedite the task of generating and using decision trees. They all have industrial successes to their credit. Westinghouse Electric's Water Reactor Division, for example, points to a fuel-enrichment application in which the company was able to boost revenue by 'more than ten million dollars per annum' through the use of one of them.[1]

3. The induction task

We now give a more precise statement of the induction task. The basis is a universe of *objects* that are described in terms of a collection of *attributes*. Each attribute measures some important feature of an object and will be limited here to taking a (usually small) set of discrete, mutually exclusive values. For example, if the objects were Saturday mornings and the classification task involved the weather, attributes might be

outlook, with values {sunny, overcast, rain}
temperature, with values {cool, mild, hot}
humidity, with values {high, normal}
windy, with values {true, false}

Taken together, the attributes provide a zeroth-order language for characterizing objects in the universe. A particular Saturday morning might be described as

outlook: overcast
temperature: cool
humidity: normal
windy: false

[1] Letter cited in the journal *Expert Systems* (January, 1985), p. 20.

Each object in the universe belongs to one of a set of mutually exclusive *classes*. To simplify the following treatment, we will assume that there are only two such classes denoted *P* and *N*, although the extension to any number of classes is not difficult. In two-class induction tasks, objects of class P and N are sometimes referred to as *positive instances* and *negative instances*, respectively, of the concept being learned.

The other major ingredient is a *training set* of objects whose class is known. The induction task is to develop a *classification rule* that can determine the class of any object from its values of the attributes. The immediate question is whether or not the attributes provide sufficient information to do this. In particular, if the training set contains two objects that have identical values for each attribute and yet belong to different classes, it is clearly impossible to differentiate between these objects with reference only to the given attributes. In such a case attributes will be termed *inadequate* for the training set and hence for the induction task.

As mentioned above, a classification rule will be expressed as a decision tree. Table 1 shows a small training set that uses the 'Saturday morning' attributes. Each object's value of each attribute is shown, together with the class of the object (here, class P mornings are suitable for some unspecified activity). A decision tree that correctly classifies each object in the training set is given in Figure 2. Leaves of a decision tree are class names, other nodes represent attribute-based tests with a branch for each possible outcome. In order to classify an object, we start at the root of the tree, evaluate the test, and take the branch appropriate to the outcome. The process continues until a leaf is encountered, at which time the object is asserted to belong to

Table 1. A small training set

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

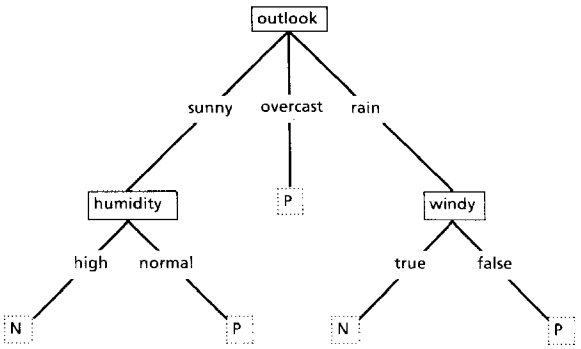


Figure 2. A simple decision tree

the class named by the leaf. Taking the decision tree of Figure 2, this process concludes that the object which appeared as an example at the start of this section, and which is not a member of the training set, should belong to class P. Notice that only a subset of the attributes may be encountered on a particular path from the root of the decision tree to a leaf; in this case, only the outlook attribute is tested before determining the class.

If the attributes are adequate, it is always possible to construct a decision tree that correctly classifies each object in the training set, and usually there are many such correct decision trees. The essence of induction is to move beyond the training set, i.e. to construct a decision tree that correctly classifies not only objects from the training set but other (unseen) objects as well. In order to do this, the decision tree must capture some meaningful relationship between an object's class and its values of the attributes. Given a choice between two decision trees, each of which is correct over the training set, it seems sensible to prefer the simpler one on the grounds that it is more likely to capture structure inherent in the problem. The simpler tree would therefore be expected to classify correctly more objects outside the training set. The decision tree of Figure 3, for instance, is also correct for the training set of Table 1, but its greater complexity makes it suspect as an 'explanation' of the training set.[2]

4. ID3

One approach to the induction task above would be to generate all possible decision trees that correctly classify the training set and to select the simplest of them. The

[2] The preference for simpler trees, presented here as a commonsense application of Occam's Razor, is also supported by analysis. Pearl (1978b) and Quinlan (1983a) have derived upper bounds on the expected error using different formalisms for generalizing from a set of known cases. For a training set of predetermined size, these bounds increase with the complexity of the induced generalization.

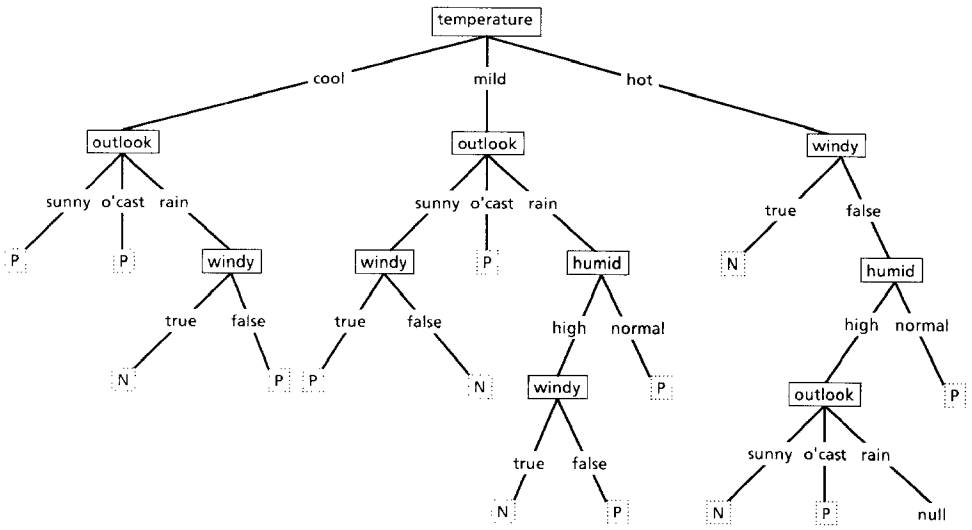


Figure 3. A complex decision tree.

number of such trees is finite but very large, so this approach would only be feasible for small induction tasks. ID3 was designed for the other end of the spectrum, where there are many attributes and the training set contains many objects, but where a reasonably good decision tree is required without much computation. It has generally been found to construct simple decision trees, but the approach it uses cannot guarantee that better trees have not been overlooked.

The basic structure of ID3 is iterative. A subset of the training set called the *window* is chosen at random and a decision tree formed from it; this tree correctly classifies all objects in the window. All other objects in the training set are then classified using the tree. If the tree gives the correct answer for all these objects then it is correct for the entire training set and the process terminates. If not, a selection of the incorrectly classified objects is added to the window and the process continues. In this way, correct decision trees have been found after only a few iterations for training sets of up to thirty thousand objects described in terms of up to 50 attributes. Empirical evidence suggests that a correct decision tree is usually found more quickly by this iterative method than by forming a tree directly from the entire training set. However, O'Keefe (1983) has noted that the iterative framework cannot be guaranteed to converge on a final tree unless the window can grow to include the entire training set. This potential limitation has not yet arisen in practice.

The crux of the problem is how to form a decision tree for an arbitrary collection C of objects. If C is empty or contains only objects of one class, the simplest decision tree is just a leaf labelled with the class. Otherwise, let T be any test on an object with possible outcomes $O_1, O_2, \ldots O_w$. Each object in C will give one of these outcomes for T, so T produces a partition $\{C_1, C_2, \ldots C_w\}$ of C with C_i containing those ob-

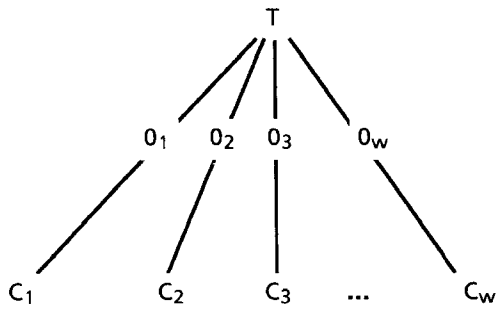


Figure 4. A tree structuring of the objects in C.

jects having outcome O_i. This is represented graphically by the tree form of Figure 4. If each subset C_i in this figure could be replaced by a decision tree for C_i, the result would be a decision tree for all of C. Moreover, so long as two or more C_i's are non-empty, each C_i is smaller than C. In the worst case, this divide-and-conquer strategy will yield single-object subsets that satisfy the one-class requirement for a leaf. Thus, provided that a test can always be found that gives a non-trivial partition of any set of objects, this procedure will always produce a decision tree that correctly classifies each object in C.

The choice of test is crucial if the decision tree is to be simple. For the moment, a test will be restricted to branching on the values of an attribute, so choosing a test comes down to selecting an attribute for the root of the tree. The first induction programs in the ID series used a seat-of-the-pants evaluation function that worked reasonably well. Following a suggestion of Peter Gacs, ID3 adopted an information-based method that depends on two assumptions. Let C contain p objects of class P and n of class N. The assumptions are:

(1) Any correct decision tree for C will classify objects in the same proportion as their representation in C. An arbitrary object will be determined to belong to class P with probability $p/(p+n)$ and to class N with probability $n/(p+n)$.
(2) When a decision tree is used to classify an object, it returns a class. A decision tree can thus be regarded as a source of a message 'P' or 'N', with the expected information needed to generate this message given by

$$I(p, n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

If attribute A with values $\{A_1, A_2, \ldots A_v\}$ is used for the root of the decision tree, it will partition C into $\{C_1, C_2, \ldots C_v\}$ where C_i contains those objects in C that have value A_i of A. Let C_i contain p_i objects of class P and n_i of class N. The expected

information required for the subtree for C_i is $I(p_i, n_i)$. The expected information required for the tree with A as root is then obtained as the weighted average

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(p_i, n_i)$$

where the weight for the ith branch is the proportion of the objects in C that belong to C_i. The information gained by branching on A is therefore

$$gain(A) = I(p, n) - E(A)$$

A good rule of thumb would seem to be to choose that attribute to branch on which gains the most information.[3] ID3 examines all candidate attributes and chooses A to maximize gain(A), forms the tree as above, and then uses the same process recursively to form decision trees for the residual subsets $C_1, C_2, \ldots C_v$.

To illustrate the idea, let C be the set of objects in Table 1. Of the 14 objects, 9 are of class P and 5 are of class N, so the information required for classification is

$$I(p, n) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.940 \text{ bits}$$

Now consider the outlook attribute with values {sunny, overcast, rain}. Five of the 14 objects in C have the first value (sunny), two of them from class P and three from class N, so

$$p_1 = 2 \quad n_1 = 3 \quad I(p_1, n_1) = 0.971$$

and similarly

$$p_2 = 4 \quad n_2 = 0 \quad I(p_2, n_2) = 0$$
$$p_3 = 3 \quad n_3 = 2 \quad I(p_3, n_3) = 0.971$$

The expected information requirement after testing this attribute is therefore

$$E(\text{outlook}) = \frac{5}{14} I(p_1, n_1) + \frac{4}{14} I(p_2, n_2) + \frac{5}{14} I(p_3, n_3)$$
$$= 0.694 \text{ bits}$$

[3] Since I(p,n) is constant for all attributes, maximizing the gain is equivalent to minimizing E(A), which is the mutual information of the attribute A and the class. Pearl (1978a) contains an excellent account of the rationale of information-based heuristics.

The gain of this attribute is then

$$\text{gain(outlook)} = 0.940 - \text{E(outlook)} = 0.246 \text{ bits}$$

Similar analysis gives

$$\text{gain(temperature)} = 0.029 \text{ bits}$$
$$\text{gain(humidity)} = 0.151 \text{ bits}$$
$$\text{gain(windy)} = 0.048 \text{ bits}$$

so the tree-forming method used in ID3 would choose outlook as the attribute for the root of the decision tree. The objects would then be divided into subsets according to their values of the outlook attribute and a decision tree for each subset would be induced in a similar fashion. In fact, Figure 2 shows the actual decision tree generated by ID3 from this training set.

A special case arises if C contains no objects with some particular value A_j of A, giving an empty C_j. ID3 labels such a leaf as 'null' so that it fails to classify any object arriving at that leaf. A better solution would generalize from the set C from which C_j came, and assign this leaf the more frequent class in C.

The worth of ID3's attribute-selecting heuristic can be assessed by the simplicity of the resulting decision trees, or, more to the point, by how well those trees express real relationships between class and attributes as demonstrated by the accuracy with which they classify objects other than those in the training set (their *predictive accuracy*). A straightforward method of assessing this predictive accuracy is to use only part of the given set of objects as a training set, and to check the resulting decision tree on the remainder.

Several experiments of this kind have been carried out. In one domain, 1.4 million chess positions described in terms of 49 binary-valued attributes gave rise to 715 distinct objects divided 65%:35% between the classes. This domain is relatively complex since a correct decision tree for all 715 objects contains about 150 nodes. When training sets containing 20% of these 715 objects were chosen at random, they produced decision trees that correctly classified over 84% of the unseen objects. In another version of the same domain, 39 attributes gave 551 distinct objects with a correct decision tree of similar size; training sets of 20% of these 551 objects gave decision trees of almost identical accuracy. In a simpler domain (1,987 objects with a correct decision tree of 48 nodes), randomly-selected training sets containing 20% of the objects gave decision trees that correctly classified 98% of the unseen objects. In all three cases, it is clear that the decision trees reflect useful (as opposed to random) relationships present in the data.

This discussion of ID3 is rounded off by looking at the computational requirements of the procedure. At each non-leaf node of the decision tree, the gain of each untested attribute A must be determined. This gain in turn depends on the values p_i

and n_i for each value A_i of A, so every object in C must be examined to determine its class and its value of A. Consequently, the computational complexity of the procedure at each such node is $O(|C| \cdot |A|)$, where $|A|$ is the number of attributes above. ID3's total computational requirement per iteration is thus proportional to the product of the size of the training set, the number of attributes and the number of non-leaf nodes in the decision tree. The same relationship appears to extend to the entire induction process, even when several iterations are performed. No exponential growth in time or space has been observed as the dimensions of the induction task increase, so the technique can be applied to large tasks.

5. Noise

So far, the information supplied in the training set has been assumed to be entirely accurate. Sadly, induction tasks based on real-world data are unlikely to find this assumption to be tenable. The description of objects may include attributes based on measurements or subjective judgements, both of which may give rise to errors in the values of attributes. Some of the objects in the training set may even have been misclassified. To illustrate the idea, consider the task of developing a classification rule for medical diagnosis from a collection of patient histories. An attribute might test for the presence of some substance in the blood and will almost inevitably give false positive or negative readings some of the time. Another attribute might assess the patient's build as slight, medium, or heavy, and different assessors may apply different criteria. Finally, the collection of case histories will probably include some patients for whom an incorrect diagnosis was made, with consequent errors in the class information provided in the training set.

What problems might errors of these kinds pose for the tree-building procedure described earlier? Consider again the small training set in Table 1, and suppose now that attribute outlook of object 1 is incorrectly recorded as overcast. Objects 1 and 3 will then have identical descriptions but belong to different classes, so the attributes become inadequate for this training set. The attributes will also become inadequate if attribute windy of object 4 is corrupted to true, because that object will then conflict with object 14. Finally, the initial training set can be accounted for by the simple decision tree of Figure 2 containing 8 nodes. Suppose that the class of object 3 were corrupted to N. A correct decision tree for this corrupted training set would now have to explain the apparent special case of object 3. The smallest such tree contains twelve nodes, half again as complex as the 'real' tree. These illustrations highlight two problems: errors in the training set may cause the attributes to become inadequate, or may lead to decision trees of spurious complexity.

Non-systematic errors of this kind in either the values of attributes or class information are usually referred to as *noise*. Two modifications are required if the tree-building algorithm is to be able to operate with a noise-affected training set.

(1) The algorithm must be able to work with inadequate attributes, because noise can cause even the most comprehensive set of attributes to appear inadequate.
(2) The algorithm must be able to decide that testing further attributes will not improve the predictive accuracy of the decision tree. In the last example above, it should refrain from increasing the complexity of the decision tree to accommodate a single noise-generated special case.

We start with the second requirement of deciding when an attribute is really relevant to classification. Let C be a collection of objects containing representatives of both classes, and let A be an attribute with random values that produces subsets $\{C_1, C_2, \ldots C_v\}$. Unless the proportion of class P objects in each of the C_i is exactly the same as the proportion of class P objects in C itself, branching on attribute A will give an apparent information gain. It will therefore appear that testing attribute A is a sensible step, even though the values of A are random and so cannot help to classify the objects in C.

One solution to this dilemma might be to require that the information gain of any tested attribute exceeds some absolute or percentage threshold. Experiments with this approach suggest that a threshold large enough to screen out irrelevant attributes also excludes attributes that are relevant, and the performance of the tree-building procedure is degraded in the noise-free case.

An alternative method based on the chi-square test for stochastic independence has been found to be more useful. In the previous notation, suppose attribute A produces subsets $\{C_1, C_2, \ldots C_v\}$ of C, where C_i contains p_i and n_i objects of class P and N, respectively. If the value of A is irrelevant to the class of an object in C, the expected value p'_i of p_i should be

$$p'_i = p \cdot \frac{p_i + n_i}{p + n}$$

If n'_i is the corresponding expected value of n_i, the statistic

$$\sum_{i=1}^{v} \frac{(p_i - p'_i)^2}{p'_i} + \frac{(n_i - n'_i)^2}{n'_i}$$

is approximately chi-square with v-1 degrees of freedom. Provided that none of the values p'_i or n'_i are very small, this statistic can be used to determine the confidence with which one can reject the hypothesis that A is independent of the class of objects in C (Hogg and Craig, 1970). The tree-building procedure can then be modified to prevent testing any attribute whose irrelevance cannot be rejected with a very high (e.g. 99%) confidence level. This has been found effective in preventing over-

complex trees that attempt to 'fit the noise' without affecting performance of the procedure in the noise-free case.[4]

Turning now to the first requirement, we see that the following situation can arise: a collection of C objects may contain representatives of both classes, yet further testing of C may be ruled out, either because the attributes are inadequate and unable to distinguish among the objects in C, or because each attribute has been judged to be irrelevant to the class of objects in C. In this situation it is necessary to produce a leaf labelled with class information, but the objects in C are not all of the same class.

Two possibilities suggest themselves. The notion of class could be generalized to allow the value $p/(p+n)$ in the interval (0,1), a class of 0.8 (say) being interpreted as 'belonging to class P with probability 0.8'. An alternative approach would be to opt for the more numerous class, i.e. to assign the leaf to class P if $p > n$, to class N if $p < n$, and to either if $p = n$. The first approach minimizes the sum of the squares of the error over objects in C, while the second minimizes the sum of the absolute errors over objects in C. If the aim is to minimize expected error, the second approach might be anticipated to be superior, and indeed this has been found to be the case.

Several studies have been carried out to see how this modified procedure holds up under varying levels of noise (Quinlan 1983b, 1985a). One such study is outlined here based on the earlier-mentioned task with 551 objects and 39 binary-valued attributes. In each experiment, the whole set of objects was artificially corrupted as described below and used as a training set to produce a decision tree. Each object was then corrupted anew, classified by this tree and the error rate determined. This process was repeated twenty times to give more reliable averages.

In this study, values were corrupted as follows. A noise level of n percent applied to a value meant that, with probability n percent, the true value was replaced by a value chosen at random from among the values that could have appeared.[5] Table 2 shows the results when noise levels varying from 5% to 100% were applied to the values of the most noise-sensitive attribute, to the values of all attributes simultaneously, and to the class information. This table demonstrates the quite different forms of degradation observed. Destroying class information produces a linear increase in error so that, when all class information is noise, the resulting decision tree classifies objects entirely randomly. Noise in a single attribute does not have a dramatic effect. Noise in all attributes together, however, leads to a relatively rapid increase in error which reaches a peak and declines. The peak is somewhat inter-

[4] ASSISTANT uses an information-based measure to perform much the same function, but no comparative results are available to date.

[5] It might seem that the value should be replaced by an incorrect value. Consider, however, the case of a two-valued attribute corrupted with 100% noise. If the value of each object were replaced by the (only) incorrect value, the initial attribute will have been merely inverted with no loss of information.

Table 2. Error rates produced by noise in a single attribute, all attributes, and class information

Noise level	Single attribute	All attributes	Class information
5%	1.3%	11.9%	2.6%
10%	2.5%	18.9%	5.5%
15%	3.3%	24.6%	8.3%
20%	4.6%	27.8%	9.9%
30%	6.1%	29.5%	14.8%
40%	7.6%	30.3%	18.1%
50%	8.8%	29.2%	21.8%
60%	9.4%	27.5%	26.4%
70%	9.9%	25.9%	27.2%
80%	10.4%	26.0%	29.5%
90%	10.8%	25.6%	34.1%
100%	10.8%	25.9%	49.6%

esting, and can be explained as follows. Let C be a collection of objects containing p from class P and n from class N, respectively. At noise levels around 50%, the algorithm for constructing decision trees may still find relevant attributes to branch on, even though the performance of this tree on unseen but equally noisy objects will be essentially random. Suppose the tree for C classifies objects as class P with probability $p/(n+p)$. The expected error if objects with a similar class distribution to those in C were classified by this tree is given by

$$\frac{p}{p+n} \cdot \left(1 - \frac{p}{p+n}\right) + \frac{n}{p+n} \cdot \left(1 - \frac{n}{p+n}\right) = \frac{2pn}{(p+n)^2}$$

At very high levels of noise, however, the algorithm will find all attributes irrelevant and classify everything as the more frequent class; assume without loss of generality that this class is P. The expected error in this case is

$$\frac{p}{p+n} \cdot 0 + \frac{n}{p+n} \cdot 1 = \frac{n}{p+n}$$

which is less than the above expression since we have assumed that p is greater than n. The decline in error is thus a consequence of the chi-square cutoff coming into play as noise becomes more intense.

The table brings out the point that low levels of noise do not cause the tree-building machinery to fall over a cliff. For this task, a 5% noise level in a single attribute produces a degradation in performance of less than 2%; a 5% noise level in all attributes together produces a 12% degradation in classification performance; while a similar

noise level in class information results in a 3% degradation. Comparable figures have been obtained for other induction tasks.

One interesting point emerged from other experiments in which a correct decision tree formed from an uncorrupted training set was used to classify objects whose descriptions were corrupted. This scenario corresponds to forming a classification rule under controlled and sanitized laboratory conditions, then using it to classify objects in the field. For higher noise levels, the performance of the correct decision tree on corrupted data was found to be inferior to that of an imperfect decision tree formed from data corrupted to a similar level! (This phenomenon has an explanation similar to that given above for the peak in Table 2.) The moral seems to be that it is counter-productive to eliminate noise from the attribute information in the training set if these same attributes will be subject to high noise levels when the induced decision tree is put to use.

6. Unknown attribute values

The previous section examined modifications to the tree-building process that enabled it to deal with noisy or corrupted values. This section is concerned with an allied problem that also arises in practice: unknown attribute values. To continue the previous medical diagnosis example, what should be done when the patient case histories that are to form the training set are incomplete?

One way around the problem attempts to fill in an unknown value by utilizing information provided by context. Using the previous notation, let us suppose that a collection C of objects contains one whose value of attribute A is unknown. ASSISTANT (Kononenko *et al*, 1984) uses a Bayesian formalism to determine the probability that the object has value A_i of A by examining the distribution of values of A in C as a function of their class. Suppose that the object in question belongs to class P. The probability that the object has value A_i for attribute A can be expressed as

$$\text{prob}(A = A_i \mid \text{class} = P) = \frac{\text{prob}(A = A_i \;\&\; \text{class} = P)}{\text{prob}(\text{class} = P)} = \frac{p_i}{p}$$

where the calculation of p_i and p is restricted to those members of C whose value of A is known. Having determined the probability distribution of the unknown value over the possible values of A, this method could either choose the most likely value or divide the object into fractional objects, each with one possible value of A, weighted according to the probabilities above.

Alen Shapiro (private communication) has suggested using a decision-tree approach to determine the unknown values of an attribute. Let C′ be the subset of C consisting of those objects whose value of attribute A is defined. In C′, the original

Table 3. Proportion of times that an unknown attribute value is replaced by an incorrect value

Replacement method	Attribute		
	1	2	3
Bayesian	28%	27%	38%
Decision tree	19%	22%	19%
Most common value	28%	27%	40%

class (P or N) is regarded as another attribute while the value of attribute A becomes the 'class' to be determined. That is, C′ is used to construct a decision tree for determining the value of attribute A from the other attributes and the class. When constructed, this decision tree can be used to 'classify' each object in C − C′ and the result assigned as the unknown value of A.

Although these methods for determining unknown attribute values look good on paper, they give unconvincing results even when only a single value of one attribute is unknown; as might be expected, their performance is much worse when several values of several attributes are unknown. Consider again the 551-object 39-attribute task. We may ask how well the methods perform when asked to fill in a single unknown attribute value. Table 3 shows, for each of the three most important attributes, the proportion of times each method fails to replace an unknown value by its correct value. For comparison, the table also shows the same figure for the simple strategy: always replace an unknown value of an attribute with its most common value. The Bayesian method gives results that are scarcely better than those given by the simple strategy and, while the decision-tree method uses more context and is thereby more accurate, it still gives disappointing results.

Rather than trying to guess unknown attribute values, we could treat 'unknown' as a new possible value for each attribute and deal with it in the same way as other values. This can lead to an anomalous situation, as shown by the following example. Suppose A is an attribute with values $\{A_1, A_2\}$ and let C be a collection of objects such that

$$p_1 = 2 \quad p_2 = 2$$
$$n_1 = 2 \quad n_2 = 2$$

giving a value of 1 bit for E(A). Now let A′ be an identical attribute except that one of the objects with value A_1 of A has an unknown value of A′. A′ has the values $\{A'_1, A'_2, A'_3 = \text{unknown}\}$, so the corresponding values might be

$$p'_1 = 1 \quad p'_2 = 2 \quad p'_3 = 1$$
$$n'_1 = 2 \quad n'_2 = 2 \quad n'_3 = 0$$

resulting in a value of 0.84 bits for E(A′). In terms of the selection criterion developed earlier, A′ now seems to give a higher information gain than A. Thus, having unknown values may apparently increase the desirability of an attribute, a result entirely opposed to common sense. The conclusion is that treating 'unknown' as a separate value is not a solution to the problem.

One strategy which has been found to work well is as follows. Let A be an attribute with values $\{A_1, A_2, \ldots A_v\}$. For some collection C of objects, let the numbers of objects with value A_i of A be p_i and n_i, and let p_u and n_u denote the numbers of objects of class P and N respectively that have unknown values of A. When the information gain of attribute A is assessed, these objects with unknown values are distributed across the values of A in proportion to the relative frequency of these values in C. Thus the gain is assessed as if the true value of p_i were given by

$$p_i + p_u \cdot \text{ratio}_i$$

where

$$\text{ratio}_i = \frac{p_i + n_i}{\sum_i (p_i + n_i)}$$

and similarly for n_i. (This expression has the property that unknown values can only decrease the information gain of an attribute.) When an attribute has been chosen by the selection criterion, objects with unknown values of that attribute are discarded before forming decision trees for the subsets $\{C_i\}$.

The other half of the story is how unknown attribute values are dealt with during classification. Suppose that an object is being classified using a decision tree that wishes to branch on attribute A, but the object's value of attribute A is unknown. The correct procedure would take the branch corresponding to the real value A_i but, since this value is unknown, the only alternative is to explore all branches without forgetting that some are more probable than others.

Conceptually, suppose that, along with the object to be classified, we have been passed a *token* with some value T. In the situation above, each branch of A_i is then explored in turn, using a token of value

$$T \cdot \text{ratio}_i$$

i.e. the given token value is distributed across all possible values in proportion to the ratios above. The value passed to a branch may be distributed further by subsequent tests on other attributes for which this object has unknown values. Instead of a single path to a leaf, there may now be many, each qualified by its token value. These token values at the leaves are summed for each class, the result of the classification being

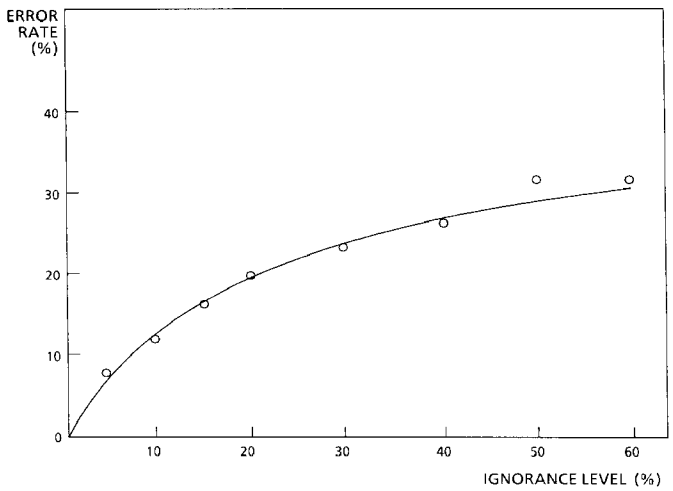


Figure 5. Error produced by unknown attribute values.

that class with the higher value. The distribution of values over the possible classes might also be used to compute a confidence level for the classification.

Straightforward though it may be, this procedure has been found to give a very graceful degradation as the incidence of unknown values increases. Figure 5 summarizes the results of an experiment on the now-familiar task with 551 objects and 39 attributes. Various 'ignorance levels' analogous to the earlier noise levels were explored, with twenty repititions at each level. For each run at an ignorance level of m percent, a copy of the 551 objects was made, replacing each value of every attribute by 'unknown' with m percent probability. A decision tree for these (incomplete) objects was formed as above, and then used to classify a new copy of each object corrupted in the same way. The figure shows that the degradation of performance with ignorance level is gradual. In practice, of course, an ignorance level even as high as 10% is unlikely – this would correspond to an average of one value in every ten of the object's description being unknown. Even so, the decision tree produced from such a patchy training set correctly classifies nearly ninety percent of objects that also have unknown values. A much lower level of degradation is observed when an object with unknown values is classified using a correct decision tree.

This treatment has assumed that no information whatsoever is available regarding an unknown attribute. Catlett (1985) has taken this approach a stage further by allowing partial knowledge of an attribute value to be stated in Shafer notation (Garvey, Lowrance and Fischler, 1981). This notation permits probabilistic assertions to be made about any subset or subsets of the possible values of an attribute that an object might have.

7. The selection criterion

Attention has recently been refocussed on the evaluation function for selecting the best attribute-based test to form the root of a decision tree. Recall that the criterion described earlier chooses the attribute that gains most information. In the course of their experiments, Bratko's group encountered a medical induction problem in which the attribute selected by the gain criterion ('age of patient', with nine value ranges) was judged by specialists to be less relevant than other attributes. This situation was also noted on other tasks, prompting Kononenko *et al* (1984) to suggest that the gain criterion tends to favor attributes with many values.

Analysis supports this finding. Let A be an attribute with values A_1, A_2, ... A_v and let A' be an attribute formed from A by splitting one of the values into two. If the values of A were sufficiently fine for the induction task at hand, we would not expect this refinement to increase the usefulness of A. Rather, it might be anticipated that excessive fineness would tend to obscure structure in the training set so that A' was in fact less useful than A. However, it can be proved that gain(A') is greater than or equal to gain(A), being equal to it only when the proportions of the classes are the same for both subdivisions of the original value. In general, then, gain(A') will exceed gain(A) with the result that the evaluation function of Section 4 will prefer A' to A. By analogy, attributes with more values will tend to be preferred to attributes with fewer.

As another way of looking at the problem, let A be an attribute with random values and suppose that the set of possible values of A is large enough to make it unlikely that two objects in the training set have the same value for A. Such an attribute would have maximum information gain, so the gain criterion would select it as the root of the decision tree. This would be a singularly poor choice since the value of A, being random, contains no information pertinent to the class of objects in the training set.

ASSISTANT (Kononenko *et al*, 1984) solves this problem by requiring that all tests have only two outcomes. If we have an attribute A as before with v values A_1, A_2, ... A_v, the decision tree no longer has a branch for each possible value. Instead, a subset S of the values is chosen and the tree has two branches, one for all values in the set and one for the remainder. The information gain is then computed as if all values in S were amalgamated into one single attribute value and all remaining values into another. Using this selection criterion (the *subset* criterion), the test chosen for the root of the decision tree uses the attribute and subset of its values that maximizes the information gain. Kononenko *et al* report that this modification led to smaller decision trees with an improved classification performance. However, the trees were judged to be less intelligible to human beings, in agreement with a similar finding of Shepherd (1983).

Limiting decision trees to a binary format harks back to CLS, in which each test was of the form 'attribute A has value A_i', with two branches corresponding to true and false. This is clearly a special case of the test implemented in ASSISTANT, which

permits a set of values, rather than a single value, to be distinguished from the others.

It is also worth noting that the method of dealing with attributes having continuous values follows the same binary approach. Let A be such an attribute and suppose that the distinct values of A that occur in C are sorted to give the sequence $V_1, V_2, \ldots, V_k$. Each pair of values V_i, V_{i+1} suggests a possible threshold

$$\frac{V_i + V_{i+1}}{2}$$

that divides the objects of C into two subsets, those with a value of A above and below the threshold respectively. The information gain of this division can then be investigated as above.

If all tests must be binary, there can be no bias in favor of attributes with large numbers of values. It could be argued, however, that ASSISTANT's remedy has undesirable side-effects that have to be taken into account. First, it can lead to decision trees that are even more unintelligible to human experts than is ordinarily the case, with unrelated attribute values being grouped together and with multiple tests on the same attribute.

More importantly, the subset criterion can require a large increase in computation. An attribute A with v values has 2^v value subsets and, when trivial and symmetric subsets are removed, there are still $2^{v-1}-1$ different ways of specifying the distinguished subset of attribute values. The information gain realized with each of these must be investigated, so a single attribute with v values has a computational requirement similar to $2^{v-1}-1$ binary attributes. This is not of particular consequence if v is small, but the approach would appear infeasible for an attribute with 20 values.

Another method of overcoming the bias is as follows. Consider again our training set containing p and n objects of class P and N respectively. As before, let attribute A have values $A_1, A_2, \ldots A_v$ and let the numbers of objects with value A_i of attribute A be p_i and n_i respectively. Enquiring about the value of attribute A itself gives rise to information, which can be expressed as

$$IV(A) = -\sum_{i=1}^{v} \frac{p_i+n_i}{p+n} \log_2 \frac{p_i+n_i}{p+n}$$

IV(A) thus measures the information content of the answer to the question, 'What is the value of attribute A?' As discussed earlier, gain(A) measures the reduction in the information requirement for a classification rule if the decision tree uses attribute A as a root. Ideally, as much as possible of the information provided by determining the value of an attribute should be useful for classification purposes or, equivalently, as little as possible should be 'wasted'. A good choice of attribute would then be one for which the ratio

$$\text{gain(A)} \;/\; \text{IV(A)}$$

is as large as possible. This ratio, however, may not always be defined – IV(A) may be zero – or it may tend to favor attributes for which IV(A) is very small. The *gain ratio* criterion selects, from among those attributes with an average-or-better gain, the attribute that maximizes the above ratio.

This can be illustrated by returning to the example based on the training set of Table 1. The information gain of the four attributes is given in Section 4 as

$$\begin{aligned} \text{gain(outlook)} &= 0.246 \text{ bits} \\ \text{gain(temperature)} &= 0.029 \text{ bits} \\ \text{gain(humidity)} &= 0.151 \text{ bits} \\ \text{gain(windy)} &= 0.048 \text{ bits} \end{aligned}$$

Of these, only outlook and humidity have above-average gain. For the outlook attribute, five objects in the training set have the value sunny, four have overcast and five have rain. The information obtained by determining the value of the outlook attribute is therefore

$$\begin{aligned} \text{IV(outlook)} &= -\frac{5}{14}\log_2\frac{5}{14} - \frac{4}{14}\log_2\frac{4}{14} - \frac{5}{14}\log_2\frac{5}{14} \\ &= 1.578 \text{ bits} \end{aligned}$$

Similarly,

$$\text{IV(humidity)} = -\frac{7}{14}\log_2\frac{7}{14} - \frac{7}{14}\log_2\frac{7}{14} = 1 \text{ bit}$$

So,

$$\begin{aligned} \text{gain ratio(outlook)} &= 0.246 \;/\; 1.578 = 0.156 \\ \text{gain ratio(humidity)} &= 0.151 \;/\; 1.000 = 0.151 \end{aligned}$$

The gain ratio criterion would therefore still select the outlook attribute for the root of the decision tree, although its superiority over the humidity attribute is now much reduced.

The various selection criteria have been compared empirically in a series of experiments (Quinlan, 1985b). When all attributes are binary, the gain ratio criterion has been found to give considerably smaller decision trees: for the 551-object task, it produces a tree of 143 nodes compared to the smallest previously-known tree of 175 nodes. When the task includes attributes with large numbers of values, the subset criterion gives smaller decision trees that also have better predictive performance, but can require much more computation. However, when these many-valued attributes are augmented by redundant attributes which contain the same information at a

lower level of detail, the gain ratio criterion gives decision trees with the greatest predictive accuracy. All in all, these experiments suggest that the gain ratio criterion does pick a good attribute for the root of the tree. Testing an attribute with many values, however, will fragment the training set C into very small subsets $\{C_i\}$ and the decision trees for these subsets may then have poor predictive accuracy. In such cases, some mechanism such as value subsets or redundant attributes is needed to prevent excessive fragmentation.

The three criteria discussed here are all information-based, but there is no reason to suspect that this is the only possible basis for such criteria. Recall that the modifications to deal with noise barred an attribute from being used in the decision tree unless it could be shown to be relevant to the class of objects in the training set. For any attribute A, the value of the statistic presented in Section 5, together with the number v of possible values of A, determines the confidence with which we can reject the null hypothesis that an object's value of A is irrelevant to its class. Hart (1985) has proposed that this same test could function directly as a selection criterion: simply pick the attribute for which this confidence level is highest. This measure takes explicit account of the number of values of an attribute and so may not exhibit bias. Hart notes, however, that the chi-square test is valid only when the expected values of p'_i and n'_i are uniformly larger than four. This condition could be violated by a set C of objects either when C is small or when few objects in C have a particular value of some attribute, and it is not clear how such sets would be handled. No empirical results with this approach are yet available.

8. Conclusion

The aim of this paper has been to demonstrate that the technology for building decision trees from examples is fairly robust. Current commercial systems are powerful tools that have achieved noteworthy successes. The groundwork has been done for advances that will permit such tools to deal even with noisy, incomplete data typical of advanced real-world applications. Work is continuing at several centers to improve the performance of the underlying algorithms.

Two examples of contemporary research give some pointers to the directions in which the field is moving. While decision trees generated by the above systems are fast to execute and can be very accurate, they leave much to be desired as representations of knowledge. Experts who are shown such trees for classification tasks in their own domain often can identify little familiar material. It is this lack of familiarity (and perhaps an underlying lack of modularity) that is the chief obstacle to the use of induction for building large expert systems. Recent work by Shapiro (1983) offers a possible solution to this problem. In his approach, called *Structured Induction*, a rule-formation task is tackled in the same style as structured programming. The task is solved in terms of a collection of notional super-attributes, after which the subtasks

of inducing classification rules to find the values of the super-attributes are approached in the same top-down fashion. In one classification problem studied, this method reduced a totally opaque, large decision tree to a hierarchy of nine small decision trees, each of which 'made sense' to an expert.

ID3 allows only two classes for any induction task, although this restriction has been removed in most later systems. Consider, however, the task of developing a rule from a given set of examples for classifying an animal as a monkey, giraffe, elephant, horse, etc. A single decision tree could be produced in which these various classes appeared as leaves. An alternative approach taken by systems such as INDUCE (Michalski, 1980) would produce a collection of classification rules, one to discriminate monkeys from non-monkeys, another to discriminate giraffes from non-giraffes, and so on. Which approach is better? In a private communication, Marcel Shoppers has set out an argument showing that the latter can be expected to give more accurate classification of objects that were not in the training set. The multi-tree approach has some associated problems – the separate decision trees may classify an animal as both a monkey and a giraffe, or fail to classify it as anything, for example – but if these can be sorted out, this approach may lead to techniques for building more reliable decision trees.

Acknowledgements

It is a pleasure to acknowledge the stimulus and suggestions provided over many years by Donald Michie, who continues to play a central role in the development of this methodology. Ivan Bratko, Igor Kononenko, Igor Mosetic and other members of Bratko's group have also been responsible for many insights and constructive criticisms. I have benefited from numerous discussions with Ryszard Michalski, Alen Shapiro, Jason Catlett and other colleagues. I am particularly grateful to Pat Langley for his careful reading of the paper in draft form and for the many improvements he recommended.

References

Buchanan, B.G., & Mitchell, T.M. (1978). Model-directed learning of production rules. In D.A. Waterman, F. Hayes-Roth (Eds.), *Pattern directed inference systems*. Academic Press.

Carbonell, J.G., Michalski, R.S., & Mitchell, T.M. (1983). An overview of machine learning, In R.S. Michalski, J.G. Carbonell and T.M. Mitchell, (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga Publishing Company.

Catlett, J. (1985). *Induction using the shafer representation* (Technical report). Basser Department of Computer Science, University of Sydney, Australia.

Dechter, R., & Michie, D. (1985). *Structured induction of plans and programs* (Technical report). IBM Scientific Center, Los Angeles, CA.

Feigenbaum, E.A., & Simon, H.A. (1963). Performance of a reading task by an elementary perceiving and memorizing program, *Behavioral Science, 8.*

Feigenbaum, E.A. (1981). Expert systems in the 1980s. In A. Bond (Ed.), *State of the art report on machine intelligence.* Maidenhead: Pergamon-Infotech.

Garvey, T.D., Lowrance, J.D., & Fischler, M.A. (1981). An inference technique for integrating knowledge from disparate sources. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* Vancouver, B.C., Canada: Morgan Kaufmann.

Hart, A.E. (1985). Experience in the use of an inductive system in knowledge engineering. In M.A. Bramer (Ed.), *Research and development in expert systems.* Cambridge University Press.

Hogg, R.V., & Craig, A.T. (1970). *Introduction to mathematical statistics.* London: Collier-Macmillan.

Hunt, E.B. (1962). *Concept learning: An information processing problem.* New York: Wiley.

Hunt, E.B., Marin, J., & Stone, P.J. (1966). *Experiments in induction.* New York: Academic Press.

Kononenko, I., Bratko, I., & Roskar, E. (1984). *Experiments in automatic learning of medical diagnostic rules* (Technical report). Jozef Stefan Institute, Ljubljana, Yugoslavia.

Langley, P., Bradshaw, G.L., & Simon, H.A. (1983). Rediscovering chemistry with the BACON system. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto: Tioga Publishing Company.

Michalski, R.S (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence 2.*

Michalski, R.S., & Stepp, R.E. (1983). Learning from observation: conceptual clustering. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto: Tioga Publishing Company.

Michie, D. (1982). Experiments on the mechanisation of game-learning 2 – Rule-based learning and the human window. *Computer Journal 25.*

Michie, D. (1983). Inductive rule generation in the context of the Fifth Generation. *Proceedings of the Second International Machine Learning Workshop.* University of Illinois at Urbana-Champaign.

Michie, D. (1985). Current developments in Artificial Intelligence and Expert Systems. In *International Handbook of Information Technology and Automated Office Systems.* Elsevier.

Nilsson, N.J. (1965). *Learning machines,* New York: McGraw-Hill.

O'Keefe, R.A. (1983). Concept formation from very large training sets. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* Karlsruhe, West Germany: Morgan Kaufmann.

Patterson, A., & Niblett, T. (1983). *ACLS user manual.* Glasgow: Intelligent Terminals Ltd.

Pearl, J. (1978a). Entropy, information and rational decisions (Technical report). Cognitive Systems Laboratory, University of California, Los Angeles.

Pearl, J. (1978b). On the connection between the complexity and credibility of inferred models. *International Journal of General Systems, 4.*

Quinlan, J.R. (1969). A task-independent experience gathering scheme for a problem solver. *Proceedings of the First International Joint Conference on Artificial Intelligence.* Washington, D.C.: Morgan Kaufmann.

Quinlan, J.R. (1979). Discovering rules by induction from large collections of examples. In D. Michie (Ed.), *Expert systems in the micro electronic age.* Edinburgh University Press.

Quinlan, J.R. (1982). Semi-autonomous acquisition of pattern-based knowledge. In J.E. Hayes, D. Michie & Y-H. Pao (Eds.), *Machine intelligence 10.* Chichester: Ellis Horwood.

Quinlan, J.R. (1983a). Learning efficient classification procedures and their application to chess endgames. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell, (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto: Tioga Publishing Company.

Quinlan, J.R. (1983b). Learning from noisy data, *Proceedings of the Second International Machine Learning Workshop.* University of Illinois at Urbana-Champaign.

Quinlan, J.R. (1985a). The effect of noise on concept learning. In R.S. Michalski, J.G. Carbonell & T.M.

Mitchell (Eds.), *Machine learning*. Los Altos: Morgan Kaufmann (in press).

Quinlan, J.R. (1985b). Decision trees and multi-valued attributes. In J.E. Hayes & D. Michie (Eds.), *Machine intelligence 11*. Oxford University Press (in press).

Sammut, C.A. (1985). Concept development for expert system knowledge bases. *Australian Computer Journal 17*.

Samuel, A. (1967). Some studies in machine learning using the game of checkers II: Recent progress. *IBM J. Research and Development 11*.

Shapiro, A. (1983). *The role of structured induction in expert systems*. Ph.D. Thesis, University of Edinburgh.

Shepherd, B.A. (1983). An appraisal of a decision-tree approach to image classification. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. Karlsruhe, West Germany: Morgan Kaufmann.

Winston, P.H. (1975). Learning structural descriptions from examples. In P.H. Winston (Ed.), *The psychology of computer vision*. McGraw-Hill.