Edge Inference for Image Interpolation

Neil Toronto
Department of Computer Science
Brigham Young University
Provo, UT 84602
email: ntoronto@cs.byu.edu

Dan Ventura
Department of Computer Science
Brigham Young University
Provo, UT 84602
email: ventura@cs.byu.edu

Bryan S. Morse
Department of Computer Science
Brigham Young University
Provo, UT 84602
email: morse@cs.byu.edu

***Abstract*— Image interpolation algorithms try to fit a function to a matrix of samples in a "natural-looking" way. This paper presents edge inference, an algorithm that does this by mixing neural network regression with standard image interpolation techniques. Results on gray level images are presented. Extension into RGB color space and additional applications of the algorithm are discussed.**

I. Introduction

The goal of image interpolation is to infer a continuous function $f(x, y)$ from a given $m \times n$ matrix of quantized samples [1]. Though the density and equal spacing of the samples simplifies the mechanics of this process, the human eye is picky—which gives rise to the quest to find techniques that yield ever-more "natural-looking" fits. In machine learning terms, the objective is to find an algorithm with a bias that approximates that of human image interpretation.

This paper presents *edge inference*, an algorithm that uses many simple neural networks to infer edges from blocks of neighboring samples and combines their outputs using bicubic interpolation. The result is a natural-looking fit that achieves much sharper output than standard interpolation algorithms but with much less blockiness.

Edge inference is similar to edge-directed interpolation [2], [3], [4], but with a crucial difference. Edge-directed methods regard an edge as a discontinuity between two areas of different value, and use thresholds to determine which discontinuities are significant. They then use the edges to guide a more standard interpolation algorithm. Edge inference regards an edge as a *gradient* between two areas of different value and uses the gradient as a model of the underlying image, avoiding thresholding altogether.

Edge inference may also be regarded as a reconstruction technique. It fits geometric primitives to samples and combines them to produce the final output. Data-directed triangulation (DDT) [5] is similar, with triangles as its geometric primitives. DDT is computationally demanding, and while edge inference produces output that is qualitatively similar to DDT's, it produces it much more quickly.

Edge-directed methods provide sharpness control in a post-processing stage, and DDT currently provides none. With edge inference, users have control over a sharpness factor: a sliding scale between the output of bicubic interpolation (which is "fuzzy") and edge inference of any sharpness.

Please note that all matrices are assumed column-major. This is for notational convenience only, as the algorithm works just as well with row-major matrices.

II. The Edge Inference Algorithm

In short, edge inference performs regression using multiple neural network basis functions, and combines their outputs using a piecewise bicubic interpolant.

The image samples are given in an $m \times n$ matrix M of gray-level pixel values, normalized to the interval $[-1, 1]$. Each sample has a location (x, y) and a value M_{xy}.

A. Neural Network Basis Functions

An $m \times n$ matrix F contains the basis functions, for a one-to-one correspondence with the samples. (This is not strictly necessary, but has given the best results so far.) It may be helpful to think of the neural networks as being placed on the image itself.

Figure 1 shows the simple two-layer network that this algorithm uses. Each trains on the sample it is associated with and its eight nearest neighbors (or fewer, if the sample lies on an image boundary). The instances in the training set are in the form

$$(x, y) \to M_{x_s y_s}$$

where (x_s, y_s) is the location of the sample, and (x, y) is the location of the sample relative to the neural network. That is, if (u, v) is the location of the network, $x = x_s - u$, and $y = y_s - v$. Each neural network represents a function in this form:

$$F_{uv}(x, y) = w_4 * \tanh(w_1 x + w_2 y + w_3) + w_5 \qquad (1)$$

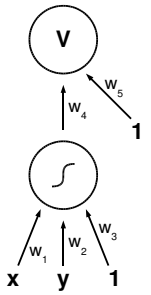
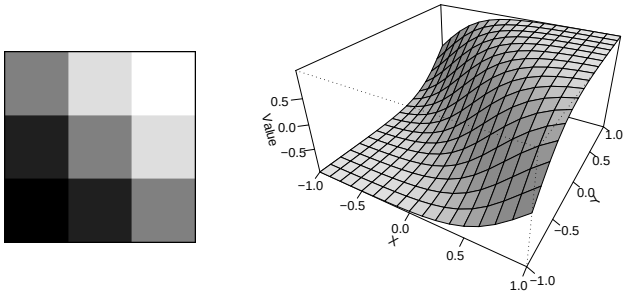


Fig. 1. The simple two-layer network

Figure 2 shows the graphical interpretation of fitting one of these simple neural networks to a 3×3 block of samples.



(a) A 3×3 block of samples

(b) $1 * \tanh(2x + 2y + 0) + 0$

Fig. 2. Fitting to a 3×3 block of samples

Unlike with most neural networks, the weights can be interpreted to have specific, geometric meanings. The equation

$$w_1 x + w_2 y + w_3 = 0$$

gives the orientation of the inferred edge as a line in implicit form. The gradient of F_{uv} is

$$\nabla F_{uv} = \begin{bmatrix} \partial F_{uv}/\partial x \\ \partial F_{uv}/\partial y \end{bmatrix} = \begin{bmatrix} w_1(1 - \tanh^2(w_1 x + w_2 y + w_3)) \\ w_2(1 - \tanh^2(w_1 x + w_2 y + w_3)) \end{bmatrix}$$

Because the steepest slope of $\tanh(x)$ is at $x = 0$, ∇F_{uv} is at its greatest magnitude when $w_1 x + w_2 y + w_3 = 0$:

$$\nabla F^*_{uv} = \begin{bmatrix} w_1(1 - \tanh^2(0)) \\ w_2(1 - \tanh^2(0)) \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

Therefore, the steepest slope of Equation 1 is given by

$$|\nabla F^*_{uv}| = \sqrt{w_1^2 + w_2^2}$$

which can be interpreted as the sharpness of the inferred edge. The values $-w_4 + w_5$ and $w_4 + w_5$ approximate the gray-level values on each side of the edge, and w_5 is the gray-level value along the line defining the edge.

Speed is critical in most image processing applications. Though these neural networks are small, special care must be taken in setting the training parameters and setting stopping criteria. The appendix describes our current implementation, and the techniques and parameters we used to reduce training time.

B. Bicubic "Distance Weighting"

Edge inference uses an inexact cubic B-spline interpolant to combine the outputs of the neural networks. Other cubic interpolants exist and may be desirable for some images [1], [6], but in our experiments, B-splines tended to produce the best results in photographs and cartoon images. For the remainder of this paper, assume that all cubics mentioned are cubic B-splines.

This section describes only what is necessary to implement bicubic interpolation. For a fuller treatment, see [1].

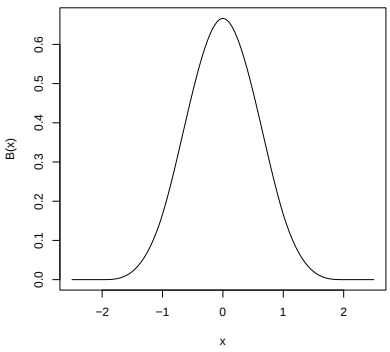


Fig. 3. B-spline kernel function

Figure 3 shows a plot of the cubic B-spline's kernel function:

$$B(x) = \frac{1}{6} \begin{cases} 3|x|^3 - 6|x|^2 + 4 & : \quad 0 \le |x| < 1 \\ -|x|^3 + 6|x|^2 - 12|x| + 8 & : \quad 1 \le |x| < 2 \\ 0 & : \quad 2 \le |x| \end{cases} \quad [1]$$

which can be used much like a distance metric. If (x, y) is the point in question, and

$$\begin{aligned} x_0 &= \max(\lfloor x \rfloor - 1, 1) \\ x_1 &= \min(\lfloor x \rfloor + 2, m) \\ y_0 &= \max(\lfloor y \rfloor - 1, 1) \\ y_1 &= \min(\lfloor y \rfloor + 2, n) \\ c &= (x_1 - x_0 + 1)(y_1 - y_0 + 1) \end{aligned}$$

(where (x_0, y_0) is one corner of the surrounding samples, (x_1, y_1) is the opposite corner, and c is the number of surrounding samples), then the interpolated value is given by

$$b(x, y) = \frac{16}{c} \sum_{u=x_0}^{x_1} \sum_{v=y_0}^{y_1} M_{uv} B(x - u) B(y - v) \qquad (2)$$

where b is a C^2-continuous function over the image domain. The term $\frac{16}{c}$ scales the result according to the number of surrounding samples, to avoid vignette effects (fuzzy, dark borders) on the image boundaries. Each of the 16 (or fewer) nearest neighbors makes a weighted contribution to the interpolated value.

What if, instead of contributing a constant value, the nearest neighbors contributed an estimate of the value at (x, y)? Under the right circumstances, this should result in more detail. Fortunately, the matrix of neural networks, F, provides a way to make these estimates.

Given the same definitions for x_0, x_1, y_0, y_1, and c above, edge inference's interpolated values are given by

$$v(x, y) = \frac{16}{c} \sum_{u=x_0}^{x_1} \sum_{v=y_0}^{y_1} F_{uv}(x - u, y - v) B(x - u) B(y - v) \qquad (3)$$

where v is also a C^2-continuous function over the image domain. In a sense, edge inference interpolates over *super*-pixels, which have not just a single value, but two values and an oriented edge between them.

(a) Nearest-neighbor (to emphasize samples) (b) Edge inference, $s = 1$ (c) Edge inference, $s = 2$

(d) Bicubic interpolation (e) Full edge inference (with noise), $s = 1$ (f) Full edge inference (with noise), $s = 2$

Fig. 4. 128×128 crop of "Man" from the USC-SIPI Image Database, scaled to 1024×1024 (8×8)

With this intuition, it is easy to see why edge inference is capable of sharper edges than bicubic interpolation. Suppose a small local area of samples has a sharp edge running through it. Multiple neural networks on either side of the edge are likely to fit to that edge, especially if it is the strongest feature in their training sets. Thus, at interpolated points near the edge, nearest neighbors from *both* sides of the edge contribute the correct value. With standard bicubic interpolation, this is not possible.

Figure 4 demonstrates that this is often the case. It shows the output of edge inference applied to a 128×128 image of an unfortunate actor from the USC-SIPI Image Database [7], and used to magnify the image to 1024×1024. Figure 4(a) shows the original image magnified using nearest-neighbor interpolation to emphasize the original samples. Figure 4(d) shows the output of Equation 2, and Figure 4(b) shows the output of Equation 3. Notice how much sharper the edges are in Figure 4(b).

C. Post-training Sharpness Control

Modifying Equation 1 to become

$$F_{uv} = w_4 * \tanh(s(w_1 x + w_2 y + w_3)) + w_5 \qquad (4)$$

introduces a new *sharpness factor*, s. This is similar to gain, except it is held constant equal to 1 during training, and may only be changed afterward.

Now, when $w_1 x + w_2 y + w_3 = 0$, $\partial F_{uv}/\partial x = s w_1$ and $\partial F_{uv}/\partial y = s w_2$. The steepest slope is given by

$$|\nabla F^*_{uv}| = \left| \begin{bmatrix} s w_1 \\ s w_2 \end{bmatrix} \right| = \sqrt{s^2 w_1^2 + s^2 w_2^2} = s\sqrt{w_1^2 + w_2^2}$$

Thus, s is a constant multiplier to the steepest slope. Figure 4(c) shows the output of Equation 3 with $s = 2$, which is even sharper than Figure 4(b).

D. Reinterpreting Noise

A problem with Figures 4(b) and 4(c) is that parts that, in the original image, have fine detail—especially the headdress—are flat and uninteresting. The neural networks have learned the edges very well and disregarded noise. (We define "noise" somewhat circularly as every feature the neural networks fail to learn.) However, the "noise" clearly contains significant features that would be desirable to have in the interpolated image.

A simple way to keep those features is to add the noise directly to the interpolated image. Let N be an $m \times n$ matrix,

defined by

$$N_{uv} = M_{uv} - F_{uv}(0,0) \tag{5}$$

In other words, each element in the noise matrix N contains the signed difference between the sample and the corresponding neural network's estimate of the sample. Figure 5 shows the calculation of N and the result: a low-contrast image with values in the range $[-2, 2]$.

The noise image is then scaled and added directly to the interpolated image. This is easy to combine with edge inference as it is defined so far. Adding the right side of Equation 2 (with N substituted for M) to the right side of Equation 3 results in

$$f(x,y) = \frac{16}{c} \sum_{u=x_0}^{x_1} \sum_{v=y_0}^{y_1} (N_{uv} + F_{uv}(x-u, y-v))B(x-u)B(y-v) \tag{6}$$

This finally completes the algorithm. Figures 4(e) and 4(f) show the results of applying it to the 128×128 "Man" image. Both seem to be fuller and have more depth than their counterparts, Figures 4(b) and 4(c). In particular, the noisy parts of the image, such as the feathers and hair, have regained lost features.

Figure 6 gives edge inference in high-level pseudocode. Notice that the noise value is calculated *only after* the neural network's sharpness is set, because any N_{uv} depends on the value of $F_{uv}(0,0)$, which depends on the sharpness. Of course, if the sharpness is changed later on, the noise value will have to be recalculated.

One very useful property of this formulation is that it turns edge inference into a generalization of bicubic interpolation. Specifically, from Equation 4, when $s = 0$,

$$\begin{aligned} F_{uv} &= w_4 * \tanh(0(w_1 x + w_2 y + w_3)) + w_5 \\ &= w_4 * 0 + w_5 \\ &= w_5 \end{aligned}$$

Equation 5 becomes

$$N_{uv} = M_{uv} - w_5$$

and the term $(N_{uv} + F_{uv}(x-u, y-v))$ in Equation 6 becomes $M_{uv} - w_5 + w_5 = M_{uv}$, yielding

$$f_{s=0}(x,y) = \frac{16}{c} \sum_{u=x_0}^{x_1} \sum_{v=y_0}^{y_1} M_{uv} B(x-u) B(y-v)$$

which is the same as Equation 2. Therefore, when $s = 0$, edge inference with a bicubic weighting function behaves exactly as bicubic interpolation. (In fact, when $s = 0$, edge inference with any interpolant's weighting function behaves exactly as that interpolant.)

This means that users of edge inference get a *sliding scale* between bicubic interpolation and edge inference of any sharpness. A much weaker but still important result is that, as long as a human being has control over the value of s, it is impossible for edge inference to perform *subjectively* worse than bicubic interpolation.

```
function learnFunctions(M, m, n, sharpness)
F <- new mxn matrix of neural networks
N <- new mxn matrix of values
for x from 1 to m
    for y from 1 to n
        F[x,y].train(x, y, M, m, n)
        F[x,y].s = sharpness
        N[x,y] = M[x,y] - F[x,y](0,0)
return F, N

function getInterpolatedValue(F, N, m, n, x, y)
x0=max(floor(x)-1,1)
x1=min(floor(x)+2,m)
y0=max(floor(y)-1,1)
y1=min(floor(y)+2,n)
c=(x1-x0+1)*(y1-y0+1)
value=0
for u from x0 to x1
    for v from y0 to y1
        value=value+(N[u,v]+F[u,v](x-u,y-v))*
            B(x-u)*B(y-v)
return 16*value/c
```

Fig. 6. Edge inference in pseudocode

III. Edge Inference with Color Images

Edge inference will work on RGB images with very little change, by treating each color plane as a separate image. However, it can be done much more quickly and with less memory by making one simplifying assumption: that the edges in each color plane are oriented approximately the same way. Thus, the neural network matrix F is defined by

$$F_{uv}(x,y) = \begin{bmatrix} w_4 \\ w_5 \\ w_6 \end{bmatrix} \tanh(s(w_1 x + w_2 y + w_3)) + \begin{bmatrix} w_7 \\ w_8 \\ w_9 \end{bmatrix} \tag{7}$$

as shown in Figure 7. It is easy to verify that the functions b (Equation 2) and f (Equation 6) do not have to be altered to use vectors as matrix elements.

Notice that Figure 7 implies that the neural network trains in YCrCb color space [8] rather than in RGB. The neural networks consistently produce better fits in YCrCb color space than they do in RGB color space. This is likely because the luminance plane (Y) offers the neural network a very strong, single feature to train on when learning data from photographic images.

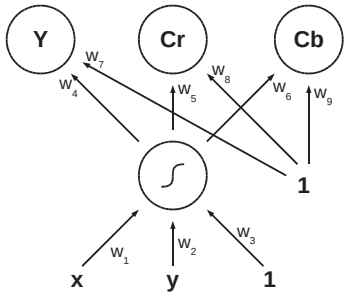


Fig. 7. The three-output network

(a) M_{uv} (b) $F_{uv}(0,0)$ (c) N_{uv}

Fig. 5. Calculation of the noise image

Figure 8 shows the results of applying edge inference to "Peppers" from the USC-SIPI Image Database [7], which has been shrunk to 128×128 and then scaled to 512×512. It also demonstrates using the sliding sharpness scale, from $s = 0$ to $s = 3$.

IV. Additional Applications

Only image superscaling has been presented here, but there are many other possible applications of edge inference. In particular, many image transformations and distortions [8] can make good use of well-interpolated sub-pixel values. Besides those, however, there are two more applications that arise from the mechanics of edge inference: noise reduction and sharpening.

A. Noise Reduction

If one makes the assumption that "noise" is every feature the neural networks fail to learn, it is possible to use edge inference to remove noise from images. The output image, I, is given by

$$I_{uv} = kN_{uv} + F_{uv}(0,0)$$

where k is a constant *noise factor* in the range [0,1]: a sliding scale between the original image and the noise-reduced image. Figures 5(a) and 5(b) demonstrate the ends of this scale, as $k = 0$ and $k = 1$, respectively.

B. Sharpening

In image processing, sharpening an image without enhancing noise is a difficult problem [8]. Using edge inference while constraining the output image to the same dimensions as the input image is one possible solution. Figure 9(b) shows the result, with $s = 4$.

Note that some of the detail is lost. This might be compensated for by inferring edges of more complex shapes than simple lines, such as quadratic curves.

(a) Original image (b) Sharpened image ($s = 4$)

Fig. 9. Sharpening with edge inference

V. Appendix: Implementation Details

Our current implementation employs a number of techniques to train the neural networks quickly and cause them to return consistent results. It is fully deterministic and, in our tests, averages about 9 seconds to train on 1024×768 RGB images on a 2 GHz Intel processor.

The details discussed in this section only apply to the YCrCb version of edge inference—the neural network depicted in Figure 7.

A. Determinism

It is desirable for edge inference to always infer the same $f(x, y)$ for each image. To achieve this, the weights are initialized to constant values. We determined experimentally that

$$w_1 = w_2 = w_7 = w_8 = w_9 = 0$$
$$w_3 = w_4 = w_5 = w_6 = 0.002$$

tends to produce good results. Also, the neural networks are trained in batch mode to avoid randomizing the order of the training set for each epoch.

(a) Nearest-neighbor (b) $s = 0$ (bicubic interpolation) (c) $s = 0.25$ (d) $s = 0.5$

(e) $s = 1.0$ (f) $s = 1.5$ (g) $s = 2.0$ (h) $s = 3.0$

Fig. 8. Scaling the 128×128 "Peppers" image to 512×512, from $s = 0$ to $s = 3$

B. Training Parameters

The momentum term is 0.9. The learning rates are per-weight, with

$$\eta_{w_1} = \eta_{w_2} = \eta_{w_3} = 0.4$$
$$\eta_{w_4} = \eta_{w_5} = \eta_{w_6} = 0.2$$

In our implementation, weights w_7, w_8 and w_9 are not trained, but are solved after every epoch (see section V-C).

We found that having good stopping criteria was the best way to speed up training over the entire image. When the majority of the neural networks in F train in fewer than 25 epochs, the algorithm runs very quickly. In our current implementation, the neural networks train for at least five epochs, and no longer than 300. They stop training when the largest weight update is smaller than 0.002.

C. Other Time-Reducing Techniques

The function *tanh* is implemented with a lookup table, which speeds up training and querying considerably.

We also derive the error function in terms of each weight, and use those partial derivatives to perform gradient descent. This uses fewer floating-point operations than backpropagation, and allows those operations to be arranged for better temporal locality. It also allows w_7, w_8 and w_9 to be solved for the minimum sum squared error directly.

Our current implementation of neural network training is written in C, and located at

`http://axon.cs.byu.edu/~neil/edge_function/`

along with a PDF file giving all the partial derivatives and solutions for w_7, w_8, and w_9.

References

[1] T. M. Lehmann, "Survey: Interpolation methods in medical image processing," *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, November 1999.

[2] J. P. Allebach and P. W. Wong, "Edge-directed interpolation," in *Proceedings of the IEEE International Conference on Image Processing*, vol. 3, 1996, pp. 707–710.

[3] X. Li and M. T. Orchard, "New edge-directed interpolation," *IEEE Transactions on Image Processing*, vol. 10, no. 10, October 2001.

[4] Q. Wang and R. Ward, "A new edge-directed image expansion scheme," in *Proceedings of the IEEE International Conference on Image Processing*, vol. 3, 2001, pp. 899–902.

[5] X. Yu, B. S. Morse, and T. W. Sederberg, "Image reconstruction using data-dependent triangulation," *IEEE Computer Graphics and Applications*, vol. 21, pp. 62–68, May 2001.

[6] D. P. Mitchell and A. N. Netravali, "Reconstruction filters in computer graphics," *Computer Graphics*, vol. 22, no. 4, August 1988.

[7] "The USC-SIPI image database." [Online]. Available: http://sipi.usc.edu/database/

[8] R. C. Gonzales and R. E. Woods, *Digital Image Processing*, 2nd ed. Pearson Education, 2002.