Neural Network Image Scaling Using Spatial Errors

Carl Staelin, Darryl Greig, Mani Fischer, Ron Maurer
HP Laboratories Israel[1]
HPL-2003-26 (R.1)
October 30th , 2003*

image error measures, image scaling, image interpolation, super-resolution, neural networks

We propose a general method for gradient-based training of neural network (NN) models to scale multi-dimensional signal data. In the case of image data, the goal is to fit models that produce images of high *perceptual* quality, as opposed to simply a high peak signal to noise ratio (PSNR). There have been a number of perceptual image error measures proposed in the literature, the majority of which consider the behavior of the error surface in some local neighborhood of each pixel. By integrating such error measures into the NN learning framework, we may fit models that minimize the perceptual error, producing results that are more visually pleasing. We introduce a spatial error measure and discuss in detail the derivative computations necessary for backpropagation. The results are compared to neural networks trained with the standard sum of squared errors (SSE) function, as well a state of the art scaling method.

* Internal Accession Date Only
Approved for External Publication
[1] HP Laboratories Israel, Technion City, Haifa 32000, Israel

Neural Network Image Scaling Using Spatial Errors

Carl Staelin, Darryl Greig, Mani Fischer, and Ron Maurer

***Abstract*— We propose a general method for gradient-based training of neural network (NN) models to scale multi-dimensional signal data. In the case of image data, the goal is to fit models that produce images of high *perceptual* quality, as opposed to simply a high peak signal to noise ratio (PSNR). There have been a number of perceptual image error measures proposed in the literature, the majority of which consider the behavior of the error surface in some local neighborhood of each pixel. By integrating such error measures into the NN learning framework, we may fit models that minimize the perceptual error, producing results that are more visually pleasing. We introduce a spatial error measure and discuss in detail the derivative computations necessary for backpropagation. The results are compared to neural networks trained with the standard sum of squared errors (SSE) function, as well as a state of the art scaling method.**

***Index Terms*— image error measures, image scaling, image interpolation, super-resolution, neural networks.**

I. Introduction

We propose a new method for fitting statistical learning models using spatial error measures, and demonstrate its use in the context of image scaling. Traditional error measures examine and penalize each output value in isolation, solely as a function of the difference between the actual output value and the target value for that input vector. However, in some applications such as image processing, the correlation or co-location of errors can have a dramatic impact on the importance and significance of the particular error. Spatial error measures were developed to take these inter-output interactions into account and to incorporate them into the penalty function used during model fitting.

The standard sum of squared errors (SSE) measure is inversely proportional to the peak signal to noise ratio (PSNR) value between the scaled image and the target image, but PSNR does not necessarily capture the perceptual quality of the scaled image (see, for example, [1], [2]). We seek other error functions that more accurately reflect the desired results. Artifacts that are visually disturbing often correspond to local correlations in the error surface. For example, a checkerboard pattern in the pixel errors along a diagonal edge corresponds to staircasing, and parallel straight lines along an edge may correspond to smoothing or over-sharpening artifacts. Appropriately defined spatial error functions can specifically target such undesirable artifacts, and produce more visually appealing images. Spatial error functions take into account not only the relationship of a single predicted value to the target value, but also the relationship of that prediction to the other predicted values in a geographical neighborhood (for images, this corresponds to pixels that are in close proximity to the target pixel in the image array).

The image scaling problem is concerned with magnifying an image while minimizing the loss of perceptual quality. Traditionally this problem has been approached using weighted sums of neighboring pixels to estimate the value of the interpolated pixel. For example, bilinear interpolation and bicubic interpolation. However these methods tend to produce undesirable blurring in the interpolated image [3]. More recently, new techniques have been introduced that use a learning procedure, such as Resolution Synthesis [4] and Example-based Super Resolution [5]. These methods reduce edge blurring, but have other problems. Resolution Synthesis can cause artifacts, mainly in textured areas, and Example-based Super Resolution produces visually pleasing results but has a high cost in memory usage.

The general architecture of our system is shown in Figure 1. During normal operation the system processes the low resolution image a pixel at a time. A neightborhood of the current pixel is transformed into a fixed length vector which is input to the neural network. The network outputs a fixed length vector of reals which are transformed into the high resolution pixels in the output image corresponding to the single input pixel. During training a high resolution image is first downscaled to create an input image with a true, or target, high resolution image. After each pixel is processed an error is computed between the output image and the true image, and the derivative of this error is backpropagated through the network to iteratively improve and refine the network approximation of the interpolation function. Note that one may use any of a number of downsampling methods, such as pixel averaging, decimation, median filtering, smoothing & decimation, and so forth. For our experiments we used pixel averaging.

More formally, suppose we are presented with a low resolution image L derived from a corresponding high resolution image H, and by means of a scaling function $f(L, \alpha)$ with parameters α, we obtain an upscaled image $I = f(L, \alpha)$. The *distance* between the upscaled image and the true high resolution image may be measured by some metric $E(I, H)$, which we shall call the error function. Then the problem of finding an optimal set of parameters α fits naturally into the general framework of optimization. Feed-forward neural networks are an appropriate candidate for the model f since they provide highly flexible models which can be fitted using general optimization algorithms. Particular examples, which will be considered in more detail below, can be found in [6], [7], [8], and [9].

Our neural network processes the low resolution image a pixel at a time. At each pixel a neighborhood of pixels surrounding the current pixel of interest is encoded to form the input vector. The network has an output for each pixel in the high resolution image encompassed by the low resolution pixel area. The network outputs are coded in a fashion similar to the input vectors, so the system decodes the outputs and places them in the high resolution image in the position cor-

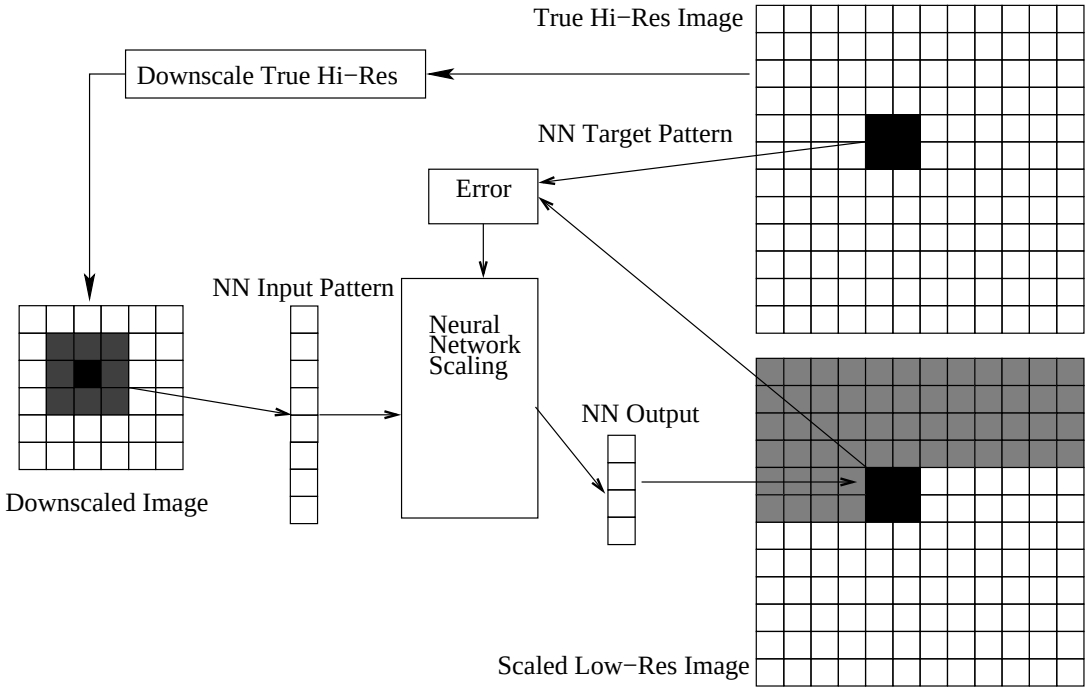


Fig. 1. NN training for image scaling.

responding to the low resolution pixel. Initially we used SSE during training, but we discovered that we often had artifacts along diagonal edges and diagonal thin lines. In consequence, we started investigating the possibility of punishing these artifacts, and discovered that using an error measure that took into account errors from neighboring pixels could reduce the frequency of these artifacts without causing other artifacts.

In this article we focus on neural network models and discuss spatial error functions in that context, detailing necessary changes to the backpropagation fitting algorithm. A simple spatial error function based on a 3x3 neighborhood is introduced, and results are compared to networks fitted using the traditional SSE error.

II. Prior Work

There has been a great deal of work done over the years on image scaling, with some of the more recent works approaching the problem using machine learning techniques. Plaziac [8] implemented a straightforward neural network model for image interpolation based on treating the low resolution image as a decimated high resolution image, and filling in the gaps using a neural network prediction. The networks were fitted using SSE, and the network input was a 24 pixel low resolution context. The networks used 16 hidden nodes and interpolated 5 high resolution values at a time. The image expansion (2x upscaling) model is based on an interpolation framework - that is, the low-resolution image is assumed to be a point sampling of the true image signal. Thus the training is done by decimating a high-resolution image, and fitting a network that supplies the missing high-resolution values based on the decimated image. The author reported a significant improvement in PSNR over bilinear interpolation, and a median based interpolation scheme.

Gustafson and Meyer [7] proposed using adaptive splines in a neural network architecture for image interpolation, and reported good results, although the article does not supply much in terms of implementation details. The results are compared only to a standard cubic spline interpolation, and comparisons are given as reductions in mean squared error.

Lazzaro and Wawrzynek [10] used a neural network for JPEG quality transcoding. This problem amounts to converting a low quality JPEG image to a larger JPEG image with reduced artifacts, which is closely related to the general image scaling problem. The networks are trained using a perceptual image metric and the network architecture is quite specific to JPEG information loss. The perceptual error measure does have a spatial component, although it includes only information from the neighboring *true* image values, rather than the fitted values.

Some interpolating schemes, such as Resolution Synthesis [4], propose a pre-processing step to segment the image into patches of some (hopefully small) number of classes. The patches in each class can then be interpolated using a standard interpolation strategy (bilinear, bicubic, etc) that is optimized for that class. Training images are used to design a set of filters which can be used to classify the image patches. For classes containing edges, interpolation is generally done using a linear filter with parameters optimized to the training set examples. Classes without edges are interpolated using a simpler method such as bilinear interpolation. The user has control over the filter size and the number of classes allowed during training.

Another segmentation approach based on neural networks is given in Admed [6]. This method uses two neural network models in sequence. The first performs principle components

analysis (PCA) on a large input vector of "candidate features" for a single pixel, such as gray level distribution, texture measures, etc. The output of this network is a non-linear projection of the original feature set onto a lower dimensional space, where the projection is supposed to capture maximal information from the higher-dimensional feature set. The smaller feature set is then passed to a second neural network which is trained as a self-organizing map. This network clusters the input observations into a set of classes that share common characteristics. Once this clustering has been obtained, the different pixel classes may be interpolated either using custom built filters, such as in Resolution Synthesis, or alternatively using existing interpolation techniques such as bilinear or bicubic.

Freeman [5] used an example based learning algorithm for image interpolation. The learning set contains 5×5 or 7×7 tiles in the low resolution image and their corresponding high resolution tile. The input image is tiled, and each tile is compared to a data-base to find the best match. Once the match is found the high resolution tile is plugged into the generated high resolution image. In order not to have boundary artifacts, the tiles are overlapped, and the tile which best matches the previous one is chosen from all the candidates. As stated previously, this method gives very good results both in textured regions, and in terms of edge sharpness. However the implementation requires a large data base of image patches, the size of which generally dictates the resultant quality. The memory complexity (and possibly the computational complexity of the data base search) required to achieve satisfactory results can be prohibitive.

The search for a "true" perceptual image measure continues to generate much activity in the image/signal processing literature, and also in psychology literature (for example see [1], [11] and references therein). In our experience, the measures proposed are very complicated, and therefore not suitable as error measures in a learning framework. In particular, they are often not differentiable and so cannot be used in gradient based optimization schemes unless a suitable approximation to the derivative can be found. An exception to this is the Structural Similarity measure [2] which is differentiable.

III. NN Training Details

The neural network models are trained using a quasi-Newton optimization algorithm (BFGS), and the network error & error derivatives are computed using backpropagation. We describe here additional architecture details, and the content of the training set.

A. Network Architecture

We use a generalized feed-forward neural network architecture with linear output nodes and tanh activation in the hidden nodes. In this architecture, each input node is connected to each hidden and output node, and we represent the connection weight between node i and node j by w_{ij}.

The hidden nodes are ordered, and each hidden node is connected to each subsequent hidden node and to each output node. The first input is a bias input with a constant value of one. Each network node has an index, with index 0 being the bias input, indices 1,...N_{in} being the input nodes, indices $(N_{in}+1)$,..., $(N_{in}+N_{hid})$ being the hidden nodes, and $(N_{in}+N_{hid}+1)$,..., $(N_{in}+N_{hid}+N_{out})$ being the output nodes.

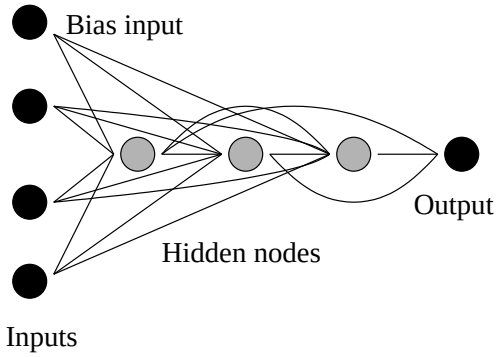


Fig. 2. Generalized feedforward architecture

Let z_i be the output from any node i, input, hidden or output. The output function for hidden node i in terms of previous hidden nodes and inputs is shown in Equation (1). Since we use linear output nodes, the output function for output node i in terms of previous output, hidden and input nodes is shown in Equation (2).

$$z_i = \tanh\left(\sum_{j=0}^{i-1} w_{ij} z_j\right) \tag{1}$$

$$z_i = \sum_{j=0}^{i-1} w_{ij} z_j \tag{2}$$

This flexible architecture is very powerful and a network of any given size can mimic any layered architecture with an equivalent number of total hidden nodes. Figure 2 shows an example network with three inputs, three hidden nodes, and a single output node. In practice we have found that only a small number of hidden nodes are required to provide satisfactory results. In particular, we have found that reasonable results can be obtained with between 10 and 30 hidden nodes.

B. Inputs and Output Scaling

For both the training and operation of the NN models we assume that the input images are single channel (either graylevels or a single color channel). We generate a single input and target pattern for each low-resolution pixel, so the target consists of $u \times u$ target values where u is the (integer) scaling factor. The input pattern consists of a window around the pixel of interest drawn from the low-resolution image. The window size should increase as the scaling factor increases - we have generally found that a window size of 5x5 is suitable for factor 2x upscaling, and 7x7 for factor 4x upscaling.

We use three data encoding schemes: direct, relative, and scaled, but in general, scaling works best. Direct encoding rescales input and output values to the range $[0...1]$ by dividing each input and output by 255. Network outputs are multiplied

by 255 and clipped to the range $[0, ..., 255]$ to get image pixel values. More formally, the translation from image pixel value to network value is specified as:

$$v_{simple}(p) = \frac{p}{255}$$

where $v_{simple}(p)$ is the network input or target value and p is the image pixel value. The translation from network value (output) to pixel value is:

$$p_{simple}(v) = \begin{cases} 0 & v < 0 \\ 255 & v > 1 \\ 255 \cdot v & o.w. \end{cases}$$

Since the network input and output values fall in the range $[0, \ldots, 1]$, we may naturally use logistic or linear output nodes.

Relative encoding builds on direct coding by first dividing inputs and outputs by 255 and then subtracting the value of the central pixel of the input window from each of the inputs and outputs, and finally removing the central pixel from the input set. The outputs are also relative to the central input pixel value, so the image pixel value is obtained by adding the central pixel value to the network output, and then multiplying the result by 255 and clipping to the range $[0, ..., 255]$. This encoding reduces the dimensionality of the input / target space by making edges of similar height look similar to the network regardless of the base value. If we define $dc \equiv v_{simple}(center)$, then we can define the translations as:

$$v_{relative}(p) = v_{simple}(p) - dc$$

$$p_{relative}(v) = p_{simple}(v + dc)$$

Since the network input, output, and target values all fall in the range $[-1, \ldots, 1]$, we may naturally use tanh or linear output nodes.

Scaled encoding builds on relative encoding by dividing the relative inputs and targets by $contrast$, which is a measure of the contrast variation in the input window. We can define $contrast$ using the following equations. Values is the set of $v_{relative}$ values for the pixels in the input window:

$$values \equiv \{v_{relative}(p) \mid p \in \{input\, window\}\}$$

Range is the difference between the maximal and minimal value in $values$:

$$range \equiv max(values) - min(values)$$

and contrast is defined as:

$$contrast \equiv max\,(thresh, range) \tag{3}$$

where threshold is used to prevent noise in flat regions from being stretched to look like edges, and is set to a value roughly equivalent to 30 grey levels. This has the effect of stretching the windows so that the underlying image structure is supplied and predicted by the network. For example, an edge is treated as an edge no matter what the contrast between the sides is, nor the overall gray-level at which it occurs. Of course, in very flat regions the contrast is small, so we impose a lower limit on the contrast value to prevent unreasonably large input and target values in these regions. We can then define the scaled transforms as:

$$v_{scaled}(p) = \frac{v_{relative}(p)}{contrast} \tag{4}$$

$$p_{scaled}(v) = p_{relative}\,(contrast \cdot v)$$

Since the output values are not bounded by $[-1, \ldots, 1]$, we use only linear output nodes to avoid clipping the network outputs and limiting the response of the system.

C. Training Images

During training the system takes each sample image and down-samples it using pixel averaging to create a low resolution input corresponding to the original (i.e. true) high resolution image. Then it uses the neural network to generate an interpolated high-resolution image. During the interpolation process the training algorithm can compute the error between each newly generated high resolution pixel and the corresponding pixels in the original image. This error is used to compute the overall error function for the network, and its derivative is back-propagated through the network to compute the error gradient for the connection weights.

The models are trained on the image set used by Atkins [4]. This set consists of 25 images of a variety of natural images (people, landscapes, man-made objects, still life, etc). The models are tested on a set of images that do not occur in the training set.

In addition to learning how to interpolate images, it may be desirable to do simple image processing at the same time, such as selective sharpening, smoothing, or darkening. This can be accomplished by training the neural network using target images that incorporate the desired imaging effects. The Resolution Synthesis method [4] takes a similar approach when training its interpolation engine.

IV. Spatial Error Measures

Suppose that the total error E between two images Z and T is given by

$$E(Z,T) = \sum_{i,j} \zeta(\nu(z_{ij}), \nu(t_{ij}))$$

where ν denotes the vector of pixels in some pre-specified neighborhood around its argument, z_{ij}, t_{ij} are the ij^{th} pixels of image Z and image T respectively, and ζ is a differentiable function. For non-spatial measure (such as SSE), the vector $\nu(z_{ij})$ contains only one value, namely z_{ij}. In that case, the derivative of the error with respect to z_{ij} is simply:

$$\frac{\partial E(Z,T)}{\partial z_{ij}} = \frac{\partial \zeta(\nu(z_{ij}), \nu(t_{ij}))}{\partial z_{ij}}.$$

However in the more general case of spatial errors, we must consider every pixel neighborhood $\nu(z_{kl})$ such that $z_{ij} \in \nu(z_{kl})$. That is,

$$\frac{\partial E(Z,T)}{\partial z_{ij}} = \sum_{kl:\, z_{ij} \in \nu(z_{kl})} \left(\frac{\partial \zeta(\nu(z_{kl}), \nu(t_{kl}))}{\partial z_{ij}} \right).$$

The computations of the error measure and its derivative are not complicated, however the form of the derivative does introduce some issues when this measure is to be used in a backpropagation algorithm.

To begin with, let us consider the steps involved in the standard backpropagation framework:

1) For some input pattern $\mathbf{x}$ and corresponding target t_{ij}, forward-propagate the input through the network to compute the internal state of the network and the network prediction z_{ij} on input $\mathbf{x}$.
2) Compute the contribution to the error of this prediction, $\zeta(z_{ij}, t_{ij})$, and the derivative of the total error with respect to this network output, $\frac{\partial E(Z,T)}{\partial z_{ij}} = \frac{\partial \zeta(z_{ij},t_{ij})}{\partial z_{ij}}$.
3) Backpropagate the derivative $\frac{\partial E(Z,T)}{\partial z_{ij}}$ through the network to compute the contribution this network output makes to the total derivative of the error with respect to the network weight vector.

The backpropagation step 3 requires that the internal state of the network (specifically, the output of each hidden node in the network) is appropriately set by the forward-propagation step 1.

For spatial measures the contribution to the error of network output z_{ij} depends on the network outputs $z_{kl} \in \nu(z_{ij})$ and the derivative of this output error contributes to the derivative of the total error with respect to each $z_{kl} \in \nu(z_{ij})$. This makes it difficult to implement step 3, since the partial derivative $\frac{\partial E(Z,T)}{\partial z_{ij}}$ cannot be known until all network outputs contributing to this derivative have been computed. At that point we need to back-propagate the partial derivative through the network with network state set by the input pattern on which output z_{ij} was computed, which is not normally the current state of the network. The following steps describe a modification of standard backpropagation that is suited to the special requirements of spatial errors.

1) For some input pattern $\mathbf{x}$ and corresponding target t_{ij}, forward-propagate the input through the network to compute the internal state of the network and the network prediction z_{ij} on input $\mathbf{x}$. The network state (output values from each hidden and output node) is saved in a buffer for referral when all the outputs in the neighborhood $\nu(z_{ij})$ have been computed.
2) Once all the outputs in neighborhood $\nu(z_{ij})$ have been computed, compute the contribution to the network error $\zeta(\nu(z_{ij}), \nu(t_{ij}))$, and the derivative of this quantity with respect to each output $z_{kl} \in \nu(z_{ij})$. These derivatives are added to a second buffer containing cumulative sums for each $\frac{\partial E(Z,T)}{\partial z_{kl}}$.
3) Once the errors $\zeta(\nu(z_{kl}), \nu(t_{kl}))$ for all $z_{kl} \in \nu(z_{ij})$ have been computed, and their corresponding derivatives added to the buffer of cumulative sums, then the derivative value $\frac{\partial E(Z,T)}{\partial z_{ij}}$ is complete. The entry in the network state buffer corresponding to the input responsible for output z_{ij} is loaded back into the network, and $\frac{\partial E(Z,T)}{\partial z_{ij}}$ is backpropagated as per the usual backpropagation algorithm.

Note that the above algorithm, although computationally quite efficient, is expensive on memory due to the fact that the previous network states are stored in step 1. If this memory overhead is too high, then we may choose a more memory efficient implementation that pays a higher computational cost. The algorithm for such an implementation is identical to the above algorithm except that in step 1 the network state is *not* stored, and in step 3 the network state is recomputed by repeating the forward propagation of the relevant input pattern.

Thus the computationally efficient implementation requires a memory buffer sufficiently large to hold as many past network states as will be needed for future error computations. For a two dimensional signal (which, at least for images, is usually read in horizontal strips), an error neighborhood with height m for a network with n hidden+output nodes would require storage for $m \times w$ floating point vectors of length n, where w is the width of the signal strips. The memory efficient implementation does not have this overhead, but must forward propagate each input pattern twice, rather than just once.

A. Matrix Error Measure

The goal of spatial error measures may be grasped intuitively by considering the error image presented in Figure 3. In this figure we see (a) a 2x upscaling of a low resolution image, (b) the true high resolution image from which that low resolution image was derived, and the error image (c) obtained by taking the channel-wise differences between (a) and (b).

The "embossing" seen in the error image corresponds to the locations in the pixel replicated image where the most visually disturbing artifacts are found. This simple observation opens up the possibility of using the error image to detect and penalize artifacts in the upscaling method. In particular, any filter that detects edges and lines will, when applied to the error image, supply a value that can be used in a suitably defined penalty function. Recall that the penalty function (or error measure) must be a metric, and must be at least once-differentiable.

As a first step we consider a complete basis of 3x3 masks for edge and line detection presented in [12] and cited in [13]. The basis consists of three different sub-bases for edge, line and "average" features. The (normalized) images resulting from applying each of the sub-bases to the grayscale error images in Figure 3(c) are shown in Figure 5. The "average" image (c) corresponds to the penalties employed in conventional neural network fitting. For example, if the network is fitted with sum of squared errors, then the network error will be the sum of squares of some linear transform of the gray-levels in this image. Images (a) and (b), on the other hand, highlight the errors corresponding to the "jaggedness" in the pixel replicated image. It is just these features that we hope to introduce penalties against in the network error function.

We now proceed to define a spatial error measure using this basis. To simplify the formulae we define the pixel errors as:

$$e_{ij} \equiv z_{ij} - t_{ij} \tag{5}$$

where z_{ij} is the network output value, and t_{ij} is the target network output. Further let $\nu(e_{ij})$ be the vector formed from taking all the pixel errors in a 3x3 neighborhood of (i, j), where the pixels indices are read row by row.

Fig. 3. (a) pixel replicated low resolution image; (b) true high resolution image; (c) error image = (a)-(b).

Edge detectors:
$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Line detectors:
$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

Average:
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Fig. 4. Orthogonal masks for edge and line detection.

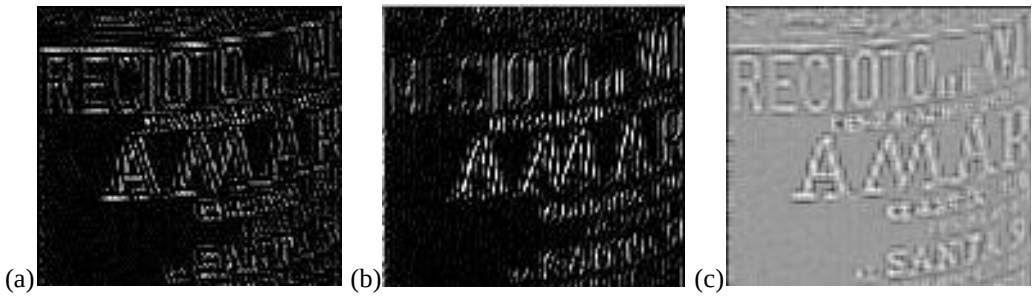


Fig. 5. Grayscale feature detection results (a) edge (b) line (c) average

If we choose a positive definite matrix A, then the error function

$$E(Z,T) = \sum_{i,j} \zeta(\nu(z_{ij}), \nu(t_{ij})) = \sum_{i,j} \nu(e_{ij})^T A \nu(e_{ij})$$

is positive for all non-zero error neighborhoods $\nu(e_{ij})$. Therefore we seek a positive definite candidate for A that penalizes the sorts of errors highlighted by the basis refered to above. From Wolfram ([14] Equation 5) we know that a real symmetric matrix A is positive definite *iff* there exists a real, non-singular matrix M such that $A = M^T M$.

An obvious choice for M is formed by the row-wise vectorization of the basis matrices from Figure 4 and is shown in Equation 6. M is real and non-singular, and detects edges and lines in the error image. However, we first want to normalize the detectors (rows) using a normalization matrix N so that $I = (N \cdot M)^T (N \cdot M)$, giving

$$N = diag\left(\frac{1}{\sqrt{diag(M \cdot M^T)}}\right) \cdot M$$

Then we want to have a weight vector $\vec{w}$ to explicity weight each detector, so that

$$A = (diag(sqrt(\vec{w})) \cdot N \cdot M)^T \cdot (diag(sqrt(\vec{w})) \cdot N \cdot M).$$

Defining $B = N \cdot M$ and $W = diag(\vec{w})$, and using a few simple transformations, we get

$$A = B^T \cdot W \cdot B$$

which is positive definite *iff* all the weights are positive and M is real and non-singular.

Note that B is a basis, so this transformation is equivalent to transforming the vector into another basis, adjusting the features according to a penalty function in the basis space, and then transforming back to the original space.

Since the feature detection basis gives a large response (in absolute value) when the specific features described above are detected, we can expect the square of this quantity to be large and positive in the presence of edges or lines in the error image, and approximate a traditional squared error metric otherwise.

The only decision that remains is the weighting scheme applied to the matrix M. It is reasonable that the rows of M

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & 1 & -1 & 0 & 1 & -1 & -1 & 0 \\ -1 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 1 \\ -1 & -1 & -1 & 2 & 2 & 2 & -1 & -1 & -1 \\ -1 & -1 & 2 & -1 & 2 & -1 & 2 & -1 & -1 \\ -1 & 2 & -1 & -1 & 2 & -1 & -1 & 2 & -1 \\ 2 & -1 & -1 & -1 & 2 & -1 & -1 & -1 & 2 \end{bmatrix} \quad (6)$$

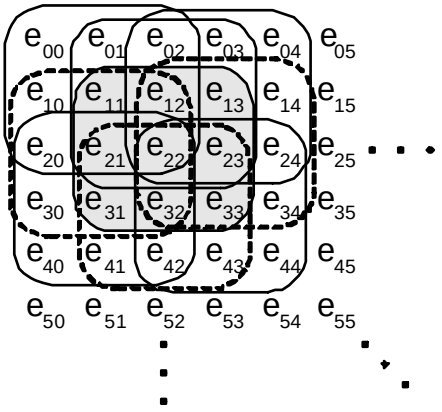


Fig. 6. Matrix Error Computation

should be normalized. We investigated a number of different weighting schemes, although in every case we applied a single weight to each of the sub-bases. This reduces the complexity of the investigation, and there does not seem to be a good reason to introduce internal weighting of the sub-bases.

1) Error Computation and Derivatives: Figure 6 shows the influence neighborhood of the pixel error e_{22}. The contribution to the total error at the point $(2, 2)$ is given by

$$E_{22} = \nu(e_{22})^T A \nu(e_{22}).$$

This requires the 9 precomputed errors in the 3x3 neighborhood $\nu(e_{22})$, and thus may only be computed when the computation of e_{33} is complete. The derivative of E_{22} with respect to the output of the network $z_{2+k,2+l}$, where $k, l \in -1, 0, 1$, is given by:

$$\frac{\partial E_{22}}{\partial z_{2+k,2+l}} = A_{4+3k+l,:}\nu(e_{22}) + (A_{:,4+3k+l})^T v(e_{22}) \quad (7)$$

where $A_{m,:}$ and $A_{:,n}$ are the row m and column n vectors of matrix A respectively. By defining $D_m = A_{m,:} + (A_{:,m})^T$, and using Equation (7) we may compute the derivative of the *total* error with respect to z_{22}:

$$\begin{aligned} \frac{\partial E}{\partial z_{22}} =& D_0\nu(e_{33}) + D_1\nu(e_{32}) + D_2\nu(e_{31}) \\ &+ D_3\nu(e_{23}) + D_4\nu(e_{22}) + D_5\nu(e_{21}) \\ &+ D_6\nu(e_{13}) + D_7\nu(e_{12}) + D_8\nu(e_{11}) \end{aligned} \quad (8)$$

We may only backpropagate this error once the computation of e_{44} (from the neighborhood $\nu(e_{33})$) is complete.

2) Connection to Mahalanobis Distance: Some insight into the behavior of this error measure may be gained by considering the well known *Mahalanobis distance* between two vectors $\vec{x}$ and $\vec{y}$ with a common covariance matrix of the feature space Σ:

$$d_M(\vec{x}, \vec{y}) = (\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y}).$$

If the features are uncorrelated, then Σ reduces to the diagonal matrix with the variance of each feature along the diagonal, and d_M simply becomes the scaled sum of squared differences of the features of $\vec{x}$ and $\vec{y}$, where the contribution of each feature is scaled by one over the variance of that feature. This has the effect of scaling the features so that differences in features with large variance are not weighted more than differences in features with small variance. In the case that correlations do exist between features, the inverse covariance term has the same result - the feature space is rescaled so that large covariances between features do not skew the distance.

The matrix error measure may be written as a Mahalanobis distance if we take $\vec{x}$ to be the error neighborhood $\nu(e_{ij})$, $\vec{y}$ to be the mean vector of the errors (which we may assume to be $\vec{0}$ if the model is unbiased), and A is an estimate of the inverse covariance matrix Σ^{-1}. Note however that the spatial error measure seeks to penalize correlations between features, whereas the Mahalanobis distance seeks to factor these correlations out of the distance measure. Therefore we choose weighting schemes that exaggerate the effect of undesirable correlations and induce the model to penalize such correlations more heavily in the training phase.

V. Similarity Measures

In order to evaluate the efficacy of the new training method using spatial error measures, we use four image similarity measures: PSNR, block-PSNR, MSSIM, and block-MSSIM. PSNR is the standard measure and is defined as:

$$PSNR(X, Y) = 20 \cdot log\left(\sqrt{\frac{255 \cdot 255}{\sum_{i=1}^{N}(X(i) - Y(i))^2}}\right)$$

Blocked PSNR [15] is the minimal PSNR taken over all fixed-size sub-blocks of the two images. In our case, we use sub-blocks where the area of the sub-block is 1% of the image. Since the human eye tends to be drawn towards artifacts, this sets the similarity as the worst PSNR for a block in the image.

MSSIM [2] was designed to more accurately reflect similarities between two images according to the human visual

system. Let $\mathbf{x}_j$ refer to the pixels in image $\mathbf{X}$ that fall within a neighborhood $\nu(j)$ centered on pixel j. For convenience, we shall refer to a generic neighborhood in the image as ν, and let $\mathbf{x}$ refer to the pixels contained in ν. Furthermore, suppose that the elements of ν are indexed $1, \ldots, N$, then we introduce a weighting scheme $\mathbf{w} = \{w_1, \ldots, w_N\}$ over ν. For the generic neighborhood ν, we shall write $\mathbf{x} = \{x_1, \ldots, x_N\}$, and for a specific neighborhood $\nu(j)$ we write $\mathbf{x}_j = \{x_{j1}, \ldots, x_{jN}\}$. The equations defining the MSSIM similarity measure from Wang et. al. are shown:

$$MSSIM(\mathbf{X}, \mathbf{Y}) = \frac{1}{M} \sum_{j=1}^{M} SSIM(\mathbf{x}_j, \mathbf{y}_j)$$

$$SSIM(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha} \cdot [c(\mathbf{x}, \mathbf{y})]^{\beta} \cdot [s(\mathbf{x}, \mathbf{y})]^{\gamma}$$

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_{\mathbf{x}}\mu_{\mathbf{y}} + C_1}{\mu_{\mathbf{x}}^2 + \mu_{\mathbf{y}}^2 + C_1}$$

$$c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + C_2}{\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{y}}^2 + C_2}$$

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{\mathbf{xy}} + C_3}{\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + C_3}$$

$$\mu_{\mathbf{x}} = \sum_{i=1}^{N} w_i x_i$$

$$\sigma_{\mathbf{x}} = \left(\sum_{i=1}^{N} w_i \left(x_i - \mu_{\mathbf{x}} \right)^2 \right)^{\frac{1}{2}}$$

$$\sigma_{\mathbf{xy}} = \sum_{i=1}^{N} w_i \left(x_i - \mu_{\mathbf{x}} \right) \left(y_i - \mu_{\mathbf{y}} \right)$$

Following Wang, we use an 11x11 window, and the w_i are set according to a circular-symmetric Gaussian function with standard deviation of 1.5 pixels normalized to unit sum $\left(\sum_{i=1}^{N} w_i = 1 \right)$.

Blocked-MSSIM is created analogously to blocked-PSNR. It is the smallest MSSIM over all sub-blocks of a given size. Again we chose blocks that are 1% of the image size.

VI. Results

We evaluated the utility of the spatial error measure both visually and statistically. Two independent visual tests were performed. In the first visual test, the subjects were shown pairs of natural images, and in the second test they were shown fragments of images containing the narrow lines and diagonal edges that often caused artifacts in SSE networks. In all cases, neural networks that failed to train properly were excluded from the analysis. These are easy to identify as all resultant images are usually either white or black.

In each visual test, one set of images and two sets of networks were used (one for each configuration). The user was shown a series of paired images and asked to select the image he preferred, with the additional option of giving a tie. The image pairs were created by randomly choosing an image from the image set, and then randomly choosing two networks, one of each configuration. The selected image was upscaled using each network, the resulting images were randomly assigned to the names *a* and *b*, and the two images were then shown to the subject. In this way the images were anonymized and the subject could not know which image was processed using which method. Each methods was assigned a value, -1 or 1, and the result of each comparison was one of -1, 0, or 1, depending on whether the subject chose the first method, tied, or chose the second method. The null hypothesis was that the sequence of values had a zero mean. The test was terminated when fifty image pairs were evaluated. Additionally, the test could be terminated early if the current results violated the null hypothesis at a 95% confidence value, or if there was no way to violate the null hypothesis at a 95% confidence value within fifty evaluations given the current results.

TABLE I
Visual Preference Test Results

subject	LiveQuality	Fragments
a	H_0	spatial
b	H_0	spatial
c	spatial	spatial
d	spatial	spatial

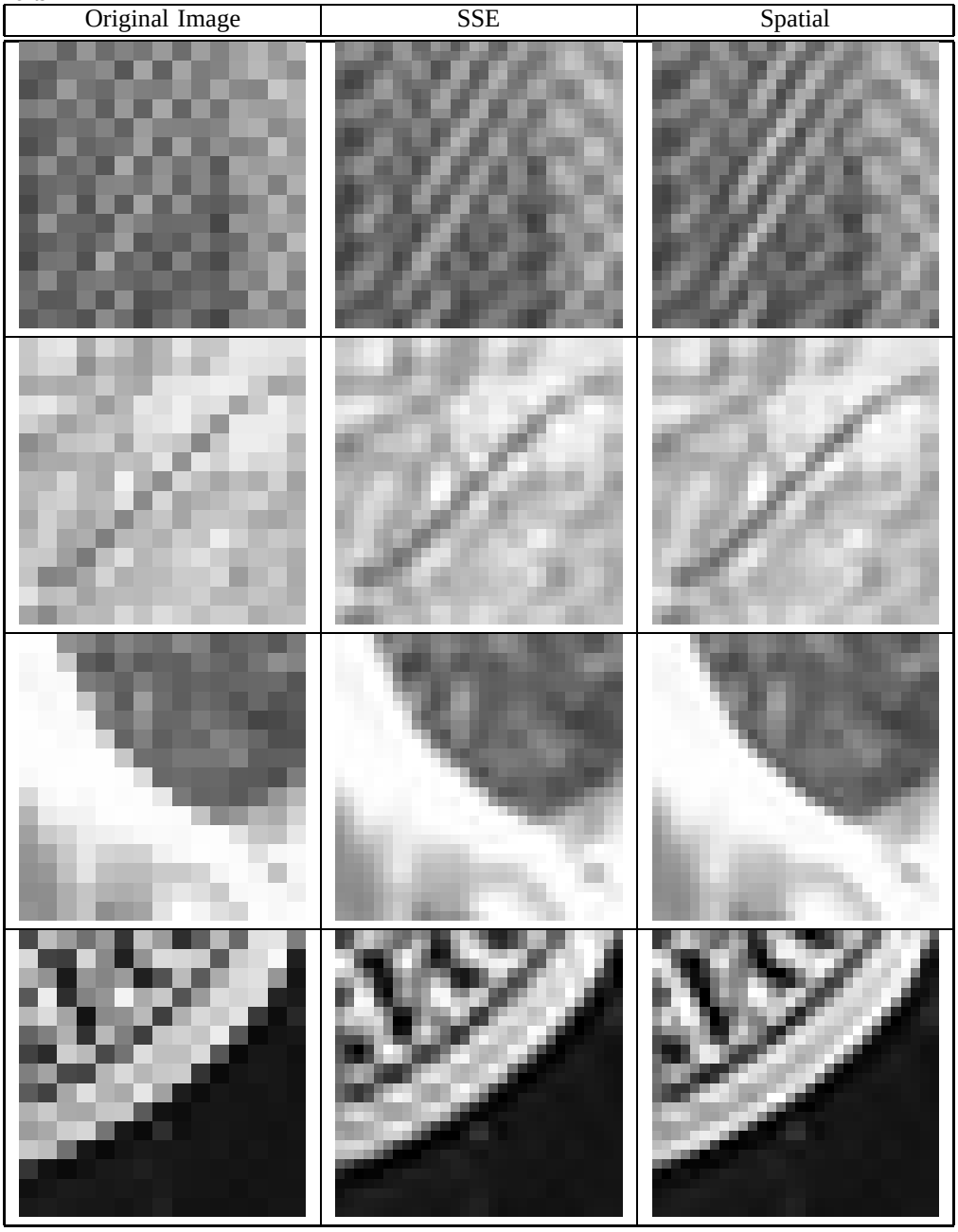
Table I shows the results for four subjects on each of two image sets. The table entries show the result of each test, and the possible values in the table are SSE, spatial, and H_0 for the null hypothesis or an inconclusive result. LiveQuality is the set of 29 natural images from the LiveQuality2002 image database [16], while Fragments is a set of four image fragments containing features such as diagonal edges and narrow lines that are known to cause problems for SSE-trained networks. From these results we can see that on whole images, spatial error measures are not worse than SSE-trained networks, and for some viewers it can improve the image. In addition, spatial error measures definitely improve the visual quality of those portions of the images where SSE-trained networks often have problems.

Some examples of portions of the image fragments illustrating the differences are shown in Figure 7. You can see that the SSE-trained networks tended to result in somewhat blockier images, while the Spatial-trained networks tended to recognize lines and edges more readily. The first image fragment is a portion of the striped headscarf from Barbara. The second fragment has a tree branch running from the bottom left to top right corner. The third fragment shows a slightly curved edge, and the fourth image shows a patterned foil wrapper on the top of a wine bottle with some curved lines and edges.

We also conducted statistical tests to determine if spatial errors affected image similarity measures, such as PSNR and MSSIM.

For the purposes of comparison, we assume that a particular neural network is a single repetition of a *training configuration* representing a specific choice of training and architecture parameters. The training configuration specifies number of hidden units, input & data scaling, training error measure, and error parameter values (where required). The comparison below is the result of training a subset of configurations for a fixed number N of repetitions. The training configurations

Fig. 7. Image Fragments



were chosen to span a range of values, and give some useful comparative analysis. A more comprehensive search of the configuration space is currently underway to chose the "best" configuration for performance purposes. The comparisons are made on a series of SNR measures to yield a more broad consensus on the results.

As a final outcome of our results, we wish to produce a ranking of the configurations considered, giving some idea of which configurations may be considered statistically indistinguishable for a given SNR measure, and which differ in a statisically significant way. This ranking can be induced by selecting all pairs of configurations (c_1, c_2) , and testing the hypotheses

$$H_0 : \ SNR(c_1) = SNR(c_2) \tag{9}$$

against the alternative

$$H_A : \ SNR(c_1) > SNR(c_2),$$

where $SNR(c)$ is a measures of the SNR performance of configuration c.

Now suppose we wish to test the above hypothesis for a particular configuration pair (c_1, c_2). We have N_{c_1} and N_{c_2} trained networks respectively for these configurations and wish to use an appropriate test to assess the hypothesis based on the SNR results the networks achieve on the LiveQuality database of images. A visual inspection of the SNR values suggested that the SNR distribution for the networks of a specific configuration on a specific image is near enough to normal to admit a test statistic based on the normal distribution. Since

the number of repetitions is typically quite small (in our case, $N = 16$), the appropriate test is a paired Student's *t*-test. Therefore, for each image i, we compute a p-value for the hypothesis test

$$H_0 : \ SNR(c_1, i) = SNR(c_2, i) \tag{10}$$

against the alternative

$$H_A : \ SNR(c_1, i) > SNR(c_2, i).$$

The result is a sequence of p-values $p_1, ..., p_I$ corresponding to the hypotheses that configuration c_1 is better than c_2 on image $i \in \{1, ..., I\}$. Now, supposing the original null hypothesis (9) were true, then distribution of p-values from the test (10) should be more or less uniformly distributed. That is, if the set of test images is suitably random, we would expect that, for example, 5% of the hypothesis tests to be significant at the 5% level. Therefore, we may test if the null hypothesis (9) is violated by testing if the sequence of p-values resulting from (10) is approximately uniformly distributed on $[0, 1]$. An appropriate test for this is the one-sided Kolmogorov-Smirnov goodness of fit test [17]. Using this test in a pairwise fashion we are able to induce a ranking on the configurations of interest, where configurations that are not significant may be grouped in equivalence classes.

We performed the above analysis on 16 repetitions of two neural network configurations, and evaluated each of the networks on a set of 29 test images, computing the PSNR, blocked PSNR, MSSIM and blocked MSSIM measures for each outcome against the true images. The configurations use a 5x5 input window, scaled inputs, linear output nodes, and 30 hidden nodes, and are trained with either SSE or the new spatial error measure. In the case of the spatial error measure only one parameter set was used, giving 0.1 weight to the average basis, and 0.45 weight to each of the line and edge bases. The resulting weight vector $\vec{w}$ is:

$$\vec{w} = \{0.1, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45\}$$

This configuration was selected after a more extensive survey of configuration parameter values (on a separate set of test images).

We also compared these two neural network configurations against Resolution Synthesis with 5x5 input window, 100 classes, and a factor of 12 which typically resulted in nearly all 100 classes participating in each pixel computation. We tried a few different configurations of Resolution Synthesis on an independent image set and found this configuration to generally perform the best for all error measures all except block-MSSIM, where a different configuration was sometimes preferred. Since Resolution Synthesis is a deterministic method, the two sample paired Student's *t*-test above was replaced by a one sample *t*-test that the mean of the networks was greater than the resolution synthesis result. The remainder of the analysis is the same.

The results from the analysis are given in Table II. In each column the methods are ranked from best (at the top) to worst (at the bottom). For each of the similarity measures the rankings were statistically signficant at the 0.05 level. In the PSNR column, for example, Spatial is a significant improvement over ResSynth, which is in turn a significant improvement over SSE. Clearly the spatial-error trained networks are statistically significantly better than both Resolution Synthesis and SSE trained networks on all four SNR measures. This concurs with the results of the human perceptual tests and suggests that spatial-error based training can utilize the flexibility of a given network architecture more effectively than SSE for the purposes of image scaling.

PSNR	B-PSNR	MSSIM	B-MSSIM
Spatial	Spatial	Spatial	Spatial
ResSynth	ResSynth	SSE	SSE
SSE	SSE	ResSynth	ResSynth

TABLE II
COMPARATIVE RANKINGS OF SPATIAL ERROR, SSE AND RESOLUTION SYNTHESIS ON THE FOUR SNR VALUES

VII. CONCLUSIONS

We have presented a general framework for using neural networks for image scaling using a spatial penalty function to train the network, and applied this framework in the specific case of the matrix error described above. This method allows more general, perceptual errors to be used for fitting the model, and can therefore target specific visual artifacts, such as staircasing. The framework is not restricted to neural networks and may be extended easily to any model that is fitted using gradient based training. Furthermore, it may be applied directly to multi-dimensional signals of all kinds, not just images. The results show a clear improvement over the classical sum of squared errors training, and also compare favorably with the state of the art Fast Resolution Synthesis.

There is an extra training overhead for using area based errors, in particular, each pixel must participate in the errors and derivatives for all the pixels in its influence neighborhood (3x3, for the spatial error above). This overhead is further compounded if there is not sufficient memory resources available to store the network state, as discussed in Section IV. On the other hand, a larger influence neighborhood is likely to yield better networks and perhaps quicker convergence times for the training algorithm. In our experience, the extra overhead for the spatial error is not significant compared with the memory and computation overhead of the NN training algorithm for this problem. Furthermore, this is an example of a model that is fit once, but used many times, so the fitting time is not usually crucial.

It is clear that any $k^2 \times k^2$ positive definite matrix defines a matrix error measure with $k \times k$ influence neighborhood, however the cost of searching the parameter space to find good matrix coefficients may become prohibitive. We are currently investigating methods for developing spatial error measures that do not depend on a parameter space search.

ACKNOWLEDGEMENTS

We should like to thank Brian Atkins for his interesting and stimulating discussions on image interpolation, and for his

generous sharing of his time, Resolution Synthesis software, and training images.

REFERENCES

[1] K. K. Makoto Miyahara and V. R. Algazi, "Objective picture quality scale (pqs) for image coding," *IEEE Trans. Commun.*, vol. 46, no. 9, pp. 1215–1226, 1998.

[2] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error measurement to structural similarity," *IEEE Trans. Image Processing*, (To Appear).

[3] H. S. Hou and H. C. Andrews, "Cubic splines for image interpolation and digital filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 26, no. 6, pp. 508–517, 1978.

[4] C. B. Atkins, C. A. Bouman, and J. P. Allebach, "Tree based resolution synthesis," in *Proceedings Conference on Image Processing, Image Quality, Image Capture Systems*, Savannah, Georgia, April 25–28 1999, pp. 405–410.

[5] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super resolution," in *Proceedings of Computer Graphics and Applications*. IEEE, March/April 2002, pp. 56–65.

[6] B. E. C. Mohamed N. Ahmed and S. T. Love, "Adaptive image interpolation using a multilayer neural network," in *Proceedings of the SPIE - Applications of Artificial Neural Networks in Image Processing IV*.

[7] S. C. Gustafson and G. J. Meyer, "Spline-based neural networks for digital image interpolation," in *Proceedings of the SPIE - Applications of Artificial Neural Networks in Image Processing IV*, N. M. Nasrabadi and A. K. Katsaggelos, Eds.

[8] N. Plaziac, "Image interpolation using neural networks," *IEEE Trans. Image Processing*, vol. 8, no. 11, pp. 1647–1651, 1999.

[9] T. Sigitani, Y. Linguni, and H. Maeda, "Image interpolation for progressive transmission by using radial basis function networks," *IEEE Trans. Neural Networks*, vol. 10, no. 2, pp. 381–390, 1999.

[10] J. Lazzaro and J. Wawrzynek, "Jpeg quality transcoding using neural networks trained with a perceptual error measure," *Neural Computation*, vol. 11, no. 1, pp. 267–296, 1999.

[11] P. Teo and D. Heeger, "Perceptual image distortion," in *Proceedings ICIP-94 (IEEE International Conference on Image Processing)*, vol. 2, 1994, pp. 982–986. [Online]. Available: citeseer.nj.nec.com/teo94perceptual.html

[12] W. Frei and C. C. Chen, "Fast boundary detection: A generalization and a new algorithm," *IEEE Trans. Comput.*, vol. C-26, no. 10, pp. 988–998, 1977.

[13] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Addison-Wesley, 1993.

[14] W. Research, "Positive definite matrix," http://mathworld.wolfram.com/PositiveDefiniteMatrix.html, accessed July 2003.

[15] A. S. Tom, "Prediction of fir pre- and post-filter performance based upon a visual model," Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1986.

[16] H. R. Sheikh, Z. Wang, L. Cormack, and A. C. Bovik, "LIVE image quality assessment database," 2002. [Online]. Available: http://live.ece.utexas.edu/research/quality

[17] R. A. Johnson, *Miller and Freund's Probability and Statistics for Engineers*, 6th ed. Upper Saddle River, NJ: Prentice-Hall, 2000.