Tree-Based Resolution Synthesis

C. Brian Atkins, Charles A. Bouman**, and Jan P. Allebach***

**Imaging Technology Department*
Hewlett-Packard Company, Hewlett-Packard Laboratories
Palo Alto, CA 94304-1126 / USA

***Electronic Imaging Systems Laboratory*
Purdue University, School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 / USA

Abstract

In this paper, we present an approach to optimal image scaling called Tree-Based Resolution synthesis (TBRS). TBRS works by first performing a fast local classification of a window around the pixel being interpolated, and then by applying an interpolation filter designed for the selected class. The idea behind TBRS is to use a regression tree as a piecewise linear approximation to the conditional mean estimator of the high-resolution image given the low-resolution image. We generate the parameters for the regression tree by training on sample images. The training is computationally demanding, but it only needs to be performed once. We will demonstrate that the resulting predictor may be used effectively on input images that were not used in the training.

1 Introduction

An image is usually only available at one or a few resolutions, so that in order to render it at a higher resolution, some image scaling technique must be applied. This involves some inference on the part of the interpolation algorithm; and for most images, the result is that the higher-resolution image exhibits some objectionable artifact of the interpolation process. In answer to this, we present an approach to optimal image scaling that we call Tree-Based Resolution Synthesis (TBRS).

TBRS works by first performing a fast local classification of a window around the pixel being interpolated, and then applying an interpolation filter designed for the selected class, as illustrated in Figure 1. The idea behind TBRS is to use a regression tree as a piecewise linear approximation to the conditional mean estimator of the high-resolution image given the low-resolution image. Intuitively, having the different regions of linear operation allows for separate filtering of distinct behaviors like edges of different orientation and smoother gradient transitions.

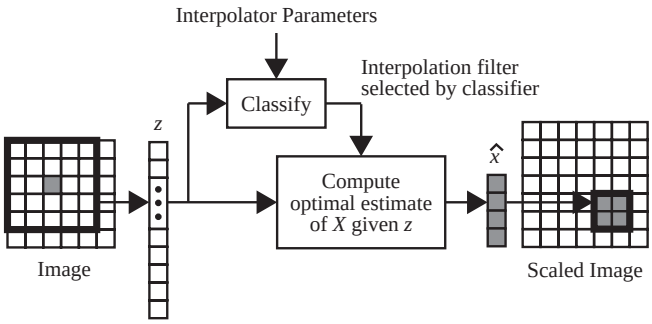

Figure 1. TBRS interpolation by a factor of 2

An overview of the TBRS algorithm appears in Figure 2. Note that before TBRS can be executed, we must already have generated the parameters for the regression tree by training on sample images. This training procedure requires considerable computation, but it only needs to be performed once. The resulting predictor may be used effectively on images that were not used in the training.

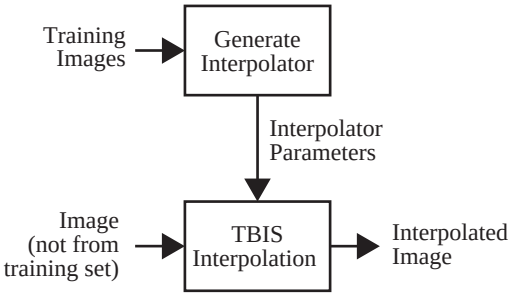

Figure 2. Overview of the TBRS algorithm.

We will see that TBRS represents a departure from traditional spline-based image interpolation techniques [1,2,3] that usually rely on some assumption about the continuous structure of an underlying interpolating function.

It is also separate from [4], which is based on a different underlying stochastic model. However, it does share some similarity with the wavelet-based approach described in [5], as well as with edge-directed techniques such as [6] and [7], since the purpose of the classification stage in TBRS is to locate edge regions and to treat them specially to lead to a scaled image having sharper appearance.

The remainder of this paper is organized as follows. In Sec. 2, we describe the TBRS procedure; and in Sec. 3, we develop the training procedure by which we obtain the parameters for the regression tree. In Secs. 4 and 5, we present results and conclusions. Note that when we refer to images, we mean monochrome images with pixel values in the range [0,255]. To interpolate color images, we interpolate the red, green, and blue planes separately.

2 TBRS Structure

In TBRS, we generate an $L \times L$ block of high-resolution pixels for every pixel in the low-resolution source image as illustrated in Figure 1. (Here, $L \geq 1$ denotes the scaling factor.) We do this by filtering the corresponding $W \times W$ window of pixels in the low-resolution image, with the filter coefficients selected based on a classification. (We will use $W = 5$, but generally W may take the value of any positive integer.) Thinking of the desired high-resolution pixels as an L^2-dimensional random vector X and the corresponding low-resolution pixels as the realization of a W^2-dimensional random vector Z, our approach is to use a regression tree which approximates the conditional mean estimator of $X \mid Z$, so that the vector $\hat{X}$ of interpolated pixels satisfies

$$\hat{X} \approx \mathrm{E}[X \mid Z]. \tag{1}$$

It is well known that the conditional mean estimator minimizes the expected mean-squared error [8]. (Note that we will use capital letters to represent random quantities, and lowercase letters for their realizations.)

A closed-form expression for the true conditional mean estimator would be difficult to obtain for the present context. However, the regression tree T that we use provides a convenient and flexible piecewise linear approximation, with the M different linear regions being polygonal subsets which comprise a partition of the sample space $\mathcal{Z}$ of low-resolution vectors Z. These polygonal subsets, or classes, correspond to visually distinct behaviors like edges of different orientation.

With the main ideas in place, we return to Figure 1 for a better look. To interpolate the shaded pixel in the low-resolution image, we first procure the vector z by stacking the pixels in the 5×5 window centered there. Then we obtain interpolated pixels as

$$\hat{x} = A_j z + \beta_j, \tag{2}$$

where A_j and β_j are respectively the $L^2 \times W^2$ matrix and L^2-dimensional vector comprising the interpolation filter for class j, and j is the index of the class obtained as

$$j = C_T(z), \tag{3}$$

where $C_T : \mathcal{Z} \to \{0,...,M-1\}$ is a function which embodies the classifying action of T. To evaluate $C_T(z)$, we begin at the top and traverse down the tree T as illustrated in Figure 3, making a decision to go right or left at each nonterminal node (circle), and taking the index j of the terminal node (square) which z lands in. Each decision has the form,

$$e_m^t (z - \mu_m) \begin{array}{c} > \\ < \end{array} 0, \tag{4}$$

where m is the index of the node, e_m and μ_m are W^2-dimensional vectors, and a superscript t denotes taking the transpose. This decision determines whether z is on one side of a hyperplane or the other, with μ_m being a point in the hyperplane and with e_m specifying its orientation. By convention, we go left if the quantity on the left-hand side is negative, and we go right otherwise.

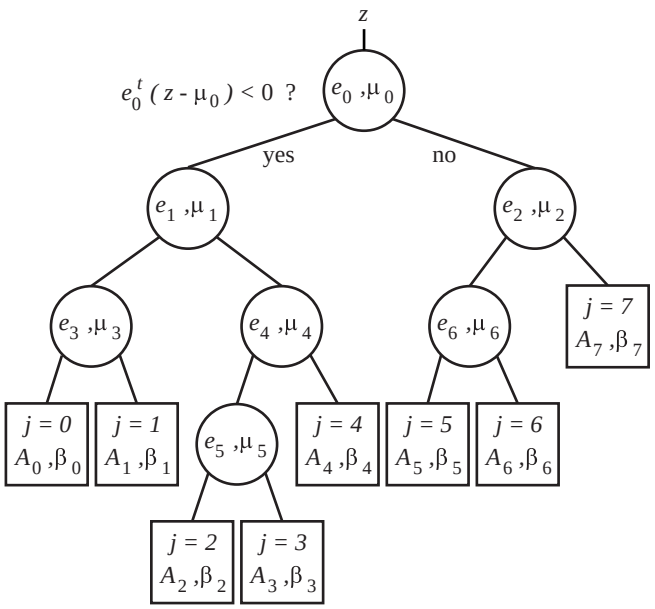

Figure 3. Binary tree structure used in TBRS.

3 Generating Parameters for TBRS

Our objective during training is to obtain numerical values for the integer number $M \geq 1$ of terminal nodes in the tree; the decision rules $\{(e_m, \mu_m)\}_{m=0}^{M-2}$ for the nonterminal nodes (assuming that $M > 1$); and the interpolation filters $\{(A_m, \beta_m)\}_{m=0}^{M-1}$ for the terminal nodes. To compute these

parameters, we use training vector pairs, which we assume are independent realizations of (X,Z). A training vector pair is extracted from low- and high-resolution renderings of the same image.

The training procedure, which is illustrated in Figure 4, is based on that given by Gelfand, Ravishankar, and Delp, in [9], suitably modified for the design of a regression tree rather than a classification tree. We first use one training set G to grow the tree, and then we use a different training set P to prune it back. Finally, we generate new interpolation filters for the terminal nodes using a very large training set D. An important difference between our procedure and that of Gelfand, Ravishankar, and Delp, is that our procedure only involves one growing phase followed by one pruning phase, rather than iterating through cycles of the growing-then-pruning process.

In the remainder of this section, we describe the tree growing phase in Sec. 3.1, the pruning phase in Sec. 3.2, and the final filter generation process in Sec. 3.3. Last, in Sec. 3.4, we describe how we generate the training images and how we procure the training sets G, P, and D.

We will refer to a tree-structured interpolator with ℓ terminal nodes as $T(\ell)$. To evaluate the quality of $T(\ell)$, we compute its sample mean-squared interpolation error. During the growing phase we compute the error from the set G, and we use the notation $\mathcal{E}_G(T(\ell))$. Note that this is a resubstitution estimate, and should not be regarded as an honest estimate of the true mean-squared error of $T(\ell)$ [10]. However, we only use $\mathcal{E}_G(\cdot)$ to compare trees generated during the growing phase. We similarly only use $\mathcal{E}_P(\cdot)$, computed relative to the pruning set P, to compare trees generated during the pruning phase.

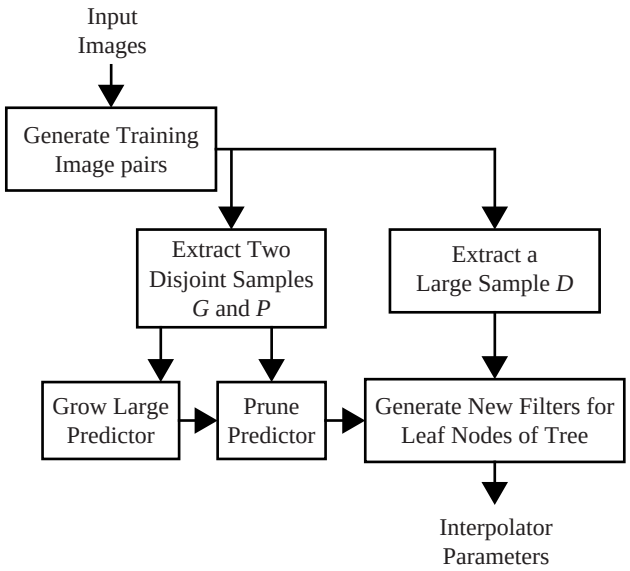

Figure 4. Overview of the TBRS training process.

3.1 Growing the tree-structured interpolator

During the tree growing phase, we generate an increasing sequence of trees

$$T(1), T(2), ..., T(M_{\max}),$$

where $M_{\max}$ is a user-specified maximum number of terminal nodes. We select $M_{\max}$ to be large enough so that $T(M_{\max})$ actually overfits the set G [9]. In our experiments, we have obtained reasonable interpolation results by setting $M_{\max}$ to 500. This tends to lead to final tree sizes (*i.e.*, after the pruning phase is complete) of between roughly 20 and 50 classes.

To generate $T(\ell+1)$ from $T(\ell)$ (for $\ell \geq 1$), we split the terminal node with index m^* that yields the greatest reduction in the sample mean-squared error. More specifically, we split terminal node m^* of $T(\ell)$ selected as

$$m^* = \arg \min_{0 \leq m < \ell} E_G(T'_m(\ell+1)), \tag{5}$$

where $T'_m(\ell+1)$ is the candidate tree with $\ell+1$ terminal nodes, generated by splitting node m in $T(\ell)$. Note here that we will use primes with variables associated with candidate trees.

To generate the candidate tree $T'_m(\ell+1)$ from $T(\ell)$, we do two things. First, we use the subset of G belonging to class m, defined as

$$G_m = \{(x,z) \in G : C_{T(\ell)}(z) = m\}, \tag{6}$$

to generate the candidate decision rule

$$(e'_{\ell-1}, \mu'_{\ell-1})$$

for the candidate nonterminal node in $T'_m(\ell+1)$. Note that we use the subscript $\ell-1$ since this will be the index of the new nonterminal node in $T(\ell+1)$. Second, we use the candidate decision rule to divide G_m into two sets G'_m and G'_ℓ, and we generate new interpolation filters

$$(\tilde{A}'_m, \tilde{\beta}'_m)$$

and

$$(\tilde{A}'_\ell, \tilde{\beta}'_\ell)$$

for the two candidate terminal nodes in $T'_m(\ell+1)$. Here, we use the indices m and ℓ since they would be the indices of the two new terminal nodes in $T(\ell+1)$. Note also that we will use tildes to denote interpolation filters generated during the growing stage of the training.

3.1.1. Splitting the *m*-th terminal node in $T(\ell)$.

By generating the candidate decision rule we split the polygonal subset of $\mathcal{Z}$ belonging to class m defined as

$$\mathcal{Z}_m = \{z \in \mathcal{Z} : C_{T(\ell)}(z) = m\}. \quad (7)$$

It is common for a split to have the form of a hyperplane in $\mathcal{Z}$, but the form which we will use is actually quite general since we make no restriction on the orientation of the hyperplane. Note that in this section, we restrict our attention to the m-th terminal node of $T(\ell)$. To keep the notation simple, we will be using X and Z to refer to the random variables $(X \mid Z \in \mathcal{Z}_m)$ and $(Z \mid Z \in \mathcal{Z}_m)$.

We define the split based on a vector $\overline{X}$, defined as

$$\overline{X} = B\tilde{A}_m Z, \quad (8)$$

where B is an $L^2 \times L^2$ matrix which subtracts the average of the elements in the vector $\tilde{A}_m Z$ from each element in $\tilde{A}_m Z$. That is, B is computed as

$$B = I - \frac{1}{L^2} O, \quad (9)$$

where I is an $L^2 \times L^2$ identity matrix and O is $L^2 \times L^2$ matrix of ones.

To understand how we selected $\overline{X}$, note that $\tilde{A}_m Z$ is the part of $\hat{X}$ that depends upon Z. We think of $\overline{X}$ as a "centered" version of the part of $\hat{X}$ that depends upon Z, since B renders $\overline{X}$ invariant to the average of the elements in $\tilde{A}_m Z$. By splitting on $\overline{X}$, we are forcing the splits to depend on the differences among the elements in $\hat{X}$, rather than on the elements of $\hat{X}$ themselves. Geometrically, this leads to more classes for structures based on differences, such as edges of different orientation. This can significantly improve interpolation quality.

The desired decision rule is obtained as

$$e'_{\ell-1} = \left(\hat{e}^{\,t}_{\overline{X}|m} B\tilde{A}_m\right)^t \quad (10)$$

and

$$\mu'_{\ell-1} = \hat{\mu}_{Z|m}, \quad (11)$$

where $\hat{e}_{\overline{X}|m}$ is the eigenvector corresponding to the largest eigenvalue of an estimate $\hat{\Sigma}_{\overline{X}|m}$ of the covariance matrix of $\overline{X}$, and $\hat{\mu}_{Z|m}$ is an estimate of the mean of Z. Observe that this decision rule splits realizations of Z, and it also yields a hyperplane which divides realizations of $\overline{X}$, since

$$\hat{e}^{\,t}_{\overline{X}|m} B\tilde{A}_m \left(Z - \hat{\mu}_{Z|m}\right) = \hat{e}^{\,t}_{\overline{X}|m} \left(\overline{X} - \hat{\mu}_{\overline{X}|m}\right), \quad (12)$$

where $\hat{\mu}_{\overline{X}|m}$ is an estimate of the mean of $\overline{X}$. The idea behind the form of this split is to divide the realizations of $\overline{X}$ with a hyperplane passing through the mean of $\overline{X}$ and perpendicular to the direction in which $\overline{X}$ varies the most. Using second-order statistics, a good way to estimate that direction is to take the eigenvector $e_{\overline{X}|m}$ corresponding to the largest eigenvalue of the covariance matrix of $\overline{X}$ [11].

We compute $\hat{\mu}_{Z|m}$ as the sample mean of the z's in G_m. Next, we obtain $\hat{e}_{\overline{X}|m}$ by computing $\hat{\Sigma}_{\overline{X}|m}$ and then performing an eigenvalue decomposition. We obtain $\hat{\Sigma}_{\overline{X}|m}$ as

$$\hat{\Sigma}_{\overline{X}|m} = B\tilde{A}_m \hat{\Sigma}_{Z|m} \tilde{A}^t_m B^t, \quad (13)$$

where $\hat{\Sigma}_{Z|m}$ is an estimate of the covariance matrix of Z, computed as

$$\hat{\Sigma}_{Z|m} = \frac{1}{N_{G_m}} \sum_{i=0}^{N_{G_m}-1} \left(z_i - \hat{\mu}_{Z|m}\right)\left(z_i - \hat{\mu}_{Z|m}\right)^t, \quad (14)$$

where N_{G_m} is the number of training vector pairs in G_m.

3.1.2. Generating interpolation filters for the candidate terminal nodes.

With the candidate decision rule computed, we separate the set G_m into two sets

$$G'_m = \left\{(x,z) \in G_m : (e'_{\ell-1})^t (z - \mu'_{\ell-1}) < 0\right\} \quad (15)$$

and

$$G'_\ell = \left\{(x,z) \in G_m : (e'_{\ell-1})^t (z - \mu'_{\ell-1}) \geq 0\right\} \quad (16)$$

Note that we use the indices m and ℓ since they would be the indices of the new terminal nodes in $T(\ell+1)$. The next step is to compute candidate interpolation filters for the candidate terminal nodes. Since the procedures are the same, we will only demonstrate for G'_m.

Our approach is to use maximum likelihood estimates of the parameters for the regression of X on $(Z \mid Z \in \mathcal{Z}'_m)$, where

$$\mathcal{Z}'_m = \left\{z \in \mathcal{Z}_m : (e'_{\ell-1})^t (z - \mu'_{\ell-1}) < 0\right\}. \quad (17)$$

This is a well-known result from the multivariate statistics [12]. The desired interpolation filters are computed as

$$\tilde{A}'_m = \hat{\Sigma}'_{XZ|m} \left(\hat{\Sigma}'_{Z|m}\right)^{-1} \quad (18)$$

and

$$\tilde{\beta}'_m = \hat{\mu}'_{X|m} - \hat{\Sigma}'_{XZ|m} \left(\hat{\Sigma}'_{Z|m}\right)^{-1} \hat{\mu}'_{Z|m}, \quad (19)$$

where

$$\hat{\mu}'_{Z|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} z_i, \quad (20)$$

$$\hat{\mu}'_{X|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} x_i, \quad (21)$$

$$\hat{\Sigma}'_{Z|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} \left(z_i - \hat{\mu}'_{Z|m}\right)\left(z_i - \hat{\mu}'_{Z|m}\right)^t , \quad (22)$$

$$\hat{\Sigma}'_{XZ|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} \left(x_i - \hat{\mu}'_{X|m}\right)\left(z_i - \hat{\mu}'_{Z|m}\right)^t . \quad (23)$$

Here, $N_{G'_m}$ is the numbers of elements in G'_m, and a superscript "–1" denotes taking the matrix inverse (or pseudo-inverse).

3.2 Pruning the tree-structured interpolator

The purpose of the pruning stage is to remove extraneous branches in $T(M_{\max})$, leaving behind a tree structure which efficiently represents a wide range of behaviors observable in images. Here we generate a decreasing sequence of pruned subtrees

$$T(M_{\max}), ..., T(M^*) ,$$

$M_{\max} \geq M^* \geq 1$, where each tree is obtained from the preceding tree by removing a branch. We point out that that $M_{\max}$ must be specified by the person programming the algorithm, but M^* actually falls out of the algorithm by itself.

We employ a procedure which is similar to that used in [13] and in [14]. The basic idea is to start at the terminal nodes and to work our way up toward the root node, pruning any nonterminal node along the way which yields a decrease in the estimated mean-squared error $\mathcal{E}_P(\cdot)$. The only requirement on the order is that we never consider pruning at any nonterminal node until after we have considered pruning at both its children. Our approach is to submit the root node of $T(M_{\max})$ to a recursive subroutine that first considers pruning at each child of the current nonterminal node (with a subroutine call to itself), and then considers pruning at the current nonterminal node.

After the pruning is complete, we regard the size and the shape of the final tree-structured interpolator T as fixed. That is,

$$M = M^* , \quad (24)$$

and the decision rules for the nonterminal nodes in T are defined to be those in $T(M^*)$ (assuming that $M^* > 1$). The interpolation filters are not defined to be those in $T(M^*)$, because we generate new ones after the pruning stage.

3.3 Generating new interpolation filters

The last step in the training procedure is to compute the interpolation filters in T using a set D of training vector pairs which is very large relative to the sizes of the sets G and P. The idea here is to use many more vector pairs than were used to generate the interpolation filters for the classes originally, during the tree growing stage. This improves performance for interpolating behaviors which are distinct enough to have warranted the formation of classes, but which may have been slightly under-represented in the growing set. We have observed that this improves performance in particular for higher-order interpolations, such as for an interpolation factor $L = 4$.

This process is formally identical to the procedure by which we generate the candidate interpolation filters, which we explained above.

3.4 Obtaining the training sets

We obtain the training vector pairs from a set of training images which consists of low-resolution images and corresponding high-resolution images. To generate the training images, we first procure a database of high-quality images from a photo CD image library. These are the input images in Figure 4. We create the low-resolution training images using simple $L \times L$ block-averaging. By sharpening the input training images to produce the high-resolution training images, we give the resulting interpolator a built-in sharpening effect.

Next we procure the training vector pairs which comprise the training sets G, P, and D. One requirement is that in order for the pruning stage to be effective, the sets G and P should be extracted from different pixels in the training images. It is also important that the pixels be from spatially separate regions of the training images, so that the set represents a wide variety of image data.

We selected the sizes of sets G and P to be equal and as large as possible, under the constraint that during either of the growing and pruning stages, all of the training set should be maintained in computer memory along with the rest of the executable. This speeds up the training process by making it unnecessary for the computer to refer to the disk for a training vector pair each time it is needed. In our experiments, each of G and P contained 300,000 training vector pairs. On the other hand, the training set D can be arbitrarily large, since each training vector pair in D only has to be accessed once. In our experiments, D contained between 5 and 20 million vector pairs.

4 Results

Here we present results of image scaling by a factor of $L = 2$, rendered at 75 dots per inch. None of the images in this section was among the training images.

In Figure 5, we show an image after default Photoshop sharpening followed by bilinear interpolation. (This was the best result we obtained using the sharpening functions in Photoshop.) In Figure 6, we show the same image scaled using TBRS with built-in sharpening described in Sec. 3.4. Note that the TBRS interpolation is sharper than the bilinear result, and the edges in the TBRS result are more continuous. We have also included Figure 7, to follow up on the part of Sec. 3.1 where we describe our method for splitting terminal nodes. Here we demonstrate the effect of centering the vector relative to which we determine the

splits, using the matrix B. Note that the result generated by splitting on a vector which was not centered exhibits objectionable "jaggie" artifacts around edges. These artifacts are not nearly so prominent in the standard TBRS result. An interesting fact is that although the same training sets were used to generate the two sets of interpolator parameters, the interpolator parameters for standard TBRS had only 23 classes, while the parameters generated without the centering had 40 classes. This implies that the centering makes the splitting process more efficient by encouraging the formation of visually distinct classes such as edges of different orientation, rather than wasting splits by defining classes for behaviors which are visually more similar.

Figure 5. Image processed by default sharpening in Photoshop followed by 2X bilinear interpolation.

Figure 6. Image scaled 2X using TBRS.

Figure 7. Result of not centering vector used to determine splits.

5 Conclusions

In this report, we have introduced TBRS, a method for image scaling. One advantage of TBRS is that it has a simple implementation. But we have also seen that since the TBRS interpolator is designed to approximate the conditional mean estimator of the high-resolution pixels given corresponding low-resolution pixels, it provides an approximately optimal (MMSE) interpolation. We have demonstrated the effectiveness of TBRS by showing results of scaling images that were not among the training images used to generate the predictor parameters.

References

1. M. Unser, A. Aldroubi, and M. Eden, Enlargement or reduction of digital images with minimum loss of information, *IEEE Trans. Image Proc.*, **4**(3), pp. 247-258, 1995.
2. R. G. Keys, Cubic convolution interpolation for digital image processing, *IEEE Trans. ASSP*, **29**(6), pp. 1153-1160, 1981.
3. H. S. Hou and H. C. Andrews, Cubic splines for image interpolation and digital filtering, *IEEE Trans. ASSP*, **26**(6), pp. 508-517, 1978.
4. R. R. Schultz and R. L. Stevenson, A Bayesian approach to image expansion for improved definition, *IEEE Trans. Image Proc.*, **3**(3), pp. 233-242, 1994.
5. S. G. Chang, Z. Cvetkovic, and M. Vetterli, Resolution enhancement of images using wavelet transform extrema extrapolation, *Proc. ICASSP*, vol. 4, pp. 2379-2382, 1995.
6. J. P. Allebach and P. W. Wong, Edge-directed interpolation, *Proc. ICIP*, pp. 707-710, 1996.
7. K. Jensen and D. Anastassiou, Subpixel edge localization and the interpolation of still images, *IEEE Trans. Image Proc.*, **4**(3), pp. 285-295, 1995.
8. L. L. Scharf, Statistical Signal Processing, Addison-Wesley, Reading, MA, 1991.
9. S. B. Gelfand, C. S. Ravishankar, and E. J. Delp, An iterative growing and pruning algorithm for classification tree design, *IEEE Trans. PAMI*, **13**(2), pp. 163-174, 1991.
10. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees, Wadsworth International Group, Belmont, CA, 1984.
11. M. T. Orchard and C. A. Bouman, Color quantization of images, *IEEE Trans. Signal Proc.*, **39**(12), pp. 2677-2690, 1991.
12. T. W. Anderson, An Introduction to Multivariate Statistics, John Wiley and Sons, New York, NY, 1958.
13. S. B. Gelfand, C. S. Ravishankar, and E. J. Delp, Tree-structured piecewise linear adaptive equalization, *IEEE Trans. Comm.*, **41**(1), pp. 70-82, 1993.
14. S. B. Gelfand, and C. S. Ravishankar, A tree-structured piecewise linear adaptive filter, *IEEE Trans. Info. Theory*, **39**(6), pp. 1907-1922, 1993.